Negation in Logic and Deductive Databases

Xuegang Wang

School of Computer Studies

The University of Leeds

September 1999

Submitted in accordance with the requirements

for the degree of Doctor of Philosophy.

In memory of my father, Caibing Wang,

and

to my mother, Shixiu Ding.

Abstract

This thesis studies negation in logic and deductive databases. Among other things, two kinds of negation are discussed in detail: strong negation and nonmonotonic negation.

In the logic part, we have constructed a first-order logic **CF**$'$ of strong negation with bounded quantifiers. The logic is based on constructive logics, in particular, Thomason's logic **CF**. However, unlike constructive logic, quantifiers in our system as in Thomason's are static rather than dynamic. For the logic **CF**$'$, the usual Kripke formal semantics is defined but based on situations instead of conventional possible worlds. A sound and complete axiomatic system of **CF**$'$ is established based on the axiomatic systems of constructive logics with strong negation and Thomason's completeness proof techniques. **CF**$'$ is proposed as the underlying logic for situation theory. Thus the connection between **CF**$'$ and infon logic is briefly discussed.

In the database part, based on the study of some main existing semantic theories for logic programs with nonmonotonic negation, we have defined a novel semantics of logic programs called quasi-stable semantics. An important observation is that a nonmonotonic negation such as is required for logic programs should be computed by a nonmonotonic, revision process. Only a process that allows one to withdraw by revising provisionally held negative information can hope to be adequate to model a non-monotonic negation. In light of this, we propose a model of negation that owes much to the stable semantics but allows, through a mechanism of consistency-recovery, for just this withdrawal of previously assumed negative information. It has been proved that our new semantics maintains the desired features of both the well-founded semantics and the stable model semantics while overcoming their shortcomings. In addition, the quasi-stable semantics has been generalised to logic programs with both strong negation and nonmonotonic negation, giving rise to the quasi-answer set semantics.

Acknowledgments

I should like to thank my supervisor, Peter Mott for his direction, supervision, criticism and encouragement throughout the whole process of my study. Without his valued help, I would have not studied at Leeds, let alone finish the thesis. Thanks also to my advisor, Professor Tony Cohn for his encouragement and instructive advice.

There have been many other people giving me various kinds of timely help in one way or another in my life. I should like to thank them too.

Finally I wish thank and acknowledge funding from an Overseas Research Students (ORS) Award (#ORS-95023007), a Tetley & Lupton Scholarship, a Leeds University Foundation Scholarship, and a departmental grant.

Contents

Chapter 1

Introduction

The notion of negation is basic not only to any formal or informal logic system (see [43]) but also to logic programming and deductive databases (see [87]). Historically, there were serious objections against the use of negation in intuitionistic mathematics by Griss (see [56] and related citations there). However, it was argued by Heyting [56] that, without negation, there would be no calculus of propositions because only true propositions make sense, and thus the logic of negationless mathematics would be difficult to formalise. In the contexts of logic programming and deductive databases, logic programs without negation would have a limited expressive power (see [1]). Thus we take it for granted that the use of negation is indispensable. Our task is to have a good understanding of negation from the logical and pragmatic viewpoints. In this introductory chapter, we have a brief informal look at various kinds of negation in different logics, in logic programming, and in deductive databases. We assume basic familiarity with logic, and logic programming.

1.1 Negation in Logic

1.1.1 Negation in Classical Logic

In logic, the most important form of negation is probably the negation in classical first-order logic, called *classical negation*. The importance of classical negation derives from the importance of classical first-order logic, which has been widely applied to various subjects, including mathematics and its foundations, physics, computer science, etc.

Informally, classical negation as a unary operator expresses something like "*it is not the case that ...* " in natural language. A negative sentence $\neg p$ is true whenever p is false, and false whenever p is true.

It is worth noting that the above simple form of classical negation is based on the assumption that any sentence is either true or false, which in turn is connected with the two basic assumptions implicitly made in the semantics of first-order logic. One is that every name in a first-order language must refer to an object in the domain of quantification, that is there are no *denotation failures*. We call this the *denotation assumption*. The other is that all predicates $p(x)$ are completely defined in the sense that for any individual a, $p(a)$ is either true or false. In other words, models used in first-order logic are assumed to represent a complete scenario of states of the whole world. We shall call the second assumption the *completeness assumption*. Under these two assumptions, first-order logic is semantically two-valued. A first-order sentence is either true or false. On this the bivalence is built the notion of classical negation: the negation of a true sentence is false, and the negation of a false sentence is true. An important feature of classical negation is that it satisfies the principle of the excluded third and the principle of the double negation.

The two semantic assumptions, among other things, have been criticised from different perspectives. Free logic and partial logic have resulted by dropping one or both of the assumptions respectively (see [14] and [18]). The denotation assumption is somewhat philosophical. From the database viewpoint, we need not worry about it. By restricting interpretations to Herbrand interpretations, this assumption will be automatically satisfied. As to the completeness assumption, we shall argue that it is not necessary to give up the simple bivalence of classical first-order logic if we drop the assumption. We shall show this in the next chapter.

1.1.2 Negation in Intuitionistic Logic

The use of classical logic, though spread wide, is not without any problems even in mathematics itself. We mentioned in the previous section that criticism of the semantic assumptions of first-order logic resulted in two non-classical logics: free logic and partial logic. The principle of the excluded middle is criticised from the Brouwerian constructive approach to mathematics, known as intuitionism[1].

One of basic tenets behind intuitionism is that any mathematical proposition must have a mathematical construction with certain given properties. A proposition p is asserted if we have a mathematical construction of the proposition; and a negative proposition $\sim p$ can be asserted if and only if there is a construction which from the assumption that a construction for p were carried out leads to a contradiction. Furthermore, all other logical connectives and quantifiers also need a constructive meaning. As a result, the principle of the excluded middle cannot be acceptable from the constructