<table>
<thead>
<tr>
<th>Title</th>
<th>Studies on Disjunctive Logic Programming (Dissertation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Sakama, Chiaki</td>
</tr>
<tr>
<td>Citation</td>
<td>Kyoto University (京都大学)</td>
</tr>
<tr>
<td>Issue Date</td>
<td>1995-01-23</td>
</tr>
<tr>
<td>URL</td>
<td><a href="https://doi.org/10.11501/3099098">https://doi.org/10.11501/3099098</a></td>
</tr>
<tr>
<td>Type</td>
<td>Thesis or Dissertation</td>
</tr>
<tr>
<td>Textversion</td>
<td>author</td>
</tr>
</tbody>
</table>

Kyoto University
STUDIES ON
DISJUNCTIVE LOGIC
PROGRAMMING

by

CHIAKI SAKAMA

A dissertation submitted to
the Faculty of Engineering of
Kyoto University
for the degree of Doctor of Engineering

July 1994
Preface

Recent studies have enriched the expressive power of logic programming as a knowledge representation tool and the growing importance of logic programming in artificial intelligence is well recognized these days. Disjunctive logic programming is one of such extensions of logic programming which provides us with the ability of reasoning with indefinite information. Due to their expressiveness, disjunctive logic programming has been given an increasing attention over the past few years.

In this dissertation, we study theoretical frameworks for disjunctive logic programming. Our particular interest is in the semantic issues of disjunctive logic programs and their correspondences to commonsense reasoning in artificial intelligence.

As a semantics of disjunctive logic programs, we propose a new declarative semantics called the possible model semantics. The possible model semantics is an alternative theoretical framework for disjunctive logic programs, which provides a flexible inference mechanism for representing knowledge and also has a computational advantage over the classical minimal model semantics.

To relate disjunctive logic programs to commonsense reasoning in artificial intelligence, we propose transformations from disjunctive logic programs to various forms of nonmonotonic reasoning such as default logic, circumscription, and autoepistemic logic. Moreover, we discuss connections between disjunctive logic programs and abductive logic programs, and reveal close relationships between each framework.

Another important issue for commonsense reasoning is the treatment of inconsistent knowledge. Since classical logic programming is not useful in inconsistent programs, we introduce paraconsistent semantics for disjunctive logic programs which provide uniform frameworks for handling both indefinite and inconsistent information in a program.

We finally discuss program optimization issues in disjunctive logic programs. A technique of partial deduction in logic programming is extended to disjunctive logic programs and its correctness is presented.
Acknowledgments

I would like to express my sincere gratitude to my thesis advisor, Shuji Doshita, for his continuous guidance and encouragement. I would also like to thank the judging committee members, Yoshihiro Matsumoto and Toru Ishida.

This research was started when I was at ICOT (Institute for New Generation Computer Technology), and was continued at ASTEM (Advanced Software Technology and Mechatronics Research Institute of Kyoto). I would like to thank Kazuhiro Fuchi and Yutaka Ohno for providing me with opportunities to pursue this research. My appreciation also goes to my research managers, Hidenori Itoh, Shun-ichi Uchida, and Koichi Furukawa for their helpful advice and encouragement.

Special thanks are due to Susumu Yamashita, who guided me to the area of logic programming when I was at Kyoto University. I am also grateful to my research colleagues for their fruitful discussions and valuable comments. Though I cannot list all their names, my special thanks go to Katsumi Inoue and Hirohisa Seki.

Discussions with researchers in the areas of logic programming and nonmonotonic reasoning are always stimulating and helpful. In particular, I would like to thank Edward P. F. Chan, Hendrik Decker, Jurgen Dix, Michael Gelfond, Georg Gottlob, John Grant, Jack Minker, and V. S. Subrahmanian for useful correspondence on the work presented in this dissertation.
3.2.1 Negation in Positive Disjunctive Programs ...
3.2.2 Possible Model Semantics ...
3.3 Possible Model Semantics for Normal Disjunctive Programs ...
3.4 Fixpoint Semantics ...
3.4.1 Fixpoint Semantics for Positive Disjunctive Programs ...
3.4.2 Fixpoint Semantics for Normal Disjunctive Programs ...
3.5 Computing Possible Models ...
3.5.1 Bottom-up Model Generation Procedure ...
3.5.2 Query Answering ...
3.6 Computational Complexity ...
3.7 Discussion ...
3.7.1 Declarative Semantics ...
3.7.2 Fixpoint Semantics ...
3.7.3 Proof Procedure ...
3.8 Summary ...
4 Relating Disjunctive Logic Programs to Default Theories ...
4.1 Introduction ...
4.2 Default Logic ...
4.3 Default Translation of Normal Disjunctive Programs ...
4.3.1 Positive Default Theory Revisited ...
4.3.2 Representing Normal Disjunctive Programs by Default Theories ...
4.4 Default Translation of Extended Disjunctive Programs ...
4.4.1 Representing Extended Disjunctive Programs by Default Theories ...
4.4.2 Relationship to Disjunctive Default Theory ...
4.5 Connections with Autoepistemic Logic and Circumscription ...
4.5.1 Autoepistemic Logic ...
4.5.2 Circumscription ...
4.5.3 Characterizing Possible Model Semantics ...
4.6 Summary ...
5 Equivalence between Disjunctive and Abductive Logic Programs ...
5.1 Introduction ...
5.2 Abductive Logic Programming ...
5.3 Connections between Disjunctive and Abductive Logic Programs ...
5.3.1 Generalized Stable Models are Possible Models ...
5.3.2 Possible Models are Generalized Stable Models ...
5.4 Abductive Disjunctive Programs ...
5.4.1 Generalized Disjunctive Stable Models and Possible Models ...
5.4.2 Generalized Possible Models are Generalized Stable Models ...
5.5 Discussion ...
5.6 Summary ...
6 Handling Inconsistency in Disjunctive Logic Programs ...
6.1 Introduction ...
6.2 Paraconsistent Semantics for Positive Extended Disjunctive Programs ...
6.2.1 Multi-valued Logic ...
6.2.2 Paraconsistent Possible Model Semantics ...
6.3 Paraconsistent Semantics for Extended Disjunctive Programs ...
6.3.1 Paraconsistent Stable Model Semantics ...
6.3.2 Paraconsistent Possible Model Semantics ...
6.3.3 Connection to the Answer Set Semantics ...
6.4 Fixpoint Semantics of Extended Disjunctive Programs ...
6.5 Reasoning with Inconsistency ...
6.5.1Preferred Stable Models ...
6.5.2 Suspicious Stable Models ...
6.5.3 Semi-Stable Models ...
6.6 Related Work ...
6.7 Summary ...
7 Partial Deduction of Disjunctive Logic Programs ...
7.1 Introduction ...
7.2 Partial Deduction of Positive Disjunctive Programs ...
7.2.1 Normal Partial Deduction ...
7.2.2 Disjunctive Partial Deduction ...
7.3 Partial Deduction of Normal Disjunctive Programs ...
7.3.1 Disjunctive Partial Deduction of Normal Disjunctive Programs ...
7.3.2 Connections between Normal and Disjunctive Partial Deduction ...
7.3.3 Preservation of the Possible Model Semantics ...
7.4 Goal-Oriented Partial Deduction ...
7.5 Discussion ...
7.6 Summary ...)
Chapter 1

Introduction

All parties to the debate agree that a central goal of research is that computers must somehow come to "know" a good deal of what every human being knows about the world and about organisms, natural and artificial, that inhabit it. This body of knowledge - indefinite, no doubt, in its boundaries - goes by the name "common sense".

— David Israel [1983].

After the discovery of resolution principle by J. Alan Robinson [1965a], logic based approaches to artificial intelligence (AI) have been developed in the area of automated theorem proving in the late 1960's [Chang and Lee, 1973]. In the early 1970's, Kowalski [1974] attached a procedural interpretation to Horn clause logic and introduced the framework of logic programming. This leads to the design of the programming language PROLOG by Colmerauer and his group [1973]. After the success of PROLOG, the logic programming framework has been employed in many AI projects in the world, including the Japanese Fifth Generation Computer Project in the 1980's.

A prominent feature of logic programming is its formal logical basis. That is, a logic program is viewed as a set of axioms, its computation corresponds to deduction, and the meaning of a program is exactly the logical consequences of the program. It is worth noting that such a formal foundation of logic programming enables researchers in this field to communicate with each other using mathematical logic as a common language. This is an important and distinguished advantage of logic programming which is never seen in the fields of expert systems nor object-oriented databases today. This unique feature of logic programming has promoted theoretical studies in its own right, and found various applications ranging from databases to artificial intelligence [Gallaire et al., 1984; Kowalski, 1991; Baral and Gelfond, 1994].
CHAPTER 1. INTRODUCTION

Another important feature of logic programming is the separation of logic and control [Kowalski, 1979]. That is, logic programs specify declarative sentences representing knowledge in the world, while their algorithms are designed independent of the contents of programs. Such a separation is also important from the viewpoint of knowledge representation in artificial intelligence [McCarthy and Hayes, 1969].

A declarative meaning of a logic program is characterized by the declarative semantics, which is usually given by the model theory of first-order logic. On the other hand, a computational aspect of a logic program is characterized by the procedural semantics, which is provided as a proof procedure of the program. Thus, declarative and procedural semantics characterize two different aspects of logic programming.

The semantics of logic programming was firstly studied by van Emden and Kowalski [1976], in which they introduced declarative and procedural semantics of Horn logic programs. Horn logic programs are the simplest class of logic programming and have applications such as the programming language PROLOG or deductive databases [Gallaire et al., 1984].

Horn logic programs have powerful computational mechanisms in the sense that they are as expressive as a Turing machine [Tärnland, 1977]. Considering Horn logic programs as a knowledge representation language, however, they describe only definite information in the world and provide no inference mechanism for reasoning with incomplete information. In fact, these limitations have often caused a criticism that knowledge representations based on formal logics and deductive inference are not useful for commonsense reasoning in AI.

The necessity of dealing with incomplete information in knowledge representation is addressed by Levesque [1983]:

The reason incomplete knowledge bases are so important is that, in many applications, the knowledge base undergoes a continual evolution. At each stage, information can be acquired that is potentially very vague or indefinite in nature. More important, a problem solving system cannot simply wait for the knowledge base to stabilize in some final and complete form since this may never happen.

Then, in order to overcome those limitations, attempts have been done to enhance the expressiveness and inference abilities of logic programming. Such extensions are mainly achieved in two ways.

First extension is the representation of negation. In a definite Horn logic program, each fact derived from the program represents true knowledge in the program, while any fact unproved in the program is assumed to be false as default negation or negation as failure to prove [Clark, 1978; Reiter, 1978]. Such default negation can augment incomplete knowledge in a program, while it makes logic programming semantics nonmonotonic. Together with this non-classical mechanism, the framework of logic programming was extended from Horn logic programs to non-Horn logic programs by incorporating default negation in programs. Stratified logic programs [Chandra and Harel, 1985; Apt et al., 1988; van Gelder, 1988] and normal logic programs [Gelfond and Lifschitz, 1988; van Gelder et al., 1991] are such extensions. Those frameworks were further extended by Gelfond and Lifschitz [1990] to extended logic programs which include classical negation as well as default negation.

Second extension is the representation of indefinite or disjunctive information. Logic programming which can represent such information is called disjunctive logic programming. A theoretical framework of disjunctive logic programming was firstly studied by Minker [1982] for positive disjunctive programs, and then generalized to normal disjunctive programs [Lobo et al., 1992], and extended disjunctive programs [Gelfond and Lifschitz, 1991] by including negation.

Negation and disjunctions are two important extensions of logic programming which provide abilities to deal with incomplete information in a program. Those extensions of logic programming greatly increase the expressive power of logic programming as a knowledge representation tool and realize commonsense reasoning in artificial intelligence.

With these backgrounds, in this dissertation we study theoretical frameworks of disjunctive logic programming.

1.1 Motivations and Objectives

Most of the semantics of logic programming have traditionally been studied based on the notion of minimal models. The notion of minimal models reflects the so-called Occam's razor in the sense that a minimal model contains exactly as many facts as required to hold in a program. Such a principle of minimality has been supported in the area of not only logic programming, but also nonmonotonic logics in artificial intelligence. Therefore, it has been recognized that the principle of minimality is one of the most basic and indispensable criteria that each semantics for commonsense reasoning should obey [Schlipf, 1992a]. In fact, the least Herbrand model semantics of Horn logic programs [van Emden and Kowalski, 1976], the perfect model semantics of stratified logic programs [Przymusinski, 1988a], and the stable model semantics of normal

---

CHAPTER 1. INTRODUCTION

logic programs [Gelfond and Lifschitz, 1988] are all based on the principle of minimal-ity. This is also the case in the context of disjunctive logic programming, namely, the minimal model semantics of positive disjunctive programs [Mugni, 1982] and the disjunctive stable model semantics of normal disjunctive programs [Przymusinski, 1991a; Gelfond and Lifschitz, 1991] are both minimal.

However, such a minimalism is not always appropriate in a program containing indefinite information. Ross and Topor [1988] have firstly noticed this problem in the context of inferring negation in disjunctive logic programs. They argue that when one infers negation from a disjunctive logic program, one should be cautious to interpret disjunctions inclusively rather than exclusively. In fact, the minimal model semantics minimizes truth facts, then it usually interprets disjunctions exclusively and maximizes negative information inferred from a program.

Then Ross and Topor gave a framework for inferring negation in inclusive disjunctive logic programs, which is different from the minimal model semantics. However, the problem is that their framework, on the contrary, has difficulty for treating exclusive disjunctions in a program. Moreover, they provided no model theoretical meaning for inclusive disjunctive logic programs as a counterpart of the minimal model semantics. Thus our first objective is to give a theoretical framework of disjunctive logic programs which can distinguish both exclusive and inclusive disjunctions uniformly in a program.

Our second objective is to clarify the relations between disjunctive logic programs and various forms of commonsense reasoning in artificial intelligence. Recent studies have revealed close relationships between logic programming and non-monotonic reasoning [Reiter, 1982; Lifschitz, 1985, Gelfond, 1987; Lifschitz, 1988; Przymusinski, 1988b; Gelfond et al., 1989; Bidoit and Froidevaux, 1991a, Bidoit and Froidevaux, 1991b], and there are increasing interests to investigate interrelations between the two areas [Nerode et al., 1991; Pereira and Nerode, 1993]. These studies are important both for logic programming and artificial intelligence. Representing logic programs in terms of nonmonotonic formalisms helps us to realize commonsense reasoning in logic programming, on the other hand, it opens possibilities for using logic programming proof procedures as inference engines for nonmonotonic reasoning. Thus, clarifying such interrelations enables us to use techniques developed in one area by the other, and it will help researchers in each community to progress their work and enrich perspectives. Until now, those interrelations have been mainly studied for normal logic programs, then our next goal is to extend those previously studied results to the case of disjunctive logic programs.

1.1. MOTIVATIONS AND OBJECTIVES

Abduction is also a form of commonsense reasoning to which much attention has been paid recently. Abduction supplies an ability to perform reasoning with hypotheses, and its growing importance in various AI problems is widely recognized. In the context of logic programming, abduction is realized by the framework of abductive logic programming [Kakas et al., 1992]. Abductive logic programming is an extension of logic programming and also realizes reasoning with incomplete information in a program. In this regard, abductive logic programs and disjunctive logic programs appear to deal with very similar problems from different viewpoints. However, each formalism has been independently developed so far and has different syntax and semantics from the other. Then our primary interest is whether there is any formal correspondences between those two frameworks.

Another important aspect of commonsense reasoning in logic programming is the treatment of inconsistent knowledge. In practical situations, inconsistency is likely to happen as well as indefiniteness when we build a large scale of knowledge base. In such a knowledge base, it may be the case that a program contains local inconsistency and yet might have a global intended meaning. However, as traditional logic programming is based on classical first-order logic, a piece of inconsistent information makes a whole program trivial. A logic which is not destructive in the presence of inconsistent information is called paraconsistent logic, and its application to logic programming is known as paraconsistent logic programming [Blair and Subrahmanian, 1989]. Then, in order to treat both inconsistent and indefinite information in a program, paraconsistent frameworks for disjunctive logic programs are necessary.

Our last objective is an optimization issue of disjunctive logic programs. In logic programming, program transformation and optimization are important from practical viewpoints since correctly specified programs are not necessarily efficient programs. Partial deduction or partial evaluation is known as one of such optimization techniques in logic programming. Partial deduction derives a more specific program through performing deduction on a part of the program, while retaining the original meaning of the program. Such a specialized program is usually more efficient than the original program when executed. Partial deduction techniques have been mainly studied for normal logic programs so far. However, since computation of disjunctive logic programs is generally expensive than normal logic programs, it is necessary to develop partial deduction techniques for disjunctive logic programs also.

To summarize, the objectives of this dissertation are to study various aspects of disjunctive logic programming from both theoretical and practical points of view.
CHAPTER 1. INTRODUCTION

1.2 Contributions

The contributions of the research reported in this dissertation are as follows.

1. We propose a new declarative semantics of disjunctive logic programs called the possible model semantics. The possible model semantics is a different theoretical framework from the classical minimal model semantics, and provides a flexible negative inference mechanism under the closed world assumption. A new fixpoint semantics of disjunctive logic programs is introduced to characterize the possible model semantics and its proof procedure is presented. It is shown that the possible model semantics also has a computational advantage over the minimal and the stable model semantics of disjunctive logic programs.

2. We show a method of representing disjunctive logic programs in terms of default logic [Reiter, 1980]. The problem of previously proposed approaches is pointed out, and an alternative correct transformation from disjunctive logic programs to default theories is proposed. We also present the correspondences between disjunctive logic programs and other forms of nonmonotonic reasoning in AI such as disjunctive default logic [Gelfond et al., 1991], circumscription [McCarthy, 1980], and autoepistemic logic [Moore, 1985].

3. We reveal a close relationship between disjunctive logic programs and abductive logic programs. It is shown that the possible models of disjunctive logic programs are essentially equivalent to the generalized stable models of abductive logic programs [Kakas and Mancarella, 1990]. We also show that the possible model semantics is useful to characterize abductive logic programs from the computational complexity point of view.

4. We propose paraconsistent frameworks for disjunctive logic programs which can distinguish inconsistent information from other information in a program. Paraconsistent semantics for extended disjunctive programs are introduced which are based on lattice-structured multi-valued logics. Fixpoint characterizations of those paraconsistent semantics are presented, and methods for reasoning with inconsistency are discussed.

5. We develop partial deduction techniques for disjunctive logic programs. In disjunctive logic programs, normal partial deduction is shown to be not useful, and a new partial deduction method for disjunctive logic programs is introduced. It is shown that the proposed partial deduction preserves the meanings of disjunctive logic programs, and its application to goal-oriented partial deduction is presented for query optimization.

1.3 Outline of the Dissertation

This dissertation consists of eight chapters. The rest of the dissertation is organized as follows.

In Chapter 2, we first give basic notions of logic programming and other related notions used in this dissertation. We introduce the framework of disjunctive logic programming and review previously studied results on logic programming and disjunctive logic programming.

In Chapter 3, we propose a new theoretical framework of disjunctive logic programs. The possible model semantics is introduced as a declarative semantics of disjunctive logic programs. Negative inference under the possible model semantics is presented and its properties are discussed. A new fixpoint semantics of disjunctive logic programs is introduced to characterize the possible model semantics. For the procedural part, a bottom-up proof procedure is provided to compute the possible model semantics of positive and normal disjunctive programs. We also compare computational complexities of various semantics of disjunctive logic programs and show that the possible model semantics has a computational advantage over other proposed semantics.

In Chapter 4, we present a method of relating disjunctive logic programs to Reiter's default logic. We first point out the problem of previously studied results, then establish a correct default translation of disjunctive logic programs. We show a one-to-one correspondence between the stable models of a disjunctive logic program and the extensions of its associated default theory. Next we extend the results to extended disjunctive programs and investigate the connection with Gelfond et al.'s disjunctive default logic. The stable model semantics of disjunctive logic programs is also characterized by Moore's autoepistemic logic and McCarthy's circumscription. We also show that the possible model semantics is naturally expressed in terms of autoepistemic logic.

In Chapter 5, we present the equivalence relationship between disjunctive logic programs and abductive logic programs. We show that the generalized stable models of abductive logic programs can be translated into the possible models of disjunctive logic programs, and vice versa. It is also shown that abductive disjunctive programs can be expressed by abductive logic programs under the possible model semantics. Moreover, we observe that when considering the disjunctive stable model semantics instead of the possible model semantics, it is unlikely that disjunctive logic programs can be efficiently expressed in terms of abductive logic programs. The usefulness
of the possible model semantics for abductive logic programs is also verified by its computational complexity.

In Chapter 6, we propose declarative semantics of possibly inconsistent disjunctive logic programs. We introduce the paraconsistent minimal and stable model semantics for extended disjunctive programs, which can distinguish inconsistent information from other information in a program. The possible model semantics introduced in Chapter 3 is also extended to the paraconsistent possible model semantics in extended disjunctive programs. These semantics are natural extensions of the answer set semantics of extended disjunctive programs [Gelfond and Lifschitz, 1991], and are based on lattice-structured multi-valued logics. Paraconsistent semantics are also characterized by a fixpoint semantics of extended disjunctive programs, and methods for reasoning in inconsistent programs are addressed.

In Chapter 7, we present methods of partial deduction for disjunctive logic programs. We first show that normal partial deduction is not useful for disjunctive logic programs, then introduce disjunctive partial deduction for disjunctive logic programs. It is proved that disjunctive partial deduction preserves the minimal model semantics of positive disjunctive programs and the disjunctive stable model semantics of normal disjunctive programs. We also show that together with suitable program transformations, normal partial deduction can compute disjunctive partial deduction and also preserves the possible model semantics. The proposed partial deduction method is applied to goal-oriented partial deduction for query optimization.

In Chapter 8, we conclude the dissertation and address future directions of the research.

1.4 Publications

Some chapters of this dissertation are based on the published papers as follows.

Chapter 3 is based on the papers [Sakama, 1989] and [Sakama and Inoue, 1993a], which were presented at the First International Conference on Deductive and Object-Oriented Databases (DOOD'89; Kyoto, Japan, December 1989) and the Tenth International Conference on Logic Programming (ICLP'93; Budapest, Hungary, June 1993), respectively. Also much of this chapter is in [Sakama and Inoue, 1994c] which will appear in the Journal of Automated Reasoning.

Chapter 4 is based on the paper [Sakama and Inoue, 1993b] which was presented at the Second International Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR'93; Lisbon, Portugal, June 1993).

Chapter 5 is based on the paper [Sakama and Inoue, 1994a] which was presented at the Eleventh International Conference on Logic Programming (ICLP'94; S. Margherita Ligure, Italy, June 1994).

Chapter 6 is based on the paper [Sakama and Inoue, 1994b] which will appear in the Journal of Logic and Computation. Related topic is also discussed in the paper [Sakama, 1992] which was presented at the International Conference on Fifth Generation Computer Systems (FGCS'92; Tokyo, Japan, June 1992).

Chapter 7 is based on the paper [Sakama and Seki, 1994] which was presented at the Fourth International Workshop on Logic Program Synthesis and Transformation (LOPSTR'94; Pisa, Italy, June 1994). Related topic is also presented in the paper [Sakama and Itoh, 1998] which was published in the Journal of New Generation Computing, vol. 6, Nos. 2-3.

The dissertation does not include those topics presented in author's published papers [Sakama and Itoh, 1988] and [Sakama and Okumura, 1988]. These papers also discuss theoretical aspects of logic programming and nonmonotonic reasoning from different viewpoints.
Chapter 2

Preliminaries

This chapter introduces basic notions and terminologies used in this dissertation and reviews previously studied results on logic programming.

2.1 First-Order Logic

A framework of logic programming stems from classical first-order logic. Then we start from a brief overview of first-order theories. Terminologies and notations presented in this section are based on [Lloyd, 1987; Apt, 1990].

2.1.1 Language

An alphabet of a first-order language consists of a set of constants, variables, function symbols, predicate symbols, and usual punctuation symbols, together with connectives ∧, ∨, ¬, =, and quantifiers ∃ and ∀.

A term is defined inductively as either a variable or a constant or an expression of the form f(t1, ..., tn) where f is a function symbol and tᵢ's are terms.

A formula is defined inductively as follows:

(i) An atom p(t₁, ..., tₙ) is a formula where p is a predicate symbol and tᵢ's are terms.


(iii) For a formula F and a variable x, ∃xF and ∀xF are formulas.

In the context of logic programming, the connective — is also used for implication.
A formula is closed if it contains no free occurrences of any variable (that is, any variable is bounded by some quantifier). A term or a formula containing no variables is called ground. A ground formula is also called a proposition or a propositional formula. A first order language \( L \) over an alphabet is defined as the set of all formulas constructed from the symbols of the alphabet. A literal is an atom \( A \) or its negation \( \neg A \). A literal is called positive if it is an atom, otherwise it is called negative.

A clause is a formula of the form:

\[
\forall x_1, \ldots, \forall x_k (A_1 \lor \ldots \lor A_l \lor \neg B_1 \lor \ldots \lor \neg B_m)
\]

where each \( A_i \) (\( 1 \leq i \leq l \)) and \( B_j \) (\( 1 \leq j \leq m \)) are atoms and \( x_1, \ldots, x_k \) are variables occurring in the formula. A clause is also written as:

\[
\forall x_1, \ldots, \forall x_k (A_1 \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_m)
\]

or simply as:

\[
A_1 \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_m
\]

The left-hand side of the clause (2.1) is called the head, and the right-hand side of the clause is called the body. Note that any closed first-order formula can be transformed to clausal form [Chang and Lee, 1973].

2.1.2 Model Theory

The meaning of a first-order theory is given by an interpretation of formulas.

An interpretation \( I \) for a first-order language \( L \), called the domain of \( I \), and an assignment defined as follows:

- each constant in \( L \) is assigned to an element in \( D \),
- each \( n \)-ary function in \( L \) is assigned to a mapping from \( D^n \) to \( D \),
- each \( n \)-ary relation in \( L \) is assigned to a mapping from \( D^n \) to \{true, false\},

where true and false are the truth values assigned to propositions.

Given an interpretation \( I \) of \( L \), a variable assignment \( V \) is an assignment to each variable in \( L \) of an element in the domain of \( I \). The term assignment (with respect to \( I \) and \( V \) of the terms in \( L \) is defined as follows:

- for each variable in \( L \), the assignment according to \( V \),
- for each constant in \( L \), the assignment according to \( I \),

2.1. FIRST-ORDER LOGIC

- for each \( n \)-ary function \( f(t_1, \ldots, t_n) \) in \( L \), the assignment \( f'(t'_1, \ldots, t'_n) \) where \( f' \) is the assignment of \( f \) according to \( I \), and \( t'_1, \ldots, t'_n \) are the term assignments of \( t_1, \ldots, t_n \).

Now given an interpretation \( I \) and a variable assignment \( V \), truth values of formulas are inductively defined as follows.

(i) if the formula is an atom \( p(t_1, \ldots, t_n) \), then its truth value is obtained by calculating the value of \( p'(t'_1, \ldots, t'_n) \), where \( p' \) is the mapping assigned to \( p \) by \( I \) and \( t'_1, \ldots, t'_n \) are the term assignments of \( t_1, \ldots, t_n \) with respect to \( I \) and \( V \).

(ii) Given formulas \( F \) and \( G \),

- \( \neg F \) is true (resp. false) iff \( F \) is false (resp. true),
- \( F \lor G \) is true (resp. false) iff either \( F \) or \( G \) is true (resp. both \( F \) and \( G \) are false),
- \( F \land G \) is true (resp. false) iff both \( F \) and \( G \) are true (resp. either \( F \) or \( G \) is false),
- \( F \equiv G \) is true (resp. false) iff either \( F \) is false or \( G \) is true (resp. both \( F \) is true and \( G \) is false),
- \( F \supset G \) is true (resp. false) iff both \( F \supset G \) and \( G \supset F \) are true (resp. either \( F \supset G \) or \( G \supset F \) is false).

(iii) The formula \( \exists x F \) is true iff there exists \( d \in D \) such that \( F \) has a truth value true with respect to \( I \) and \( V(x/d) \), where \( V(x/d) \) is \( V \) except that \( x \) is assigned to \( d \); otherwise its truth value is false.

(iv) The formula \( \forall x F \) is true iff for all \( d \in D \), \( F \) has a truth value true with respect to \( I \) and \( V(x/d) \); otherwise its truth value is false.

A formula \( F \) is satisfied in an interpretation \( I \) (written \( I \models F \)) iff \( F \) is true in \( I \) for any variable assignment \( V \). In particular, when \( F \) is closed, the truth value of the formula is independent of \( V \). Given a set of formulas \( S \), an interpretation \( I \) is called a model of \( S \) if \( I \) satisfies every formula in \( S \). A set of formulas \( S \) is called satisfiable or consistent if it has a model; otherwise \( S \) is called unsatisfiable or inconsistent. When every interpretation is a model for \( S \), we say that \( S \) is valid.
2.2 Logic Programming

In computer science, logic plays an important role for designing, analyzing, and reasoning about computer programs. On the other hand, logic programming, advocated by Kowalski [1974], uses first-order predicate logic itself as a procedural programming language. Logic programming has a very simple syntax and a clear semantics based on the formal mathematical logic and the theory of resolution proof procedures. Compared with conventional programming languages, logic programming provides capabilities to conventional relational databases [Ullman, 1982].

In this section, we review the framework of logic programming and present previously studied results (not necessarily in a comprehensive manner).

2.2.1 Horn Logic Programs

An early stage of logic programming is defined as a subset of first-order clauses of the form (2.1), called Horn clauses [Kowalski, 1974].

A Horn logic program is a finite set of Horn clauses of the form:

\[ A_1 \leftarrow B_1 \land \ldots \land B_m \quad (0 \leq l \leq 1; m \geq 0) \]  

(2.2)

where \( A_1 \) and \( B_i \)'s are atoms.

A Horn clause with a non-empty head \( (l = 1) \) is called a definite clause, and a definite clause with the empty body \( (m = 0) \) is called a unit clause. A Horn clause with the empty head \( (l = 0) \) and a non-empty body \( (m \neq 0) \) is called a negative clause, which is also called a goal or an integrity constraint depend on its usage. A Horn logic program including only definite clauses is called a definite logic program.

From the database point of view, a set of Horn clauses attaches deductive capabilities to conventional relational databases in the context of deductive databases [Gallaire et al., 1984]. For example, in a definite logic program it is easy to specify the transitive closure of a relation by recursive clauses like that:

\[ \text{ancestor}(x, y) \leftarrow \text{parent}(x, z) \land \text{ancestor}(z, y), \]

\[ \text{ancestor}(x, y) \leftarrow \text{parent}(x, y), \]

which cannot be expressed in relational databases [Ullman, 1982].

The declarative semantics of logic programming is usually defined by means of particular models of a program, called Herbrand models.
P has an Herbrand model $M$ if its ground program has the Herbrand model $M$. Thus, any logic program $P$ is semantically identified with its ground program, so we usually assume ground programs in this dissertation when considering semantics of logic programs.

The declarative semantics of Horn logic programs is defined as the smallest Herbrand model, called the least Herbrand model. The least Herbrand model is also obtained as the intersection of all Herbrand models of the program [van Emden and Kowalski, 1976]. A definite logic program is always consistent and has the least Herbrand model.

The procedural semantics of logic programming is defined by proof procedures in two ways. A bottom-up proof procedure is based on usual modus ponens and starts from the given facts and proceeds forward to the conclusions. On the other hand, Kowalski [1974] introduced a procedural interpretation of Horn logic programs in which the head of a definite clause is viewed as a procedure name and the body of the clause is viewed as a set of procedural calls. A top-down proof procedure of logic programming is based on this idea. It starts from a given goal to be proved and goes backward to premises in a program by iteratively producing its subgoals.

For definite logic programs, van Emden and Kowalski [1976] present bottom-up hyperresolution and show that the set of all ground atoms derived from a definite logic program by hyperresolution coincides with the least Herbrand model of the program. The least Herbrand model is also characterized by the success set of top-down SLD-resolution [Apt and van Emden, 1982]. Various proof procedures for (function-free) definite logic programs are also studied in the context of query-answering in deductive databases [Bancilhon and Ramakrishnan, 1988].

The least Herbrand model assigns the truth value true to each atom included in the model, and remained other atoms are interpreted as false. Proof-theoretically speaking, however, definite logic programs allow to derive only positive consequences and negative facts are never derived from a program. To infer negative information in a program, we can use negative clauses to represent false facts in a Horn logic program. However, it is impractical to represent negative information explicitly in a program, since a program usually specifies true sentences in the actual world while other false facts not written in the program are relatively huge (possibly infinite) in their amount. Then it is more convenient to consider a collection of definite clauses representing known positive information, and assume any fact non-derivable from the program to be false by negation as failure to prove. Such negation is also called default negation and is distinguished from classical negation in first-order logic.

2.2 LOGIC PROGRAMMING

To formalize an inference rule for default negation in logic programming, Reiter [1978] introduced the closed world assumption (CWA). Given a program $P$ and for any ground atom $A$, the CWA is formally stated as follows:

$$CW_A(P) \equiv \neg A \iff P \not\models A.$$  

An alternative formalization of default negation is known as Clark’s program completion [Clark, 1972]. Program completion characterizes the finite failure set of SLD-resolution, while it has some drawbacks compared with the CWA [Shepherdson, 1984; 1988].

Note that default negation in logic programming is nonmonotonic in its nature. That is, an addition of a new fact might withdraw a previously presumed default fact. It is known that such nonmonotonic reasoning is useful for commonsense reasoning in artificial intelligence. Thus introduction of default negation enhances inference abilities of logic programming beyond monotonic deduction, while it makes the theory of logic programming depart from classical first-order logic.

2.2.2 Normal Logic Programs

A (definite) Horn logic program is the simplest form of logic programming, however, as a knowledge representation language, its expressive power is very limited since it only allows one to specify conjunctions and implications between relations. With this restriction, we cannot compute even the complement of a given relation. We have already seen that default negation introduces a mechanism of negative inference into Horn logic programs. Then the first extension of logic programming is to incorporate such negation into a program and use it during deductive inference. To express default negation in a program, Horn logic programs are extended to include normal clauses as follows.

A normal logic program is a finite set of normal clauses of the form:

$$A_1 \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n \quad (0 \leq l \leq 1; \ n \geq m \geq 0)$$  \hfill (2.3)

where $A_1$ and $B_i$'s are atoms and $\neg$ is the negation-as-failure operator. A normal clause with the empty head and a non-empty body is called an integrity constraint.

Note here that negation in normal clauses represents default negation and is distinguished using a new connective $\neg$. The above normal clause (2.3) is read as that $A_1$ is true if $B_1, \ldots, B_m$ are true and $B_{m+1}, \ldots, B_n$ are not true. Clearly, normal logic programs reduce to Horn logic programs in the absence of default negation.
A normal logic program is called stratified [Chandra and Halter, 1985; Apt et al., 1988; van Gelder, 1988] if it contains no predicate derived recursively through its negation. More precisely, a normal logic program \( P \) is stratified if there is a partition \( P = P_1 \cup \ldots \cup P_n \) such that the following conditions hold for \( i = 1, \ldots, n \):

(i) If a predicate occurs positively in a clause in \( P_i \), then its definition (i.e., clauses containing the predicate in their heads) is contained within \( \bigcup_{j \leq i} P_j \).

(ii) If a predicate occurs negatively in a clause in \( P_i \), then its definition is contained within \( \bigcup_{j < i} P_j \).

\( P_n \) can be empty.

For example, the popular flying-bird example is written in a stratified logic program like that:

\[
\begin{align*}
\text{fly}(x) & \leftarrow \text{bird}(x) \land \neg \text{abnormal}(x), \\
\text{bird}(x) & \leftarrow \text{penguin}(x), \\
\text{abnormal}(x) & \leftarrow \text{penguin}(x),
\end{align*}
\]

which means that a bird flies if she is not an abnormal bird.

The notion of stratified logic programs is further extended to *locally stratified logic programs* [Przymusinski, 1988a] which contain no self-recursive atom through negation, and *call-consistent/order-consistent logic programs* [Kunen, 1989; Sato, 1990] which contain no self-recursive predicate/atom through an odd number of negation.

In contrast to stratified logic programs, possibly unstratified normal logic programs are often called *general logic programs* [van Gelder, 1988; Gelfond and Lifschitz, 1988; van Gelder et al., 1991]. The following game program [van Gelder et al., 1991] is an example of such programs:

\[
\begin{align*}
\text{winning}(x) & \leftarrow \text{move}(x, y) \land \neg \text{winning}(y),
\end{align*}
\]

which presents a game situation that one is in a winning position if there is a move to a losing position.

The model theoretical meaning of a normal logic program is defined in the same manner as that of a Horn logic program with an additional statement that an interpretation \( I \) satisfies default negation \( \neg A \) whenever \( A \) is not true in \( I \) (\( I \models \neg A \) if \( I \not\models A \)). Then a model of a normal logic program is defined as an interpretation satisfying every normal clause from the program. A normal logic program is consistent if it has a model, otherwise it is inconsistent.

A declarative semantics of normal logic programs is defined in terms of the particular Herbrand models called the minimal models. An Herbrand model \( M \) is a minimal model of a program \( P \) if no proper subset of \( M \) is a model of \( P \).

In case of definite logic programs, every program has exactly one minimal model, the least Herbrand model. However, in the presence of non-Horn clauses, a program may have several minimal models in general. In this situation, some canonical models are usually chosen as the intended meaning of a program. By the "intended meaning" we mean that one who writes/reads the program is likely to expect it.

One of the criteria for choosing canonical models is the supportedness [Apt et al., 1988]. For a program \( P \), a model \( M \) is called supported if for any atom \( A \) included in \( M \), there is a ground clause of the form (2.3) from \( P \) such that \( A = A_i \) and \( M \) satisfies the body of the clause.

The condition of supportedness is quite an appealing, since we are interested in a model which includes atoms actually derived from a program. For example, the program \( \{ a \leftarrow \neg b \} \) has two minimal models \( \{ a \} \) and \( \{ b \} \), where the first one is supported while the second one is not. In fact, \( b \) is never derived from the program and the intended meaning of the above program is \( \{ a \} \) reflecting the sentence that "\( a \) is true if \( b \) is not proved".

The stable model semantics proposed by Gelfond and Lifschitz [1988] characterizes such canonical models for normal logic programs.

Given a normal logic program \( P \), an interpretation \( I \) is called a stable model of \( P \) if \( I \) coincides with the least Herbrand model of the Horn logic program \( P^I \) defined as

\[
P^I = \{ A \leftarrow B_1 \land \ldots \land B_m \mid \text{there is a ground clause of the form (2.3) from } P \text{ such that } \{B_{m+1}, \ldots, B_n \} \cap I = \emptyset \}.
\]

An intuitive meaning of the above definition is that we first consider an assumption set \( I \), and compute the reduct \( P^I \) of \( P \) with respect to the assumption set \( I \). Then, if \( I \) coincides with the set of logical consequences of \( P^I \), the assumption set \( I \) is justified as an intended meaning.

Stable models are minimal and supported models, but not vice versa [Marek and Subrahmanian, 1992]. A normal logic program may have none, one, or multiple stable models in general. In particular, a consistent (locally) stratified logic program has a unique stable model called the perfect model [Przymusinski, 1988a].

Note that a consistent program does not always have a stable model. For example, the program \( \{ a \leftarrow, b \leftarrow \neg b \} \) is consistent since it has a model \( \{ a, b \} \), while the program has no stable model. A program having at least one stable model is called coherent, otherwise it is called incoherent. For sufficient conditions of the existence
of a stable model, call-consistent logic programs or order-consistent logic programs are always coherent [Pajes, 1990; Dung, 1992b].

Another well-known semantics for normal logic programs is the well-founded semantics, originally proposed by van Gelder et al. [1991]. We do not address the detailed definition of the well-founded semantics here, but refer the reader to the literature [van Gelder et al., 1991; Przymusinski, 1989a; van Gelder, 1989].

The stable model semantics and the well-founded semantics provide alternative theoretical frameworks for normal logic programs. An essential difference between two formalisms is that the stable model semantics is based on the classical two-valued logic, while the well-founded semantics is based on the notion of three-valued well-founded partial model.

The well-founded semantics has an advantage that the well-founded (partial) model is uniquely defined for any consistent normal logic program, while this is not the case for the stable model semantics. However, this deterministic feature often makes the well-founded semantics too skeptical compared with the stable model semantics. For example, the program \{ c → a, c → b, a → not b, b → not a \} has two stable models \{ a, c \} and \{ b, c \}, while its well-founded model is the empty set (i.e., each atom has truth value unknown). Thus, the stable model semantics or the well-founded semantics is usually chosen according to applications.

Recent studies of logic programming have extended the stable model semantics and the well-founded semantics in various ways, while their original forms still serve as two representative semantics for normal logic programs at present. Moreover, both formalisms are revealed to be closely related with each other [Przymusinski, 1990c; Dung, 1992b; Baral and Subrahmanian, 1993].

For proof procedures, bottom-up algorithms are proposed for computing stable models in [Sacca and Zaniolo, 1990; Eshghi, 1990; Inoue et al., 1992; Fernandez et al., 1993; Subrahmanian et al., 1993; Bell et al., 1993], and for computing well-founded models in [Kemp et al., 1992; Subrahmanian et al., 1993].

As a top-down proof procedure for normal logic programs, SLD-resolution is extended to SLDNF-resolution [Clark, 1978; Lloyd, 1987]. For stratified logic programs, SLDNF-resolution is modified to cope with infinite failure in [Kemp and Topor, 1988; Seki and Itoh, 1988; Przymusinski, 1989c] under the perfect model semantics. An abductive proof procedure by Eshghi and Kowalski [1989], is a procedure based on SLDNF-resolution and is correct with respect to the stable model semantics for call-consistent logic programs. For the stable model semantics, however, there is no simple top-down proof procedure applicable for any normal logic program in general. This is due to the irrelevance property of the stable model semantics [Dix, 1992b], such that

2.2. LOGIC PROGRAMMING

the truth value of an atom is often affected by a clause irrelevant to its derivation. On the other hand, some general top-down proof procedures are known for the well-founded semantics such as [Ross, 1989a; Przymusinski, 1989a; Bidot and Legay, 1990; Chen and Warren, 1993].

2.2.3 Extended Logic Programs

In logic programming, it is often useful to represent explicit negation as well as default negation in a program. Gelfond and Lifschitz [1990] have extended the framework of normal logic programs to include classical negation in a program. An extended logic program is a finite set of extended clauses of the form:

\[ L_k \leftarrow L_1 \land \ldots \land L_m \land \text{not } L_{m+1} \land \ldots \land \text{not } L_n \quad (0 \leq k \leq n; m \geq 0) \]

where each \( L_k \) is a literal. A not-free extended logic program is called a positive extended logic program. An extended clause with the empty head and a non-empty body is called an integrity constraint.

Note that in extended logic programs, two kinds of negation, classical negation (\( \sim \)) and default negation (\( \text{not} \)), are distinguished. For example, awarding scholarships to college students is written by the following extended logic program [Gelfond and Lifschitz, 1990]:

\[
\begin{align*}
\text{Eligible}(z) & \leftarrow \text{HighGP}(z), \\
\sim \text{Eligible}(z) & \leftarrow \text{NotFairGP}(z), \\
\text{Interview}(z) & \leftarrow \text{not Eligible}(z) \land \text{not NotEligible}(z),
\end{align*}
\]

where \( \sim \text{Eligible}(z) \) means that \( z \) is non-eligible, while \( \text{not Eligible}(z) \) means that there is no evidence that \( z \) is eligible.

Introduction of classical negation enables us to represent negative knowledge explicitly in a program. While positive and negative knowledge are now equivalently specified in a program, those not explicitly represented in a program are considered to be unknown. This means that the closed world assumption is not assumed in an extended logic program any more, but is specified for each predicate by the following CWA-rule:

\[ \sim P(z) \leftarrow \text{not } P(z). \]

Thus, in an extended logic program we can freely specify the closed world assumption depending on whether a relation is complete or not. In this sense, extended logic programs can represent incomplete knowledge in a program.

\[ \text{In contrast to default negation, classical negation is also called explicit negation.} \]
2.3 DISJUNCTIVE LOGIC PROGRAMMING

Logic programs introduced in the previous section specify knowledge having a definite consequence in the head of each clause. By contrast, logic programs possibly including clauses with disjunctive heads are called indefinite logic programs or disjunctive logic programs [Gallaire et al., 1984; Lobo et al., 1992]. Indefinite or disjunctive logic programs are more expressive than definite logic programs since they can represent indefinite information as well as definite one in a program. Our primary interest in this dissertation is in such programs, and hereafter we use the term disjunctive logic programs or simply disjunctive programs.

2.3.1 Positive Disjunctive Programs

A foundation of disjunctive programs is firstly studied by Minker [1982], in which he provided a theoretical framework of positive disjunctive programs.

A positive disjunctive program is a finite set of clauses of the form

\[ A_1 \lor \ldots \lor A_i \leftarrow B_1 \land \ldots \land B_m \ (i, m \geq 0) \]  

where \( A_1 \)'s and \( B_1 \)'s are atoms. A clause is called disjunctive if its head contains more than one atom \( (i > 1) \); otherwise it is a Horn clause. Thus a positive disjunctive program is a generalization of a Horn logic program.
CHAPTER 2. PRELIMINARIES

In a disjunctive program, indefinite information is specified using disjunctive clauses. That is, when the condition of a disjunctive clause (2.3) holds, at least one of the disjuncts \( A_i \) (1 \( \leq \) i \( \leq \) l) becomes true, but it is generally unknown which one is true.

In real life situations, disjunctive knowledge appears in two ways. For example, the clause:

\[
male \lor female \leftarrow \text{human}
\]

is intended to mean an exclusive disjunction such that human is either male or female but not both. On the other hand, the clauses:

\[
\text{land-animal} \lor \text{aquatic} \leftarrow \text{animal},
\]

\[
\text{amphibian} \leftarrow \text{land-animal} \land \text{aquatic}
\]

have an inclusive disjunction meaning that if an animal is both a land-animal and aquatic, it is an amphibian.

A declarative semantics of positive disjunctive programs is given by the minimal model semantics [Minker, 1982], which is defined as the set of all minimal models of a program. A consistent positive disjunctive program has at least one minimal model, while an inconsistent program has none. The minimal model semantics coincides with the least Herbrand model semantics in Horn logic programs. The minimal model semantics also has its origin in McCarthy's circumscription [McCarthy, 1980] and close relationships between circumscription and logic programming semantics are studied in [Gelfond et al., 1989].

Inference rules for default negation are also introduced in disjunctive programs. In the presence of disjunctive information in a program, however, Reiter [1978] pointed out that the CWA causes an inconsistency in a program. For example, in the program \( \{ A \lor B \leftarrow \} \), the CWA implies both \( \neg A \) and \( \neg B \), which are inconsistent with the program. To improve such a situation in disjunctive programs, two alternative extensions of the CWA are known. One is the generalized closed world assumption (GCWA) by Minker [1982], and the other is the weak generalized closed world assumption (WGCWA) by Rajasekar et al. [1989]. The WGCWA is also independently proposed by Ross and Topor [1988] under the name of the disjunctive database rule (DDR). Comparing each rule, the GCWA provides a strong inference rule for negation, while the WGCWA or the DDR provides a weak inference rule for negation. Both extensions fairly generalize the CWA, however, each rule has some limitation in its inference ability as discussed in Chapter 3. The completion semantics of normal logic programs is also extended to disjunctive programs in [Lobo et al., 1992].

2.3. DISJUNCTIVE LOGIC PROGRAMMING

Proof procedures for the minimal model semantics and the GCWA are studied in the literature such as [Bossu and Siegel, 1985; Yahya and Henschen, 1985; Grant and Minker, 1986; Henschen and Park, 1988; Manthey and By, 1988; Przymusinski, 1988; Lobo et al., 1989; Minker and Rajasekar, 1990; Fernandcz and Minker, 1991a], while those for the WGCWA are in [Ross and Topor, 1988; Rajasekar et al., 1989].

2.3.2 Normal Disjunctive Programs

As the case of Horn logic programs, positive disjunctive programs are extended to normal disjunctive programs by incorporating default negation in a program.

A normal disjunctive program is a finite set of clauses of the form:

\[
A_1 \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_n \land \neg B_{n+1} \land \ldots \land \neg B_m \quad (i \geq 0, \ n \geq m \geq 0)
\]

where \( A_i \)'s and \( B_j \)'s are atoms and \( \neg \) is the negation-as-failure operator. A clause is called a (normal) disjunctive clause if its head contains more than one atom; otherwise it is a normal clause. A normal disjunctive program reduces to a positive disjunctive program in the absence of default negation, and reduces to a normal logic program in the absence of disjunctive clauses.

The notions of models and consistent/inconsistent normal disjunctive programs are defined in the same way as those of normal logic programs. The stable model semantics of normal logic programs is directly extended to normal disjunctive programs as follows.

Given a normal disjunctive program \( P \), an interpretation \( I \) is called a (disjunctive) stable model \(^4\) of \( P \) if \( I \) coincides with a minimal model of the positive program \( P^I \) (called a reduct) defined as:

\[
P^I = \{ A_1 \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_n \mid \text{there is a ground clause of the form (2.6) from } P \text{ such that } \{B_{n+1}, \ldots, B_m\} \cap I = \emptyset \}.
\]

The disjunctive stable model semantics [Przymusinski, 1991a; Gelfond and Lifschitz, 1991] is defined as the set of all stable models of the program. By definition, the disjunctive stable model semantics reduces to Gelfond and Lifschitz's stable model semantics in normal logic programs, and also coincides with the perfect model semantics in stratified disjunctive programs [Przymusinski, 1988a].

\(^4\)We use the term disjunctive stable models when we distinguish them from stable models of normal logic programs. Otherwise, we simply say stable models.

\(^5\)The definition of a stratified disjunctive program is a direct extension of that of a stratified logic program.
The notion of supported models is also extended to disjunctive programs. Given a normal disjunctive program $P$, a model $M$ of $P$ is supported if for any atom $A$ included in $M$, there is a ground clause of the form (2.6) from $P$ such that $A = A_i$ $(1 \leq i \leq l)$ and $M$ satisfies the body of the clause. Stable models of a normal disjunctive program are minimal and supported models.

Like normal logic programs, there are consistent normal disjunctive programs having no stable model. A normal disjunctive program is called coherent (resp. incoherent) if it has a (resp. no) stable model.

The well-founded semantics of normal logic programs is also extended to the disjunctive well-founded semantics [Ross, 1989b; Baral et al., 1992a] or the stationary semantics [Przymusinski, 1991b; 1991a] in normal disjunctive programs. Proof procedures for normal disjunctive programs are developed in the literature such as [Ross, 1989b; Baral et al., 1992a; Inoue et al., 1992; Dung, 1992a; Dung, 1993; Fernandez and Minker, 1992].

Now we compare the stable model semantics approach and the well-founded semantics approach to normal disjunctive programs. Both the disjunctive stable model semantics and the disjunctive well-founded/stationary semantics specialize the minimal model semantics of positive disjunctive programs. However, there are essential differences between two approaches as follows.

1. The disjunctive stable model semantics characterizes non-deterministic behavior of a disjunctive program in terms of multiple stable models. On the other hand, the disjunctive well-founded/stationary semantics presents the meaning of a program by a set of ground disjunctions called a state, which is different from classical approaches based on Herbrand interpretations.

2. The stable model semantics is closely related to nonmonotonic reasoning in AI. However, the well-founded semantics is based on the three-valued logic, and in order to relate the well-founded semantics to nonmonotonic formalisms, three-valued extensions of the formalisms are needed [Przymusinski, 1991c].

3. The stable model semantics is also known to be useful to characterize abductive reasoning in AI, while the well-founded semantics is not usually used for abduction due to its deterministic feature.

Thus, approaches based on the stable model semantics still stay in classical two-valued logic and are useful to characterize commonsense reasoning in disjunctive programs. With these reasons, we take the stable model approach rather than the well-founded model approach in this dissertation.

### 2.3. Disjunctive Logic Programming

#### 2.3.3 Extended Disjunctive Programs

In disjunctive programs, classical negation is also included under the framework of extended disjunctive programs.

An extended disjunctive program is a finite set of clauses of the form:

$$L_1 \lor \ldots \lor L_l \leftarrow L_{l+1} \land \ldots \land L_m \land \neg L_{m+1} \land \ldots \land \neg L_n \quad (n \geq m \geq 0)$$

where each $L_i$ is a literal. A clause is called an (extended) disjunctive clause if its head contains more than one literal; otherwise it is an extended clause. An extended disjunctive program containing no $\neg$ is called a positive extended disjunctive program.

An extended disjunctive program reduces to a normal disjunctive program when all $L_i$'s are atoms, and in the absence of disjunctive clauses it reduces to an extended logic program.

The answer set semantics of extended logic programs is directly extended to extended disjunctive programs as follows.

Let $P$ be a positive extended disjunctive program and $L_P$ be the set of all ground literals from the language of $P$. Then, an answer set of $P$ is defined as the minimal subset $S$ of $L_P$ satisfying the conditions:

1. For each ground clause $L_1 \ldots \lor L_l \leftarrow L_{l+1} \land \ldots \land L_m$ from $P$, $\{L_{l+1}, \ldots, L_m\} \subseteq S$ implies $L_i \in S$ for some $i$ $(1 \leq i \leq l)$. In particular, for each ground clause $\leftarrow L_1 \land \ldots \land L_m$ from $P$, $\{L_1, \ldots, L_m\} \not\subseteq S$; and

2. If $S$ contains a pair of complementary literals $L_i$ and $\neg L_i$, then $S = L_P$.

Next, let $P$ be an extended disjunctive program and $S \subseteq L_P$. The reduct $P^S$ of $P$ with respect to $S$ is defined as

$$P^S = \{ L_1 \lor \ldots \lor L_l \leftarrow L_{l+1} \land \ldots \land L_m \} \text{ there is a ground clause of the form (2.7) from } P \text{ such that } \{L_{l+1}, \ldots, L_m\} \cap S = \emptyset \}$$

Then $S$ is an answer set of $P$ if $S$ is an answer set of $P^S$. The notions of consistent/contradictory answer sets, and coherent/incoherent programs are correspondingly defined as in extended logic programs. A positive form of an extended disjunctive program is also defined in the same manner as that of an extended logic program.

Proof procedures for extended disjunctive programs are studied in [Inoue et al., 1992; Ben-Elieyahu and Dechter, 1992].
2.4 Fixpoint Theory

Fixpoint theory describes denotational aspects of programming languages. In logic programming, it is usually used to characterize declarative semantics of logic programs. For a definite logic program, van Emden and Kowalski (1976) introduced a closure operator which acts over the lattice of Herbrand interpretations $2^{|\mathcal{H}|}$.

Given a definite logic program $P$ and its interpretation $I$, an immediate consequence operator $T_P$ is defined as the mapping $T_P : 2^{|\mathcal{H}|} \rightarrow 2^{|\mathcal{H}|}$ such that

$$T_P(I) = \{ A | A \leftarrow B_1 \land \ldots \land B_n \text{ is a ground clause from } P \text{ and } \{B_1, \ldots, B_n\} \subseteq I \}.$$

The ordinal powers of $T_P$ is defined as:

$$T_P \uparrow 0 = \emptyset,$$

$$T_P \uparrow n + 1 = T_P(T_P \uparrow n),$$

$$T_P \uparrow \omega = \bigcup_{n<\omega} T_P \uparrow n,$$

where $n$ is a successor ordinal and $\omega$ is a limit ordinal. Then the least fixpoint of $T_P$ is given by $T_P \uparrow \omega$.

Van Emden and Kowalski have shown that the above least fixpoint coincides with the least Herbrand model of a definite logic program $P$. The result is extended by Apt, Blair and Walker (1988), in their paper they developed a nonmonotonic fixpoint operator and its iterative fixpoint for logic programs with negation. Przymusinski (1988a) showed that Apt et al.'s fixpoint characterizes the perfect model of a stratified logic program. Further extensions of fixpoint semantics for normal logic programs are studied for the stable model semantics (Sacco and Zaniolo, 1990; Fages, 1991) and the well-founded semantics (van Gelder, 1988; Przymusinski, 1989a; van Gelder et al., 1991).

In the context of disjunctive programs, Minker and Rajasekar (1990) introduced a new fixpoint semantics for positive disjunctive programs. They first introduced the extended Herbrand base as the set of all ground positive disjunctions from a program, and defined the notion of a state as a set of positive disjunctions from the extended base. Then they developed a fixpoint operator which operates on states and showed that its least fixpoint contains disjunctions which are true in every minimal model of a program.

Such a state based fixpoint semantics is extended to stratified disjunctive programs (Rajasekar and Minker, 1989) and normal disjunctive programs (Baral et al., 1992a; Przymusinski, 1991b). Yet other fixpoint semantics are also presented in the literature such as (Ross and Topor, 1988; Reed et al., 1991; Fernandez and Minker, 1991b; Decker, 1992, Inoue and Sakama, 1992). Detailed discussion on the fixpoint semantics of disjunctive programs is presented in Chapter 3.

2.5 Computational Complexity

In this section, we briefly review basic concepts of complexity theory. More on the subject can be found in (Garey and Johnson, 1979; Johnson, 1990).

Complexity theory is used to classify problems according to their intrinsic computational difficulties. For classification, some complexity classes are introduced to characterize problems. In this dissertation, we are mainly concerned with decision problems which admit boolean answers (yes/no).

The class $P$ represents the set of all decision problems solvable by deterministic Turing machines in polynomial time. A problem in this class is called tractable or efficiently solvable.

On the other hand, the class $NP$ represents the set of all decision problems solvable by non-deterministic Turing machines in polynomial time. The class co-$NP$ is the set of problems whose complements are in $NP$. Apparently, the class $P$ is included both in the class $NP$ and in co-$NP$. The question whether this inclusion is strict remains open, but it is usually considered that $P \neq NP$. 
The polynomial hierarchy consists of classes $\Delta^p_k$, $\Sigma^p_k$, and $\Pi^p_k$ defined as follows:

\[
\Delta^p_0 = \Sigma^p_0 = \Pi^p_0 = P,
\]
\[
\Delta^p_{k+1} = \Sigma^p_k \cup \Pi^p_k,
\]
\[
\Sigma^p_{k+1} = \text{co-} \Sigma^p_k,
\]
\[
\Pi^p_{k+1} = \text{co-} \Sigma^p_k. \quad (k \geq 0)
\]

In particular, $\Delta^p_0 = P$, $\Sigma^p_0 = \text{NP}$, and $\Pi^p_0 = \text{co-NP}$.

In the above, $\Delta^p_k$ (resp. $\Sigma^p_k$) is the set of problems solvable deterministically (resp. non-deterministically) in polynomial time with an oracle for the problems in $\Sigma^p_{k+1}$, while $\Pi^p_{k+1}$ consists of the complements of $\Sigma^p_{k+1}$. An oracle is considered as a subroutine without cost, that is, it is decided in unit time for any problem.

In the polynomial hierarchy, the relations $\Delta^p_k \subseteq \Sigma^p_k \cap \Pi^p_k$ and $\Sigma^p_k \cup \Pi^p_k \subseteq \Delta^p_{k+1}$ hold. Problems included in the upper levels of the hierarchy are more difficult to solve than those in lower levels unless the polynomial hierarchy collapses.

A problem is called C-hard if every problem in a class $C$ is efficiently reducible to the problem by a polynomial-time transformation. The C-hard problem is also called C-complete if the problem itself belongs to the class $C$. For instance, checking the satisfaction of any propositional theory is an NP-complete problem.

When we discuss the complexity issue of logic programming, (finite) propositional logic programs are usually assumed. This is because in the presence of function symbols, a program containing variables has the infinite set of ground clauses, then checking its satisfiability is undecidable even in the case of Horn logic programs. For propositional Horn logic programs, deciding whether a ground atom is derived from a program is tractable. This is also the case for the decision problem under the perfect model semantics of propositional stratified logic programs [Apt et al., 1988].

On the other hand, when a propositional logic program has multiple canonical models, the computational complexity of the program is usually characterized by the following three decision problems.

**Existence**: Deciding the existence of a canonical model of a program.

**Set-Membership**: Deciding whether a ground atom is true in some canonical model of a program.

**Set-Entailment**: Deciding whether a ground atom is true in every canonical model of a program.

The set-membership problem also characterizes credulous reasoning, while the set-entailment problem characterizes skeptical reasoning. For example, the existence problem and the set-membership problem for the stable model semantics of normal logic programs are both NP-complete, while the set-entailment problem is coNP-complete [Marek and Trusczynski, 1991a; 1991b]. In disjunctive programs, problems become harder than normal logic programs in general. The set-membership problem for the disjunctive stable model semantics is $\Sigma^p_k$-complete, while the set-entailment problem is $\Pi^p_k$-complete. It is known that similar results also hold for nonmonotonic logics [Gottlob, 1992].

For surveys on the complexity issues in logic programming, see [Schlipf, 1992b; Eiter and Gottlob, 1993a; Cadoli and Schaerf, 1993].
Chapter 3

Possible Model Semantics for Disjunctive Logic Programs

In this chapter, we introduce a new declarative semantics of disjunctive programs called the possible model semantics. The possible model semantics is an alternative theoretical framework to the classical minimal model semantics and provides a flexible inference mechanism for inferring negation in disjunctive programs. The possible model semantics is characterized by a new fixpoint semantics of disjunctive programs. We also present a proof procedure for the possible model semantics and show that the possible model semantics has an advantage from the computational complexity point of view.

3.1 Introduction

Traditionally, the declarative semantics of logic programming and deductive databases has been studied based on the notion of minimal models. For instance, the least Herbrand model semantics for Horn logic programs [van Emden and Kowalski, 1976], the perfect model semantics for stratified logic programs [Przymusinski, 1988a], and the stable model semantics for normal logic programs [Gelfond and Lifschitz, 1988] are all minimal models. The minimal models reflect the so-called Occam's razor such that “only those objects should be assumed to exist which are minimally required by the context”. Such a principle of minimality plays a fundamental role in the area of not only logic programming but also nonmonotonic reasoning in artificial intelligence [McCarthy, 1980]. Therefore, it has been recognized that the principle of minimality is one of the most basic and indispensable criteria that each semantics for commonsense reasoning should obey [Schlipf, 1992b].
**POSSIBLE MODEL SEMANTICS**

This is also the case in the context of disjunctive programs, namely, the *minimal model semantics* for positive disjunctive programs [Minker, 1982] and the *disjunctive stable model semantics* for normal disjunctive programs [Przymusinski, 1988b; Gelfond and Lifschitz, 1991] are both minimal. However, it has also been pointed out that such a minimalism is not always appropriate in a theory containing indefinite information. Ross and Topor [1988] have firstly noticed this problem in the context of inferring negation in disjunctive programs. They argue that when one infers negation from a disjunctive program, one should be cautious to interpret disjunctions exclusively rather than *exclusively*. In fact, the minimal model semantics minimizes truth extensions of predicates as much as possible, then it usually interprets disjunctions exclusively and maximizes negative information inferred from a program.

In positive disjunctive programs, Minker [1982] has extended Reiter's CWA to the *generalized closed world assumption* (GCWA). On the other hand, Ross and Topor [1988] have proposed an alternative rule called the *disjunctive database rule* (DDR), which turns out to be equivalent to the weak generalized closed world assumption (WGCA) independently proposed by Rajasekar et al. [1989].

Comparing these two rules, the GCWA is based on the minimal model semantics and usually interprets disjunctions exclusively, while the DDR and the WGCA are weaker than the GCWA and interpret disjunctions inclusively. Thus, both the GCWA and the WGCA (or DDR) fairly extend the CWA, but the problem is that they are inherently different from each other. In fact, in the absence of a single uniform framework, one has to use these separate rules in order to treat both exclusive and inclusive disjunctions in the same program. Such a situation actually happens in our real life situations. For instance, consider the program:

\[
\text{land-animal} \lor \text{aquatic} \rightarrow \text{animal},
\]

\[
\text{biped} \lor \text{quadruped} \rightarrow \text{land-animal},
\]

\[
\text{amphibian} \rightarrow \text{land-animal} \lor \text{aquatic},
\]

in which the disjunction in the first clause is inclusive, while the disjunction in the second clause is exclusive. As another example, a disjunctive clause possibly contains hybrid disjunctions such as:

\[
\text{Sunday} \lor \text{national-holiday} \lor \text{weekday} \rightarrow \text{calendar-days},
\]

in which the disjunction in the first clause is inclusive, while the disjunction in the second clause is exclusive. As another example, a disjunctive clause possibly contains hybrid disjunctions such as:

3.2 Possible Model Semantics for Positive Disjunctive Programs

In this section, we first consider positive disjunctive programs, that is, disjunctive programs containing no default negation.
3.2.1 Negation in Positive Disjunctive Programs

As presented in Section 2.3, Reiter's CWA is not useful in the context of disjunctive programs. Then, for inferring negation in positive disjunctive programs, Reiter's CWA is mainly extended in two ways: one is Minker's generalized closed world assumption (GCWA) and the other is Ross and Topor's disjunctive database rule (DDR) or Rajasekar et al.'s weak generalized closed world assumption (WGCWA).

We first review definitions and properties of those two frameworks.

Given a positive disjunctive program $P$, let us denote by $MMP$ the set of all minimal models of $P$. Then the GCWA is defined as follows.

**Definition 3.1** ([Minker, 1982]) Let $P$ be a consistent positive disjunctive program. Then $GCWA(P)$ is defined as:

$$GCWA(P) = \{ \neg A \mid A \in HSP \text{ and } A \notin I \text{ for any } I \in MMP \}. \quad \Box$$

On the other hand, the WGCWA provides a weaker form of closed world reasoning in positive disjunctive programs as follows.

The Horn transformation [Ross and Topor, 1988] of a positive disjunctive program $P$ is defined as:

$$Horn(P) = \{ A_i \leftarrow B_1 \land \ldots \land B_m \mid A_1 \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_m \in P$$

$$\text{and } 1 \leq i \leq l, \ l \geq 1 \}.$$

Note here that $Horn(P)$ is always consistent since it does not contain integrity constraints.

**Definition 3.2** ([Ross and Topor, 1988; Rajasekar et al., 1989]) Let $P$ be a consistent positive disjunctive program and $Horn(P)$ be its Horn transformation. Let $M_{Horn(P)}$ be the least Herbrand model of $Horn(P)$. Then $WGCWA(P)$ is defined as:

$$WGCWA(P) = \{ \neg A \mid A \in HSP \text{ and } A \notin M_{Horn(P)} \}. \quad \Box$$

Properties of the GCWA and the WGCWA are as follows.

**Theorem 3.1** ([Minker, 1982; Ross and Topor, 1988; Rajasekar et al., 1989]) Let $P$ be a consistent positive disjunctive program and $A$ be a ground atom. Then,

1. According to [Rajasekar et al., 1989; Lobo et al., 1992], the DDR and the WGCWA are equivalent. Then we use the term WGCWA hereafter.

### POSITIVE DISJUNCTIVE PROGRAMS

(i) $P \cup GCWA(P)$ is consistent.

$P \cup WGCWA(P)$ is consistent.

(ii) $P \models A$ if $P \cup GCWA(P) \models A$.

$P \models A$ if $P \cup WGCWA(P) \models A$.

(iii) $P \subseteq P'$ does not imply $GCWA(P') \subseteq GCWA(P)$.

$P \subseteq P'$ implies $WGCWA(P') \subseteq WGCWA(P)$.

(iv) For a definite logic program $P$, $GCWA(P) = WGCWA(P) = CW(P)$. \quad \Box$

That is, (i) both $GCWA(P)$ and $WGCWA(P)$ are consistent with $P$, (ii) positive facts proven from $P$ are invariant, (iii) the GCWA (resp. WGCWA) is non-decreasing (resp. decreasing), (iv) the GCWA is stronger than the WGCWA, and (v) for definite logic programs each rule reduces to the CWA.

**Example 3.1** Let $P$ be the program:

$$\{ a \lor b \lor c \leftarrow, \quad d \leftarrow a \land b, \quad e \leftarrow b \land c, \quad \neg b \land c \}$$

where $MMP = \{ \{ a \}, \{ b \}, \{ c \} \}$ and $M_{Horn(P)} = \{ a, b, c, d, e \}$. Then, $GCWA(P) = \{ \neg d, \neg e \}$, while $WGCWA(P) = \emptyset$. \quad \Box

In the above example, the GCWA interprets each disjunction exclusively, while the WGCWA interprets them inclusively. Then, $GCWA(P)$ excludes the inclusive interpretation of $a \lor b$ and infers $\neg d$ from the program. On the other hand, the integrity constraint $\leftarrow b \land c$ inhibits an inclusive interpretation of $b \lor c$, nevertheless, $WGCWA(P)$ cannot infer $\neg e$. This is because the WGCWA does not consider the model theoretical meaning of a given program, and ignores the effect of integrity constraints in a program. In fact, $M_{Horn(P)}$ is no longer a model of $P$. Generally speaking, the GCWA is too strong to interpret inclusive disjunctions, while the WGCWA is too weak to treat exclusive disjunctions. Then, to treat both types of disjunctions in a program, one has to use two different rules in the same program.

To improve such a situation, in the next section we introduce a new declarative semantics called the possible model semantics, which can distinguish two types of disjunctions uniformly in a program.
3.2.2 Possible Model Semantics

A disjunctive program is considered to represent a set of possible facts that might have been true in the actual world. The possible model semantics is intended to formulate this situation.

Given a positive disjunctive program \( P \), a split program is defined as a ground Horn logic program obtained from \( P \) by replacing each ground disjunctive clause of the form:

\[
A_1 \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_m
\]

with the following ground definite clauses (called split clauses):

\[
A_i \leftarrow B_1 \land \ldots \land B_m
\]

for every \( A_i \in S \)

where \( S \) is some non-empty subset of \( \{A_1, \ldots, A_l\} \). Note that every ground Horn clause from \( P \) is included in any split program of \( P \). Then, a possible model of \( P \) is defined as the least Herbrand model of any split program of \( P \). The set of all possible models of \( P \) is denoted by \( \mathcal{PM}_P \).

Example 3.2 Let \( P \) be the program:

\[
\{ a \lor b \leftarrow, \ b \lor c \leftarrow, \ \lnot b \land c \}.
\]

Then the split programs of \( P \) are

\[
\{ a \leftarrow, \ b \leftarrow, \ \lnot b \land c \},
\]

\[
\{ a \leftarrow, \ c \leftarrow, \ \lnot b \land c \},
\]

\[
\{ b \leftarrow, \ \lnot b \land c \},
\]

\[
\{ b \leftarrow, \ c \leftarrow, \ \lnot b \land c \},
\]

\[
\{ a \leftarrow, \ b \leftarrow, \ c \leftarrow, \ \lnot b \land c \}.
\]

Since the last two split programs are inconsistent, the set of all possible models of \( P \) is \( \mathcal{PM}_P = \{\{a, b\}, \{a, c\}, \{b\}\} \).

Possible models have the following properties.

Proposition 3.2 A consistent positive disjunctive program has at least one possible model.

Proof: Since a consistent positive disjunctive program has a consistent split program, the result immediately follows.

Proposition 3.3 A possible model of a positive disjunctive program \( P \) is a model of \( P \).

Proof: Let \( M \) be a possible model of \( P \) such that \( M \) is the least Herbrand model of a consistent split program \( P' \). Then, for each clause \( C_i : A_i \leftarrow B_1 \land \ldots \land B_m \) in \( P \), there is a corresponding clause \( C : A_i \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_m \) in \( P' \) where \( 1 \leq i \leq l \). Since \( M \) satisfies each \( C_i \), it also satisfies \( C \). Also each integrity constraint in \( P \) is included in \( P' \) and is satisfied in \( M \). Hence \( M \) is a model of \( P \).

The notion of possible models is different from minimal models. In fact, in Example 3.2, \( \{a, b\} \) is a possible model, but not a minimal model. The intuitive meaning of the possible model is that each atom included in a possible model has its possible justification in a program. Thus, both inclusive and exclusive interpretations of disjunctions are considered whenever there is no integrity constraint to inhibit inclusive interpretations. The following property directly follows from the definition.

Proposition 3.4 A possible model is a supported model.

Note that the converse of the above proposition does not hold in general. For example, \( \{a\} \) is a supported model of the program \( \{a \lor b \leftarrow a\} \), but not a possible model. The following relationship holds between possible models and minimal models of a program.

Proposition 3.5 Let \( P \) be a consistent positive disjunctive program. Then the set of all minimal elements from \( \mathcal{PM}_P \) coincides with \( \mathcal{MMP}_P \).

Proof: Let \( M \) be a minimal model of \( P \). Then, for each clause \( A_1 \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_m \) in \( P \), \( \{B_1, \ldots, B_m\} \subseteq M \) implies \( A_i \in M \) for some \( i \), \( 1 \leq i \leq l \). In this case, there is a split program \( P_i \) of \( P \) which contains \( A_i \leftarrow B_1 \land \ldots \land B_m \). Since \( M \) satisfies each integrity constraint in \( P \), \( M \) is the least Herbrand model of the consistent Horn logic program \( P_i \), hence a possible model of \( P \). Thus, \( \mathcal{MMP}_P \) is included in \( \mathcal{PM}_P \). On the other hand, since each possible model is a model of \( P \) by Proposition 3.3, the set of all minimal elements from \( \mathcal{PM}_P \) coincides with \( \mathcal{MMP}_P \).

Thus the set of minimal models is a subset of the set of possible models. In particular, a definite logic program has a unique possible model which is the least Herbrand model of the program. The above proposition implies that as for the inference of positive facts, the possible model semantics coincides with the minimal model semantics.
Theorem 3.6 Let $P$ be a consistent positive disjunctive program. An atom $A$ is true in $P$ iff $A$ is included in every possible model of $P$. $\square$

Note that possible models depend on the syntax of a program. For instance, two programs \{ $a \lor b \leftarrow \cdot$ \} and \{ $a \leftarrow \cdot$ \} are equivalent under first-order logic, while the first program has the possible model \{ $a, b$ \} which is not a possible model of the second program. This is because the first program is intended to specify some indefinite information about $b$, while it is not the case in the second program. Such a distinction is in fact meaningful in knowledge representation. Suppose a situation like this:

There is a visitor at my house whom I do not know. I am living with my parents so that I guess either my mother or father must know him: $\text{know(mother, visitor)} \lor \text{know(father, visitor)}$. After a while, mother comes back and she actually knows him: $\text{know(mother, visitor)}$, and at this moment the possibility is open that my father's knowing him too. However, if we replace the previous belief $\text{know(mother, visitor)} \lor \text{know(father, visitor)}$ with $\text{know(mother, visitor)} \land \text{know(father, visitor)}$, the negation $\text{\neg know(father, visitor)}$ is assumed under the closed world assumption.

Thus, logically equivalent sentences do not necessarily have the same meaning from the viewpoint of knowledge representation. Generally speaking, the syntax of a program plays an important role in logic programming to specify our intended knowledge, then it appears natural that the possible model semantics also shares such syntax-dependent properties.

Next, we consider the inference of negative facts under the possible model semantics. Under the possible model semantics, negation is defined as follows.

Definition 3.3 Let $P$ be a consistent positive disjunctive program. Then the possible world assumption (PWA) of $P$ is defined as:

$$PWA(P) = \{ \neg A \mid A \in \mathcal{P}_m(P) \land A \not \in I \text{ for any } I \in \mathcal{P}_m(P) \}. \quad \square$$

Theorem 3.7 Let $P$ be a consistent positive disjunctive program and $A$ be a ground atom. Then,

(i) $P \cup PWA(P)$ is consistent.

(ii) $P \models A$ iff $P \cup PWA(P) \models A$.

POSITIVE DISJUNCTIVE PROGRAMS

(iii) $P \subseteq P'$ does not imply $PWA(P') \subseteq PWA(P)$.

(iv) For a definite logic program $P$, $PWA(P) = CW(P)$.

Proof: (i) Since $P$ is consistent, it has a minimal model $M$ which is also a possible model of $P$. Then, by definition, $M$ is also a model of $P \cup PWA(P)$, hence the result follows. (ii) The result directly follows from Theorem 3.6. (iii) Let $P = \{ a \lor b \leftarrow, c \leftarrow \neg a \land b \}$ and $P' = P \cup \{ \neg a \land b \}$. Then $PWA(P) = \emptyset$, while $PWA(P') = \{ \neg c \}$. (iv) For a definite logic program $P$, $\mathcal{P}_m(P)$ contains the unique least Herbrand model of $P$, hence the result follows. $\square$

The next theorem presents that the PWA is stronger than the WGCWA and weaker than the GCWA.

Theorem 3.8 Let $P$ be a consistent positive disjunctive program. Then the following relationship holds:

$$WGCWA(P) \subseteq PWA(P) \subseteq GCWA(P).$$

In particular, $WGCWA(P) = PWA(P)$ if $P \cup \text{Horn}(P)$ is consistent.

Proof: The relationship $PWA(P) \subseteq GCWA(P)$ immediately follows from the fact that $\mathcal{P}_m(P) \subseteq \mathcal{P}_m(P)$. Since the least Herbrand model $M_{\text{Horn}(P)}$ of $\text{Horn}(P)$ is a super set of any possible model in $\mathcal{P}_m(P)$, the relationship $WGCWA(P) \subseteq PWA(P)$ also holds. In particular, $\text{Horn}(P)$ is the set of all definite clauses included in the maximal split program of $P$. Then, if $P \cup \text{Horn}(P)$ is consistent, $M_{\text{Horn}(P)}$ coincides with the maximal element in $\mathcal{P}_m(P)$, hence the result follows. $\square$

Example 3.3 (cont. from Example 3.1) Let $P$ be the program:

$$\{ a \lor b \lor c \leftarrow, d \leftarrow a \land b, c \leftarrow b \land c \}\$$

where $\mathcal{P}_m(P) = \{ \{a\}, \{b\}, \{c\}, \{a, b, d\}, \{a, c\} \}$. Then, $PWA(P)$ implies $\neg c$, but not $\neg d$. $\square$

Note that in the above example $P \cup \text{Horn}(P)$ is inconsistent and $WGCWA(P)$ fails to capture the intended meaning of $P$. By contrast, the possible model semantics can infer proper negation by distinguishing two kinds of disjunctions using integrity constraints.

The possible model semantics was independently discovered by Chan [1993] under the name of the possible world semantics. In [Chan, 1993], both notions are proven to be equivalent in positive disjunctive programs.
3.3 Possible Model Semantics for Normal Disjunctive Programs

In this section, we extend the possible model semantics of positive disjunctive programs to normal disjunctive programs.

The notion of split programs is defined in the same manner as positive disjunctive programs. Given a normal disjunctive program $P$, its split program is defined as a ground normal logic program obtained from $P$ by replacing each ground disjunctive clause of the form:

$$A_1 \lor \ldots \lor A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n$$

with the following ground normal clauses (called split clauses):

$$A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n$$

where $S$ is some non-empty subset of $\{A_1, \ldots, A_i\}$. Then, a possible model of $P$ is defined as a stable model of any split program of $P$. The set of all possible models of $P$ is denoted by $\mathcal{PM}_P$. A normal disjunctive program having at least one possible model is called p-coherent, otherwise it is called p-incoherent.

The possible models defined above reduce to those presented in the previous section in a positive disjunctive program. Also, possible models coincide with stable models in normal logic programs. The following property holds.

**Proposition 3.9** A possible model of a normal disjunctive program $P$ is a model of $P$. □

**Proposition 3.10** A possible model is a supported model, but not vice versa. □

**Proposition 3.11** Let $P$ be a consistent normal disjunctive program and $ST_P$ be the set of all stable models of $P$. Then the set of all minimal elements from $\mathcal{PM}_P$ contains $ST_P$.

*Proof:* By definition, a stable model $M$ of $P$ is also a stable model of some split program of $P$. Then $M$ is also a possible model of $P$. Since $M$ is minimal, it is also a minimal element in $\mathcal{PM}_P$. □

The above proposition implies that a coherent normal disjunctive program is also p-coherent (but not vice versa, see Example 3.9). The converse of the above proposition does not hold in general. That is, minimal possible models are not always stable models.

---

**NORMAL DISJUNCTIVE PROGRAMS**

**Example 3.4** Let $P$ be the program:

$$\{ a \lor b \leftarrow, \ b \leftarrow a, \ c \leftarrow \neg a \}.$$  

Then $\mathcal{PM}_P = \{(a, b), (b, c)\}$, and $(a, b)$ is a minimal element in $\mathcal{PM}_P$ but not a stable model of $P$. □

In the above example, $(b, c)$ is the unique stable model of $P$, hence $c$ is true under the disjunctive stable model semantics. However, this is not the case under the possible model semantics, since there is an inclusive interpretation of the disjunction $(a, b)$ in which $c$ is not true. Thus, in contrast to the case of positive disjunctive programs, positive facts true under the possible model semantics (viz. positive facts true in every possible model) differ from those ones under the disjunctive stable model semantics. At the end of this subsection, we will also show the case that the possible model semantics implies more positive facts than the disjunctive stable model semantics.

Now we consider negative inference in normal disjunctive programs. We first define an extension of the GCWA as negation under the disjunctive stable model semantics.

**Definition 3.4** Let $P$ be a coherent normal disjunctive program. Then GCWA$^+(P)$ is defined as:

$$\text{GCWA}^+(P) = \{ \neg A \mid A \in \text{HBP} \text{ and } A \notin I \text{ for any } I \in ST_P \}. \quad \Box$$

Next, in order to define a suitable extension of the WGCWA, we introduce a transformation from a normal disjunctive program to a normal logic program.

The NLP-transformation of a normal disjunctive program $P$ is defined as:

$$\text{NLP}(P) = \{ A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n \mid A_i \lor \ldots \lor A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n \in P \text{ and } 1 \leq i \leq l, 1 \geq l \}.$$  

The $\text{NLP}(P)$ is a direct extension of $\text{Horn}(P)$, and $\text{NLP}(P) = \text{Horn}(P)$ holds for a positive disjunctive program $P$. For a coherent normal disjunctive program $P$, $\text{NLP}(P)$ is not always coherent.

**Example 3.5** Let $P$ be the program:

$$\{ a \lor b \leftarrow \neg a \}.$$  

Then its NLP-transformation becomes

$$\text{NLP}(P) = \{ a \leftarrow \neg a, \ b \leftarrow \neg a \}.$$  

Here $ST_P = \{(b)\}$, while $ST_{\text{NLP}(P)} = \emptyset$. □
Conversely, there is an incoherent program whose NLP-transformation has a stable model.

**Example 3.6** Let \( P \) be the program:
\[
\{ a \lor b \leftarrow \text{not } a, \text{not } b \}.
\]
Then \( ST_P = \emptyset \), while \( ST_{NLP(P)} = \{ \{ a, b \} \} \).

We say that a normal disjunctive program \( P \) is *weakly coherent* if either \( P \) or \( NLP(P) \) has a stable model. Clearly, a coherent program is also weakly coherent, but not vice versa.

**Definition 3.5** Let \( P \) be a weakly coherent normal disjunctive program. Then \( WGCW^+(P) \) is defined as:
\[
WGCW^+(P) = \{ \neg A \mid A \in \mathcal{H}_P \text{ and } A \not\in I \text{ for any } I \in ST_P \cup ST_{NLP(P)} \}.
\]

This definition is natural in the sense that \( WGCW^+(P) \) restricts its negative inference like the WGCW by taking into account the stable models of \( NLP(P) \). A similar definition is also given in [Dix, 1992a] in a different context. Note that in the above definition, one may consider that by analogy with the WGCW, considering \( ST_{NLP(P)} \) is enough instead of \( ST_P \cup ST_{NLP(P)} \). But this is not the case. For instance, let \( P = \{ a \lor b \leftarrow \text{not } a, \text{not } b \} \). Then \( ST_P = \{ \{ a, c \}, \{ b, c \} \} \), whereas \( ST_{NLP(P)} \) does not imply \( \neg c \). On the other hand, \( NLP(P) = \{ \{ a \leftarrow \text{not } a, b \leftarrow \text{not } b \} \) then \( ST_{NLP(P)} = \{ \{ a, b \} \} \) which implies \( \neg c \). Thus, without \( ST_P \), the WGCW is not weaker than the GCWA any more.

The WPA is also extended in normal disjunctive programs as follows.

**Definition 3.6** Let \( P \) be a p-coherent normal disjunctive program. Then \( PW\!A^+(P) \) is defined as:
\[
PW\!A^+(P) = \{ \neg A \mid A \in \mathcal{H}_P \text{ and } A \not\in I \text{ for any } I \in \mathcal{P}\mathcal{M}_P \}.
\]

Now we investigate the properties of each rule. In the following, we write \( P \models_{ST} A \) (resp. \( P \models_{PM} A \) if \( A \in I \) for any \( I \in ST_P \) (resp. \( I \in \mathcal{P}\mathcal{M}_P \)).

**Theorem 3.12** Let \( P \) be a normal disjunctive program and \( A \) be a ground atom. Then the following properties hold.

---

**NORMAL DISJUNCTIVE PROGRAMS**

1. (i) If \( P \) is coherent, \( P \cup GCWA^+(P) \) is coherent.
   (ii) \( P \models_{ST} A \) if \( P \cup GCWA^+(P) \models_{ST} A \).
   (iii) \( P \subseteq P' \) does not imply \( GCWA^+(P) \subseteq GCWA^+(P') \).
   (iv) For a positive disjunctive program \( P \), \( GCWA^+(P) = GCWA(P) \).

2. (i) If \( P \) is weakly coherent, \( P \cup WGCWA^+(P) \) is weakly coherent.
   (ii) \( P \models_{ST} A \) if \( P \cup WGCWA^+(P) \models_{ST} A \).
   (iii) \( P \subseteq P' \) does not imply \( WGCWA^+(P) \subseteq WGCWA^+(P') \).
   (iv) For a positive disjunctive program \( P \), \( WGCWA^+(P) = WGCWA(P) \).

3. (i) If \( P \) is p-coherent, \( P \cup PW\!A^+(P) \) is p-coherent.
   (ii) \( P \models_{PM} A \) if \( P \cup PW\!A^+(P) \models_{PM} A \).
   (iii) \( P \subseteq P' \) does not imply \( PW\!A^+(P') \subseteq PW\!A^+(P) \).
   (iv) For a positive disjunctive program \( P \), \( PW\!A^+(P) = PW\!A(P) \).

**Proof.** 1. (i) Since \( P \) is coherent, it has at least one stable model for any negated atom in \( GCWA^+(P) \) not included in any stable model of \( P \), hence \( P \cup GCWA^+(P) \) is coherent. (ii) Any negated atom in \( GCWA^+(P) \) is not included in any stable model. Then adding such negative facts to \( P \) does not affect the construction of stable models. Hence the result follows. (iii) The GCWA is non-decreasing since the GCWA includes the GCWA by (iv) which is non-decreasing. (iv) Since stable models reduce to minimal models in a positive disjunctive program, the result immediately follows.

2. (i) When \( P \) is weakly coherent, every negated atom in \( WGCWA^+(P) \) is not included in any stable model of \( P \) and \( NLP(P) \). Hence, \( P \cup WGCWA^+(P) \) is also weakly coherent. (ii) The result follows from (i). (iii) For the non-decreasing property of \( WGCWA^+(P) \), see Example 3.7 (by (iv)) which is non-decreasing. (iv) Since \( ST_P \cup ST_{NLP(P)} \) reduces to \( \mathcal{M}\mathcal{MP} \cup \{ \mathcal{M}_{Horn} \} \) in a positive disjunctive program \( P \), and each minimal model in \( \mathcal{M}\mathcal{MP} \) is a subset of \( \mathcal{M}_{Horn} \), the result also holds.

3. (i) and (ii) follow directly from the definition. (iii) The \( PW\!A^+ \) is non-decreasing since the \( PW\!A^+ \) includes the \( PW\!A \) (by (iv)) which is non-decreasing. (iv) Since possible models in a normal disjunctive program reduce to those presented in the previous section in a positive disjunctive program, the result immediately follows. □
POSSIBLE MODEL SEMANTICS

Notice that in contrast to the WGCWA, the WGCWA^~ is non-decreasing.

Example 3.7 Let P be the program:
\[ \{ a \lor b \leftarrow \neg c \}, \ c \leftarrow d \]
and \( P' = P \cup \{ d \leftarrow \} \). Then, \( ST_P \cup ST_{NLP(P)} = \{ \{ a \}, \{ b \}, \{ a, b \} \} \) and \( ST_{P'} \cup ST_{NLP(P')} = \{ \{ c, d \} \} \). Hence, \( WGCWA^\sim(P) = \{ \neg c, \neg d \} \), while \( WGCWA^\sim(P') = \{ \neg a, \neg b \} \). □

Thus, in the presence of default negation in a program, the non-monotonic decreasing property of the WGCWA does not hold any more.

For coherent normal logic programs, the three rules coincide.

Proposition 3.13 Let P be a coherent normal logic program. Then, \( GCWA^\sim(P) = WGCWA^\sim(P) = PWA^\sim(P) \).

Proof: For a coherent normal logic program \( P \), \( ST_P \cup ST_{NLP(P)} = ST_P \).
Then the relation \( GCWA^\sim(P) = WGCWA^\sim(P) \) holds by each definition.

The relation \( GCWA^\sim(P) = PWA^\sim(P) \) also holds since \( ST_P = PM_P \)
holds for a coherent normal logic program \( P \). □

The next theorem presents the relationship among the three rules in normal disjunctive programs.

Theorem 3.14 Let \( P \) be a coherent normal disjunctive program. Then,
(i) \( GCWA^\sim(P) \subseteq GCWA^\sim(P) \).
(ii) \( PWA^\sim(P) \subseteq GCWA^\sim(P) \).

Proof: Since \( ST_P \subseteq ST_P \cup ST_{NLP(P)} \), (i) follows from definitions. The part (ii) also follows from the fact that \( ST_P \subseteq PM_P \). □

As for the WGCWA^~ and the PWA^~, there is no inclusion relationship.

Example 3.8 Consider the program:

\[ P = \{ a \lor b \lor c \leftarrow \neg d \}, \ c \leftarrow a \land b \land \neg c \} \]

Then \( ST_P = \{ \{ a \}, \{ b \}, \{ c \} \}, ST_{NLP(P)} = \{ \{ a, b, c \} \} \), and \( WGCWA^\sim(P) = \{ \neg d, \neg e \} \), while \( PM_P = \{ \{ a \}, \{ b \}, \{ c \}, \{ a, b, c \}, \{ b, c \}, \{ c, a \}, \{ a, b, c \} \} \) and \( PWA^\sim(P) = \{ \neg d \} \).

Hence, \( WGCWA^\sim(P) \not\subseteq PWA^\sim(P) \). The converse inclusion relation does not hold by Theorem 3.3 either, since each rule reduces to the WGCWA or the PWA in positive disjunctive programs. □

3.4 Fixpoint Semantics

In this section, we present a new fixpoint semantics for disjunctive programs to characterize the possible model semantics presented in the previous sections. We first give a fixpoint semantics for positive disjunctive programs, and generalize it to normal disjunctive programs.

3.4.1 Fixpoint Semantics for Positive Disjunctive Programs

In a definite logic program, van Emde and Kowalski [1976] introduced a fixpoint operator which computes the definite consequences of the program. However, the
situation becomes more complicated in the presence of indefinite consequences in
disjunctive programs. In order to characterize such non-deterministic behavior of
disjunctive programs, we first introduce a closure operator which operates over the
lattice of sets of Herbrand interpretations $2^{2^{2\cal P}}$.

Definition 3.7 Let $P$ be a positive disjunctive program and $I$ be a set of interpreta-
tions. Then a mapping $T_{P} : 2^{2^{2\cal P}} \rightarrow 2^{2^{2\cal P}}$ is defined as

$$T_{P}(I) = \bigcup_{I' \subseteq I} T_{P}(I')$$

where the mapping $T_{P} : 2^{2^{2\cal P}} \rightarrow 2^{2^{2\cal P}}$ is defined as follows:

$$T_{P}(I) = \begin{cases} 
\emptyset, & \text{if } \{B_{1}, \ldots, B_{m}\} \subseteq I \text{ for some ground integrity constraint} \\
\{ J | \text{for each ground clause } C_{i} : A_{i} \leftarrow B_{i} \text{ from } P \} & \text{from } P \text{ such that } \{B_{1}, \ldots, B_{m}\} \subseteq I,
\\
J = I \cup \{ A_{i} \} & \text{for each } 1 \leq i \leq m \}, \text{ otherwise).
\end{cases}$$

Thus, $T_{P}(I)$ is the set of interpretations $J$'s such that for each clause $C_{i}$ whose
body is satisfied by $I$, $I$ is expanded into $J$ by adding one disjunct $A_{i}$ from the heads
of every such clause $C_{i}$. In particular, if $I$ does not satisfy an integrity constraint from $P$, $I$ is
removed in $T_{P}(I)$.

Example 3.10 Let $P$ be the program:

$$\{ a \leftarrow c, \ d \leftarrow c, \ c \leftarrow \neg a \land b \}.$$  

Then, $T_{P}(\{c\}) = \{ \{c, d, a\}, \{c, d, b\} \}$ and $T_{P}(\{\{c, d, a\}, \{c, d, b\}\}) = \{ \{c, d, a\}, \{c, d, b\}, \{c, d, a, b\}\}$.  

Definition 3.8 The ordinal powers of $T_{P}$ are defined as follows:

$$T_{P} \uparrow 0 = \{\emptyset\},$$

$$T_{P} \uparrow n + 1 = T_{P}(T_{P} \uparrow n),$$

$$T_{P} \uparrow \omega = \bigcup_{n < \omega} T_{P} \uparrow n,$$

where $n$ is a successor ordinal and $\omega$ is a limit ordinal.  

The above definition means that at the limit ordinal $\omega$ the closure retains interpre-
tations which are persistent in the preceding iterations. That is, for any interpreta-
tion $I$ in $T_{P} \uparrow \omega$, there is an ordinal $\alpha$ smaller than $\omega$ such that for every $n \in [\alpha, \alpha + 1)$, $I$ is included in $T_{P} \uparrow n$. Such a closure definition is also used in [Fages, 1991; Teusink, 1993a] for computing stable models of normal logic programs.

Fixpoint Semantics

Theorem 3.15 $T_{P} \uparrow \omega$ is a fixpoint.

Proof: When $I \in T_{P} \uparrow \omega$, suppose that there is no interpretation $J$ in $T_{P} \uparrow \omega$ such that $I \in T_{P}(\{J\})$. In this case, for any $n \in [\alpha, \omega)$ such that $J$ is not included in $T_{P} \uparrow n$. Then $I \notin T_{P} \uparrow n + 1$. This contradicts the fact that $I \in T_{P} \uparrow \omega$. Thus, $J \in T_{P} \uparrow \omega$. Conversely, if $I \in T_{P}(T_{P} \uparrow \omega)$, there is an interpretation $J$ in $T_{P} \uparrow \omega$ such that $I \in T_{P}(\{J\})$. Then $J$ is included in any $T_{P} \uparrow n$ for $n \in [\alpha, \omega)$ by definition. Thus $I \in T_{P} \uparrow n$ for any $n + 1 \leq n < \omega$. Hence, $I \in T_{P} \uparrow \omega$.  

Example 3.11 (cont. from Example 3.10) Given the program $P$, it becomes:

$$T_{P} \uparrow 1 = \{\{c\}\},$$

$$T_{P} \uparrow 2 = \{\{c, d, a\}, \{c, d, b\}\},$$

$$T_{P} \uparrow 3 = \{\{c, d, a\}, \{c, d, b\}, \{c, d, a, b\}\},$$

$$T_{P} \uparrow 4 = \{\{c, d, a\}, \{c, d, b\}, \{c, d, a, b\}\},$$

where $T_{P} \uparrow \omega = T_{P} \uparrow 3$.  

In the above example, the interpretation $\{c, d, a, b\}$ in $T_{P} \uparrow 3$ is pruned in $T_{P} \uparrow 4$ by the integrity constraint $\neg a \land b$, while the same interpretation is also generated from $\{c, d, a\}$ and $\{c, d, b\}$ in $T_{P} \uparrow 3$, hence $\{c, d, a, b\}$ remains in $T_{P} \uparrow 4$.

By Definition 3.8, the fixpoint closure presented above exists for any program and is
uniquely determined. Intuitively, the fixpoint characterizes a set of interpretations which are "generated" in a program by starting from the empty interpretation. Next we show that the fixpoint closure in fact contains what we want, i.e., the set of all possible models.

Lemma 3.16 Let $P$ be a positive disjunctive program. Then $I$ is a model of $P$ iff $I \in T_{P}(\{I\})$.

Proof: $I$ is a model of $P$ if and only if $I$ satisfies integrity constraints and for each clause $A_{i} \leftarrow B_{i} \land \ldots \land B_{m}$ in $P$, $\{B_{1}, \ldots, B_{m}\} \subseteq I$ implies $A_{i} \in I$ for some $A_{i}$ ($1 \leq i \leq l$).

Let $\mu(T_{P} \uparrow \omega) = \{ I | I \in T_{P} \uparrow \omega \text{ and } I \in T_{P}(\{I\}) \}$. Then, by Lemma 3.16, $\mu(T_{P} \uparrow \omega)$ represents the set of models of $P$ which are included in the fixpoint closure. Also let $\min(I) = \{ I \in I | \beta J \in I \text{ such that } J \subseteq I \text{ } \}$. Then the following relations hold.
Theorem 3.17 Let $P$ be a positive disjunctive program. Then,

(i) $\mathcal{PM}_P = \wp (T_P \uparrow \omega)$.

(ii) $\mathcal{MM}_P = \min (\wp (T_P \uparrow \omega))$.

Proof: (i) $I \in \wp (T_P \uparrow \omega)$

iff $I$ is included in $T_P \uparrow \omega$ and is a model of $P$ (by Lemma 3.16)

iff each $A_i$ in $I$ is included in the derived head of a ground clause $A_1 \vee \ldots \vee A_i \leftarrow B_1 \wedge \ldots \wedge B_m$ ($1 \leq i \leq l$) in $P$ and $I$ satisfies every integrity constraint in $P$

iff $I$ is the least Herbrand model of a consistent split Horn logic program $P$ of $P$ and each $A_i$ in $I$ is derived by the split clause $A_i \leftarrow B_1 \wedge \ldots \wedge B_m$ in $P$.

iff $I$ is in $\mathcal{PM}_P$.

(ii) The result follows from (i) and Proposition 3.5. $\square$

Corollary 3.18 A positive disjunctive program $P$ is inconsistent iff $\wp (T_P \uparrow \omega) = \emptyset$. $\square$

Example 3.12 (cont. from Example 3.11)

$\wp (T_P \uparrow \omega) = \{ \{c, d, a\}, \{c, d, b\}\}$, which coincides with the set of all possible models of $P$. $\square$

For definite logic programs, our fixpoint construction reduces to van Emden and Kowalski's fixpoint semantics [van Emden and Kowalski, 1976].

Corollary 3.19 Let $P$ be a definite logic program. Then $T_P \uparrow \omega$ contains the unique least Herbrand model of $P$. $\square$

3.4.2 Fixpoint Semantics for Normal Disjunctive Programs

Now we extend the fixpoint semantics of positive disjunctive programs to normal disjunctive programs. To this end, we first introduce a program transformation which translates a normal disjunctive program into a semantically equivalent positive disjunctive program.\footnote{The transformation is originally introduced in [Jonae et al., 1992] in a different form.}

Definition 3.9 Let $P$ be a normal disjunctive program. Then its 
\textit{epistemic transformation} is defined as the positive disjunctive program $P^*$ obtained from $P$ by replacing each clause containing default negation:

\[ A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge \neg B_{m+1} \wedge \ldots \wedge \neg B_n \ (m \neq n) \]  

with the following not-free clauses in $P^*$:

\[ \lambda_1 \vee \ldots \vee \lambda_i \vee KB_{m+1} \vee \ldots \vee KB_n \leftarrow B_1 \wedge \ldots \wedge B_m, \]  

\[ A_i \leftarrow \lambda_i \text{ for } i = 1, \ldots, l, \]  

\[ \lambda_i \leftarrow A_i \wedge \lambda_{k_i} \text{ for } i = 1, \ldots, l \text{ and } k = m+1, \ldots, n. \]  

In particular, each integrity constraint containing default negation is transformed into

\[ KB_{m+1} \vee \ldots \vee KB_n \leftarrow B_1 \wedge \ldots \wedge B_m. \]

Note here that each not-free clause in $P$ is included in $P^*$ as it is. $\square$

In the epistemic transformation, the newly introduced atom $KB_j$ means that $B_j$ is believed. With this epistemic reading, each default negation not $B_j$ in the body of a clause is rewritten in $\neg KB_j$, which means that $B_j$ is disbelieved, and shifted to the head of the clause. Also each $A_i$ is a newly introduced atom appearing nowhere in $P$ and is uniquely associated with each ground instance of a clause (3.1) from $P$.

An intuitive reading of the transformed clauses is that if $B_1, \ldots, B_m$ are true, then some $A_i$ ($1 \leq i \leq l$) becomes true via $\lambda_i$ when $B_{m+1}, \ldots, B_n$ are not true; otherwise, some $B_j$ ($m+1 \leq j \leq n$) is believed. The clause (3.5) has an effect to associate $\lambda_i$ with $A_i$, whenever $A_i$ is true and another disjunct $A_j$ is derived from (3.2) via $\lambda_k$. In this way, every normal disjunctive program $P$ is transformed into a positive disjunctive program $P^*$. Then we can construct the fixpoint of $P^*$ as presented in the previous section.

Let $I^*$ be an interpretation of $P^*$. Then $I^*$ is called canonical if $KB \in I^*$ implies $A \in I^*$ for any atom $A$ in $KB$. That is, in a canonical interpretation each believed atom has a justification. Given a set of interpretations $\mathcal{I}_P$, let

\[ \text{obj}_{I^*}(\mathcal{I}_P) = \{ I^* \in \mathcal{I}_P \mid I^* \uparrow \omega \text{ and } I^* \text{ is canonical}\}. \]
Now we present the fixpoint characterization of possible models in normal disjunctive programs. We first prove preliminary lemmas.

Lemma 3.20 Let $P$ be a normal disjunctive program. Then,

$$ST_P = \text{obj}_J(\min(\mu(T_P \downarrow \omega)))$$

Proof: Suppose that $I$ is in $\text{obj}_J(\min(\mu(T_P \downarrow \omega)))$. Let $I^*$ be a canonical interpretation in $\min(\mu(T_P \downarrow \omega))$ such that $I^* \cap \mathcal{H}B_P = I$. Then, for each ground clause of the form (3.1) from $P$, $(B_1, \ldots, B_n) \subseteq I^*$ implies either (i) $A_i \in I^*$ $(1 \leq i \leq l)$, or (ii) $3 \exists \, B_j \in I^* (m + 1 \leq j \leq n)$ by (3.2), (3.3), and (3.4).\footnote{When a clause (3.1) contains no atom, $(B_1, \ldots, B_n) \subseteq I^*$ implies $A_i \in I^*$ $(1 \leq i \leq l)$ as a special case of (i).}

In case of (i), $(B_{m+1}, \ldots, B_n) \cap I^* = \emptyset$ implies $(B_{m+1}, \ldots, B_n) \cap I = \emptyset$. Then there is a clause of the form:

$$A_1 \lor \ldots \lor A_i \leftarrow B_j \land \ldots \land B_m \quad (*)$$

in $P^I$. Since $(B_1, \ldots, B_m) \subseteq I^*$ and $A_i \in I^*$ implies $(B_1, \ldots, B_m) \subseteq I$ and $A_i \in I$, $I$ satisfies the clause $(*)$ in $P^I$. In case of (ii), since $I^*$ is canonical, $\mathcal{K}B_j \in I^*$ implies $B_j \in I^*$, and thus $B_j \in I$. In this case, the clause $(*)$ is not included in $P^I$. Thus, in both cases, $I$ satisfies every clause in $P^I$.

Suppose that there is an interpretation $J$ such that (a) $J \subset I$ and (b) $J$ satisfies each clause from $P^I$. Then, two conditions (a) and (b) are satisfied only if there is a clause $(*)$ such that $(B_1, \ldots, B_m) \subseteq J$ and for some two atoms $A_{i_1}$ and $A_{i_2}$ $(1 \leq i_1, i_2 \leq l, i_1 \neq i_2)$, $A_{i_1} \in J$ but $A_{i_2} \not\in J$. Without loss of generality, we can assume that just one such clause exists in $P^I$. Since $I$ does not contain atoms $B_{m+1}, \ldots, B_n$, the corresponding canonical interpretation $I^*$ does not contain $\mathcal{K}B_{m+1}, \ldots, \mathcal{K}B_n$ either. Thus, $(B_1, \ldots, B_n) \subseteq I$ implies $\exists \lambda_k \in I^*$ for some $1 \leq k \leq l$. Since $A_{i_1}, A_{i_2} \in I$ implies $A_{i_1}, A_{i_2} \in I^*$, $\lambda_k \in I^*$ implies $\lambda_k \in I^*$ by (3.5). Let $J^* = I^* \setminus \{A_{i_1}, \lambda_k\}$. Then, the interpretation $J^*$ satisfies all the clauses (3.2), (3.3), and (3.4) in $P^I$. This contradicts the fact that $I^*$ is a minimal model of $P^I$. Hence $I$ is also a minimal model of $P^I$, hence a stable model of $P$.\footnote{When a clause (3.1) contains no atom, $(B_1, \ldots, B_n) \subseteq I^*$ implies $A_i \in I^*$ $(1 \leq i \leq l)$ as a special case of (i).}

Conversely, suppose that $I$ is a stable model of $P$. Then, for each clause $C$ of the form (3.1) from $P$, let $I_1 = \cup \{\{A_1, B_1, \ldots, B_n\} \subseteq I, \{B_{m+1}, \ldots, B_n\} \cap I = \emptyset, A_i \in I (1 \leq i \leq l)\}$ and $I_K = \cup \{KB_j | (B_1, \ldots, B_n) \subseteq I \land B_j \in I (m + 1 \leq j \leq n)\}$. Let $I^* = I_1 \cup I_K$. Then, $I^*$ satisfies each clause (3.2), (3.3), (3.4), and (3.5) from $P^I$, and by the construction of $I^*$, $I^* \subseteq \mu(T_P \downarrow \omega)$. Now let us define $I^* = I \cup S$ where $S$ is a minimal subset of $I_1 \cup I_K$ such that each $A_i$ or $KB_j$ is chosen in a way that $I^*$ satisfies every clause in $P^I$. Note that for each atom $KB$ in $I^*$, $B \in I$ by definition, so $B \in I^*$. Hence $I^*$ is canonical. Next assume that there exists $J^* \subseteq \mu(T_P \downarrow \omega)$ such that $J^* \subseteq I^*$. Since we have defined $I^*$ as a minimal set with respect to the atoms from $I_1 \cup I_K$, the inclusion relation implies $J^* \cap \mathcal{H}B_P \subseteq I^* \cap \mathcal{H}B_P$. Then $3 \exists \lambda \in I^* \cap \mathcal{H}B_P$. Then there is a clause (3.2) in $P^I$ such that $(B_1, \ldots, B_m) \subseteq J^*$, $\lambda \in I^* \cap \mathcal{H}B_P$ for some $1 \leq i \leq l$ and $m + 1 \leq j \leq n$. Since $J^* \subseteq I^*$, $KB_j \in I^*$. As $I^*$ is canonical, $KB_j \in I^*$ implies $B_j \in I^*$. This is impossible from the condition (3.4). Thus, there is no $J^*$ which is smaller than $I^*$, hence $I^* \in \mu(T_P \downarrow \omega))$. Since $I^*$ is canonical, $I \in \text{obj}_J(\min(\mu(T_P \downarrow \omega)))$.

Lemma 3.21 Let $P$ be a normal logic program. Then,

$$ST_P = \text{obj}_J(\min(\mu(T_P \downarrow \omega)))$$

Proof: By Lemma 3.20, \text{obj}_J(\min(\mu(T_P \downarrow \omega))) is the set of all stable models of $P$. Since $I \in \text{obj}_J(\min(\mu(T_P \downarrow \omega)))$ implies $I \in \text{obj}_J(\mu(T_P \downarrow \omega))$, we show that the converse is also true. Assume that the converse does not hold. That is, there is a non-minimal model $I$ in $\text{obj}_J(\mu(T_P \downarrow \omega))$ and $3J \in \text{obj}_J(\min(\mu(T_P \downarrow \omega)))$ such that $J \subset I$. In this case, there exists an atom $A$ such that $A \in I \setminus J$. Let $I = I^* \cap \mathcal{H}B_P$ and $J = J^* \cap \mathcal{H}B_P$ for some canonical interpretations $I^*$ and $J^*$. Then, corresponding to (3.2), (3.3), and (3.4), there exist clauses:

$$\lambda \lor KB_{m+1} \lor \ldots \lor KB_n \leftarrow B_1 \land \ldots \land B_m$$

$$A \leftarrow \lambda$$

in $P^I$, where $(B_1, \ldots, B_n) \subseteq J^*$, $(B_1, \ldots, B_n) \subseteq J^*$, $\lambda \in I^*$, and $KB_j \in J^*$ $(m + 1 \leq j \leq n)$. Note here that the clause (3.5) becomes $\lambda \leftarrow A \land \lambda$ and is neglected. Since $J^*$ is canonical, $B_j \in J^*$. Then $J \subset I$ implies $B_j \in I^*$. But this is impossible from the third clause above.\footnote{When a clause (3.1) contains no atom, $(B_1, \ldots, B_n) \subseteq I^*$ implies $A_i \in I^*$ $(1 \leq i \leq l)$ as a special case of (i).}
Now we are ready to prove the main theorem.

**Theorem 3.22** Let $P$ be a normal disjunctive program. Then,

$PM_P = obj_j(\mu(TP^\top \uparrow \omega))$.

**Proof:** Let $I$ be a possible model of $P$. Then $I$ is a stable model of some coherent split normal logic program $P_s$ of $P$. By Lemma 3.21, for the transformed program $P_s^*$ of $P_s$, $I$ is in $obj_j(\mu(TP^s_\uparrow \uparrow \omega))$. Since $P_s$ is a program obtained by splitting each disjunctive clause in $P$, every interpretation included in $TP^\top \uparrow \omega$ is also obtained by splitting during the computation of $TP^\top \uparrow \omega$. Moreover, since $I$ satisfies each integrity constraint in $P_s^*$, it also satisfies the same integrity constraints in $P^\top$. Hence $I$ is also in $obj_j(\mu(TP^\top \uparrow \omega))$. The converse is also shown in the same manner. □

The above lemmas and theorem are direct extensions of Theorem 3.17.

**Corollary 3.23** Let $P$ be a normal disjunctive program. Then,

(i) $P$ is inconsistent iff $\mu(TP^\top \uparrow \omega) = \emptyset$.

(ii) $P$ is incoherent iff $obj_j(min(\mu(TP^\top \uparrow \omega))) = \emptyset$.

(iii) $P$ is $p$-incoherent iff $obj_j(\mu(TP^\top \uparrow \omega)) = \emptyset$. □

**Example 3.13** (cont. from Example 3.4) The program

$P = \{a \lor b \leftarrow, b \leftarrow a, c \leftarrow \neg a\}$

is transformed into the epistemic form:

$P^* = \{a \lor b \leftarrow, b \leftarrow a, \lambda \lor Ka \leftarrow, c \leftarrow \lambda, \leftarrow \lambda \land a\}$.

Then it becomes

$\mu(TP^\top \uparrow \omega) = \{a, b, Ka\}, \{b, Ka\}, \{b, c, \lambda\}, \{b, c, \lambda, Ka\}\}.$

Thus,

$\mu(TP^\top \uparrow \omega) = \{(a, b), (b, c)\}$,

which contains the possible models of $P$. On the other hand,

$min(\mu(TP^\top \uparrow \omega)) = \{(b, Ka), (b, c, \lambda)\}$,

then

$\mu(min(\mu(TP^\top \uparrow \omega))) = \{(b, c)\}$,

which contains the stable model of $P$. □

### 3.5 Computing Possible Models

In this section, we provide a bottom-up proof procedure to compute possible models of disjunctive programs. We assume in this section that a program is function-free and range-restricted, that is, a program containing no function symbol and any variable in a clause has an occurrence in a positive atom in the body. Such conditions are usually imposed on a program in the context of deductive databases.

#### 3.5.1 Bottom-up Model Generation Procedure

The algorithm we use to compute possible models in disjunctive programs is based on a bottom-up model generation proof procedure.

Let $P$ be a positive disjunctive program and $T_P$ be a set of interpretations of $P$. Let $T^0_P = \{\emptyset\}$. For $i \geq 0$ do:

1. For any $I \in T_P$, if there is an integrity constraint in $P$ of the form:

   $\leftarrow B_1 \land \ldots \land B_m$

   such that $I \models (B_1 \land \ldots \land B_m)\sigma$ for some ground substitution $\sigma$, then remove $I$ from $T_P$.

2. For any $I \in T_P$, for every clause $C_k$ in $P$ of the form:

   $C_k : A_1 \lor \ldots \lor A_l \leftarrow B_1 \land \ldots \land B_m \ (l \geq 1)$

   such that $I \models (B_1 \land \ldots \land B_m)\sigma$ for some ground substitution $\sigma$, put $I \cup C_k(A_j\sigma)$ into $T^+_{P_{k+1}}$ for every $j = 1, \ldots, l$.

3. Iterate the above two steps until it reaches the fixpoint $T^*_{P_{k+1}} = T^+_{P_{k+1}}$ which is closed under the above two operations.

In step 1, the procedure prunes interpretations which do not satisfy integrity constraints in the program. In step 2, the procedure generates the new set of interpretations $T^+_{P_{k+1}}$ from the given interpretations $T_P$ by performing forward reasoning.
based on hyperresolution\(^7\) and case-splitting on non-united derived clauses. Note here that since a program's range-restricted, each disjunct \(A_j\) generated in step 2 is always ground. Hence the soundness for unsatisfiability by case-splitting is guaranteed [Manthey and Bry, 1988]. Moreover, since we consider a finite function-free program, the above procedure always terminates in a finite step.

The connection between the above closure computation and the fixpoint semantics with the mapping \(T_p\) given in Section 3.4.1 is obvious. This correspondence can also be regarded as an extension of the relation between hyperresolution and van Emde and Kowalski's fixpoint semantics for definite logic programs [van Emde and Kowalski, 1976, Section 8].

Now we characterize the possible model semantics using the algorithm presented above. Let \(T_p\) be the fixpoint closure obtained by the above procedure. Then the following results directly follow from Theorem 3.17.

**Theorem 3.24** Let \(P\) be a positive disjunctive program. Then the following relations hold.

(i) \(PM_P = \mu(T_p^*)\).
(ii) \(MM_P = \min(\mu(T_p^*))\).

In particular, \(P\) is inconsistent iff \(\mu(T_p^*) = \emptyset\).

**Corollary 3.25** For a consistent positive disjunctive program \(P\) and a ground atom \(A\),

(i) \(GCWA(P) \models \neg A \iff A \not\in I \text{ for any } I \in \min(\mu(T_p^*))\).
(ii) \(WGCWA(P) \models \neg A \iff A \not\in I \in T_{\text{horn}(P^*)}\).
(iii) \(PWCA(P) \models \neg A \iff A \not\in I \text{ for any } I \in \mu(T_p^*)\).

**Proof:** (i) and (iii) directly follow from each definition and the results in Theorem 3.24. Since \(T_{\text{horn}(P^*)}\) contains a unique element which is the least Herbrand model of \(\text{horn}(P)\), (ii) also follows from the definition of the WGCWA.

For normal disjunctive programs, the following results hold by Lemmas 3.20, 3.21, and Theorem 3.22.

---

\(^7\)A \(1 \lor \ldots \lor A_i\) is said to be obtained from \(A_1 \lor \ldots \lor A_i \Rightarrow B_1 \land \ldots \land B_m\) and \(B_1, \ldots, B_m\) by hyperresolution [Robinson, 1965b].
Theorem 3.28 Let \( P \) be a normal disjunctive program and \( Q \) be a query. Then,
(i) \( Q \) is true in \( P \) iff for any \( I \in \text{obj}(\mu(T_{\text{pm}})) \), \( (x) \sigma \in I \) for some \( \sigma \).
(ii) \( Q \) is possibly true in \( P \) iff for some \( I \in \text{obj}(\mu(T_{\text{pm}})) \), \( (x) \sigma \in I \) for some \( \sigma \).
(iii) \( Q \) is false in \( P \) iff for any \( I \in \text{obj}(\mu(T_{\text{pm}})) \), \( (x) \sigma \notin I \) for any \( \sigma \). \( \blacksquare \)

Example 3.14 Let \( P \) be the program:
\[ \{ p(a) \lor p(b) \leftarrow \} \]
Then, \( q_1(x) \leftarrow p(x) \) is true, \( q_2 \leftarrow p(a) \) is possibly true, and \( q_3 \leftarrow p(c) \) is false. \( \blacksquare \)

By using Theorem 3.26 (iii) instead, the above result is also applicable to query answering under the disjunctive stable model semantics.

3.6 Computational Complexity

In a propositional positive disjunctive program, a minimal model exists whenever the program is satisfiable. Then the complexity of the existence problem (resp. the set-entailment problem) for the minimal model semantics is \( \text{NP} \)-complete (resp. \( \text{co-NP} \)-complete). Since the possible model semantics coincides with the minimal model semantics for positive inference, those complexity results also hold for the possible model semantics in propositional positive disjunctive programs. On the other hand, it is known that the complexity of the set-membership problem for the minimal model semantics is \( \Sigma_2^p \)-complete, and thus inferring negation under the GCWA is \( \Pi_2^p \)-complete [Eiter and Gottlob, 1993c]. By contrast, Chan [1993] has shown that, in a propositional positive disjunctive program, inferring negation under the WGCA or the PWA is still \( \text{NP} \)-complete. In particular, in the absence of integrity constraints both the WGCA and the PWA are tractable.

These observations tell us that the possible model semantics has the computational advantage over the minimal model semantics for inferring negation, since it does not increase the complexity more than the classical propositional entailment. Moreover, as shown in Section 3.2, since the PWA is more intuitive than the WGCA, it is concluded that the possible model semantics is the best choice among others from both the reasoning and computational points of view.

In this section, we prove that the complexity results for the possible model semantics is still within (co-)\( \text{NP} \), even in normal disjunctive programs. We show this fact by transforming possible models in a normal disjunctive program into stable models in a normal logic program.

**Definition 3.10** Let \( P \) be a normal disjunctive program. Then the pm-transformation transforms \( P \) into the normal logic program \( p(P) \) which is obtained from \( P \) by replacing each disjunctive clause:
\[ A_1 \lor \ldots \lor A_l \leftarrow \Gamma \]
in \( P \) with the following normal clauses and an integrity constraint:
\[ A_i \leftarrow \Gamma \land \text{not} A_i' \quad \text{for } i = 1, \ldots, l, \]
\[ A_i' \leftarrow \Gamma \land \text{not} A_i \quad \text{for } i = 1, \ldots, l, \]
\[ \leftarrow \Gamma \land A_i' \land \ldots \land A_l' \]
in \( p(P) \) where \( \Gamma \) denotes a conjunction in the body of the clause and each \( A_i' \) is a new atom not appearing in \( P \) and is uniquely introduced for each \( A_i \) in \( HBP \). \( \blacksquare \)

The intuitive meaning of the pm-transformation is that: when \( A_i \) becomes true by the disjunctive clause (3.7), we can make it true also by the corresponding normal clause (3.8) in \( p(P) \) by assuming that its complementary atom \( A_i' \) is not true. Else when \( A_i \) does not become true by (3.7), we will make \( A_i' \) true by assuming that \( A_i \) is not true in the corresponding normal clause (3.9) in \( p(P) \). The condition (3.10) states that when \( \Gamma \) is true, every \( A_i' \) cannot become true at the same time, that is, at least one \( A_i \) should be true. Thus, the transformed clauses represent every possible selection of disjuncts from the disjunctive head of the clause, which exactly characterizes every set of split clauses of (3.7).

Now we show that there is a one-to-one correspondence between the possible models of \( P \) and the stable models of \( p(P) \). We first present a preliminary lemma.

**Lemma 3.29** Let \( P \) be a normal disjunctive program. Then \( M \) is a possible model of \( P \) iff \( M \) is a possible model of \( P^M \).

**Proof:** \( M \) is a possible model of \( P \)
iff \( M \) is a stable model of some split normal logic program \( P_c \) of \( P \)
iff \( M \) is the least Herbrand model of \( M_{P_c} \)
iff \( M \) is the least Herbrand model of some split Horn logic program \( P^M \)
of \( P^M \)
iff \( M \) is a possible model of \( P^M \) \( \blacksquare \)

**Theorem 3.30** Let \( P \) be a normal disjunctive program and \( p(P) \) be its pm-transformation. Then \( \mathcal{M}_P = ST_{p(P)} \cap HB_P \) holds, where \( ST_{p(P)} \cap HB_P = \{ I \cap HB_P \mid I \in ST_{p(P)} \} \).
POSSIBLE MODEL SEMANTICS

Proof: (i) First we show that $\mathcal{P}_N = \mathcal{S}_N \cap \mathcal{H}_N$ holds for a positive disjunctive program $P$. Let $M$ be a possible model of a positive disjunctive program $P$. Then $M$ is the least Herbrand model of a split program $P_s$ of $P$. In this case, $M$ is also the least Herbrand model of a program in which each disjunctive clause (3.7) in $P$ is replaced with its split clauses $\{A_i \models \Gamma \mid A_i \in M \cap \{A_1, \ldots, A_j\}\}$. Now let us consider a Horn logic program $P_s$, which is obtained from $P$ by replacing each disjunctive clause (3.7) with clauses of $\{A_i \models \Gamma \mid A_i \in M \cap \{A_1, \ldots, A_j\}\} \cup \{A'_i \models \Gamma \mid A_i \in \{A_1, \ldots, A_j\} \setminus M\}$. Let $M'$ be the least Herbrand model of $P_s$. Then clearly $M = M' \cap \mathcal{H}_N$ holds. Here a program $P_s$ together with the integrity constraint (3.10) coincides with the program $\psi(P)^{M'}$. Since $M = \Gamma$ implies at least one $A_i \in M$, $M'$ satisfies the condition (3.10). Then $M'$ is also the least Herbrand model of $\psi(P)^{M'}$, hence a stable model of $\psi(P)$.

Conversely, let $M$ be a stable model of $\psi(P)$. Since $M$ satisfies the condition (3.10), $M \models \Gamma$ implies that at least one of the clauses (3.8) becomes $A_i \models \Gamma$ in $\psi(P)^{M'}$, and for each such clause $M \models \Gamma$ implies $A_i \in M$. In this case, there is a split program $P_s$ of $P$ in which each disjunctive clause (3.7) is replaced with its split clauses $\{A_i \models \Gamma \mid A_i \in M \cap \{A_1, \ldots, A_j\}\} \cup \{A'_i \models \Gamma \mid A_i \in \{A_1, \ldots, A_j\} \setminus M\}$. Since $M$ is the least Herbrand model of $\psi(P)^{M'}$, $M \cap \mathcal{H}_N$ is also the least Herbrand model of $P_s$, hence a possible model of $P$. (ii) Next we show that $\mathcal{P}_N = \mathcal{S}_N \cap \mathcal{H}_N$ holds for a normal disjunctive program $P$. Let $M$ be a possible model of a normal disjunctive program $P$. By Lemma 3.29, $M$ is also a possible model of a positive disjunctive program $P^M$. Then, by (i), there is a stable model $M'$ of $\psi(P^M)$ such that $M = M' \cap \mathcal{H}_N$, which is also the least Herbrand model of $\psi(P^M)^{M'}$. Since $\psi(P^M)^{M'} = \psi(P^{M'})^{M'} = \psi(P)^{M'}$, $M'$ is also a stable model of $\psi(P)$.

Conversely, let $M$ be a stable model of $\psi(P)$. Then $M$ is the least Herbrand model of $\psi(P)^{M'}$. Since $\psi(P)^{M'} = \psi(P^M)^{M'}$, $M$ is also the least Herbrand model of $\psi(P^M)^{M'}$, and a stable model of $\psi(P^M)$. Then, by (i), $M \cap \mathcal{H}_N$ is a possible model of $P^M$. Since $\psi(P^M) = \psi(P^{M'})^{M'}$, $M \cap \mathcal{H}_N$ is a possible model of $P$ by Lemma 3.29.

The above theorem presents that the possible models of any normal disjunctive program are expressed by the stable models of the corresponding transformed normal logic program.

COMPUTATIONAL COMPLEXITY

For the possible model semantics in propositional normal logic programs, both the existence problem and the set-membership problem are NP-complete, while the set-entailment problem is co-NP-complete [Marek and Truszczynski, 1991a; 1991b].

Theorem 3.31 Let $P$ be a propositional normal disjunctive program. Then, (i) deciding the existence of a possible model of $P$ is NP-complete. (ii) Deciding whether an atom is true in some possible model of $P$ is NP-complete. (iii) Deciding whether an atom is true in every possible model of $P$ is co-NP-complete.

Proof: Since possible models coincide with stable models in normal logic programs, each decision problem under the possible models semantics is (co-)NP-hard. To see that it is in (co-)NP, the pm-transformation efficiently translates each decision problem for possible models into the corresponding problem for stable models which is in (co-)NP, then the membership in (co-)NP follows.

Corollary 3.32 Inferring negation under the PWA* is co-NP-complete.  

Corollary 3.33 Inferring negation under the GCWA* or the WGCWA is both NP-complete.

The complexity results for disjunctive programs are summarized in Table 3.1. These results show that the frameworks based on the minimal/disjunctive stable model semantics introduce an additional source of complexity for minimality-checking, while this is not the case for computation of possible models due to its non-minimal feature.

We have already seen in the previous sections that the possible model semantics can provide flexible reasoning mechanisms compared with the minimal/disjunctive stable model semantics thanks to its non-minimal nature. The results of this section present that this unique feature of the possible model semantics also brings a computational advantage over those minimal model based semantics.

1 In [Marek and Truszczynski, 1991a; 1991b], integrity constraints are not included in a program. However, a program containing integrity constraints is easily reducible to the one without them by rewriting each integrity constraint — $G$ by the normal clause $\text{false} \leftarrow G$. 
The possible model semantics was also independently discovered by Chan [1993] under the name of the possible world semantics. Lately it was rediscovered by Decker [1992] under the name of the sustained model semantics. Decker and Casamayor [1993] have also shown that their sustained world assumption, which corresponds to the PWA, satisfies the properties such as cautious monotonicity, cumulativity and rationality in the sense of [Kraus et al., 1990]. These works have characterized the possible model semantics from different viewpoints, while they consider only positive disjunctive programs and extensions to normal disjunctive programs are not studied in the literature.

To distinguish two types of disjunctions, one may consider that instead of inserting integrity constraints, inserting cyclic clauses under the usual minimal model semantics is enough to interpret inclusive disjunctions. But this is not the case. Consider to
make the disjunction $a \lor b$ inclusive by adding cyclic clauses $a \leftarrow b$ and $b \leftarrow a$ to it. The resultant program now implies the equivalence $a \equiv b$. Applying it to the introductory example, we obtain $\text{land-animal} \equiv \text{aquatic}$, which is of course not our intention.

We have used integrity constraints to distinguish exclusive disjunctions from inclusive ones. Then if one wishes to simulate the GCWA under the PWA, it is enough to insert integrity constraints for each exclusive disjunction. Such a simulation is discussed in [Chan, 1993].

### 3.7.2 Fixpoint Semantics

A fixpoint semantics for disjunctive programs has been studied by several researchers. An early approach to provide a fixpoint semantics for positive disjunctive programs was given by [Minker and Rajasekar, 1990]. In the paper, they developed a fixpoint operator which operates on states, sets of positive disjunctions from the extended Herbrand base. Then they showed that its fixpoint closure characterizes the minimal model semantics of positive disjunctive programs. Our fixpoint semantics is basically different from theirs in the following points. First, our fixpoint operator is designed to compute possible models as well as minimal models of disjunctive programs. Second, our fixpoint semantics is well-defined not only for positive disjunctive programs, but also for every normal disjunctive program. Third, our fixpoint construction is based on the manipulation of standard Herbrand interpretations and does not require any extension of the Herbrand base. The state based fixpoint semantics have also been developed for stratified disjunctive programs in [Rajasekar and Minker, 1989] and for normal disjunctive programs in [Baral et al., 1992a; Przymusinski, 1990b; Przymusinski, 1991b; Reed et al., 1991] provide a different fixpoint semantics which characterizes logical consequences of a positive disjunctive program.

Fernandez and Minker [1991b] have presented a fixpoint semantics for stratified disjunctive programs using a fixpoint operator over sets of interpretations. With this fixpoint operator, they have shown that its iterative fixpoint characterizes the perfect models of a stratified disjunctive program. In [Fernandez and Minker, 1992; Fernandez et al., 1993], the result is further extended to normal disjunctive programs, in which they have developed a method for computing stable models by transforming a normal disjunctive program into a stratified disjunctive program with integrity constraints. Their approach is close to ours, however, an essential difference is that their fixpoint operator computes minimal and stable models, while ours also computes possible models. Inoue and Sakama [1992] developed yet another fixpoint semantics for computing minimal and stable models but not possible models.

### Discusssion

Decker [1992; 1994] has also developed a fixpoint semantics of disjunctive programs. His fixpoint operator maps a disjunction of interpretations into a disjunction of interpretations, and computes sustained models which is equivalent to possible models. However, he provides the fixpoint semantics only for positive disjunctive programs, and its extension to normal disjunctive programs is not discussed.

Ross and Topor [1988] have given a fixpoint construction for positive disjunctive programs to characterize the semantics of the DDR, but their fixpoint operator computes the least Herbrand model of a transformed Horn program and does not characterize any model theoretical meaning of the original program.

### 3.7.3 Proof Procedure

Proof procedures for disjunctive programs are developed by several researchers. Fernandez et al. [Fernandez and Minker, 1991a; Fernandez et al., 1993] develop a model generation proof procedure for computing minimal and stable models of disjunctive programs using a similar program transformation to ours. Compared with their approach, our algorithm is designed for computing not only minimal/stable models but also possible models of a program, and is easily realizable in a non-deterministic or-parallel environment like [Inoue et al., 1992]. The model generation procedure presented in Section 3.5 might be considered as a variant of SATCHMO [Manthey and Bry, 1988] or MGTP [Inoue et al., 1992], but these procedures are designed to compute minimal/stable models and different from ours. For computing possible models, Chan [1993] presents a different procedure which, given a positive disjunctive program $P$ and its model $M$, finds a subset of $M$ that is also a possible model of $P$.

We have also presented a method of using a bottom-up procedure to evaluate queries under the possible model semantics. As an alternative approach, we can design a top-down proof procedure for the possible model semantics as follows. In Section 3.6 we have presented that possible models of a normal disjunctive program can be expressed in terms of stable models of a normal logic program by using the pm-transformation. This means that, using the pm-transformation, a top-down proof procedure for the stable model semantics of normal logic programs can also be used as a procedure for the possible model semantics of normal disjunctive programs. For instance, Eshghi and Kowalski's [1989] abductive proof procedure is known to be correct with respect to call-consistent normal logic programs. Since the pm-transformation preserves the call-consistency, the abductive procedure is also used as a proof procedure for the possible model semantics. For positive disjunctive programs, yet other top-down procedures are developed in [Sakama, 1989; Decker, 1992; Decker and Casamayor, 1993].
3.8 Summary

In this chapter, we have introduced the possible model semantics for positive and normal disjunctive programs, which is an alternative non-minimal model approach to the declarative semantics of disjunctive programs. The possible model semantics gives a uniform framework to treat both inclusive and exclusive disjunctions in a program, and provides a flexible negative inference mechanism compared with the previously proposed closed world assumptions. The possible model semantics was also characterized by a new fixpoint semantics of disjunctive programs.

For computing possible models, we have presented a bottom-up model generation proof procedure for positive and normal disjunctive programs. The procedure is sound and complete with respect to the possible model semantics as well as the minimal/stable model semantics in function-free range-restricted programs. We have also shown that the possible model semantics has a computational advantage over the minimal/stable model semantics.

In normal disjunctive programs, we have defined the possible model semantics based on the stable model semantics. However, since its definition is given through the set of split normal logic programs, it is easy to construct another version of the possible model semantics based on any semantics of normal logic programs other than the stable model semantics. In this sense, the possible model semantics presented in this chapter provides a fairly general framework independent of any specific semantics. In other words, it establishes the principle of possibilism as a semantical counterpart of the traditional minimalism, which contributes to enrich our perspectives for commonsense reasoning in logic programming and artificial intelligence.

Chapter 4

Relating Disjunctive Logic Programs to Default Theories

In this chapter, we present the relationship between disjunctive programs and Reiter's default logic. We first point out the problem of previously proposed approaches, and propose an alternative default translation of normal disjunctive programs. The results are applied to extended disjunctive programs, and a correspondence between default logic and Gelfond et al.'s disjunctive default logic is investigated. We also address the connections between disjunctive programs and other major nonmonotonic formalisms such as Moore's autoepistemic logic and McCarthy's circumscription. The possible model semantics of disjunctive programs is also characterized by autoepistemic logic.

4.1 Introduction

Logic programming realizes a kind of default reasoning in the presence of default negation in a program. Such default reasoning is, on the other hand, known as nonmonotonic reasoning in artificial intelligence. Recent studies have shed light on the relationship between logic programming semantics and nonmonotonic reasoning, and it is now known that each of these areas relate to the other in a wide scope.

Default logic initially introduced by Reiter [1980] is known as one of the major formalisms of nonmonotonic reasoning in AI, and it turned out that default logic is closely related to the declarative semantics of logic programming. Bidoit and Pradeaux [1991a; 1991b] have firstly investigated the relationship between logic programming and default logic and introduced a positivist default theory for stratifiable and non-stratifiable logic programs. Marek and Truszczyński [1988] have also developed transformations from logic programs to default theories, and shown a one-to-one
correspondence between the stable models of a logic program and its corresponding default extensions. The result was further extended by Gelfond and Lifschitz [1991] to logic programs with classical negation, in which they present a connection between answer sets of a program and default extensions of its corresponding default theory.

It is often said that a difficulty of Reiter's default logic arises when one considers default reasoning with disjunctive information. Using a popular example from [Poole, 1989], when we consider the default rules:

\[ \text{lh-usable} \land \neg \text{lh-broken} \quad : \quad \text{lh-usable} \]

\[ \text{rh-usable} \land \neg \text{rh-broken} \quad : \quad \text{rh-usable} \]

with the disjunctive formula:

\[ \text{lh-broken} \lor \text{rh-broken}. \]

they have a single extension containing both \text{lh-usable} and \text{rh-usable}, which is unintuitive since the justifications of the defaults \( \neg \text{lh-broken} \) and \( \neg \text{rh-broken} \) cannot hold at the same time.

In the context of disjunctive programs, Bidoit and Hull [1986] present a one-to-one correspondence between the minimal models of a positive disjunctive program \( P \) and the extensions of a default theory which is obtained from \( P \) by adding defaults:

\[ \neg A \]

for each atom \( A \) from \( \text{HL}_P \). In the presence of default negation in a program, Bidoit and Froidevaux [1991a] present a relationship between a stratified disjunctive program and its associated positivist default theory. However, as pointed out in this chapter, Bidoit and Froidevaux's positivist default theory contains a problem and cannot be applicable to a disjunctive program with negation even if it is stratifiable. Recently, Gelfond et al. [1991] proposed a new framework called disjunctive default logic which is a direct extension of Reiter's default logic. While the disjunctive default logic is closely related to the answer set semantics of extended disjunctive programs, it remains open whether there is a correspondence between Reiter's default logic and disjunctive programs in general.

In this chapter, we study the relation between disjunctive programs and default theories. We first point out the problem of Bidoit and Froidevaux's positivist default theories and propose an alternative correct default translation of normal disjunctive programs. The result is further extended to a transformation from extended disjunctive programs to default theories. We then present a connection between default theories and disjunctive default theories through extended disjunctive programs. Furthermore, we investigate correspondences between disjunctive programs and other major nonmonotonic formalisms, circumscription and autoepistemic logic, and present methods of expressing stable models and possible models of disjunctive programs in terms of those nonmonotonic formalisms.

The rest of this chapter is organized as follows. In Section 4.2, we introduce basic notations of default logic. In Section 4.3, we point out problems of previously studied results and introduce an alternative default translation of normal disjunctive programs. In Section 4.4, we extend the results to extended disjunctive programs, and present a connection between default and disjunctive default theories. In Section 4.5, we relate disjunctive programs to autoepistemic logic and circumscription, and show an autoepistemic translation of the possible model semantics. Section 4.6 summarizes this chapter.

### 4.2 Default Logic

Classical first-order logic is always monotonic, that is, adding new axioms will never invalidate old theorems. However, this monotonic feature of classical logic is not necessarily adequate for formulating human commonsense reasoning, since we often confront situations where complete knowledge is not available, but nevertheless must draw conclusions. In such situations, we have to jump to conclusions with suitable default assumptions, and such conclusions might be revised after getting more accurate information.

Nonmonotonic logics are frameworks for such reasoning and they play important roles today as theoretical tools for commonsense reasoning in AI. Among many nonmonotonic formalisms, Reiter's default logic [Reiter, 1980] is known as a simple and powerful framework.

In default logic, nonmonotonic inference is presented by a default rule. For instance, the sentence "birds normally fly" can be represented as

\[ \text{bird}(x) : \text{fly}(x) \]

\[ \text{fly}(x) \]

The above rule is read as "if \( x \) is a bird and it is consistent to assume that \( x \) flies, then conclude that \( x \) flies". Thus, if all we know about Tweety is that she is a bird \( \text{bird(Tweety)} \), then it is concluded that she can fly \( \text{fly(Tweety)} \). However, if we learn that Tweety is a penguin, and we already know that penguins cannot fly, then the
A default theory may have none, one or multiple extensions in general. Any extension of a default theory is minimal, that is, for any two extensions \( E \) and \( F \), \( E \subseteq F \) implies \( E = F \). In particular, if the set of all justification-free defaults from \( D \) is inconsistent,\(^2\) \( D \) has the unique contradictory extension which consists of every formula in the language of \( D \). An extension is called consistent if it is not contradictory.

The following results are due to [Reiter, 1980], which present basic properties of default extensions.

**Proposition 4.1** Let \( D \) be a default theory. Then \( E \) is an extension of \( D \) if \( E = \bigcup_{i=0}^{\infty} E_i \) where

\[
E_0 = \{ F \mid F \text{ is a first-order formula in } D \}.
\]

\[
E_{i+1} = Th(E_i) \cup \{ \gamma \mid \frac{\alpha: \beta_1, \ldots, \beta_n}{\gamma} \in D \text{ where } \alpha \in E, \text{ and } \neg \beta_1, \ldots, \neg \beta_n \notin E \}. \quad \square
\]

**Proposition 4.2** Let \( D \) be a default theory and \( E \) be an extension of \( D \). Then,

\[
E = Th(\{ \gamma \mid \frac{\alpha: \beta_1, \ldots, \beta_n}{\gamma} \in D \text{ where } \alpha \in E, \text{ and } \neg \beta_1, \ldots, \neg \beta_n \notin E \}). \quad \square
\]

Note that the converse of the above proposition does not hold in general.

**Example 4.1** Let \( D \) be the default theory:

\[
\frac{a: b}{\neg a}, \frac{b: a}{\neg b}.
\]

Then \( E = Th(\langle a, b \rangle) \) satisfies the above equation, while the extension of \( D \) is \( \emptyset \). \( \square \)

A set \( E \) satisfying the equation in Proposition 4.2 is called a weak extension of \( D \) [Marek and Truszczyński, 1989b].

### 4.3 Default Translation of Normal Disjunctive Programs

In this section, we first review previously studied results on translating normal disjunctive programs into default theories. The problem of those approaches is pointed out, and an alternative correct default translation of normal disjunctive programs is presented.

---

1 Here we consider quantifier-free defaults for simplicity reasons. Such a convention is also assumed in [Gelfond et al., 1991].

2 A set of justification-free defaults is inconsistent iff its extension is inconsistent.
4.3.1 Positivist Default Theory Revisited

To relate logic programming with default theories, Bidoit and Froidevaux [1991a] have presented a transformation from disjunctive programs to so-called positivist default theories. According to [Bidoit and Froidevaux, 1991a], the transformation is presented as follows.

Definition 4.2 ([Bidoit and Froidevaux, 1991a]) Let $P$ be a normal disjunctive program. Then the positivist default theory $D$ associated with $P$ is constructed as follows:

(i) For each not-free clause $A_1 \lor \ldots \lor A_i \leftarrow B_1 \land \ldots \land B_m$ in $P$, its corresponding first-order formula $B_1 \land \ldots \land B_m \supset A_1 \lor \ldots \lor A_i$ is in $D$.

(ii) Each clause containing not in its body $A_1 \lor \ldots \lor A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n$ in $P$ is transformed into the following default in $D$:

\[
\frac{B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n}{A_1 \lor \ldots \lor A_i}
\]

(iii) For each atom $A$ in $\mathcal{HB}_P$, the following CWA-default is in $D$:

\[
\frac{\neg A}{\neg A}
\]

(iv) Nothing else is in $D$. $\square$

Then, [Bidoit and Froidevaux, 1991a] claims that a positivist default theory associated with a stratified disjunctive program always has at least one extension ([Bidoit and Froidevaux, 1991a, Theorem 3.5]). Moreover,

([Bidoit and Froidevaux, 1991a, Theorem 4.13]) Let $P$ be a stratifiable logical database. Then $M$ is a perfect model for $P$ iff $M$ is a default model for its positivist default theory.

In the above theorem, a “default model” means an Herbrand model of an extension and a “logical database” corresponds to a disjunctive program in our terminology. However, the following example shows that there exists a stratified disjunctive program whose positivist default theory does not have any extension.

4.3.2 Representing Normal Disjunctive Programs by Default Theories

Now we present an alternative transformation from disjunctive programs to default theories.

Definition 4.3 Let $P$ be a normal disjunctive program. Then its associated default theory $D_P$ is constructed as follows:

(i) Each clause $A_1 \lor \ldots \lor A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n$ in $P$ is transformed into the following default in $D_P$:

\[
\frac{\neg B_{m+1} \land \ldots \land \neg B_n}{B_1 \land \ldots \land B_m \supset A_1 \lor \ldots \lor A_i}
\]

(4.2)

In particular, each integrity constraint: $\leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n$ in $P$ is transformed into the following default in $D_P$:

\[
\frac{\neg B_{m+1} \land \ldots \land \neg B_n}{B_1 \land \ldots \land B_m \supset \text{false}}
\]

According to our analysis, the proof of Lemma 3.3 in [Bidoit and Froidevaux, 1991a] seems to contain an error. However, if a disjunctive program contains no not, the positivist default theory reduces to the defaults presented in [Bidoit and Hull, 1998] and it works well.
RELATING DISJUNCTIVE PROGRAMS TO DEFAULT THEORIES

(ii) For each atom \( A \) in \( HB_P \), the following CWA-default is in \( D_P \):

\[
\frac{\neg A}{A}
\]

(4.3)

(iii) Nothing else is in \( D_P \). \( \square \)

Notice that the difference between the positivist default theory and the associated default theory is the transformation of clauses containing default negation. That is, in our transformation, a normal disjunctive program is translated into a prerequisite-free default theory. Both transformations coincide for positive disjunctive programs.

Now several remarks are in order. Marek and Truszczynski [1989a] have developed three kinds of transformations \( tr_1 \), \( tr_2 \) and \( tr_3 \) from normal logic programs to default theories. Considering these transformations in the context of disjunctive programs, the transformation (4.2) can be regarded as an extension of their transformation \( tr_2 \) except that we are considering the CWA-default (4.3). While a transformation based upon \( tr_3 \) corresponds to the positivist default theory presented in the previous section, it has already turned out inappropriate to characterize disjunctive programs. A \( tr_1 \)-based transformation translates each clause into the default:

\[
\frac{B_1 \land \ldots \land B_n : \neg B_{n+1}, \ldots, \neg B_m}{A_1 \lor \ldots \lor A_l}
\]

together with CWA-defaults for each atom.

The difference between \( tr_1 \) and \( tr_3 \) is that in \( tr_3 \) each not-free clause is transformed into a first-order formula in \( D \), while in \( tr_1 \) it is transformed into a justification-free default in \( D \). However, this \( tr_3 \)-based transformation is also inappropriate to characterize disjunctive programs as the following example shows.

Example 4.3 Let us consider the program:

\[
\{ a \leftarrow b, \quad b \leftarrow a, \quad a \lor b \leftarrow \}
\]

Then, by the above \( tr_3 \)-based transformation, it is translated into the set of defaults:

\[
\{ \frac{b}{a}, \quad \frac{a}{b}, \quad a \lor b, \quad \frac{\neg a}{\neg b}, \quad \frac{\neg b}{\neg a} \}
\]

which has no extension. \( \square \)

DEFAULT TRANSLATION OF NORMAL DISJUNCTIVE PROGRAMS

These observations tell us that, from the viewpoint of extending three default transformations for normal logic programs in [Marek and Truszczynski, 1989a], the \( tr_2 \)-based transformation is the only candidate that can be used to characterize the semantics of disjunctive programs.

Then we verify the correctness of the transformation. We first address some features of prerequisite-free default theories.

Lemma 4.3 Let \( D \) be a prerequisite-free default theory. Then \( E \) is an extension of \( D \) iff

\[
E = Th(\{ \gamma \mid \beta_1, \ldots, \beta_n \in D \text{ where } \neg \beta_1, \ldots, \neg \beta_n \notin E \}).
\]

Proof: By Proposition 4.1, \( E \) is an extension of \( D \) iff \( E = \bigcup_{i=0}^{\infty} E_i \) where

\[
E_0 = \{ F \mid F \text{ is a first-order formula in } D \},
\]

\[
E_{i+1} = Th(E_i) \cup \{ \gamma \mid \beta_1, \ldots, \beta_n \in D \text{ where } \neg \beta_1, \ldots, \neg \beta_n \notin E \}.
\]

Then \( E_i = Th(E_1) \) for \( i \geq 2 \), and the result immediately follows. \( \square \)

The above lemma presents that prerequisite-free default theories provide a sufficient condition to assure the converse of Proposition 4.2. That is, the notions of weak extensions and extensions coincide for prerequisite-free default theories [Marek and Truszczynski, 1989b].

The above result is further simplified as follows. Let \( D \) be a default theory and \( E \) be a set of formulas. Then, let \( D^E \) be a default theory which is obtained from \( D \) by

\[
D^E = \{ \frac{\alpha}{\gamma}, \quad \frac{\neg \beta_1, \ldots, \neg \beta_n}{\gamma} \text{ is a ground instance of a default in } D \}
\]

and \( \neg \beta_1, \ldots, \neg \beta_n \notin E \}

where \( D^E \) is called the reduct of \( D \) with respect to \( E \) [Gelfond et al., 1991]. Then the following property holds.

Lemma 4.4 ([Gelfond et al., 1991]) A set of formulas \( E \) is an extension of a default theory \( D \) iff \( E \) is an extension of the justification-free default theory \( D^E \). \( \square \)

From the above two lemmas, we get the following result.

Proposition 4.5 Let \( D \) be a prerequisite-free default theory. Then \( E \) is an extension of \( D \) iff \( E = Th(D^E) \). \( \square \)
Now we are ready to prove the main result of this section. Before that, we recall the following result for positive disjunctive programs.

**Lemma 4.6** ([Bidoit and Hull, 1986; Lobo and Subrahmanian, 1992])
Let \( P \) be a positive disjunctive program. If \( E \) is a consistent extension of \( D_p \), then \( E \cap \mathcal{HB}_p \) is a minimal model of \( P \).

**Theorem 4.7** Let \( P \) be a normal disjunctive program and \( D_p \) be its associated default theory. Then the following relations hold.

(i) If \( M \) is a stable model of \( P \), then there is an extension \( E \) of \( D_p \) such that \( M = E \cap \mathcal{HB}_p \).

(ii) If \( E \) is a consistent extension of \( D_p \), then \( M = E \cap \mathcal{HB}_p \) is a stable model of \( P \).

**Proof** (i) Suppose that \( M \) is a stable model of \( P \) and let \( E = Th(M \cup \neg M) \) where \( \neg M = \{ \neg A \mid A \in \mathcal{HB}_p \setminus M \} \). Then, for each clause \( A_1 \lor \ldots \lor A_t \leftarrow B_1 \land \ldots \land B_m \) in \( P^M \), the corresponding formula \( B_1 \land \ldots \land B_m \lor A_1 \lor \ldots \lor A_t \) is in \( D_p^E \). Since \( M \) is a minimal model of \( P^M \) and \( D_p^E = P^M \cup \{ \neg A \mid A \notin M \} \), \( M \) is also a minimal model of \( D_p^E \). Thus, \( Th(M \cup \neg M) = Th(D_p^E) \) holds. Therefore, by Proposition 4.5, \( Th(M \cup \neg M) \) is an extension of \( D_p \), and since \( Th(M \cup \neg M) \cap \mathcal{HB}_p = M \), the result follows.

(ii) When \( E \) is a consistent extension of \( D_p \), \( E = Th(D_p^E) \) holds by Proposition 4.5. Let \( M = E \cap \mathcal{HB}_p \). Then, for each formula \( B_1 \land \ldots \land B_m \lor A_1 \lor \ldots \lor A_t \leftarrow B_1 \land \ldots \land B_m \) in \( D_p^E \), the corresponding clause \( A_1 \lor \ldots \lor A_t \leftarrow B_1 \land \ldots \land B_m \) is in \( P^M \). Since \( E \) is also an extension of \( D_p^E \), \( M \) is a minimal model of \( P^M \) (by Lemma 4.6). Hence \( M \) is a stable model of \( P \).

**Corollary 4.8** Let \( P \) be a normal disjunctive program. Then,

(i) \( P \) is inconsistent iff \( D_P \) has the contradictory extension.

(ii) \( P \) is consistent but incoherent iff \( D_P \) has no extension.

The above theorem presents a one-to-one correspondence between the stable models of a normal disjunctive program and the consistent extensions of its associated default theory. The above results also reduce to the corresponding results in [Marek and Truszczynski, 1989a] for normal logic programs.

**DEFAULT TRANSLATION OF EXTENDED DISJUNCTIVE PROGRAMS**

**Example 4.4** ([Gelfond et al., 1991]) Let \( P \) be the program consisting of the clauses:

\[
\begin{align*}
lh\text{-usable} & \leftarrow \neg ab_1, \\
rh\text{-usable} & \leftarrow \neg ab_2, \\
ab_1 & \leftarrow lh\text{-broken}, \\
ab_2 & \leftarrow rh\text{-broken}, \\
lh\text{-broken} \lor rh\text{-broken} & \leftarrow .
\end{align*}
\]

These clauses are transformed into the following defaults in \( D_p \):

\[
\begin{align*}
\neg ab_1 & \leftarrow lh\text{-usable}, \\
\neg ab_2 & \leftarrow rh\text{-usable}, \\
lh\text{-broken} & \leftarrow ab_1, \\
rh\text{-broken} & \leftarrow ab_2
\end{align*}
\]

with the CWA-defaults:

\[
\begin{align*}
\neg lh\text{-broken} & \leftarrow \neg rh\text{-broken}, \\
\neg rh\text{-broken} & \leftarrow \neg lh\text{-usable}, \\
\neg lh\text{-usable} & \leftarrow \neg rh\text{-usable}, \\
\neg ab_1 & \leftarrow \neg ab_2
\end{align*}
\]

Then \( D_P \) has two extensions such that the sets of all atoms from them become 

\[
\{lh\text{-usable}, rh\text{-broken}, ab_2\} \quad \text{and} \quad \{rh\text{-usable}, lh\text{-broken}, ab_1\},
\]

which coincide with the stable models of \( P \).
4.4.1 Representing Extended Disjunctive Programs by Default Theories

An extended disjunctive program is a disjunctive program containing classical negation along with default negation in the program [Gelfond and Lifschitz, 1991]. As presented in Chapter 2, an extended disjunctive program $P$ is translated into a normal disjunctive program by considering its positive form $P^+$. Using this translation, we extend the results in the previous section to extended disjunctive programs.

Given an extended disjunctive program and its answer sets, let $S^+$ be a positive form of an answer set $S$ where each negative literal $\neg A$ in $S$ is rewritten by $A'$ in $S^+$. Then the following relationship holds, which is a straightforward extension of Theorem 4.10.

\[ S^+ = E \cap \mathcal{H}B_{P^+} \]

We say that $S^+$ is a positively consistent extension of $P^+$ if it does not contain a pair of complementary atoms $A$ and $A'$. The next theorem directly follows from the above proposition and lemma, which presents a one-to-one correspondence between the consistent answer sets of a program and the (positively) consistent extensions of its associated default theory.

Theorem 4.11 Let $P$ be an extended disjunctive program.

(i) If $S$ is a consistent answer set of $P$, then there is an extension $E$ of $D_{P^+}$ such that $S^+ = E \cap \mathcal{H}B_{P^+}$.

(ii) If $E$ is a positively consistent extension of $D_{P^+}$, then $S^+ = E \cap \mathcal{H}B_{P^+}$ is a positive form of an answer set $S$ of $P$. □

4.11 Theorem Let $P$ contain a pair of complementary atoms.

Then the result for extended logic programs [Gelfond and Lifschitz, 1991, Proposition 2].

Proposition 4.9 Let $P$ be an extended disjunctive program. Then a consistent set $S$ is an answer set of $P$ iff $S^+$ is a stable model of $P^+$. □

Since an extended disjunctive program reduces to a normal disjunctive program by considering its positive form, we can directly apply Definition 4.3 to give an associated default theory for an extended disjunctive program. We first rephrase Theorem 4.7 for our current use.

Lemma 4.10 Let $P$ be an extended disjunctive program.

(i) If $M$ is a stable model of $P^+$, then there is an extension $E$ of $D_{P^+}$ such that $M = E \cap \mathcal{H}B_{P^+}$.

(ii) If $E$ is an extension of $D_{P^+}$, then $M = E \cap \mathcal{H}B_{P^+}$ is a stable model of $P^+$. □

4.11 Theorem Let $P$ be an extended disjunctive program.

(i) If $S$ is a consistent answer set of $P$, then there is an extension $E$ of $D_{P^+}$ such that $S^+ = E \cap \mathcal{H}B_{P^+}$.

(ii) If $E$ is a positively consistent extension of $D_{P^+}$, then $S^+ = E \cap \mathcal{H}B_{P^+}$ is a positive form of an answer set $S$ of $P$. □

4.12 Corollary Let $P$ be an extended disjunctive program. Then, $P$ has a positively consistent extension $D_{P^+}$ such that $S^+$ is the unique extension $\text{Th}(\{a, \neg b, a'\})$. □

4.13 Corollary Let $P$ be an extended disjunctive program. Then, $P$ has a positively consistent extension $D_{P^+}$ such that $S^+$ is the unique extension $\text{Th}(\{a, \neg b, a'\})$. □

4.14 Theorem Let $P$ be an extended disjunctive program. Then, $P$ has a positively consistent extension $D_{P^+}$ such that $S^+$ is the unique extension $\text{Th}(\{a, \neg b, a'\})$. □

The results of Theorem 4.11 and 4.13 present that the answer set semantics of extended disjunctive programs is also characterized by Reiter's default theories.

4.15 Theorem Let $P$ be an extended disjunctive program. Then, $P$ has a positively consistent extension $D_{P^+}$ such that $S^+$ is the unique extension $\text{Th}(\{a, \neg b, a'\})$. □

Clearly the above results reduce to the case of extended logic programs in the absence of disjunctions in a program. It should be noted that when a program has no consistent answer set, we cannot apply Theorem 4.11 in a straightforward manner.

Corollary 4.12 Let $P$ be an extended disjunctive program. If $L_P$ is the unique answer set of $P$, then $D_{P^+}$ has no positively consistent extension. □

The converse of the above corollary does not hold in general.

Example 4.5 Let $P$ be the extended logic program:

\[ \{ a \leftarrow \neg b, \neg a \} \]

which has no answer set. In this case, its positive form $P^+$ becomes

\[ \{ a \leftarrow \neg b, a' \} \]

and its associated default theory $D_{P^+}$ is

\[ \{ a', \vdash b, : a, : \neg b, : \neg a', : \neg a' \} \]

which has the unique extension $\text{Th}(\{a, \neg b, a'\})$. □

To characterize a program having no consistent answer set, consider a program $P_{E^F}$ which is the reduct of $P$ with respect to $L_P$. By the definition of answer sets, $L_P$ is the answer set of $P$ iff $L_P$ is the answer set of $P_{E^F}$. Let $P_{E^F}$ be a positive form of $P_{E^F}$. Then the following result holds.

Theorem 4.13 Let $P$ be an extended disjunctive program. Then,

(i) $P$ has a positively consistent extension $D_{P_{E^F}^+}$ which is a consistent extension but no positively consistent extension.

(ii) $P$ has no answer set iff either $D_{P_{E^F}^+}$ has a positively consistent extension or $D_{P_{E^F}^+}$ has the contradictory extension. □

The results of Theorem 4.11 and 4.13 present that the answer set semantics of extended disjunctive programs is also characterized by Reiter's default theories.

\[ \text{Gelfond and Lifschitz, 1991] presents a default translation of extended logic programs, which is an extension of} \text{fr. of [Marek and Truszczyński, 1989] and different from ours.} \]
4.4.2 Relationship to Disjunctive Default Theory

Disjunctive default logic, recently proposed by Gelfond et al. [1991], is known as one of the extensions of Reiter’s default logic, which is devised to treat default reasoning with disjunctive information. In this section, we investigate the connection between disjunctive default theories and associated default theories presented in the previous sections.

A disjunctive default theory \( \Delta \) is a set of defaults of the form:

\[
\Delta = \{ \alpha : \beta_1, \ldots, \beta_m \mid \gamma_1 \lor \cdots \lor \gamma_n \}
\]

(4.4)

where \( \alpha, \beta_1, \ldots, \beta_m, \gamma_1, \ldots, \gamma_n \) are quantifier-free first-order formulas and respectively called the prerequisite, the justifications and the consequents. The formula \( \gamma_1 \lor \gamma_2 \) represents a disjunction meaning \( \gamma_1 \text{ is true or } \gamma_2 \text{ is true} \), rather than \( \gamma_1 \lor \gamma_2 \text{ is true} \).

An extension \( E \) of a disjunctive default theory \( \Delta \) is defined in the same manner as that of a default theory except that it is a minimal deductively closed set of disjunctive default theories and associated default theories presented in the previous sections.

Given an extended disjunctive program \( P \), each clause

\[
L_1 \lor \cdots \lor L_l \leftarrow L_{l+1} \land \cdots \land L_m \land \text{not } L_{m+1} \land \cdots \land \text{not } L_n
\]

in \( P \) is translated into the following disjunctive default in its associated disjunctive default theory \( \Delta_P \):

\[
\Delta_P = \{ L_{l+1} \land \cdots \land L_m \mid \neg L_{m+1} \land \cdots \land \neg L_n \}
\]

(4.5)

Note here that any CWA-default is not included in \( \Delta_P \). The following proposition is due to [Gelfond et al., 1991] which presents the relation between an extended disjunctive program and its associated disjunctive default theory.

**Proposition 4.14** Let \( P \) be an extended disjunctive program and \( \Delta_P \) be its associated disjunctive default theory. Then a consistent set \( S \) is an answer set of \( P \) iff \( S \) is the set of all literals from an extension of \( \Delta_P \).

In the previous section, we have presented the relationship between extended disjunctive programs and default theories. Then the next results follow from Theorem 4.11, Theorem 4.13, and Proposition 4.14. Recall here that \( S^* \) is a positive form of an answer set \( S \).

---

**DEFAULT TRANSLATION OF EXTENDED DISJUNCTIVE PROGRAMS**

**Theorem 4.15** Let \( P \) be an extended disjunctive program.

(i) If \( E_\Delta \) is a consistent extension of \( \Delta_P \) and \( S = E_\Delta \cap \mathcal{L}_P \), then there is an extension \( E \) of \( D_P \) such that \( S^* = E \cap \mathcal{H}_P \).

(ii) If \( E \) is a positively consistent extension of \( D_P \) and \( S^* = E \cap \mathcal{H}_P \), then there is an extension \( E_\Delta \) of \( \Delta_P \) such that \( S = E_\Delta \cap \mathcal{L}_P \).

**Corollary 4.16** Let \( P \) be an extended disjunctive program. Then,

(i) \( \Delta_P \) has the contradictory extension iff either \( D_{P^+} \) has a consistent extension but no positively consistent extension, or \( D_{P^+} \) has the contradictory extension.

(ii) \( \Delta_P \) has no extension iff \( D_{P^+} \) has a positively consistent extension but \( D_P \) has no positively consistent extension.

The above results bridge the gap between disjunctive default theories and Reiter’s default theories in terms of extended disjunctive programs.

In [Gelfond et al., 1991], the difficulty of expressing disjunctive information in Reiter’s default theory is discussed using some examples. However, we have already seen that Poole’s paradox is eliminated by considering the CWA-defaults in its associated disjunctive theory (Example 4.4). The following examples, which are also given in [Gelfond et al., 1991] to differentiate each formalism, present that we do not lose any information under Reiter’s default theory in the presence of disjunctive information.

**Example 4.6** Let \( \Delta_P \) be the disjunctive default theory:

\[
\{ a \equiv b, \ a \mid b \}
\]

Then the corresponding default theory

\[
D_P = \{ a \equiv b, \ a \lor b, \ \neg a, \ \neg b \}
\]

has the unique extension \( Th(\{a, b\}) \) which is equivalent to the extension of \( \Delta_P \).

**Example 4.7** Let \( \Delta_P \) be the disjunctive default theory:

\[
\{ a \mid b, \ a \mid \neg a \}
\]

Then the corresponding default theory

\[
D_P = \{ a \lor b, \ a \lor \neg a, \ \neg a, \ \neg b, \ \neg c \}
\]

has the unique extension \( Th(\{a, b, c\}) \) where \( Th(\{a, b, c\} \cap \mathcal{H}_P) = Th(\{b, c\}) \) coincides with the unique extension of \( \Delta_P \).
It remains open whether there is a general correspondence between the disjunctive default theory and the default theory. However, the results presented in this section show that Reiter’s default theory has the same expressiveness as the disjunctive default theory to characterize the stable model semantics and the answer set semantics of normal and extended disjunctive programs.

4.5 Connections with Autoepistemic Logic and Circumscription

In this section, we consider connections between disjunctive programs and other two representative nonmonotonic formalisms. The one is Moore’s autoepistemic logic, and the other is McCarthy’s circumscription.

4.5.1 Autoepistemic Logic

Autoepistemic logic [Moore, 1985] is a modal non-monotonic logic which is developed as a reconstruction of nonmonotonic logic by [McDermott and Doyle, 1980].

In autoepistemic logic, nonmonotonic reasoning is achieved as reasoning with an agent’s own belief. For instance, the bird fly sentence is represented as

$$bird(x) \land \neg L - fly(x) \supset fly(x)$$

where L is a modal belief operator. The above formula is read as “if x is a bird and there is no evidence to believe that x does not fly, then x flies”. Thus, if we know the fact $bird(Tweety)$ and there is no reason to believe the opposite fact $\neg fly(Tweety)$, then $fly(Tweety)$ is concluded.

An autoepistemic theory is defined in a similar way to a first-order theory except that its language includes modal formulas built from modal operator L and usual first-order connectives, with the restriction that quantification into the scope of the modal operator is not allowed.

An expansion of an autoepistemic theory is defined as follows.

$^5$A recent study [Eiter and Gottlob, 1993b] indicates that both default theories and disjunctive default theories are at the same second level of the polynomial hierarchy. Therefore, “there must exist a polynomial transformation from reasoning tasks in disjunctive default theories to analogous reasoning tasks in default theories” (G. Gottlob, private communication).

$^6$Thus, free variables appearing in an autoepistemic formula are viewed as representing its ground instances.

A correspondence between autoepistemic logic and logic programming is firstly studied by Gelfond [1987] for the perfect model semantics of stratified logic programs. The result is extended to the stable model semantics of normal logic programs in [Gelfond and Lifschitz, 1988].

We present an autoepistemic translation of normal disjunctive programs using the relation between default theories and autoepistemic theories. It is known that the extensions of a default theory are related to the expansions of an autoepistemic theory [Konolige, 1988; Marek and Truszczynski, 1989b]. Marek and Truszczynski [1989b] have shown that there is a one-to-one correspondence between a weak expansion of a default theory and an expansion of its corresponding autoepistemic theory. Since weak extensions coincide with extensions in prerequisite-free default theories, the above result implies that the default translation of disjunctive programs presented in Section 4.3 is also rephrased in the context of autoepistemic logic. That is, in Definition 4.3 (i), instead of transforming each clause in a program into the corresponding default rule, we can transform each clause into the following autoepistemic formula:

$$B_1 \land \ldots \land B_m \land \neg L B_{m+1} \land \ldots \land \neg L B_n \supset A_1 \lor \ldots \lor A_t,$$

and instead of the CWA-defaults in (ii), we have the CWA-formula:

$$\neg L A \supset \neg A.$$

In this way, the autoepistemic theory $AE_P$ associated with a disjunctive program $P$ is defined. Then the following result holds.

Theorem 4.17 Let $P$ be a consistent normal disjunctive program and $AE_P$ be its associated autoepistemic theory. Then $M$ is a stable model of $P$ iff $E$ is an expansion of $AE_P$ and $M = E \cap HB_P$. □
The above theorem provides an autoepistemic characterization of disjunctive programs. Such an autoepistemic translation is also presented in [Przymusinski, 1990a] in the context of the 3-valued stable model semantics. Using the same techniques presented in the previous sections, we can also provide autoepistemic characterizations of extended disjunctive programs and associated disjunctive default theories. An alternative autoepistemic translation for extended disjunctive programs is presented in [Lifschitz and Schwarz, 1993; Chen, 1993].

4.5.2 Circumscription

Circumscription [McCarthy, 1980; 1986] is also known as one of the popular frameworks of nonmonotonic reasoning in AI. Different from default logic and autoepistemic logic, circumscription is still within classical logic but augmented with an ability of nonmonotonic reasoning.

Using the bird-fly example again, let us consider the first-order formula:

$$\forall x \, \text{bird}(x) \land \neg \text{abnormal}(x) \supset \text{fly}(x).$$

Then, if bird(Tweety) is the all known facts about Tweety, by minimizing the extension of the predicate abnormal (in this case the empty set), circumscription concludes fly(Tweety). Else if there is an evidence that Tweety is an abnormal bird abnormal(Tweety), fly(Tweety) is not derived any more.

Several variations of circumscription have been proposed so far. In what follows, we use predicate circumscription originally proposed by McCarthy [1980].

Given a first-order theory $T$ and a set of predicates $\Pi = \{p_1, \ldots, p_n\}$ from $T$, the circumscription of $T$ with respect to $\Pi$ is defined as the second-order axiom that

$$\text{Circ}(T; \Pi) \equiv T(p_1', \ldots, p_n') \land \bigwedge_{i=1}^{n} (\forall x \, p'_i(x) \supset p_i(x)) \lor \bigwedge_{s \in \Pi} (\forall x \, p_s(x) \supset p'_s(x)) \quad (4.9)$$

where each $p'_i$ is a predicate variable with the same arities as $p_i$.

The above schema means that for any theory $T(p_1', \ldots, p_n')$ obtained from $T$ by replacing each $p_i$ with $p'_i$, there is no theory which has smaller extensions of $p_1, \ldots, p_n$ with respect to $T$.

Predicates in $\Pi$ are said to be minimized, while the rest of predicates in the language of $T$ are said to be fixed. Any model of Circ$(T; \Pi)$ is called a $\Pi$-minimal model of $T$, in which extensions of each predicate from $\Pi$ are minimized with fixed interpretations for the rest of predicates.

Proof: Suppose that $M \cup LM$ is an Herbrand model of Circ$(P_L^{LM}; \Pi) \land \Pi \equiv \Pi$. Then, for each ground formula $B_1 \land \ldots \land B_m \supset A_1 \lor \ldots \lor A_k$

The meaning of $LA$ is the same as $KA$ in Chapter 3, but here we use the notation $LA$ to compare it with an autoepistemic formula $LA$. 
in $P^{LM}, \{B_1, \ldots, B_m\} \subseteq M$ implies $A_i \in M$ for some $i (1 \leq i \leq l)$. In this case, there is a corresponding ground formula $B_i \land \ldots \land B_m \supset A_1 \lor \ldots \lor A_l \lor L B_{m+1} \lor \ldots \lor L B_n$ from $P_L$, such that $\{B_1, \ldots, B_m\} \subseteq M$ and $A_i \in M$ hold. Since $M$ is $\Pi$-minimal, $M \cup LM$ is also an Herbrand model of $Circ(P_L; \Pi) \land \Pi \equiv \Pi$.

Conversely, suppose that $M \cup LM$ is an Herbrand model of $Circ(P_L; \Pi) \land \Pi \equiv \Pi$. Then, for each ground formula $B_i \land \ldots \land B_m \supset A_1 \lor \ldots \lor A_l \lor L B_{m+1} \lor \ldots \lor L B_n$, $M$ implies either (i) $A_i \in M$ for some $i (1 \leq i \leq l)$ or (ii) $LB_j \in LM$ for some $j (m + 1 \leq j \leq n)$.

In case of (i), when $LB_j \notin LM$, there is a corresponding ground formula $B_i \land \ldots \land B_m \supset A_1 \lor \ldots \lor A_l \lor L B_{m+1} \lor \ldots \lor L B_n$ from $P_L$. Then, for each ground formula $B_i \land \ldots \land B_m \supset A_1 \lor \ldots \lor A_l$ in $P^{LM}$ such that $\{B_1, \ldots, B_m\} \subseteq M$ implies $A_i \in M$. In case of (ii), there is no corresponding ground formula in $P_L^{LM}$. In each case, $M \cup LM$ is also a model of $P_L^{LM}$. Since $M$ is $\Pi$-minimal, $M \cup LM$ is an Herbrand model of $Circ(P_L^{LM}; \Pi) \land \Pi \equiv \Pi$. □

Now we present the theorem which provides a connection between disjunctive programs and circumscription.

**Theorem 4.19** Let $P$ be a normal disjunctive program. Then $M$ is a stable model of $P$ if and only if $M \cup LM$ is an Herbrand model of $Circ(P_L; \Pi) \land \Pi \equiv \Pi$.

**Proof:** $M$ is a stable model of $P$ if

- $M$ is a minimal model of $P^M$,
- $M$ is an Herbrand model of $Circ(P_L^{LM}; \Pi)$,
- $M \cup LM$ is an Herbrand model of $Circ(P_L^{LM}; \Pi) \land \Pi \equiv \Pi$,
- $M \cup LM$ is an Herbrand model of $Circ(P_L^{LM}; \Pi) \land \Pi \equiv \Pi$ (by Lemma 4.18).

The above theorem generalizes the corresponding results for normal logic programs [Lifschitz, 1988; Lin and Shoham, 1992; Youn and You, 1993], and also provides a method of characterizing extended disjunctive programs and associated disjunctive default theories in terms of circumscription.8

---

8For normal disjunctive programs, a similar result is also reported in [Lin and Shoham, 1992] without proof.

### 4.5.3 Characterizing Possible Model Semantics

We now discuss the issue of characterizing the possible model semantics by nonmonotonic formalisms.

We have already seen that possible models of normal disjunctive programs are translated into stable models of normal logic programs by the pm-transformation in Section 3.6. Since stable models of normal logic programs can be expressed by the nonmonotonic frameworks as presented in the previous sections, possible models are also expressed by each nonmonotonic framework via stable models.

However, the pm-transformation introduces extra atoms like $A_i'$'s, which are not included in the language of the original program. Hence nonmonotonic translations of possible models through the pm-transformation also include those extra-language formulas, which does not necessarily coincide with the original meaning of the program.

Then, in this section we consider a possibility to express possible models directly in terms of each nonmonotonic formalism. For this purpose, we can first exclude (disjunctive) default logic and circumscription. This is because those formalisms are based on the principle of minimality, that is, default extensions and circumscribed models are always minimal. Thereby the remaining candidate is autoepistemic logic.

Fortunately, autoepistemic expansions are not necessarily minimal with respect to first-order formulas. For instance, the autoepistemic theory \{La $\supset$ a\} presents that an an anomalous expansion, and efforts have been done to eliminate such expansions [Konolige, 1988]. However, as shown below, non-minimal expansions are useful to express possible models in an autoepistemic theory.

Let us consider the autoepistemic theory $AE = \{La \supset a, Lb \supset b\}$, which has four expansions containing the sets, $E_1 = \{a, La, \neg Lb\}$, $E_2 = \{b, Lb, \neg La\}$, $E_3 = \{a, b, La, Lb\}$, and $E_4 = \{\neg La, \neg Lb\}$, respectively. Then we can observe that the first three expansions correspond to the possible models of the program $P = \{a \lor b \lor \}$. Rewriting $AE$ by $AE' = \{a \lor \neg La, b \lor \neg Lb\}$ will help to understand the correspondence. Each formula in $AE'$ presents that an atom is true or not believed, which exactly characterizes interpretations of split programs of $P$. That is, $E_1, E_2, E_3$ express epistemic interpretations of split programs $P_1 = \{a \},$ $P_2 = \{b \},$ and $P_3 = \{a \lor b \},$ respectively. On the other hand, since there is no split program corresponding to $E_4$, we add the formula $La \lor Lb$ to $AE'$ to remove this expansion from $AE$. This formula presents that when $a \lor b$ is true in $P$, one of the disjuncts is believed.

Formally, expressing possible models in autoepistemic logic is achieved as follows.
Definition 4.5 Let \( P \) be a normal disjunctive program. Then its associated pm-autoepistemic theory \( AE_P^m \) is constructed as follows:

1. Each disjunctive clause \( A_1 \land \ldots \land A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n \) from \( P \) is transformed into the following autoepistemic formulas in \( AE_P^m \):
   \[
   B_1 \land \ldots \land B_m \land \neg L_{B_{m+1}} \land \ldots \land \neg L_{B_n} \supset B_i \lor \neg L_A \quad \text{for } i = 1, \ldots, l
   \]
   \[
   B_1 \land \ldots \land B_m \land \neg L_{B_{m+1}} \land \ldots \land \neg L_{B_n} \supset L_A \lor \ldots \lor L_A
   \]
   (4.10)
   (4.11)
   In particular, each normal clause \( A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n \) from \( P \) is transformed into the following autoepistemic formula in \( AE_P^m \):
   \[
   B_1 \land \ldots \land B_m \land \neg L_{B_{m+1}} \land \ldots \land \neg L_{B_n} \supset A_i
   \]

2. Nothing else is in \( AE_P^m \).

Note here that \( AE_P^m \) contains no CWA formula (4.8).

The first formula (4.10) represents that if the antecedent of the formula is true, \( A_i \) is true or not believed. The second formula (4.11) represents that if the antecedent is true, at least one of the disjuncts is believed. Recall that an autoepistemic expansion \( E \) satisfies the conditions that \( \neg L_A \in E \) implies \( A \not\in E \) and \( LA \in E \) implies \( A \in E \).

Now we have the following result.

Theorem 4.20 Let \( P \) be a consistent normal disjunctive program and \( AE_P^m \) be its associated pm-autoepistemic theory. Then \( M \) is a possible model of \( P \) iff \( E \) is an expansion of \( AE_P^m \) and \( M = E \cap \mathcal{H}_P \).

Proof: Let \( M \) be a possible model of \( P \). Then there is a split program \( P' \) of \( P \) such that \( M \) is a stable model of \( P' \). Suppose that each ground disjunctive clause \( A_1 \lor \ldots \lor A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n \) in \( P \) is replaced with the split clauses: \( A_i \leftarrow B_1 \land \ldots \land B_m \land \neg B_{m+1} \land \ldots \land \neg B_n \) in \( P' \), where \( A_i \in S \) for some non-empty subset \( S \) of \( \{A_1, \ldots, A_l\} \).

In this case, it is easy to see that the corresponding autoepistemic theory \( AE_P^m \) has an expansion \( E \) such that \( A_i \in M \) iff \( A_i \in E \cap \mathcal{H}_P \), and \( A_i \not\in M \) iff \( \neg LA \in E \). Hence, the result follows. The converse is also shown in the same manner. □

SUMMARY 4.6 Summary

This chapter has presented the relationship between disjunctive programs and default theories.

We first pointed out the problem of Bidoit and Froidevaux's positivist default theories, and developed an alternative correct default translation of normal disjunctive programs. It was shown a one-to-one correspondence between the stable models of a disjunctive program and the default extensions of its associated default theory.

We also extended the results to default translation of extended disjunctive programs and the answer set semantics. The results indicate that Reiter's default theory is as expressive as Gelfond et al.'s disjunctive default theory to characterize the semantics of disjunctive programs.

We finally presented the connections between disjunctive programs and autoepistemic logic, and circumscription. The possible model semantics of disjunctive programs was also characterized using non-minimal feature of autoepistemic expansions.
Chapter 5

Equivalence between Disjunctive and Abductive Logic Programs

Abductive logic programming is a recently proposed framework which enhances logic programming by supplying the ability of abductive reasoning in AI. In this chapter, we consider such an abductive logic programming framework and reveal its close relationship to disjunctive programs. We show that the generalized stable model semantics of abductive logic programs are viewed as the possible model semantics of disjunctive programs, and vice versa. We also demonstrate that abductive disjunctive programs do not increase expressiveness of disjunctive programs. Interrelations between various semantics of disjunctive and abductive logic programs are discussed in terms of computational complexity.

5.1 Introduction

Abduction is a form of hypothetical reasoning and is widely used today in various AI problems such as diagnosis and planning. Abduction is a reasoning for hypothetical generation from a given observation, and provides a weak kind of non-deductive inference as a tool for commonsense reasoning.

In logic programming, abduction is realized in the framework of abductive logic programming. The framework was firstly proposed by Eshghi and Kowalski [1989], in which they gave an abductive interpretation of negation as failure in normal logic programs. They showed a one-to-one correspondence between the stable models of a normal logic program and the extensions of its associated abductive framework. Kakas and Mancarella [1990] extended their framework to abductive logic programs containing abducibles which represent not necessarily default negation but positive
hypothesis facts. They also introduced the generalized stable model semantics as a theoretical framework of such programs. Further extensions of abductive logic programming have been studied by several researchers in the last few years [Kakas et al., 1992].

Comparing between disjunctive programs and abductive logic programs, both frameworks enhance logic programming by supplying the ability of reasoning with incomplete information. It is achieved in disjunctive programs by reasoning with disjunctive information, while in abductive logic programs by reasoning with hypotheses. Disjunctive programs and abductive logic programs have been independently developed so far and have different syntax and semantics from each other. However, in disjunctive programs, each disjunction is considered to represent knowledge about possible alternative beliefs, and such beliefs can also be regarded as a kind of hypotheses. In abductive logic programs, on the other hand, each candidate hypothesis is examined whether it is adopted or not, and this situation can be considered as meta-level disjunctive knowledge that either a hypothesis is true or not. Thus, each formalism appears to deal with very similar problems from different viewpoints. Then the question naturally arises whether there is any formal correspondence between these two frameworks.

There are some studies which can be related to the above question. Dung [1992a] presents a program transformation from acyclic disjunctive programs to normal logic programs under the stable model semantics and uses Eshghi and Kowalski's abductive proof procedure for such programs. However, Dung's transformation is restricted to acyclic disjunctive programs and not applicable in general. Console et al. [1991] characterize abduction using Clark's completion technique in abductive logic programs. They show that abductive solutions are obtained by deduction in the only-if part of the completed abductive program which is viewed as a kind of disjunctive program, but the technique is applicable only to acyclic abductive programs. Inoue and Sakama [1992] present a program transformation from abductive logic programs to disjunctive programs under the stable model semantics and use a bottom-up model generation proof procedure for computing abduction. While their transformation is fairly general, it is a one-way transformation from abductive logic programs to disjunctive programs.

In this chapter, we investigate a general correspondence between disjunctive programs and abductive logic programs. For the part from abductive logic programs to disjunctive programs, we show that the generalized stable models of an abductive logic program are characterized by the possible models of the transformed disjunctive program. Conversely, from disjunctive programs to abductive logic programs, we show that the possible models of a disjunctive program are exactly the generalized stable models of the transformed abductive logic program. Moreover, if the disjunctive stable model semantics is taken as the underlying semantics instead of the possible model semantics, it is unlikely that disjunctive programs can be efficiently expressed in terms of the generalized stable model semantics. It is also shown that abductive disjunctive programs can be expressed by abductive logic programs under the possible model semantics.

The rest of this chapter is organized as follows. In Section 5.2, we introduce the notion of abductive logic programming. In Section 5.3, we present program transformations between abductive logic programs and disjunctive programs. It is shown that the generalized stable models of an abductive logic program are characterized by the possible models of the transformed disjunctive program, and vice versa. In Section 5.4, we introduce abductive disjunctive programs and present their translation into disjunctive programs. It is shown that abductive logic programs are as expressive as abductive disjunctive programs under the possible model semantics. Section 5.5 discusses the relation between disjunctive programs and abductive logic programs from the computational complexity viewpoint. Section 5.6 summarizes this chapter.

5.2 Abductive Logic Programming

Abduction is firstly introduced by Peirce [1932] who characterized three distinguished forms of reasoning, abduction, induction, and deduction. Abduction is a non-deductive inference and is formally presented as follows.

Given a theory $T$ and an observation $O$, abduction is defined as an inference of an explanation $E$ such that

$$T \cup E \models O \text{ where } T \cup E \text{ is consistent.} \quad (5.1)$$

Thus, abduction can be thought of as a form of hypothetical reasoning which produces, with background knowledge $T$, hypothetical sentences $E$ that are sufficient to account for $O$.

Abductive logic programming is a form of such abductive frameworks in which $T$ is given as a logic program.

An abductive logic program is a pair $(P, A)$ where $P$ is a normal logic program and $A$ is a finite set of atoms called the abducibles.\(^1\)

\(^1\)We slightly modified the original definition of [Kakas and Mancarella, 1990] by including integrity constraints in a program and considering abducible atoms instead of abducible predicates. Here, an abducible containing variables is identified with its ground instances.
Abducibles are pre-specified sentences, which represent candidate hypotheses used for explanations. Such a specification is needed to get “best” explanations since there might be many candidate explanations which can deduce a given observation from the condition (5.1).

A declarative semantics of abductive logic programs is given by Kakas and Man­carella [1990] who extended the stable model semantics of normal logic programs to the generalized stable model semantics of abductive logic programs. Similar extensions are done in [Gelfond, 1990; Inoue, 1991] in the context of extended logic programs.

Let \((P,A)\) be an abductive logic program and \(E\) be a subset of \(A\). An interpretation \(I\) is a generalized stable model of \((P,A)\) if \(I\) is a stable model of the normal logic program \(P \cup E\). A generalized stable model \(I\) is called \(A\)-minimal if there is no generalized stable model \(J\) such that \(J \land A \subset I \land A\). Clearly, \((A\)-minimal\) generalized stable models coincide with stable models if \(A = \emptyset\).

Let \((P,A)\) be an abductive logic program and \(O\) be an atom which represents observation. Then a set \(E \subseteq A\) is an explanation of \(O\) if there is a generalized stable model \(I\) of \((P,A)\) such that \(I\) satisfies \(O\) and \(E = I \land A\). An explanation \(E\) of \(O\) is minimal if no \(E' \subset E\) is an explanation of \(O\).

Note that the problem of finding explanations is essentially equivalent to the problem of finding generalized stable models since \(E\) is a (minimal) explanation of \(O\) with respect to \((P,A)\) iff \(I\) is a \((A\)-minimal\) generalized stable model of \((P \cup \{\neg O\}, A)\) such that \(I \land A = E\).

Also note that without loss of generality an observation is assumed to be a non-abducible ground atom. This is because if \(O\) is an abducible, its explanations trivially contain \(O\). Else if \(O(x)\) contains a tuple of free variables \(x\), we can introduce a new proposition \(O\) which is considered as an observation in the program \(P \cup \{O \leftarrow O(x)\}\). Else if multiple observations are given like that \(O_1, \ldots, O_m\) are observed and \(O_{m+1}, \ldots, O_n\) are not observed, they are also realized by introducing a clause \(O_1 \land \cdots \land O_m \land \neg O_{m+1} \land \cdots \land \neg O_n\) into \(P\) and computing explanations of \(O\) [Inoue and Sakama, 1993].

**Example 5.1** Let \((P,A)\) be an abductive logic program such that

\[
P = \{\ p(x) \leftarrow q(x) \land \neg r(x), \ q(x) \leftarrow s(x), \ q(x) \leftarrow t(x) \}
\]

and \(A = \{s(x), t(x)\}\). Then, for a given observation \(O = p(a)\), the \((A\)-minimal\) generalized stable model \(I = \{p(a), q(a), s(a)\}\) of \((P,A)\) satisfies \(O\) and its (minimal) explanation is \(E = I \land A = \{s(a)\}\). Here, \(I\) is also the unique generalized stable model of \((P \cup \{\neg \neg p(a)\}, A)\). ☐

5.3 Connections between Disjunctive and Abductive Logic Programs

In this section, we characterize disjunctive programs in terms of abductive logic programs, and vice versa. Then we reveal close relationships between the generalized stable models of abductive logic programs and the possible models of disjunctive programs.

5.3.1 Generalized Stable Models are Possible Models

We first present a program transformation from abductive logic programs to disjunctive programs, then show that the generalized stable models of an abductive logic program can be expressed by the possible models of the transformed disjunctive program.

In an abductive logic program, each candidate hypothesis is either assumed or not. Such a situation is naturally expressed by disjunctions in a program.

**Definition 5.1** Let \((P,A)\) be an abductive logic program. Then its **dlp-transformation** is defined by a normal disjunctive program \(dlp((P,A))\) which is obtained from \(P\) by adding the following disjunctive clauses for each abducible \(A \in A\):

\[
A \lor \varepsilon \qquad (5.2)
\]

where \(\varepsilon\) is an atom not appearing elsewhere in \(P\). ☐

The intuitive meaning of the dlp-transformation is that when an abducible \(A\) is assumed in an abductive logic program \((P,A)\), the corresponding disjunct \(A\) is chosen from (5.2) in the transformed disjunctive program \(dlp((P,A))\). Else when \(A\) is not assumed, the newly introduced atom \(\varepsilon\) is chosen from (5.2). Thus the dlp-transformation specifies meta-level knowledge representing whether each abducible is assumed or not.

Now we express the generalized stable model semantics in terms of \(dlp((P,A))\). Let \(I\) be a possible model of a normal disjunctive program \(P\). We say that \(I\) is \((A\)-minimal\) if there is no possible model \(J\) of \(P\) such that \(J \land A \subset I \land A\). In the following, an atom \(A\) is identified with the unit clause \(\neg \neg A \leftarrow \neg \neg \neg A\) in \(E\).

**Theorem 5.1** Let \((P,A)\) be an abductive logic program. Then,

(i) \(I \cup \{\varepsilon\}\) is a generalized stable model of \((P,A)\) iff \(I\) is a possible model of \(dlp((P,A))\).
(ii) $I \setminus \{e\}$ is an $A$-minimal generalized stable model of $(P,A)$ iff $I$ is an $A$-minimal possible model of $dip((P,A))$.

**Proof.** (i) Let $I'$ be a generalized stable model of $(P,A)$. Then $I'$ is a stable model of $P \cup E$ for some $E$ from $A$. Now let us consider the transformed disjunctive program $dip((P,A))$. Then there is a split program $P'$ of $dip((P,A))$ such that for each disjunctive clause (5.2), $A \rightarrow$ is in $P'$ if $A \in E$; $e \rightarrow$ is in $P'$, otherwise. When $e \leftarrow$ is in $P'$, $P' \cup \{e\}$ is a stable model of $P'$ and also a possible model of $dip((P,A))$. Else when $e \leftarrow$ is not in $P'$, $I'$ is a stable model of $P'$ and also a possible model of $dip((P,A))$. Hence the result of only-if part follows.

Conversely, when $I$ is a possible model of $dip((P,A))$, it is a stable model of some split program $P'$ of $dip((P,A))$. Let $E$ be the set of all split clauses included in $P'$. Then $I$ is a stable model of $P' \cup E$. Since $E \setminus \{e\}$ consists of instances from $A$, $I \setminus \{e\}$ is a generalized stable model of $(P,A)$. (ii) The result directly follows from (i) and the definitions of $A$-minimal generalized models/ $A$-minimal possible models.

**Corollary 5.2** Let $(P,A)$ be an abductive logic program. Then, for a given observation $O$, there is a (minimal) explanation $E$ of $O$ if there is an $(A$-minimal) possible model $I$ of $dip((P,A))$ satisfying $O$ and $I \cap A = E$.

**Example 5.2** Let $(P,A)$ be an abductive logic program such that

$$P = \{ \text{wet-shoes} \leftarrow \text{wet-grass} \land \neg \text{driving-car}, \text{wet-grass} \leftarrow \text{rained}, \text{wet-grass} \leftarrow \text{sprinkler-on} \},$$

and $A = \{ \text{rained, sprinkler-on} \}$. Then,

$$dip((P,A)) = P \cup \{ \text{rained} \lor e \leftarrow, \text{sprinkler-on} \lor e \leftarrow \}$$

which has the five possible models:

$$\{\text{rained, sprinkler-on, wet-grass, wet-shoes}\},$$

$$\{e\},$$

$$\{\text{e, rained, wet-grass, wet-shoes}\}.$$

Thus, the generalized stable models of $(P,A)$ coincide with the sets which are obtained by removing $e$ from each possible model. In particular, $\{e\}$ is the $A$-minimal possible model and it corresponds to the $A$-minimal generalized stable model $\emptyset$ of $(P,A).$

The result of this section indicates that abductive logic programs are also considered as disjunctive programs. In the next section, we present that the converse is also the case.

### 5.3.2 Possible Models are Generalized Stable Models

As presented in the introduction, indefinite information in disjunctive programs is viewed as possible hypotheses in a program. Then it is natural to represent disjuncts in terms of abducibles in an abductive logic program. However, the problem is that disjunctive clauses possibly have conditions in their bodies, while abductive logic programs introduced in Section 5.2 lack the ability of expressing assumptions with preconditions. Then our first task is to extend the framework of abductive logic programs to possibly include such hypothetical rules.

An abductive logic program considering in this section is a pair $(P,C)$ where $P$ is a normal logic program and $C$ is a finite set of normal clauses called the **abducible rules**. The abducible rule intuitively means that if the rule is abduced then it is used for inference together with the background knowledge from $P$. In this sense, abductive logic programs presented in the previous sections are considered as a special case where each abducible rule has the empty precondition. The generalized stable model semantics of such an extended framework is defined as follows.

**Definition 5.2** Let $(P,C)$ be an abductive logic program and $F$ be a subset of $C$. An interpretation $I$ is a **generalized stable model of $(P,C)$** if it is a stable model of the normal logic program $P \cup F$.

The generalized stable model introduced above is a direct extension of the one presented in the previous sections, and it reduces to the usual notion when $C = A$.

Next we provide a program transformation which translates normal disjunctive programs into abductive logic programs. For a normal disjunctive program $P$, we define $P = \text{disj}(P) \cup \text{idisj}(P)$ where $\text{disj}(P)$ is the set of all disjunctive clauses from $P$ and $\text{idisj}(P)$ is the set of all normal clauses and integrity constraints from $P$. In the following, $\Gamma$ denotes the conjunction in the body of a clause.
Definition 5.3 Given a normal disjunctive program $P$, let us consider the set of normal clauses

$$C = \{ A_i \leftarrow \Gamma \mid A_1 \lor \ldots \lor A_l \leftarrow \Gamma \in \text{disj}(P) \text{ and } 1 \leq i \leq l \}$$  \hfill (5.3)

and the integrity constraints

$$IC = \{ \leftarrow \Gamma \land \neg A_1 \land \ldots \land \neg A_l \mid A_1 \lor \ldots \lor A_l \leftarrow \Gamma \in \text{disj}(P) \}.$$  \hfill (5.4)

Then we define the $\text{alp}$-transformation of $P$ by $\text{alp}(P) = (\overline{\text{disj}}(P) \cup IC, C)$. \hfill \Box

The intuitive meaning of the $\text{alp}$-transformation is that each disjunctive clause in a program is replaced with a set of abducible rules (5.3) in $C$. The integrity constraints (5.4) in $IC$ impose the condition that at least one of disjuncts is chosen as an abducible whenever the body of a disjunctive clause is true. In this way, by the $\text{alp}$-transformation each disjunctive clause is rewritten by a set of abducible rules.

Now we present the relationship between the possible models of a normal disjunctive program $P$ and the generalized stable models of the transformed abductive logic program $\text{alp}(P)$.

Theorem 5.3 Let $P$ be a normal disjunctive program. Then $I$ is a possible model of $P$ iff $I$ is a generalized stable model of $\text{alp}(P)$.

Proof: Let $I$ be a possible model of $P$. Then there is a split program $P'$ of $P$ such that $I$ is a stable model of $P'$. Suppose that each ground disjunctive clause $C^\varepsilon : A_1 \lor \ldots \lor A_k \leftarrow \Gamma$ from $P$ is replaced with the split clauses in $C^\varepsilon_S = \{ A_i \leftarrow \Gamma \mid A_i \in S \}$ in $P'$ where $S$ is a non-empty subset of $\{A_1, \ldots, A_k\}$. Then $I$ is a stable model of $\overline{\text{disj}}(P) \cup \bigcup_\varepsilon C^\varepsilon_S$. Since $\bigcup_\varepsilon C^\varepsilon_S$ consists of instances from $C$ and $I$ satisfies integrity constraints $IC$, $I$ is also a generalized stable model of $\text{alp}(P)$.

Conversely, let $I$ be a generalized stable model of $\text{alp}(P)$. Then $I$ is a stable model of $F \cup C$ where $F$ is a subset of $C$. For each normal clause $A_i \leftarrow \Gamma$ in $F$, there is a corresponding disjunctive clause $C : A_1 \lor \ldots \lor A_l \leftarrow \Gamma$ in $\text{disj}(P)$ such that $1 \leq i \leq l$. Also, since $I$ satisfies integrity constraints $IC$, when $I$ satisfies $\Gamma$, at least one normal clause $A_i \leftarrow \Gamma$ is included in $F$. In this case, there is a split program $P'$ of $P$ in which each ground instance of a disjunctive clause $C$ is split into a corresponding ground instance of a normal clause $A_i \leftarrow \Gamma$. Thus $I$ is also a stable model of $P'$, hence a possible model of $P$. \hfill \Box

Example 5.3 ([Chan, 1993]) Let

$$P = \{ \text{violent} \lor \text{psychopath} \leftarrow \text{suspect}, \text{dangerous} \leftarrow \text{violent} \land \text{psychopath}, \text{suspect} \leftarrow \}.$$  

Then, $\text{alp}(P) = (\overline{\text{disj}}(P) \cup IC, C)$ where

$$\overline{\text{disj}}(P) \cup IC = \{ \text{dangerous} \leftarrow \text{violent} \land \text{psychopath}, \text{suspect} \leftarrow, \leftarrow \text{not violent} \land \neg \text{psychopath} \},$$

$$C = \{ \text{violent} \leftarrow \text{suspect}, \text{psychopath} \leftarrow \text{suspect} \}.$$  

Thus, $\text{alp}(P)$ has three generalized stable models:

$$\{ \text{suspect, violent} \},$$

$$\{ \text{suspect, psychopath} \},$$

$$\{ \text{suspect, violent, psychopath, dangerous} \},$$

which coincide with the possible models of $P$. \hfill \Box

Note that in the above example there is no minimal model of $P$ containing $\text{dangerous}$. By contrast, $\text{alp}(P)$ has a generalized stable model in which $\text{dangerous}$ is true, which corresponds to a possible model in which the disjunction is inclusively true.

The abductive logic programming framework presented in this section is also introduced by Inoue [1991] in the context of the knowledge system for extended logic programs. He also shows that an abductive logic program $(P, C)$ can be translated into a semantically equivalent usual abductive logic program $(P, A)$. Given an abductive logic program $(P, C)$, let us consider a program $P'$ which is obtained from $P$ by including the clause:

$$A \leftarrow A' \land \Gamma$$

for each abducible rule $A \leftarrow \Gamma$ in $C$. Here $A'$ is a newly introduced atom not appearing elsewhere in $P$ and is uniquely associated with each $A$. Also let $A'$ be a set of abducibles which consists of every newly introduced atom $A'$. Then he proves that there is a one-to-one correspondence between the generalized stable models of $(P, C)$ and the generalized stable models of $(P', A')$. This fact implies that the possible models of a normal disjunctive program are also expressed by the generalized stable models of a usual abductive logic program.
5.4 Abductive Disjunctive Programs

This section extends a framework of abductive logic programs to abductive disjunctive programs, and discusses their correspondence to disjunctive programs and abductive logic programs.

5.4.1 Generalized Disjunctive Stable Models and Possible Models

Abductive disjunctive programs are disjunctive programs with abducibles. The definition of an abductive disjunctive program \((P, A)\) is the same as an abductive logic program except that \(P\) is a normal disjunctive program. For a given set \(E \subseteq A\), an interpretation \(I\) is a generalized disjunctive stable model of \((P, A)\) if \(I\) is a disjunctive stable model of the normal disjunctive program \(P \cup E\). On the other hand, \(I\) is a generalized possible model of \((P, A)\) if \(I\) is a possible model of the normal disjunctive program \(P \cup E\). A generalized disjunctive stable model (resp. generalized possible model) \(I\) is \(A\)-minimal if there is no generalized disjunctive stable model (resp. generalized possible model) \(J\) such that \(J \cap A \subseteq I \cap A\).

The above definitions are direct extensions of the previously proposed notions. In fact, generalized disjunctive stable models (resp. generalized possible models) reduce to disjunctive stable models (resp. possible models) in normal disjunctive programs with \(A = \emptyset\), and both generalized disjunctive stable models and generalized possible models reduce to generalized stable models in abductive logic programs.

A difference between generalized disjunctive stable models and generalized possible models is illustrated in the following example.

Example 5.4 Let \((P, A)\) be an abductive disjunctive program such that

\[
P = \{ a \lor b \leftarrow c, \quad d \leftarrow a \land b \}
\]

and \(A = \{ c \}\). Then, \(\emptyset, \{c, a\}, \{c, b\}, \{c, a, b, d\}\) are all generalized possible models, while \(\{c, a, b, d\}\) is not a generalized disjunctive stable model. Thus, for a given observation \(O = d\), it has an explanation \(c\) under the generalized possible models, while no explanation is available under the generalized disjunctive stable models.

In this way, the generalized possible model semantics can provide explanations which come from inclusive disjunctions, while the generalized disjunctive stable model semantics cannot in general.

5.4.2 Generalized Possible Models are Generalized Stable Models

Abductive disjunctive programs are generalization of disjunctive programs. However, in this section we present a somewhat unexpected result that abductive disjunctive programs do not increase expressive power of disjunctive programs. We show this fact by introducing a translation from abductive disjunctive programs into disjunctive programs.

For an abductive disjunctive program \((P, A)\), we define its \(dlp\)-transformation \(dlp((P, A))\) in the same manner as presented in Definition 5.1. Then the following results hold.

Theorem 5.4 Let \((P, A)\) be an abductive disjunctive program. Then,

(i) \(I \setminus \{c\}\) is a generalized possible model of \((P, A)\) iff \(I\) is a possible model of \(dlp((P, A))\).

(ii) \(I \setminus \{c\}\) is an \(A\)-minimal generalized possible model of \((P, A)\) iff \(I\) is an \(A\)-minimal possible model of \(dlp((P, A))\).

Proof: Similar to the proof of Theorem 5.1.

The above theorem, together with Theorem 5.3, implies the following result.

Corollary 5.5 Let \((P, A)\) be an abductive disjunctive program. Then \(I \setminus \{c\}\) is a generalized possible model of \((P, A)\) iff \(I\) is a generalized stable model of \(dlp(dlp((P, A))))\).

By Theorem 5.4, normal disjunctive programs are as expressive as abductive disjunctive programs under the possible model semantics. Moreover, Corollary 5.5 presents that abductive disjunctive programs can be expressed even by abductive logic programs under the generalized possible model semantics.

Next we consider the corresponding relations under the disjunctive stable model semantics.

Given an abductive disjunctive program \((P, A)\), its \(dlp\)-transformation is defined as a normal disjunctive program \(dlp((P, A))\) which is obtained from \(P\) by adding the following disjunctive clauses for each abducible \(A \in A\):

\[
A \lor \varepsilon_A \leftarrow \]

(5.5)
where \( e_A \) is an atom not appearing elsewhere in \( P \), and is uniquely associated with each \( A \).

Note here that different from the \( dlp \)-transformation presented in Definition 5.1, the unique \( e_A \) is associated with each \( A \) in (5.5). This modification is needed to assure the existence of a disjunctive stable model corresponding to any generalized stable model of \((P, A)\) (see Example 5.5 below).

Let us call \( I \) an \( A \)-minimal disjunctive stable model if it is a disjunctive stable model such that \( I \cap A \) is minimal. Then the following results hold.

**Theorem 5.6** Let \((P, A)\) be an abductive disjunctive program. Then,

(i) \( I \setminus \{ e_A \} \) is a generalized disjunctive stable model of \((P, A)\) iff \( I \) is a disjunctive stable model of \( dlp_{\text{A}}((P, A)) \).

(ii) \( I \setminus \{ e_A \} \) is an \( A \)-minimal generalized disjunctive stable model of \((P, A)\) iff \( I \) is an \( A \)-minimal disjunctive stable model of \( dlp_{\text{A}}((P, A)) \).

**Proof:** (i) Let \( I' \) be a generalized disjunctive stable model of \((P, A)\). Then \( I' \) is a disjunctive stable model of \( P \cup \{ e_A \} \) for some \( e \) from \( A \). Now let us consider the transformed disjunctive program \( dlp_{\text{A}}((P, A)) \). Then there is a disjunctive stable model \( I \) of \( dlp_{\text{A}}((P, A)) \) such that for each disjunctive clause \( A \vee e_A \leftarrow \), \( A' \in I \) iff \( A' \in I' \), and \( e_A \in I \) iff \( A \notin I' \). Hence the result follows. The converse is also shown in the same manner.

(ii) The result holds from (i) and the definition of \( A \)-minimal (generalized) disjunctive stable models.

**Corollary 5.7** Let \((P, A)\) be an abductive logic program. Then \( I \setminus \{ e_A \} \) is a generalized stable model of \((P, A)\) iff \( I \) is a disjunctive stable model of \( dlp_{\text{A}}((P, A)) \).

**Example 5.5** Let \((P, A)\) be an abductive disjunctive program such that \( P = \emptyset \) and \( A = \{ a, b \} \). Then, \( \emptyset, \{ a \}, \{ b \}, \) and \( \{ a, b \} \) are the generalized disjunctive stable models of \((P, A)\). On the other hand,

\[
dlp_{\text{A}}((P, A)) = \{ \text{a} \lor e_a \leftarrow, \; \text{b} \lor e_b \leftarrow \},
\]

and \( \{ e_a, e_b \}, \{ a, e_b \}, \{ b, e_a \}, \) and \( \{ a, b \} \) are the corresponding disjunctive stable models of \( dlp_{\text{A}}((P, A)) \).

Note here that if we do not distinguish \( e_a \) and \( e_b \), \( \{ a, e_b \} \) and \( \{ b, e_a \} \) do not become disjunctive stable models of \( dlp_{\text{A}}((P, A)) \).

**Discussion**

The above theorem presents that normal disjunctive programs are also as expressive as abductive disjunctive programs under the disjunctive stable model semantics. However, in contrast to the case of the possible model semantics, abductive disjunctive programs cannot be efficiently reduced to abductive logic programs under the generalized disjunctive stable model semantics. We verify this fact in the next section from the computational complexity viewpoint.

**5.5 Discussion**

In this section, we discuss the computational aspects of disjunctive and abductive logic programs. Throughout the section, programs are assumed to be propositional programs.

When abductive logic programs do not contain default negation, Selman and Levesque [1990] and Eiter and Gottlob [1992] show that the decision problem of the existence of explanations for a given observation in an abductive Horn program is \( \text{NP} \)-complete. In other words, in an abductive Horn program, deciding whether there is a generalized stable model satisfying an observation is \( \text{NP} \)-complete.

Inoue [1991] and Satoh and Iwayama [1991] show that an abductive logic program can be translated into a semantically equivalent normal logic program. For an abductive logic program \((P, A)\), consider a normal logic program obtained from \( P \) by adding the following clauses for each abducible \( A \) in \( A \):

\[
A \leftarrow \neg A',
\]

\[
A' \leftarrow \neg A,
\]

where \( A' \) is a newly introduced atom not appearing elsewhere in \( P \) and is uniquely associated with each \( A \). Then these authors show that there is a one-to-one correspondence between the generalized stable models of \((P, A)\) and the stable models of the transformed normal logic program. Since it is known that the set-membership problem under the stable model semantics is \( \text{NP} \)-complete [Marek and Truszczyński, 1991a], the above polynomial-time translation implies that deciding whether there is a generalized stable model satisfying a given observation is also \( \text{NP} \)-complete.

\[\]
Table 5.1: Comparison of Computational Complexity

<table>
<thead>
<tr>
<th>Program</th>
<th>Semantics</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abductive LP</td>
<td>Horn Abduction</td>
<td>NP-complete</td>
</tr>
<tr>
<td></td>
<td>Generalized Stable Model</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Normal DLP</td>
<td>Possible Model</td>
<td>NP-complete</td>
</tr>
<tr>
<td></td>
<td>Disjunctive Stable Model</td>
<td>$\Sigma_2^p$-complete</td>
</tr>
<tr>
<td>Abductive DLP</td>
<td>Generalized Possible Model</td>
<td>NP-complete</td>
</tr>
<tr>
<td></td>
<td>Generalized Disjunctive Stable Model</td>
<td>$\Sigma_2^p$-complete</td>
</tr>
</tbody>
</table>

From the results presented in Section 5.4, generalized possible models can be efficiently translated into possible models. Since we have already seen in Section 3.6 that the set-membership problem under the possible model semantics is NP-complete, the corresponding decision problem for generalized possible models is also NP-complete.

On the other hand, we have also shown in Section 5.4 that abductive disjunctive programs are reducible to normal disjunctive programs under the disjunctive stable model semantics. Since the set-membership problem under the disjunctive stable model semantics is $\Sigma_2^p$-complete [Eiter and Gottlob, 1993a], deciding whether there is a generalized disjunctive stable model satisfying a given observation is also $\Sigma_2^p$-complete. These results are summarized in Table 5.1.

The above complexity measures verify the results of this chapter that the generalized stable model semantics of abductive logic programs can be expressed in terms of the possible model semantics of normal disjunctive programs by a polynomial-time transformation, and vice versa. Moreover, we can observe that there is no efficient way to express the disjunctive stable model semantics in terms of the generalized stable model semantics unless the polynomial hierarchy collapses. This observation extends the fact that disjunctive stable models cannot be expressed by stable models of a normal logic program in polynomial time [Eiter and Gottlob, 1993b]. Also we can observe that when considering to extend the framework of abductive logic programs to abductive disjunctive programs, the generalized possible model semantics enables us to extend the framework without increasing computational complexity, while this is not the case for the generalized disjunctive stable model semantics.

The possible model semantics is originally introduced in order to provide a flexible mechanism for closed world assumptions in disjunctive programs. However, the results of this chapter reveal that the possible model semantics is also useful to characterize abductive logic programs. Moreover, in Section 3.6 we have shown that normal disjunctive programs are reducible to normal logic programs under the possible model semantics. Since abductive disjunctive programs are reducible to normal disjunctive programs, we can conclude that adding disjunctions and abducibles does not increase the expressive power of normal logic programs under the possible model semantics.

5.6 Summary

This chapter has investigated formal correspondences between disjunctive programs and abductive logic programs. We first presented program transformations between abductive logic programs and disjunctive programs. It was shown that the generalized stable models of an abductive logic program are characterized by the possible models of the transformed normal disjunctive program, and vice versa.

Next we showed that disjunctive programs are as expressive as abductive disjunctive programs. Moreover, normal disjunctive programs, abductive logic programs, and abductive disjunctive programs are all equivalent under the possible model semantics. On the other hand, we have argued that expressing the disjunctive stable model semantics in terms of generalized stable models is most unlikely possible in polynomial time.

The results of this chapter indicate that disjunctive programs and abductive logic programs are just different ways of looking at the same problem if we choose the appropriate semantics. Also the usefulness of the possible model semantics was verified not only for disjunctive programs but also for abductive logic programs.
Chapter 6

Handling Inconsistency in Disjunctive Logic Programs

An extended disjunctive program is an extension of a normal disjunctive program, which contains classical negation along with default negation. In the presence of explicit negation, however, an extended disjunctive program possibly becomes inconsistent, and classical logic programming semantics are of no use in such inconsistent programs. In this chapter, we present theoretical frameworks for possibly inconsistent disjunctive programs. To achieve this goal, we introduce paraconsistent semantics for extended disjunctive programs, which can distinguish inconsistent information from other information in a program. These semantics are based on lattice-structured multi-valued logics, and are characterized by a fixpoint semantics of extended disjunctive programs. The proposed paraconsistent semantics are used for reasoning in inconsistent programs.

6.1 Introduction

Representing and reasoning with incomplete information in a program is one of the central issues in recent studies of logic programming. Extended disjunctive programs introduced by Gelfond and Lifschitz [1991] provide a fairly general framework for that purpose. An extended disjunctive program can specify incomplete information by using classical negation as well as disjunctions in a program. In the presence of such explicit negation in a program, however, an extended disjunctive program possibly becomes contradictory, since negative consequences are allowed in the program. In [Gelfond and Lifschitz, 1991], a declarative semantics of extended disjunctive programs is given by the notion of answer sets, which is a generalization of stable models.
of normal disjunctive programs. However, the problem of the answer set semantics is that the answer set becomes trivial in an inconsistent program and implies every formula from the program. This is also the case for most of the traditional logic programming semantics in which local inconsistency might spoil the whole program. Practically speaking, when we build a large-scale knowledge base in logic programming framework, inconsistent information as well as incomplete information is likely to happen in the knowledge base. In such a knowledge base, a piece of contradictory information would make the whole program inconsistent, but still the program may contain meaningful information which is not affected by the local inconsistency.

Paraconsistent logic is a logic which is not destructive in the presence of inconsistent information. In this logic, the contradictory statement $A \land \neg A$ does not deduce an arbitrary formula, hence would not trivialize the whole theory. In this regard, paraconsistent logic can localize inconsistent information in a theory and serves as a useful inference tool in artificial intelligence. Historically, paraconsistent logic has been developed in the area of philosophical logic [Arruda, 1980], and a formal framework for inconsistent theories was given by da Costa [1974]. Applications of paraconsistent logic to logic programming have also been investigated by several researchers. Blair and Subrahmanian [1989] firstly introduced a framework of paraconsistent logic programming. They extended Fitting's three-valued semantics of logic programming [Fitting, 1985] and developed a theory for possibly inconsistent logic programs using Belnap's four-valued logic [Belnap, 1975]. The result was generalized by Subrahmanian [1992] to programs containing disjunctive information. Recently, the paraconsistent logic programming framework was further extended to treat default negation along with explicit negation in a program [Pimentel and Rodi, 1991; Wagner, 1991; Kifer and Lozinskii, 1992; Grant and Subrahmanian, 1992]. However, in the context of extended disjunctive programs, a suitable paraconsistent extension of the answer set semantics has not been studied in the literature.

In this chapter, we present declarative semantics of possibly inconsistent extended disjunctive programs. The disjunctive stable model semantics and the possible model semantics of normal disjunctive programs are extended to the corresponding paraconsistent semantics for extended disjunctive programs. The proposed paraconsistent semantics are based on lattice-structured multi-valued logics, and are characterized by the fixpoint semantics of extended disjunctive programs. We also present applications of the paraconsistent semantics for reasoning in inconsistent programs.

The rest of this chapter is organized as follows. In Section 6.2, we first present the paraconsistent minimal and possible model semantics for positive extended disjunctive programs. The results are generalized in Section 6.3 to the paraconsistent stable and possible model semantics for extended disjunctive programs. In Section 6.4, a fixpoint semantics for extended disjunctive programs is presented to characterize the paraconsistent semantics of extended disjunctive programs. In Section 6.5, we discuss the issue of reasoning in inconsistent programs. The notions of preferred stable models, suspicious stable models, and semi-stable models are introduced as variants of the paraconsistent stable model semantics. Section 6.6 addresses comparisons with related work, and Section 6.7 summarizes this chapter.

6.2 Paraconsistent Semantics for Positive Extended Disjunctive Programs

In this section, we first consider positive extended disjunctive programs, that is, extended disjunctive programs containing no default negation.

6.2.1 Multi-valued Logic

To formalize the semantics of possibly inconsistent logic programs, multi-valued logics are often used instead of the traditional two-valued logic. Then we start from introducing such a new logic and define its model theory.

The set of truth values of four-valued logic is defined as $IV = \{ t, f, T, \bot \}$, in which $t$, $f$, $T$, $\bot$ are propositions in the language of a program and respectively denote true, false, contradictory, and undefined. The set of truth values $IV$ makes a complete lattice under the ordering $\leq$ such that $\bot \leq x \leq T$ for $x \in \{ t, f \}$ (Figure 6.1). Such a lattice is also known as Belnap's four-valued logic [Belnap, 1975].

In extended disjunctive programs, negative literals have the same status as positive literals, then it is natural to consider the Herbrand base of a program $P$ as the set of all ground literals $L_P$ from the language of $P$. For simplicity, we assume that programs do not contain the reserved propositions from $IV$ and $IV \not\subseteq L_P$.

Let $I$ be a subset of $L_P$. An interpretation of a positive extended disjunctive program $P$ is defined as a function $I : L_P \rightarrow IV$ such that for each literal $L \in L_P$,

$$I(L) = \begin{cases} 
  t & \text{if } L \in I \text{ and } \neg L \not\in I, \\
  f & \text{if } L \in I \text{ and } L \not\in I, \\
  T & \text{if both } L \in I \text{ and } \neg L \not\in I, \\
  \bot & \text{otherwise}.
\end{cases}$$

Note that the order considering here is the so-called knowledge ordering, while there is the alternative truth ordering in the context of bilogics [Cerny, 1988; Fitting, 1991].

2This assumption is not essential and can be removed, but this issue is not discussed here.
Note that \( I(L) = t \) if \( I(\neg L) = f \), \( I(L) = T \) if \( I(\neg L) = T \), and \( I(L) = \perp \) if \( I(\neg L) = \perp \).

In this chapter, when no confusion arises, we identify a set of literals \( \{L, I= \} \) with \( I= \).

Definition 6.1 Let \( P \) be a positive extended disjunctive program and \( I \) be an interpretation. Then,

1. For any literal \( L \in \mathcal{L}_P \),
   (a) \( I \models L \) if \( I \preceq L \),
   (b) \( I \models \neg L \) if \( I \preceq \neg L \).
2. For any disjunction of ground literals \( F = L_1 \vee \ldots \vee L_n \),
   \( I \models F \) if \( I \models L_i \) for some \( i \) (\( 1 \leq i \leq n \)).
3. For any conjunction of ground literals \( G = L_1 \wedge \ldots \wedge L_n \),
   \( I \models G \) if \( I \models L_i \) for every \( i \) (\( 1 \leq i \leq n \)).
4. For any ground clause \( C = F \leftarrow G \), \( I \models C \) if \( I \models F \) or \( I \not\models G \).
   In particular, \( I \models \neg G \) if \( I \not\models G \), and \( I \models F \) if \( I \models F \).

Figure 6.1: Four-valued lattice IV

Proposition 6.1 If a positive extended disjunctive program has a model, it has at least one \( p \)-minimal model.

Proof: Let us consider a decreasing sequence of models \( I_1 \supseteq I_2 \supseteq \cdots \) and their greatest lower bound \( I = \bigcap \{I_i \} \). Then, for each ground clause \( F \leftarrow G \) from a positive extended disjunctive program \( P \), if \( I \models G \), \( I_i \models G \) for any \( i \geq 1 \). In this case, since each \( I_i \) is a model of \( P \), \( F \) is not empty and \( I \models F \) holds for any \( i \geq 1 \). Thus \( I \models F \). Hence, \( I \) is also a model of \( P \), and by definition it is a \( p \)-minimal model.

Proposition 6.2 A consistent positive extended disjunctive program has a consistent \( p \)-minimal model.

Proof: When a positive extended disjunctive program \( P \) is consistent, it has a consistent model \( I \) by definition. If \( I \) is not \( p \)-minimal, there is a \( p \)-minimal model \( J \) such that \( J \prec I \) by Proposition 6.1. Then, \( I(L) \neq T \) for any \( L \in \mathcal{L}_P \).

Corollary 6.3 If a positive extended logic program has a model, it has the unique \( p \)-least model. In particular, a consistent positive extended logic program has the consistent \( p \)-least model.

Example 6.1 Let \( P \) be the program:

\[ \{ a \vee b \leftarrow, \neg a \leftarrow, \neg b \leftarrow, c \leftarrow \}. \]

Then \( P \) has two \( p \)-minimal models \( \{a, \neg a, \neg b, c\} \) and \( \{b, \neg a, \neg b, c\} \).
Note that the above program is inconsistent and the classical minimal model semantics makes the program trivial, while the p-minimal models retain truth information about c that is not affected by the inconsistency.

Remark:
1. As presented in Chapter 2, the meanings of the clauses $\leftarrow L$ and $\neg L \leftarrow$ are different in extended disjunctive programs. In our multi-valued setting, this is also the case. In fact, $I(L) = T$ is a model of the first clause, while it is not a model of the second clause. Such a difference is due to the non-contrapositive feature of the connective $\leftarrow$ in extended disjunctive programs.

2. Corresponding to the above fact, the program \{L $\leftarrow$, $\neg L \leftarrow$\} has a model $I(L) = T$ while \{L $\leftarrow$, $\neg L \leftarrow$\} has no model. That is, we consider any interpretation meaningless if it does not satisfy integrity constraints. However, it is easy to construct a paraconsistent theory for integrity violation if desired.

6.2.2 Paraconsistent Possible Model Semantics

The possible model semantics for positive disjunctive programs is extended to positive extended disjunctive programs in a straightforward manner.

Given a positive extended disjunctive program $P$, a split program is defined as a ground positive extended logic program obtained from $P$ by replacing each ground disjunctive clause $C : L_1 \vee \ldots \vee L_i \leftarrow L_{i+1} \wedge \ldots \wedge L_m$ with the following ground extended clauses (called split clauses):

$L_i \leftarrow L_{i+1} \wedge \ldots \wedge L_m$ for every $L_i \in S$

where $S$ is some non-empty subset of \{L_1, \ldots, L_i\}. Then a p-possible model of $P$ is defined as the p-least model of any split program of $P$.

By definition, any p-possible model is a model of $P$.

Example 6.2 Let $P$ be the program:

\[
(a \vee b \leftarrow, \ c \leftarrow a \land b, \ \neg c \leftarrow).
\]

Then $P$ has the three p-possible models \{a, $\neg$\}, \{b, $\neg$\} and \{a, b, c, $\neg$\}. \(\square\)

Note here that if $P$ contains the integrity constraint $\leftarrow c$ instead of the clause $\neg c \leftarrow$, the positive disjunctive program has two p-possible models \{a\} and \{b\}.

The following properties are direct consequences from Propositions 6.1 and 6.2.

### Proposition 6.4
When a positive extended disjunctive program has a model, it has at least one p-possible model. \(\square\)

### Proposition 6.5
A consistent positive extended disjunctive program has a consistent p-possible model. \(\square\)

As for the relation to p-minimal models, the following property holds.

### Proposition 6.6
Any p-minimal model is a minimal p-possible model, and vice versa. \(\square\)

6.3 Paraconsistent Semantics for Extended Disjunctive Programs

This section extends the results presented in the previous section to extended disjunctive programs in general.

6.3.1 Paraconsistent Stable Model Semantics

We first extend the model theory given for positive extended disjunctive programs in the previous section to extended disjunctive programs containing default negation.

In an extended disjunctive program, the notion of interpretations and satisfaction of literals and clauses are defined in the same manner as in Definition 6.1 except that for each formula not $L$, we include the additional statements:

- $I \models not L$ iff $I(L) \leq f$.
- $I \models not \neg L$ iff $I(L) \leq t$.

The first condition indicates that if $L$ is false in $I$, its default negation not $L$ holds in $I$; else if $L$ is undefined in $I$, not $L$ holds in $I$ as negation as failure to prove; otherwise $I \not\models not L$. The second condition gives the counterpart statement.

The notion of (p-minimal) models is defined in the same way as in the previous section.

The paraconsistent stable model semantics of an extended disjunctive program is defined as follows.
Definition 6.2 Let $P$ be an extended disjunctive program and $I$ be a subset of $\mathcal{L}_P$. The reduct of $P$ with respect to $I$ is the positive extended disjunctive program $P^I$ such that a clause

$$L_1 \lor \ldots \lor L_l \leftarrow L_{l+1} \land \ldots \land L_m$$

is in $P^I$ iff there is a ground clause of the form:

$$L_1 \lor \ldots \lor L_l \leftarrow L_{l+1} \land \ldots \land L_m \land \neg L_{m+1} \land \ldots \land \neg L_n \ (l \geq 0)$$

from $P$ such that $\{L_{m+1}, \ldots, L_n\} \cap I = \emptyset$. Then $I$ is called a paraconsistent stable model (shortly, p-stable model) of $P$ if $I$ is a p-minimal model of $P$. □

Example 6.3 Let $P$ be the program:

$$\{a \lor b \leftarrow, \neg a \leftarrow, \neg b \leftarrow, c \leftarrow \neg d \}.$$  

Then $P$ has two p-stable models $\{a, \neg a, \neg b, c\}$ and $\{b, \neg a, \neg b, c\}$. □

A p-stable model $I$ is consistent if $I(L) \neq \top$ for any $L \in \mathcal{L}_P$, otherwise $I$ is inconsistent. There is a program which has no p-stable model.

Example 6.4 The program

$$P = \{a \leftarrow \neg a, \ b \leftarrow \}$$

has no p-stable model. □

A program which has at least one p-stable model is called coherent, while a program having no p-stable model is called incoherent. By definition, the notion of p-stable models reduces to that of p-minimal models in positive extended disjunctive programs. In extended disjunctive programs, every p-stable model is minimal.

Proposition 6.7 A p-stable model is a p-minimal model.

Proof: Let $I$ be a p-stable model of a program $P$. Assume that there is a p-minimal model $J$ of $P$ such that $J \leq I$. Since $J$ is a model of $P$ and satisfies each clause (6.2) in $P$, and $\{L_{m+1}, \ldots, L_n\} \cap I = \emptyset$ and $J \leq I$ imply $\{L_{m+1}, \ldots, L_n\} \cap J = \emptyset$, $J$ also satisfies each clause (6.1) in $P^I$. Thus $J$ is a model of $P^I$, and since $I$ is a p-minimal model of $P^I$, $J \leq I$ implies $J = I$. □

The converse of the above proposition does not hold in general. For instance, in Example 6.4, $\{a, b\}$ is the p-minimal model of $P$, but not p-stable.

6.3.2 Paraconsistent Possible Model Semantics

The paraconsistent possible model semantics for positive extended disjunctive programs is also directly extended to extended disjunctive programs.

Given an extended disjunctive program $P$, a split program is defined as a ground extended logic program obtained from $P$ by replacing each ground disjunctive clause $C : L_1 \lor \ldots \lor L_l \leftarrow L_{l+1} \land \ldots \land L_m \land \neg L_{m+1} \land \ldots \land \neg L_n$ with the following ground extended clauses (called split clauses):

$$L_i \leftarrow L_{l+1} \land \ldots \land L_m \land \neg L_{m+1} \land \ldots \land \neg L_n \text{ for every } L_i \in S$$

where $S$ is some non-empty subset of $\{L_1, \ldots, L_l\}$. Then a p-possible model of $P$ is defined as a p-stable model of any split program of $P$.

It is easy to see that any p-possible model is a model of $P$. The following properties are direct extensions of those presented in Section 3.3.

Proposition 6.8 A coherent extended disjunctive program has at least one p-possible model. □

Proposition 6.9 Any p-stable model is a minimal p-possible model, but not vice versa. □

As is the case of normal disjunctive programs, there are incoherent programs having p-possible models.

Example 6.5 Let $P$ be the program:

$$\{a \lor \neg b \leftarrow, \neg b \leftarrow a, \neg \neg a \}.$$  

Then $P$ has the p-possible model $\{a, \neg b\}$, while it has no p-stable model. □

6.3.3 Connection to the Answer Set Semantics

For extended disjunctive programs, Gelfond and Lifschitz [1991] have introduced the answer set semantics. The answer sets are defined in the same manner as p-stable models in Definition 6.2 except that the definition of p-minimal models of a positive extended disjunctive program $P^I$ is changed in a way that $I = \mathcal{L}_P$ if a model $I$ contains a pair of complementary literals $L$ and $\neg L$. For instance, in Example 6.3, $P$ has two inconsistent p-stable models, while it has the unique answer set $\mathcal{L}_P$. Thus p-stable models are paraconsistent, while answer sets are not.
In this section, we provide a connection between the p-stable model semantics and the answer set semantics in extended disjunctive programs. As presented above, the essential difference between the two semantics is the treatment of inconsistency. Then we relate p-stable models to answer sets by trivializing inconsistent p-stable models.

Let us consider a program \( P \), obtained from \( P \) by incorporating the trivialization rule:

\[
N \leftarrow L \land \neg L
\]

(6.3)

for all literals \( L \) and \( N \) from \( \mathcal{L}_P \). Then the relationship between answer sets and p-stable models of extended disjunctive programs is as follows.

**Theorem 6.10** Let \( P \) be an extended disjunctive program. Then \( I \) is an answer set of \( P \) iff \( I \) is a p-stable model of \( P_r \).

**Proof:** Since consistent answer sets coincide with consistent p-stable models, the result follows when \( I \) is a consistent answer set. Otherwise, suppose the case that \( P \) has the contradictory answer set \( \mathcal{L}_P \). Then, by the definition of answer sets, the positive extended disjunctive program \( P^{+}_r \) has the answer set \( \mathcal{L}_P \). In this case, \( P^{+}_r \) has no consistent p-minimal model, but an inconsistent p-minimal model. Thus, in the presence of the trivialization rule (6.3), \( P^{+}_r \) has the p-minimal model containing every literal \( N \) from \( \mathcal{L}_P \). Hence, \( \mathcal{L}_P \) is the unique p-stable model of \( P_r \). On the other hand, if \( I \) is an inconsistent p-stable model of \( P_r \), it contains every literal \( N \) from \( \mathcal{L}_P \) by (6.3). In this case, \( \mathcal{L}_P \) is the p-minimal model of \( P^{+}_r \), and thus each p-minimal model of \( P^{+}_r \) contains a pair of complementary literals. Hence, \( \mathcal{L}_P \) is the unique answer set of \( P \). \( \square \)

The above theorem indicates that we can easily simulate the "classical" meaning of logic programming by a simple program transformation. Note that without the trivialization rule, there is no one-to-one correspondence between inconsistent p-stable models and the answer set \( \mathcal{L}_P \) in general.

**Example 6.6** The program

\[
\{ \neg a \leftarrow, \ a \leftarrow \neg b \}
\]

has no answer set, while it has an inconsistent p-stable model \( \{\neg a, a\} \). On the other hand, the program

\[
\{ a \leftarrow, \ \neg a \leftarrow, \ b \leftarrow \neg b \}
\]

has the answer set \( \mathcal{L}_P \), while it has no p-stable model. \( \square \)

**Example 6.7** The program

\[
\{ a \leftarrow, \ b \leftarrow \}
\]

has no answer set, while it has an inconsistent p-stable model \( \{\neg a \leftarrow, \ \neg b \leftarrow \} \). On the other hand, the program

\[
\{ a \leftarrow, \ \neg a \leftarrow, \ b \leftarrow \neg b \}
\]

has the answer set \( \mathcal{L}_P \), while it has no p-stable model. \( \square \)

**Example 6.8** The program

\[
\{ a \leftarrow, \ b \leftarrow \}
\]

has no answer set, while it has an inconsistent p-stable model \( \{a \leftarrow, \ b \leftarrow \} \). On the other hand, the program

\[
\{ a \leftarrow, \ \neg a \leftarrow, \ b \leftarrow \neg b \}
\]

has the answer set \( \mathcal{L}_P \), while it has no p-stable model. \( \square \)

**Example 6.9** The program

\[
\{ a \leftarrow, \ b \leftarrow \}
\]

has no answer set, while it has an inconsistent p-stable model \( \{a \leftarrow, \ b \leftarrow \} \). On the other hand, the program

\[
\{ a \leftarrow, \ \neg a \leftarrow, \ b \leftarrow \neg b \}
\]

has the answer set \( \mathcal{L}_P \), while it has no p-stable model. \( \square \)

**Example 6.10** The program

\[
\{ a \leftarrow, \ b \leftarrow \}
\]

has no answer set, while it has an inconsistent p-stable model \( \{a \leftarrow, \ b \leftarrow \} \). On the other hand, the program

\[
\{ a \leftarrow, \ \neg a \leftarrow, \ b \leftarrow \neg b \}
\]

has the answer set \( \mathcal{L}_P \), while it has no p-stable model. \( \square \)

**Example 6.11** The program

\[
\{ a \leftarrow, \ b \leftarrow \}
\]

has no answer set, while it has an inconsistent p-stable model \( \{a \leftarrow, \ b \leftarrow \} \). On the other hand, the program

\[
\{ a \leftarrow, \ \neg a \leftarrow, \ b \leftarrow \neg b \}
\]

has the answer set \( \mathcal{L}_P \), while it has no p-stable model. \( \square \)

**Example 6.12** The program

\[
\{ a \leftarrow, \ b \leftarrow \}
\]

has no answer set, while it has an inconsistent p-stable model \( \{a \leftarrow, \ b \leftarrow \} \). On the other hand, the program

\[
\{ a \leftarrow, \ \neg a \leftarrow, \ b \leftarrow \neg b \}
\]

has the answer set \( \mathcal{L}_P \), while it has no p-stable model. \( \square \)

The example illustrates that a program possibly has an inconsistent p-stable model even when there is no answer set of the program. When a program has no p-stable model, on the other hand, the program has either no answer set or the trivial answer set \( \mathcal{L}_P \). Since the answer set semantics brings no useful information, the absence of p-stable models in this case is not a serious drawback. From this observation, the paraconsistent stable model semantics is considered to be more useful than the answer set semantics.

By Proposition 6.9, the answer set semantics is related to the paraconsistent possible model semantics as follows.

**Corollary 6.11** Let \( P \) be an extended disjunctive program. If \( I \) is an answer set of \( P \), then \( I \) is a minimal p-possible model of \( P_r \). \( \square \)

As previously discussed, paraconsistent possible models are well-defined whenever paraconsistent stable models are. Then, together with the above observation, we can conclude that the paraconsistent possible model semantics is the best choice among others as a semantics of extended disjunctive programs.

P-stable models of an extended disjunctive program are also characterized by stable models of the positive form of the program. Recall that a positive form of an extended disjunctive program \( P \) is defined as a normal disjunctive program \( P^+ \) which is obtained by replacing each negative literal \( \neg L \) in \( P \) with a corresponding newly introduced atom \( L' \). Let \( I^+ \) be a model of such \( P^+ \). Then the following relation holds by definition.

**Proposition 6.12** Let \( P \) be an extended disjunctive program and \( P^+ \) be its positive form. Then \( I \) is a p-stable model of \( P \) iff \( I^+ \) is a stable model of \( P^+ \). \( \square \)

Note that in case of the answer set semantics the above relation holds only for consistent answer sets [Gelfond and Lifschitz, 1991].

### 6.4 Fixpoint Semantics of Extended Disjunctive Programs

A fixpoint semantics of positive extended disjunctive programs is defined in the same manner as that of positive disjunctive programs in Section 3.4 except that in this case the closure operator acts over the lattice of sets of interpretations \( 2^{\mathcal{L}_P} \).

Given a positive extended disjunctive program \( P \), a mapping \( T_P : 2^{\mathcal{L}_P} \rightarrow 2^{\mathcal{L}_P} \) is defined as in Definition 3.7, and its fixpoint closure \( T_P \upharpoonright \omega \) is defined as well.
Let $\mathcal{PMM}_P$ be the set of all $p$-minimal models, and $\mathcal{PPM}_P$ be the set of all $p$-possible models of a program $P$. Then the following results hold.

**Theorem 6.13** Let $P$ be a positive extended disjunctive program. Then,

(i) $\mathcal{PPM}_P = \mu(T_P \uparrow \omega)$.

(ii) $\mathcal{PMM}_P = \min(\mu(T_P \uparrow \omega))$.

**Proof:** Viewing each negative literal in $P$ as an atom, the results directly follow from Theorem 3.17 in Section 3.4.

**Corollary 6.14** Let $P$ be a positive extended logic program. Then $T_P \uparrow \omega$ contains the unique $p$-least model of $P$.

The above corollary corresponds to Blair and Subrahmanian's fixpoint semantics of paraconsistent logic programs [Blair and Subrahmanian, 1989].

The fixpoint semantics is also generalized to extended disjunctive programs. Given an extended disjunctive program $P$, let us define the notions of the epistemic transformation $P^*$, canonical interpretations, and the function $\text{objc}(I)$ in the same way as those of normal disjunctive programs. Let $\mathcal{PST}_P$ be the set of all $p$-stable models of an extended disjunctive program $P$. Then the following results hold.

**Theorem 6.15** Let $P$ be an extended disjunctive program. Then,

(i) $\mathcal{PMM}_P = \text{objc}(\mu(T_P \uparrow \omega))$.

(ii) $\mathcal{PST}_P = \text{objc}(\min(\mu(T_P \uparrow \omega)))$.

In particular, when $P$ is an extended logic program, $\mathcal{PST}_P = \text{objc}(\mu(T_P \uparrow \omega))$.

**Proof:** The results follow from Lemma 3.20, Lemma 3.21, and Theorem 3.22 in Section 3.4.

As for the answer set semantics, the following relation holds by Theorem 6.10.

**Corollary 6.16** Let $P$ be an extended disjunctive program and $\mathcal{AS}_P$ be the set of all answer sets of $P$. Then,

$$\mathcal{AS}_P = \text{objc}(\min(\mu(T_P \uparrow \omega)))$$

where $P^*$ is the epistemic transformation of $P$. In particular, when $P$ is an extended logic program, $\mathcal{AS}_P = \text{objc}(\mu(T_P \uparrow \omega))$.

### 6.5 Reasoning with Inconsistency

This section presents applications of paraconsistent semantics for reasoning with inconsistent information. In this section, we discuss the issue based on the paraconsistent stable model semantics, while the techniques are directly applicable to the paraconsistent possible model semantics.

#### 6.5.1 Preferred Stable Models

In the previous section, we have defined the paraconsistent stable model semantics by the collection of all $p$-stable models. However, when a program has consistent models as well as inconsistent ones, a rational reasoner may prefer consistent models to inconsistent ones and consider truth values only in consistent models.

**Example 6.7** Let $P$ be the program:

$$\{ \neg a \lor b \leftarrow, \ a \leftarrow, \ c \leftarrow, \ \neg c \leftarrow \not c \}$$

which has two $p$-stable models $\{a, \neg a, \neg c\}$ and $\{a, b, \neg c\}$. In this case, however, it seems natural to prefer the consistent model $\{a, b, \neg c\}$ and conclude the truth of $a$ and $b$.

When an extended disjunctive program has consistent $p$-stable models, we distinguish these consistent $p$-stable models as **preferred $p$-stable models**. Thus preferred $p$-stable models characterize "consistent" meaning of a program. In fact, preferred $p$-stable models coincide with consistent answer sets of extended disjunctive programs.

**Theorem 6.17** Let $P$ be an extended disjunctive program. Then $I$ is a preferred $p$-stable model of $P$ if $I$ is a consistent answer set of $P$.

**Proof:** Since consistent answer sets coincide with consistent $p$-stable models, the result follows.

#### 6.5.2 Suspicious Stable Models

When a program contains inconsistent information, it is useful to distinguish facts affected by such information from other information in a program.
Example 6.8 Let $P$ be the program:

$$
\{ a \leftarrow b \land \neg c, \quad b \leftarrow, \quad \neg b \leftarrow, \quad d \leftarrow. \}
$$

Then $P$ has the $p$-stable model $\{a, b, \neg b, d\}$. However, the truth of $a$ is less credible than the truth of $d$, since $a$ is derived through the contradictory fact $b$. \[ \square \]

In order to distinguish such suspicious facts from others, we present suspicious reasoning under the paraconsistent stable model semantics. To this end, we first introduce two new truth values $st$ and $sf$, which respectively denote suspiciously true and suspiciously false. These newly introduced values together with the values in $IV$ constitute a lattice of six-valued logic $VI$ such that $\perp \leq sx \leq x \leq T$ for $x \in \{t, f\}$ (Figure 6.2).

Let $L_P = L_P \cup \{L | L \in L_P\}$ and $P^*$ be a subset of $L_P$, where each adorned literal $L^*$ denotes a suspicious literal. Then an interpretation under the logic $VI$ is defined as a function $P^* : L_P \to VI$ such that for each literal $L \in L_P$,

$$
P^*(L) = \text{lub}\{x \mid x = t \text{ if } L \in P^*, \quad x = f \text{ if } \neg L \in P^*, \quad x = st \text{ if } -L \in P^*, \quad x = sf \text{ if } L^* \in P^*, \quad x = \perp \text{ otherwise}\}.
$$

That is, the truth value of each literal $P^*(L)$ is defined as the least upper bound of each value $x$ which is determined by the literal occurrences in $P^*$. Thus, $P^*(L) = T$ if either $L \in P^*$ and $\neg L \in P^*$, or $L^* \in P^*$ and $\neg L^* \in P^*$, or $L \in P^*$ and $\neg L^* \in P^*$, or $\neg L \in P^*$ and $\neg L^* \in P^*$.

**Figure 6.2: Six-valued lattice VI**

**REASONING WITH INCONSISTENCY**

$\neg L \in P^*$ and $L^* \in P^*$. Note that $P^*(\neg L) = st$ iff $P^*(L) = sf$. Under the logic $VI$, satisfaction of literals and default negation is defined as follows.

- $P^* \models L$ iff $st \leq P^*(L)$,
- $P^* \models \neg L$ iff $sf \leq P^*(L)$,
- $P^* \models L$ iff $P^*(L) \leq f$,
- $P^* \models \neg L$ iff $P^*(L) \leq t$.

Satisfaction of clauses is the same as before.

Next, for a positive extended disjunctive program $P$ and an interpretation $P^*$, let $T_P^*$ be a mapping which is defined in the same way as in Definition 3.7 except that we consider the mapping $T_P$ instead of $T_P$ as follows: if $P^* \models L_1 \land \ldots \land L_m$ for some ground integrity constraint $\neg L_1 \land \ldots \land L_m$ from $P$, then $T_P^*(P^*) = \emptyset$; otherwise,

$$
T_P^*(P^*) = \{ P^* \mid \text{for each ground clause } C : L_1 \lor \ldots \lor L_m \models L_{i+1} \land \ldots \land L_m, \text{from } P \text{ such that } P^* \models L_{i+1} \land \ldots \land L_m, \text{ where } L_{i+1} = L_i \text{ if } L_i \in P^* \text{ and } \neg P^* \not\models \neg L_i \text{ for each literal } L_i \text{ for all } 1 \leq i \leq m; \text{ and } L_{i+1} = L_i \text{ otherwise}\}.
$$

The intuitive meaning of $T_P^*$ is that when the body of a clause $C_i$ is satisfied by $P^*$, each derived disjunct $L_{i+1} = L_i$ is added to $P^*$ if any literal $L_i$ in the body is derived without suspicion and its negative counterpart $\neg L_i$ or $\neg L_i^*$ is not included in $P^*$. Otherwise, the derived disjunct is suspicious $L_{i+1} = L_i^*$, since it is derived through inconsistent information in a program.

Given an extended disjunctive program $P$ and its epistemic transformation $P^*$, let us consider the fixpoint closure $\text{SPST}^* = \text{obj}_\varepsilon(\min(\mu(T_P^*, \uparrow \omega)))$. We call $\text{SPST}^*$ the suspicious $p$-stable models of $P$.

**Theorem 6.18** Let $P$ be an extended disjunctive program. Then a suspicious $p$-stable model is a model of $P$.

**Proof:** In a suspicious $p$-stable model, the truth value of each literal possibly becomes $st$ or $sf$ when its truth value is respectively $t$ or $f$ in its corresponding $p$-stable model. Let $P^*$ be a suspicious $p$-stable model and $I$ be its corresponding $p$-stable model in which each literal $L^*$ is identified
with \( L \). By definition, \( I^* \models L \) if and only if there exists a p-stable model \( I \) of \( P \) such that \( I \models L \).

Thus \( I^* \) satisfies each clause of \( P \) whenever \( I \) is a p-stable model of \( P \).

Hence the result follows. \( \square \)

**Corollary 6.19** For each suspicious p-stable model \( I^* \) of \( P \), \( I = \{ L \mid I^* \models L \} \) is a p-stable model of \( P \).

Conversely, for each p-stable model \( J \) of \( P \), there is a suspicious p-stable model \( J^* \) of \( P \) such that \( J = \{ L \mid J^* \models L \} \). \( \square \)

Thus suspicious p-stable models can distinguish information derived through contradictory facts from other information. Clearly, suspicious p-stable models reduce to p-stable models in the absence of suspicious information.

Note that a proved fact is considered to be suspicious if every proof of the fact includes inconsistent information. Then if an interpretation includes both contradictory beliefs, it means that there is a proof of \( L \) depending on no inconsistent information. In this case, by taking the least upper bound of \( t \) and \( st \), the truth value of \( L \) becomes \( t \).

**Example 6.9** (cont. from Example 6.8)

\( P^* \) becomes

\[
\{ \lambda \lor Kc \leftarrow b, \ a \leftarrow \lambda, \ \leftarrow \lambda \land c, \ b \leftarrow, \ \neg b \leftarrow, \ d \leftarrow \}.
\]

and

\[
\text{obj}_I(\min(T_\phi, \top \omega)) = \{ \{a^*, b, \neg b, d\} \}.
\]

Then, \( I^*(a) = st, I^*(b) = T, I^*(c) = \perp, \) and \( I^*(d) = t \). \( \square \)

### 6.5.3 Semi-Stable Models

There is an extended disjunctive program which has no p-stable model but still contains useful information. For instance, in Example 6.4, \( P \) has no p-stable model but it seems reasonable to conclude the truth of \( b \). Roughly speaking, incoherence arises when a literal is implied by its default negation in a program. Since incoherence is viewed as a kind of inconsistency, it is desirable to provide a framework which is paraconsistent for such incoherence. In this section, we introduce the notion of semi-stable models which is paraconsistent for incoherent programs.

To represent incoherent facts, we first introduce five extra truth values \( bt, bf, bT, tcb, \) and \( fcb \) which respectively denote **believed true**, **believed false**, **believing contradictory**, **true with contradictory belief** and **false with contradictory belief**. These values together with the values in \( IV \) constitute a lattice of nine-valued logic \( IX \) such that \( \perp \leq bx \leq x \leq xcb \leq T \) and \( bx \leq bT \leq xcb \) for \( x \in \{ t, f \} \) (Figure 6.3).

---

**REASONING WITH INCONSISTENCY**

Let \( \mathcal{L}^*_p = \mathcal{L}_p \cup \{ KL \mid L \in \mathcal{L}_p \} \) and \( I^* \) be a subset of \( \mathcal{L}^*_p \). Then an interpretation under the logic \( IX \) is defined as a function \( I^* : \mathcal{L}_p^* \rightarrow IX \) such that for each literal \( L \in \mathcal{L}_p \),

\[
I^*(L) = \text{lub} \{ x \mid x = t \text{ if } L \in I^*,
\]

\[
x = f \text{ if } \neg L \in I^*,
\]

\[
x = bt \text{ if } KL \in I^*,
\]

\[
x = bf \text{ if } K\neg L \in I^*,
\]

\[
x = \perp \text{ otherwise.}
\]

Thus, \( I^*(L) = bT \) if both \( KL \in I^* \) and \( K\neg L \notin I^* \); \( I^*(L) = xcb \) if both \( KL \in I^* \) and \( L \notin I^* \), and so on. Note that \( I^*(L) = bt \) if \( I^*(\neg L) = bf \), \( I^*(L) = bT \) if \( I^*(\neg L) = bT \), and \( I^*(L) = tcb \) if \( I^*(\neg L) = fcb \).

The intuitive reading of each newly introduced truth value is that if \( I^*(L) = bt \), \( I^* \) contains a belief \( KL \) without its justification \( L \). On the other hand, if \( I^*(L) = tcb \), \( I^* \) contains a fact \( L \) with its opposite belief \( K\neg L \).

Under this logic, satisfaction of literals and default negation is defined in the same way as Section 6.3, i.e., \( I \models L \) if \( t \in I(L); \ I \models \neg L \text{ if } f \in I(L); \ I \models \text{not } L \text{ if } L \notin I(L); \ I \models \text{not not } L \text{ if } L \leq f; \) and \( I \models \text{not not } L \text{ if } I(L) \leq t \). Satisfaction of clauses is also defined as before.

According to the above definition, when \( I(L) = bt \) or \( I(L) = bT \), it holds that \( L \models L \) and \( L \models \text{not not } L \). This means when \( L \) is believed true, it is too weak to conclude the truth of \( L \), but enough to reject \( \text{not not } L \). Else when \( I(L) = tcb \), \( I \models L \) while...

---

\( ^3 \)Recall that \( \text{not not } L \) corresponds to \( K\neg L \).
\[ I \not \models \neg \neg \neg \neg \neg \neg \neg \neg \neg \neg L. \] This means when \( L \) is true with contradictory belief, \( I \) concludes the truth of \( \neg L \) but rejects \( \neg \neg \neg \neg \neg \neg \neg \neg \neg \neg L \) in the presence of its opposite belief \( \neg \neg \neg \neg \neg \neg \neg \neg \neg \neg L \).

Next let \( I^* \) be a set of interpretations of a program \( P^* \) obtained by the epistemic transformation of an extended disjunctive program \( P \). The epistemic transformation \( I^* \) is an interpretation \( I^* \) such that the canonical condition is satisfied as much as possible. In particular, if \( I^* \) contains an interpretation \( I^p \) which is canonical, it is also maximally canonical. Now let

\[ \text{obj}_c^m(I^*) = \{ I^* \cap L^*_P \mid I^* \in I^P, \text{ and } I^* \text{ is maximally canonical} \} \]

**Theorem 6.20** Let \( P \) be an extended disjunctive program. Then any interpretation included in \( \text{SST}_P = \text{obj}_c^m(\min(\mu(T_P \uparrow \omega))) \) is a model of \( P \).

**Proof:** By definition, each maximally canonical interpretation \( I^* \) included in \( \min(\mu(T_P \uparrow \omega)) \) is a model of \( P^* \). Then, for each transformed clauses:

\[ \lambda_1 \land \ldots \land \lambda_i \lor K L_{i+1} \land \ldots \land K L_n \leftarrow L_{i+1} \land \ldots \land L_m, \]

in \( P \). In case of \( K L_j \in I^* \), when \( L_j \in I^* \), \( I^* \) satisfies the clause \( (*) \) in \( P \). Else when \( L_j \notin I^* \), (i) if \( \neg \lambda_i \notin I^* \), the truth value of \( I_j \) becomes \( \text{bf} \) or \( \text{bt} \), then \( I^* \models \neg \lambda_i \). (ii) Else if \( \neg \lambda_i \notin I^* \), the truth value of \( \lambda_i \) becomes \( \text{bf} \), then \( I^* \models \neg \lambda_i \). In either case, \( I^* \) satisfies the clause \( (*) \) in \( P \). Therefore, \( I^* \) satisfies each clause in \( P \). Hence, \( I^* \cap L^*_P \), which is obtained from \( I^* \) by removing every \( \lambda_i \), is also a model of \( P \).

We call models \( \text{SST}_P \) the semi-stable models of \( P \).

The notion of semi-stable models reduces to p-stable models in coherent programs.

**Corollary 6.21** Let \( P \) be a coherent program. Then its semi-stable models coincide with the p-stable models.

**Proof:** When \( \min(\mu(T_P \uparrow \omega)) \) contains canonical interpretations, they are also maximally canonical. Hence the result follows by definition.

**REASONING WITH INCONSISTENCY**

The existence of semi-stable models is guaranteed for any program which has models.

**Theorem 6.22** When a program has a model, it has a semi-stable model.

**Proof:** When a program \( P \) has a model, it is easy to see that \( P^* \) also has a model. Then the closure \( \min(\mu(T_P \uparrow \omega)) \) contains models which are maximally canonical, hence \( \text{SST}_P \) is not empty.

Thus incoherent programs get the meaning by considering semi-stable models.

**Example 6.10** (Barbar’s Paradox)

Let \( P \) be the program:

\[ \{ \text{shave}(N, x) \leftarrow \neg \text{shave}(x, x), \text{mayor(Casanova)} \} \]

Then its epistemic transformation \( P^* \) becomes

\[ \{ \lambda(x) \lor K \text{shave}(x, x) \leftarrow, \text{shave}(N, x) \leftarrow \lambda(x), \text{mayor(Casanova)} \} \]

Thus,

\[ \min(\mu(T_P \uparrow \omega)) = \{ \{ K \text{shave}(N, N), \text{shave}(N, C), \text{mayor}(C) \} \}

In the above closure, the second interpretation is maximally canonical, hence

\[ \text{obj}_c^m(\min(\mu(T_P \uparrow \omega))) = \{ \{ K \text{shave}(N, N), \text{shave}(N, C), \text{mayor}(C) \} \}

which contains the unique semi-stable model of \( P \) such that \( \text{shave}(N, C) \) and \( \text{mayor}(C) \) are true, while \( \text{shave}(N, N) \) is believed true.

Note that the above program has neither standard two-valued stable models nor answer sets.

In the incoherent program \( \{ a \leftarrow \neg a \} \), it is known that interpretations “oscillate” between \( \emptyset \) and \( \{ a \} \) under the stable class semantics [Baral and Subrahmanian, 1992]. Then it is interesting to observe that the truth value \( l(a) = \text{bt} \) in its semi-stable model correspondingly lies between \( \text{bt} \) and \( \text{bf} \).
6.6 Related Work

A framework of paraconsistent logic programming is firstly developed by Blair and Subrahmanian [1989] in the context of annotated logic programs. They employ Belnap's four-valued logic as a theoretical basis, but their framework does not treat default negation in a program. Fitting [1991] provides a general framework for logic programming in terms of bilattices, but he does not discuss programs containing two kinds of negation. Kifer and Lozinskii [1992] extend Blair and Subrahmanian's annotated logic programming framework to a theory possibly containing default negation, and Wagner [1991a; 1991b] also develops a theory of inconsistent logic programs with two kinds of negation. Compared with our approach, they do not treat disjunctions in a program and the underlying logics presented in these literature are different from our stable model semantics.

Subrahmanian [1992] has extended the work of [Blair and Subrahmanian, 1989] to programs containing disjunctive information. However, he does not treat default negation in a program. He also provides a fixpoint semantics of paraconsistent disjunctive programs based on Minker and Rajaseker's model state fixpoint semantics, which is different from ours as presented in Section 3.7. Lu and Henschen [1992] consider specifying the closed world assumption in paraconsistent definite and disjunctive logic programs. However, they neither consider programs containing two kinds of negation nor develop any fixpoint theory for disjunctive programs.

Paraconsistent stable model semantics is also proposed by several researchers. Pimentel and Rodi [1981], Grant and Subrahmanian [1992], and Wagner [1993] study paraconsistent stable model semantics from different viewpoints. The differences between these approaches and ours are as follows. First, their paraconsistent stable model semantics are defined for extended logic programs and do not treat disjunctive information in a program. Second, they do not provide any mechanism to compute their stable models, while our fixpoint computation realizes constructive computation of paraconsistent stable models by using bottom-up model generation techniques as presented in Section 3.5. Third, we have introduced the notion of semi-stable models which are paraconsistent for incoherent programs, while they do not discuss the issue of handling incoherence. Recently, Fitting [1993] provided a framework of stable model semantics in terms of bilattices, but did not treat disjunctive programs. Another point is that our paraconsistent extensions are not only for the stable model semantics, but also for the possible model semantics. Sakama [1992] has also developed a paraconsistent well-founded semantics for extended logic programs and disjunctive programs, which is different from the stable model approach presented in this chapter.

6.7 Summary

The paraconsistent semantics presented in this chapter is intended to localize inconsistent information in a program. There is an alternative approach which tries to detect the source of inconsistency and recover the consistency of a program. Generally speaking, however, it is a hard task to automatically resolve inconsistency in a program. When inconsistency arises from default assumptions, Pereira et al. [1991] and Dung et al. [1991] present methods of removing the inconsistency by preferring a fact that does not depend on any default assumption. However, their approaches are of no use in a program where inconsistency is derived without default assumptions. Kowalski and Sadri [1990] resolve contradiction by giving a higher priority to one of the conflicting conclusions as an exception, but such an approach generally requires one to specify a preference for each individual rule. Inoue [1991] and Baral et al. [1992b] consider the meaning of an inconsistent program as a collection of maximally consistent subsets of the program, but such a collection exponentially grows according to the increase of inconsistent information.

This chapter has presented paraconsistent frameworks for extended disjunctive programs. We have introduced the paraconsistent minimal, stable, and possible model semantics for extended disjunctive programs based on lattice-structured multi-valued logics. The paraconsistent semantics are characterized by a fixpoint semantics of extended disjunctive programs. We have also discussed applications of the paraconsistent semantics for reasoning with inconsistency.

The paraconsistent semantics are natural extensions of those corresponding semantics for normal disjunctive programs, and compared with Gelfond and Lifschitz's answer set semantics, the proposed semantics do not trivialize a program in the presence of inconsistent information. The paraconsistent semantics presented in this chapter generalize previous studies of paraconsistent logic programming and provide a uniform framework of logic programming possibly containing inconsistent information, disjunctive information, integrity constraints, and both explicit and default negation in a program. From the computational viewpoint, the bottom-up model generation procedure presented in Section 3.5 is used to compute the p-stable/p-possible models, and complexity results presented in Section 3.6 are directly applicable to the corresponding paraconsistent semantics. Thus the paraconsistent possible model semantics has a computational advantage over the paraconsistent minimal and stable model semantics, and the answer set semantics.
Chapter 7

Partial Deduction of Disjunctive Logic Programs

In this chapter, we present partial deduction of disjunctive programs. We first show that normal partial deduction in logic programming is not applicable as it is in the context of disjunctive programs. Then we introduce a new partial deduction technique for disjunctive programs, and show that it preserves the minimal model semantics of positive disjunctive programs, and the disjunctive stable model semantics of normal disjunctive programs. Normal partial deduction is also used together with suitable program transformations from disjunctive programs to normal logic programs, and the possible model semantics is preserved through such transformations. Partial deduction techniques are also applied to goal-oriented partial deduction for query optimization.

7.1 Introduction

Logic programming provides a methodology as a declarative programming language, while a correctly specified logic program is not necessarily efficient as a practical programming language. In order to bridge the gap between declarative and practical programming, studies have been devoted to develop techniques for optimizing logic programs.

Partial deduction or partial evaluation is known as one of the optimization techniques in logic programming. Given a logic program, partial deduction derives a more specific program through performing deduction on a part of the program, while preserving the meaning of the original program. Such a specialized program is usually more efficient than the original program when executed.
Partial deduction in logic programming was firstly introduced by Komorowski [1981] and has been developed by several researchers from various viewpoints [Sestoft and Zamulin, 1988; Komorowski, 1992]. From the semantic point of view, Lloyd and Shepherdson [1991] formalized partial evaluation for normal logic programs and provide the conditions to assure the correctness with respect to Clark's program completion semantics. On the other hand, Tamaki and Sato [1984] showed that partial deduction preserves the least Herbrand model semantics of definite logic programs in the context of unfold/fold transformation. The result is extended to the perfect model semantics for stratified logic programs [Seki, 1991; Maher, 1993], and the well-founded semantics for normal logic programs [Seki, 1993].

When we consider disjunctive programs, they increase expressive power of logic programming on one hand, but their computation is generally expensive on the other hand. Then optimizations of disjunctive programs are important issues for practical usage, however, the partial deduction technique in normal logic programs is not applicable to disjunctive programs in its present form. This is because normal partial deduction is based on unfolding between normal clauses, and it supplies no mechanism for unfolding between disjunctive clauses.

In this chapter, we develop partial deduction techniques for disjunctive programs. We first show that normal partial deduction is not useful in the presence of disjunctive information in a program, then introduce disjunctive partial deduction for disjunctive programs. We prove that the proposed partial deduction method preserves the minimal model semantics of positive disjunctive programs, and the disjunctive stable model semantics of normal disjunctive programs. We also discuss the preservation of the possible model semantics, and present goal-oriented partial deduction for query optimization.

The rest of this chapter is organized as follows. In Section 7.2, we present disjunctive partial deduction for positive disjunctive programs and show its correctness with respect to the minimal model semantics. Section 7.3 extends the result to normal disjunctive programs containing default negation, and shows that proposed partial deduction also works well for the disjunctive stable model semantics. A connection between normal and disjunctive partial deduction is presented, and the preservation of the possible model semantics is discussed. In Section 7.4, the partial deduction technique is applied to goal-oriented partial deduction for query optimization. Section 7.5 discusses further issues, and Section 7.6 summarizes this chapter.

### Section 7.2 Partial Deduction of Positive Disjunctive Programs

In this section, we first present partial deduction of positive disjunctive programs. In the following, when we write \( A \lor \Sigma \leftarrow \Gamma \), \( \Sigma \) denotes a disjunction (possibly false) in the head, and \( \Gamma \) denotes a conjunction (possibly true) in the body. Note here that when we write a clause as \( A \lor \Sigma \leftarrow \Gamma \), it does not necessarily mean that \( A \) should be the leftmost atom in the head of the clause. That is, any two clauses are identified modulo the permutation of disjuncts/conjuncts in their heads/bodies.

Since a program is semantically identified with its ground program, we consider ground programs throughout this chapter unless stated otherwise. We also assume without loss of generality that a disjunction in the head of a ground clause is already factored, that is, each atom in the disjunctive head of a clause is different.

#### 7.2.1 Normal Partial Deduction

Partial deduction in logic programming is usually defined as unfolding of clauses in a program. For a Horn logic program \( P \), partial deduction is formally presented as follows.

Given a Horn clause \( C \) from \( P \):

\[
C : H \leftarrow A \land \Gamma, \quad \text{suppose that } C_1, \ldots, C_k \text{ are all of the clauses in } P \text{ such that each of which has the atom } A \text{ in its head:}
\]

\[
C_i : A \leftarrow \Gamma_i \quad (1 \leq i \leq k).
\]

Then normal partial deduction of \( P \) (with respect to \( C \) on \( A \)) is defined as the program \( \pi_{N[A]}(P) \) (called a residual program) such that:

\[
\pi_{[C;A]}(P) = (P \setminus \{C\}) \cup \{C'_1, \ldots, C'_k\}
\]

where each \( C'_i \) is defined as:

\[
C'_i : H \leftarrow \Gamma \land \Gamma_i.
\]

When we simply say normal partial deduction of \( P \) (written \( \pi_N(P) \)), it means normal partial deduction of \( P \) with respect to any clause on any atom.

---

1Partial deduction is also called partial evaluation. However, we prefer to use the term partial deduction, since partial evaluation often includes non-deductive procedures.
Example 7.1 Let \( P \) be the program:
\[
P = \{ a \leftarrow b, \ b \leftarrow c, \ b \leftarrow a, \ c \leftarrow \}
\]
Then normal partial deduction of \( P \) with respect to \( a \leftarrow b \) on \( b \) becomes
\[
\pi_{(a \leftarrow b)}^N(P) = \{ a \leftarrow c, \ a \leftarrow a, \ b \leftarrow c, \ b \leftarrow a, \ c \leftarrow \}
\]

In the context of unfold/fold transformation of logic programs, Tamaki and Sato [1984] showed that normal partial deduction preserves the least Herbrand model semantics of definite logic programs.

Lemma 7.1 ([Tamaki and Sato, 1984]) Let \( P \) be a definite logic program and \( M_P \) be its least Herbrand model. Then, for any residual program \( \pi^N(P) \) of \( P \), \( M_P = M_{\pi^N(P)} \).

The result also holds for Horn logic programs, that is, programs containing integrity constraints.

Theorem 7.2 Let \( P \) be a Horn logic program and \( \pi^N(P) \) be any residual program of \( P \). Then
\[
\pi^N(P) = (P \setminus \{ C \}) \cup \{ C_1, \ldots, C_k \}
\]
where each \( C_i \) is defined as
\[
C_i : \Sigma \cup \Sigma_i \leftarrow \Gamma \land \Gamma_i,
\]
in which \( \Sigma \cup \Sigma_i \) is factored.

Disjunctive partial deduction is a natural extension of normal partial deduction. In fact, the clause (7.3) is a resolvent of the clauses (7.1) and (7.2). In Horn logic programs, disjunctive partial deduction coincides with normal partial deduction.

Now we show that disjunctive partial deduction preserves the minimal model semantics of positive disjunctive programs. We first present a preliminary lemma.

Lemma 7.3 Let \( P \) be a positive disjunctive program and \( M \) be its minimal model.
Then an atom \( A \) is in \( M \) if there is a clause \( C : \Sigma \leftarrow A \land \Gamma \) in \( P \) such that \( M \setminus \{ A \} \models \Gamma \) and \( M \setminus \{ A \} \not\models \Sigma \).
Theorem 7.4 Let \( \text{program of} \ P \) be a positive disjunctive program and \( \pi^P(P) \) be any residual program of \( P \). Then \( \mathcal{M} M_P = \mathcal{M} M_{\pi^P(P)} \).

Proof: \((\Rightarrow)\) Suppose that for some atom \( A \) in \( M \), there is no clause \( C \) in \( P \) such that \( M \setminus \{ A \} \models \Gamma \) and \( M \setminus \{ A \} \not\models \Sigma \). Then, for each clause \( C, M \setminus \{ A \} \not\models \Gamma \) or \( M \setminus \{ A \} \models \Sigma \), and hence it holds that \( M \setminus \{ A \} \models \Gamma \) implies \( M \setminus \{ A \} \not\models \Sigma \). In this case, since \( M \setminus \{ A \} \) satisfies each clause \( C \), it becomes a model of \( P \), which contradicts the assumption that \( M \) is a minimal model. Hence the result follows.

\((\Leftarrow)\) Assume that \( A \) is not in \( M \). Then \( M \setminus \{ A \} = M \) and for a clause \( C \) in \( P, M \models \Gamma \) and \( M \not\models \Sigma \) imply \( A \in M \), contradiction. \( \Box \)

**Corollary 7.5** Let \( P \) be a positive disjunctive program. Then \( P \) is inconsistentiff \( \pi^P(P) \) is inconsistent. \( \Box \)

**Example 7.3** (cont. from Example 7.2) Given the program \( P \), its disjunctive partial deduction \( \pi^P_{(c \leftarrow a)}(P) \) becomes

\[
\pi^P_{(c \leftarrow a)}(P) = \{ a \leftarrow b, \quad a \leftarrow d, \quad c \leftarrow d, \quad b \lor c \leftarrow \}\,
\]

and \( \mathcal{M} M_{\pi^P_{(c \leftarrow a)}(P)} = \{\{a, c\}, \{b\}\} \), which is exactly the same as \( \mathcal{M} M_P \). \( \Box \)

### 7.3 Partial Deduction of Normal Disjunctive Programs

In this section, we extend disjunctive partial deduction to normal disjunctive programs.

#### 7.3.1 Disjunctive Partial Deduction of Normal Disjunctive Programs

The definition of disjunctive partial deduction for normal disjunctive programs is the same as Definition 7.1, except that in this case each clause possibly contains default negation.

**Example 7.4** Let \( P \) be the normal disjunctive program:

\[
P = \{ a \lor b \leftarrow \neg c, \quad a \leftarrow d, \quad c \leftarrow a \}.
\]

Then disjunctive partial deduction of \( P \) with respect to \( c \leftarrow a \) on \( a \) becomes

\[
\pi^P_{(c \leftarrow a)}(P) = \{ a \lor b \leftarrow \neg c, \quad a \leftarrow d, \quad c \leftarrow d, \quad b \lor c \leftarrow \neg c \}.
\]

As shown in the above example, disjunctive partial deduction is not affected by the presence of default negation in a program. Thus we can directly apply previously defined disjunctive partial deduction to normal disjunctive programs and the following result holds.
Theorem 7.6 Let $P$ be a normal disjunctive program and $ST_P$ be the set of all stable models of $P$. Then $ST_P = ST_{P(D)}$.

Proof: Let $M$ be a stable model of $P$. Then $M$ is a minimal model of $P^M$. Since $P^M$ is a positive disjunctive program, by Theorem 7.4, $M$ is also a minimal model of $D(P^M)$. Now let us consider the clauses:

$$\Sigma \leftarrow A \land \Gamma \land \text{not} \Gamma'$$

and

$$A \lor \Sigma_i \leftarrow \Gamma, \land \text{not} \Gamma'_i \quad (1)$$

in $P$, where $\text{not} \Gamma'$ is the conjunction of default-negation formulas in the body.

(i) If $M \models \Gamma'$ and $M \not\models \Gamma'_i$ for some $i$ $(1 \leq i \leq k)$, the clauses:

$$\Sigma \leftarrow A \land \Gamma$$

and

$$A \lor \Sigma_i \leftarrow \Gamma, \land \text{not} \Gamma'_i$$

are in $P^M$. From these clauses, disjunctive partial deduction generates the clauses:

$$\Sigma \lor \Sigma_i \leftarrow \Gamma \land \Gamma'_i$$

in $D(P^M)$. On the other hand, from $(*)$ and $(\dagger)$ in $P$, there are the clauses:

$$\Sigma \lor \Sigma_i \leftarrow \Gamma \land \Gamma'_i \land \text{not} \Gamma' \land \text{not} \Gamma'$$

in $D(P)$, which become $(\dagger')$ in $D(P)^M$.

(ii) Else if $M \not\models \Gamma'$ or $M \models \Gamma'_i$ for any $i$ $(1 \leq i \leq k)$, the clauses $(*)$ or $(\dagger)$ is respectively eliminated in $P^M$. Then the clauses $(\dagger')$ are not included in $D(P^M)$. In this case, each clause $(1)$ in $D(P)$ is also eliminated in $D(P)^M$.

Thus, there is a one-to-one correspondence between the clauses in $D(P^M)$ and the clauses in $D(P)^M$, hence $D(P^M) = D(P)^M$. Therefore $M$ is also a minimal model of $D(P)^M$, and a stable model of $D(P)$.

The converse is also shown in the same manner.

Corollary 7.7 Let $P$ be a normal disjunctive program. Then $P$ is incoherent iff $D(P)$ is incoherent.

NORMAL DISJUNCTIVE PROGRAMS

The above theorem also implies that in normal logic programs, normal partial deduction preserves Gelfond and Lifschitz's stable model semantics.

Corollary 7.8 Let $P$ be a normal logic program. Then $ST_P = ST_{N(P)}$.

The above result is also presented in [Seki, 1990].

7.3.2 Connections between Normal and Disjunctive Partial Deduction

In this section, we consider connections between normal and disjunctive partial deduction. We first give a sufficient condition such that normal partial deduction preserves the meaning of disjunctive programs.

Theorem 7.9 Let $P$ be a normal disjunctive program and $C$ be a clause of the form $\Sigma \leftarrow A \land \Gamma$ from $P$. If $A$ does not appear in the head of any disjunctive clause in $P$, then $ST_P = ST_{N(P)}$. That is, normal partial deduction of $P$ with respect to $C$ on $A$ preserves the disjunctive stable model semantics.

Proof: In this case, disjunctive partial deduction coincides with normal one, hence the result follows from Theorem 7.6.

Next we present a method to compute disjunctive partial deduction in terms of normal partial deduction.

Definition 7.2 Let $P$ be a normal disjunctive program. Then the normal transformation transforms $P$ into the normal logic program $\pi(P)$ which is obtained from $P$ by replacing each disjunctive clause:

$$C : A_1 \lor \ldots \lor A_l \leftarrow \Gamma$$

with $l$ normal clauses:

$$C^*_l : A_1 \leftarrow \Gamma \land A_1^* \land \ldots \land A_i^* \land A_{i+1}^* \land \ldots \land A_l^* \quad (1 \leq i \leq l).$$

where each $A_i^*$ is a new atom introduced for each $A_i$.

In particular, $C = C^*_l$ if $l \leq 1$.

Now we show that disjunctive partial deduction of a normal disjunctive program $P$ with respect to a clause $C$ is obtained through normal partial deduction of $\pi(P)$ with respect to each $C^*_l$. In the following, the function $\pi^{-1}$ is the reverse transformation which shifts each atom $A_i^*$ appearing in the body of each clause in a program to the atom $A_i$ in the head of the clause. Also $\Sigma^*$ means $A_1^* \land \ldots \land A_l^*$ where $\Sigma = A_1 \lor \ldots \lor A_l$. [Seki, 1990].
Theorem 7.10  Let \( P \) be a normal disjunctive program. Then, 
\[ \pi^D_{[C;A]}(P) = \eta^{-1}(\pi^N_{[C;A]}(\eta(P))) \]
where \( \pi^N_{[C;A]}(\eta(P)) \) means normal partial deduction of \( \eta(P) \) with respect to each normal clause \( C_i \) on \( A \).

**Proof:** Corresponding to the clauses (7.1) and (7.2) in \( P \), there are the clauses:
\[ A' \leftarrow A \land \Gamma \land \Sigma' \quad (\text{where } \Sigma = \Sigma' \lor A') \quad (\ast) \]
and
\[ A \leftarrow \Gamma, \land \Sigma_i^- \quad (1 \leq i \leq k) \quad (\dag) \]
in \( \eta(P) \), respectively. Then the clauses:
\[ A' \leftarrow \Gamma \land \Gamma \land \Sigma_i'^- \land \Sigma_i^- \quad (\dag) \]
are obtained from (\ast) and (\dag) by normal partial deduction in \( \eta(P) \). In this case, by the reverse transformation \( \eta^{-1} \), each clause of the form (\dag) becomes a disjunctive clause of the form (7.3). Hence, 
\[ \pi^D_{[C;A]}(P) = \eta^{-1}(\pi^N_{[C;A]}(\eta(P))). \]

**Example 7.5** Let \( P \) be the program:
\[ P = \{ a \lor b \leftarrow, \ a \leftarrow, \ b \leftarrow a \}. \]

Then,
\[ \pi^D_{[a \leftarrow b]}(P) = \{ a \lor b \leftarrow, \ a \leftarrow, \ a \leftarrow a, \ b \leftarrow a \}. \]

On the other hand, the nlp-transformation of \( P \) becomes
\[ \eta(P) = \{ a \leftarrow b^-, \ b \leftarrow a^-, \ a \leftarrow b, \ b \leftarrow a \}, \]
and
\[ \pi^N_{[a \leftarrow b]}(\eta(P)) = \{ a \leftarrow b^-, \ b \leftarrow a^-, \ a \leftarrow a^-, \ a \leftarrow a, \ b \leftarrow a \}. \]

Thus,
\[ \eta^{-1}(\pi^N_{[a \leftarrow b]}(\eta(P))) = \{ a \lor b \leftarrow, \ a \leftarrow, \ a \leftarrow a, \ b \leftarrow a \}. \]

Therefore,
\[ \pi^D_{[a \leftarrow b]}(P) = \eta^{-1}(\pi^N_{[a \leftarrow b]}(\eta(P))). \]

The above theorem presents that disjunctive partial deduction \( \pi^D_{[C;A]}(P) \) is obtained by the transformation sequence: \( P \rightarrow \eta(P) \rightarrow \pi^N_{[C;A]}(\eta(P)) \rightarrow \eta^{-1}(\pi^N_{[C;A]}(\eta(P))). \) That is, together with the nlp-transformation, normal partial deduction can also be used for normal disjunctive programs.

### 7.3.3 Preservation of the Possible Model Semantics

We consider partial deduction for disjunctive programs under the possible model semantics. Unfortunately, disjunctive partial deduction does not preserve the possible model semantics in general.

**Example 7.6** Let \( P \) be the disjunctive program:
\[ P = \{ a \lor b \leftarrow, \ c \leftarrow b \}. \]

Then disjunctive partial deduction of \( P \) with respect to \( c \leftarrow b \) on \( b \) becomes
\[ \pi^D_{[c \leftarrow b]}(P) = \{ a \lor b \leftarrow, \ a \lor c \leftarrow \}. \]

In this case, \( \mathcal{P}M_P = \{ \{a\}, \{b,c\}, \{a,b,c\} \} \), while the residual program has the possible models \( \{a,b\} \) and \( \{a,c\} \) in addition to \( \mathcal{P}M_P \).

In the above example, a dependency relation between \( c \) and \( b \) is not expressed after partial deduction. This observation tells us that resolution-based disjunctive partial deduction often fails to preserve syntax-dependent logic programming semantics. In fact, the above example shows that even the supported models of \( P \) are not preserved during partial deduction; \( \{a,c\} \) is a supported model of the residual program but it is not a supported model of \( P \).

Then we first consider a sufficient condition for preserving the possible model semantics. A result similar to Theorem 7.9 holds.

**Theorem 7.11** Let \( P \) be a normal disjunctive program and \( C \) be a clause of the form \( \Sigma \leftarrow A \land \Gamma \) from \( P \). If \( A \) does not appear in the head of any disjunctive clause in \( P \), then \( \mathcal{P}M_P = \mathcal{P}M_{\pi^D_{[C;A]}(P)} \).

**Proof:** In this case, stable models of each split program of \( P \) are preserved, hence the result follows. \( \Box \)

Next we present a method of partial deduction for preserving possible models in general. We have already seen in Section 3.6 that possible models of any disjunctive program are expressed by stable models of a normal logic program obtained by the pm-transformation. Then, we compute partial deduction for the possible model semantics in terms of normal partial deduction in such a transformed program.

The following result follows from Theorem 3.30 and Corollary 7.8.

**Theorem 7.12** Let \( P \) be a normal disjunctive program and \( \phi(P) \) be a normal logic program obtained by pm-transformation. Then \( \mathcal{P}M_P = ST_{\phi(N(\phi(P)))} \cap \mathcal{H}B_P \). \( \Box \)
7.4 Goal-Oriented Partial Deduction

In this section, we present goal-oriented partial deduction in disjunctive programs. Goal-oriented partial deduction specializes a program with respect to a given goal, which is useful to optimize programs for query-answering. Lloyd and Shepherdson [1991] discuss a framework of goal-oriented partial evaluation for normal logic programs with respect to SLDNF proof procedures. In our framework, goal-oriented partial deduction is presented as follows.

Let us consider a query of the form:

\[ Q: Q(x) \leftarrow B_1 \land \ldots \land B_m \land \text{not } B_{m+1} \land \ldots \land \text{not } B_n \]  

(7.6)

where \(Q(x)\) is a new atom not appearing elsewhere in a program and \(x\) represents variables appearing in the body of the clause.

Then, given a normal disjunctive program \(P\), partial deduction of \(P\) with respect to \(Q\) is defined as \(\pi^D_\emptyset_{Q,B_1}(P_Q)\) where \(B_i\) is any atom occurring positively in the body of \(Q\) and \(P_Q\) is the program \(P \cup \{ Q \}\). When a query contains variables, we consider partial deduction with respect to its ground instances.

As introduced in Section 3.5, an answer to a query is defined as a ground substitution \(\sigma\) for variables in \(Q(x)\).

A query \(Q\) is true in \(P\) under the disjunctive stable model semantics if for every stable model \(I\) of \(P_Q\) there is an answer \(\sigma\) such that \(Q(x)\sigma\) is included in \(I\). Else if for some stable model \(I\) of \(P_Q\) there is an answer \(\sigma\) such that \(Q(x)\sigma\) is included in \(I\), the query is possibly true. Otherwise, if there is no such answer, the query is false.

By Theorem 7.6, the following results hold.

Theorem 7.13 Let \(P\) be a normal disjunctive program and \(Q\) be a query. Then, under the disjunctive stable model semantics,

(i) \(Q\) is true in \(P\) iff \(Q\) is true in \(\pi^D_\emptyset_{Q,B_1}(P_Q)\).

(ii) \(Q\) is possibly true in \(P\) iff \(Q\) is possibly true in \(\pi^D_\emptyset_{Q,B_1}(P_Q)\).

(iii) \(Q\) is false in \(P\) iff \(Q\) is false in \(\pi^D_\emptyset_{Q,B_1}(P_Q)\). \(\Box\)

Example 7.7 Let \(P\) be the program:

\[ \{ p(a) \lor p(b) \leftarrow \} \]

in which the query \(Q: q(x) \leftarrow p(x)\) is true. Then,

\[ \pi^D_{\emptyset_{Q,p(a)}}(P_Q) = \{ q(a) \lor p(b) \leftarrow, q(b) \lor p(a) \leftarrow, q(a) \lor q(b) \leftarrow, p(a) \lor p(b) \leftarrow \} \]

and \(Q\) is also true in \(\pi^D_{\emptyset_{Q,p(a)}}(P_Q)\) under the disjunctive stable model semantics. \(\Box\)

DISCUSSION

Note that in the above example, we assume that the ground queries \(q(a) \leftarrow p(a)\) and \(q(b) \leftarrow p(b)\) are unfolded consecutively in the program. That is, \(\pi^D_{\emptyset_{Q,p(a)}}(P_Q)\) means \(\pi^D_{\emptyset_{Q,p(a)}}(\pi^D_{\emptyset_{Q,p(b)}}(P \cup \{ q(a) \leftarrow p(a), q(b) \leftarrow p(b) \}))\). In this case, the order of unfolding does not affect the result of partial deduction since each partial deduction preserves the stable models of the program \(P_Q\).

Using the technique presented in the previous section, corresponding results also hold for goal-oriented partial deduction under the possible model semantics.

7.5 Discussion

So far, we have considered partial deduction of propositional programs, while disjunctive partial deduction is also directly extended to programs containing variables.

Let \(P\) be a normal disjunctive program and \(C\) be a clause in \(P\) of the form:

\[ C: \Sigma \leftarrow A \land \Gamma. \]

Then, for any clause

\[ C_i: A_i \lor \Sigma_i \leftarrow \Gamma_i \ (1 \leq i \leq k) \]

from \(P\) such that \(A\sigma_i = A_i\sigma_i\) holds with an mgv \(\sigma_i\), the following clauses

\[ C'_i: (\Sigma \lor \Sigma_i \leftarrow \Gamma \land \Gamma_i)\sigma_i \]

are in the residual program \(\pi^D_{\emptyset_{\Sigma,A}}(P) = (P \setminus \{ C \}) \cup \{ C'_1, \ldots, C'_l \}\).

Correctness of such disjunctive partial deduction with respect to the disjunctive stable model semantics is proved in a similar way to the propositional case.

Disjunctive partial deduction is implemented on the bottom-up model generation procedure presented in Section 3.5, and it preserves the disjunctive stable model semantics in range-restricted function-free normal disjunctive programs. On the other hand, a transformation from disjunctive partial deduction to normal partial deduction presented in the previous section enables us to use a partial evaluator for normal logic programs also as a procedure for disjunctive partial deduction. Partial deduction under the possible model semantics is also executed using a partial evaluator of normal logic programs with respect to the stable model semantics.

In the context of deductive databases, a program usually consists of a large amount of data (called an extensional database) and a comparatively small set of derivation rules (called an intensional database). For query processing in such a database, it is effective to partially evaluate a query in the intensional database at first, and subsequently evaluate it in the extensional database [Gallaire et al., 1984]. Such a
PARTIAL DEDUCTION

142

technique can also be realized in our goal-oriented partial deduction presented in Section 7.4. Related topics are discussed in [Sakama and Itoh, 1988] for optimizing queries in deductive databases.

The partial deduction technique presented in this chapter is also directly applicable to extended disjunctive programs. Moreover, since positive disjunctive programs are identified with first-order theories, disjunctive partial deduction has potential application to first-order theorem provers. Recently, Brass and Dix [1994] independently developed a partial deduction technique for disjunctive programs which is equivalent to ours. They discuss several abstract properties of disjunctive programs and conclude partial deduction as one of the fundamental properties that logic programming semantics should satisfy.

7.6 Summary

In this chapter, we have presented a method of partial deduction for disjunctive programs. We first showed that normal partial deduction is not applicable to disjunctive programs in its present form. Then we introduced disjunctive partial deduction for disjunctive programs, which is a natural extension of normal partial deduction for normal logic programs. Disjunctive partial deduction was shown to preserve the minimal model semantics of positive disjunctive programs, and the disjunctive stable model semantics of normal disjunctive programs. We also showed a method of translating disjunctive partial deduction into normal partial deduction, and presented an application to goal-oriented partial deduction for query optimization.

Chapter 8
Conclusion

8.1 Summary and Contributions

In this dissertation, we have studied theoretical aspects of disjunctive logic programming from various viewpoints.

It has been considered that the principle of minimality is a basic criterion for any rational semantics of logic programming and commonsense reasoning in AI. However, it was also noticed that this principle is not always appropriate in the presence of indefinite information in a program. Then we started to develop a new theoretical framework of disjunctive logic programs, which violates this common principle. To this end, in Chapter 3 we proposed a new declarative semantics of disjunctive logic programs called the possible model semantics. Due to its non-minimal property, the possible model semantics can freely specify both inclusive and exclusive disjunctions in a program, and provides a flexible negative inference mechanism under the closed world assumption. The possible model semantics was also characterized by a new fixpoint semantics of disjunctive logic programs, and a bottom-up proof procedure for computing possible models was presented. The advantage of the possible model semantics lies not only in its representational power, but also in its computational complexity. That is, computation of possible models does not need an extra mechanism for minimality-checking, which makes computing the possible model semantics much easier than the minimal-model based semantics. We have verified this fact by comparing the computational complexity of each semantics.

Logic programming semantics is known to be closely related to nonmonotonic formalisms in AI. There have been studied methods of representing normal logic programs in terms of nonmonotonic formalisms, and their extensions to disjunctive logic programs were also proposed by several researchers. In Chapter 4, however, we have
pointed out that the previously studied result on default translations of disjunctive logic programs is incorrect. Then we introduced an alternative correct transformation from disjunctive logic programs to default theories, and showed that the disjunctive stable model semantics is characterized in terms of default extensions. The result indicates that Reiter's default logic still works well to characterize disjunctive logic programs, which breaks the folklore that Reiter's default logic is inappropriate to characterize disjunctive logic programs in general. In fact, Gelfond et al.'s disjunctive default logic is proposed to treat disjunctions "properly" in a default theory, while we have shown that Reiter's default logic has the same expressiveness as disjunctive default logic to characterize the semantics of disjunctive logic programs. Disjunctive logic programs were also characterized by autoepistemic logic and circumscription. Moreover, we have shown that the possible model semantics of disjunctive logic programs is expressed by the non-minimal feature of autoepistemic expansions.

Abduction is also a form of commonsense reasoning in AI, and its application to logic programming is known as abductive logic programming. Disjunctive logic programs and abductive logic programs are two extensions of logic programming which provide frameworks for reasoning with incomplete information, while little attention has been paid for their interrelations. Then, in Chapter 5, we revealed a close relationship between disjunctive logic programs and abductive logic programs. It was shown that the possible model semantics of disjunctive logic programs is essentially equivalent to the generalized stable models of abductive logic programs. This fact indicates that disjunctive logic programs and abductive logic programs are just different ways of looking at the same problem if we choose the appropriate semantics. Moreover, the possible model semantics is useful not only for disjunctive logic programs but also for abductive logic programs, and establishes links between each framework. The usefulness of the possible model semantics in abductive logic programming was also verified from the computational complexity viewpoint.

Another important issue for commonsense reasoning in logic programming is the treatment of inconsistent information in a program. In the context of extended logic programs, a program may become inconsistent in the presence of explicit negation. However, classical logic programming framework is useless in the presence of inconsistent information, and paraconsistent extensions are needed in order to treat inconsistent information properly in a program. In Chapter 6, we then proposed paraconsistent frameworks for disjunctive logic programs which can cope with inconsistent information as well as indefinite information in a program. The paraconsistent stable and possible model semantics were introduced for extended disjunctive programs, and those semantics were shown to be useful compared with the answer set semantics.

SUMMARY AND CONTRIBUTIONS

Each paraconsistent semantics was also characterized by the fixpoint semantics of extended disjunctive programs, and various methods for reasoning with inconsistent information were discussed.

Disjunctive logic programs increase expressive power of logic programming, while their computation are generally more expensive than normal logic programs. Then, in Chapter 7, we developed partial deduction techniques for optimizing disjunctive logic programs. We first argued that normal partial deduction for logic programs is not directly applicable to disjunctive logic programs, then introduced a new partial deduction method for disjunctive logic programs. The proposed disjunctive partial deduction was shown to preserve the minimal model semantics of positive disjunctive programs and the disjunctive stable models semantics of normal disjunctive programs. We also showed a method of translating disjunctive partial deduction into normal partial deduction, and presented a method of using normal partial deduction for the possible model semantics. An application to goal-oriented partial deduction is also presented for query optimization in disjunctive deductive databases.

In this dissertation, we have discussed semantic issues for various extensions of logic programming. Interrelations between those extensions are presented in Figure 8.1.

Now we know the following relationships between each class of programs.

- Normal disjunctive programs are reducible to positive disjunctive program by the epistemic transformation presented in Section 3.4. This transformation preserves both the disjunctive stable model semantics and the possible model semantics.
- Normal disjunctive programs are reducible to normal logic programs preserving the possible model semantics by the pm-transformation presented in Section 3.6. By contrast, under the disjunctive stable model semantics, such a transformation is most unlikely possible in polynomial time.
- Abductive logic programs are transferable to normal disjunctive programs and vice versa under the possible model semantics. Also abductive disjunctive programs are reducible to normal disjunctive programs under the disjunctive stable model semantics and the possible model semantics (Chapter 5).
- Extended disjunctive programs and extended logic programs are respectively reducible to normal disjunctive programs and normal logic programs by considering their positive forms. Each semantics for extended programs is expressed by the corresponding semantics of normal programs (Chapter 6).
8.2 Future Research

In this dissertation, we have discussed various extensions of logic programming and their theoretical frameworks. Syntactically speaking, further extensions besides those presented in the dissertation are considered. One extension is incorporating classical negation in abductive logic/disjunctive programs. Abductive extended logic programs and abductive extended disjunctive programs are extensions of abductive logic programs and abductive disjunctive programs respectively, which are obtained by replacing normal logic/disjunctive programs with extended logic/disjunctive programs in each abductive framework. However, such frameworks are reducible to normal abductive logic/disjunctive programs by considering their positive forms, so it is easy to extend the results presented in Chapter 5 to abductive extended logic/disjunctive programs. Inoue and Sakama [1993] have introduced a fixpoint semantics of abductive extended disjunctive programs and shown that abductive extended disjunctive programs are transferable to extended disjunctive programs under the answer set semantics.

Another extension is to incorporate default negation in the head as well as in the body of each clause in extended disjunctive programs. Such an extension of logic programming is known as logic programs with "positive not". A unique feature of such programs is that answer sets of those programs are not necessarily minimal, however, its application to knowledge representation was remained open [Lifschitz and Woo, 1992]. Inoue and Sakama [1994] have recently shown that such non-minimal answer sets are useful to characterize abductive reasoning and inclusive disjunctions in logic programming. Moreover, they show that the possible model semantics plays an important role for such characterizations. This extension of logic programming is fairly new, and further applications of such programs are remained to be seen. However, since the framework is closely related to the possible model semantics, further investigation would also exploit new applications of the possible model semantics.

Considering the relationship to nonmonotonic reasoning, we have shown in Chapter 4 that disjunctive logic programs are translated into nonmonotonic formalisms in AI. Since abductive logic/disjunctive programs are transferable to disjunctive logic programs as presented in Chapter 5, by combining these two transformations, we can also relate abductive logic programming to each nonmonotonic formalism. From the computational complexity viewpoint, it is known that disjunctive stable model semantics and those nonmonotonic formalisms are interrelated at the second level of the polynomial hierarchy [Eiter and Gottlob, 1993b]. This fact implies that reasoning tasks in each nonmonotonic formalism can also be efficiently realized in disjunctive
logic programs. In other words, disjunctive logic programs have potential applications for nonmonotonic reasoning in AI.

On the other hand, we have shown that non-minimal nature of autoepistemic logic is useful to characterize inclusive disjunctions in knowledge representation. It is shown in [Inoue and Sakama, 1994] that such non-minimal autoepistemic expansions are also useful for characterizing abduction. These facts suggest that the principle of minimality is not a dominant rule for commonsense reasoning in AI any more. So it is interesting to seek further applications of non-minimal nonmonotonic reasoning, which would open new perspectives of nonmonotonic formalisms as knowledge representation tools.

As for the procedural aspects of disjunctive logic programs, in Section 3.5 we have presented a bottom-up proof procedure which is sound and complete with respect to the possible model semantics/disjunctive stable model semantics for function-free range-restricted normal disjunctive programs. However, the proposed bottom-up proof procedure is rather naive for query-answering, and in the presence of huge databases, it might need some optimization techniques as presented in [Bancilhon and Ramakrishnan, 1988]. For instance, partial deduction presented in Chapter 7 is one of such optimization techniques, and it would be used to reduce search space for a given query. In this dissertation, we have not presented a top-down proof procedure for disjunctive logic programs since it is generally inefficient. However, as discussed in Section 3.7, concerning the possible model semantics a top-down proof procedure is realized using a proof procedure for the stable model semantics in pre-transformed normal logic programs.

From the standpoint of deductive databases, integrity checking and view update in disjunctive logic programs are also important. We have considered the meaning of a disjunctive logic program as the collection of all possible models satisfying integrity constraints. Such selection is done during the bottom-up computation of possible models and any model violating integrity constraints is pruned away. Thus, if the procedure generates no possible model, we know that a database violates integrity constraints. Note that we consider integrity constraints in the form of negative clauses, while more general form of integrity constraints are transformed into negative clauses as presented in [Sadri and Kowalski, 1988].

There are difficulties in view update in disjunctive deductive databases. Without disjunctive information, view update is usually achieved by translating an update request on a virtual relation into a real update on the underlying database relations. For example, in a database containing the clause $p(x) \leftarrow q(x)$ where $p$ is a virtual relation and $q$ is a real database relation, the addition of the virtual fact $p(a)$ is translated into the addition of the real fact $q(a)$. Then $p(a)$ is implied in the updated database. However, given the disjunctive clause $p(x) \lor r(x) \leftarrow q(x)$, it is not so easy to translate the addition of $p(a)$ into the real update on $q(x)$. This is because the addition of $q(a)$ in this case does not necessarily imply the truth of $p(a)$. Several problems arise in updating indefinite knowledge bases and further investigation is needed.

Last but not least, practical applications of disjunctive logic programming are also very important. In this dissertation, we are mainly concerned with the theoretical aspects of disjunctive logic programming, however, we consider that disjunctive logic programming has promising applications due to its rich expressiveness. For instance, in Section 5 we have presented an equivalence relationship between disjunctive and abductive logic programs. This fact implies that applications of abductive logic programming, such as diagnosis and planning, can also become applications of disjunctive logic programming. On the other hand, recent studies in artificial intelligence recognize the need of standardized knowledge representation for developing reusable and sharable knowledge bases [Neches et al., 1991]. Such technologies are important since they would greatly reduce the cost of designing and maintaining knowledge bases. Of course it might be difficult to develop a single common language which is useful for different kinds of multi-purpose knowledge bases. However, logic programming has potential possibilities to serve as an archetype of the common language because it provides a universal framework based on mathematical logic. Moreover, since logic programming has nice relations with databases and nonmonotonic reasoning, it enables us to develop knowledge bases combining existing database and AI technologies.

Thus we believe that logic programming and disjunctive logic programming play important roles in knowledge representation and artificial intelligence in the next generation, and we hope that the studies presented in this dissertation will contribute to further development of research in the fields.
List of Publications


LIST OF PUBLICATIONS


Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


[BIBLIOGRAPHY]


BIBLIOGRAPHY


BIBLIOGRAPHY


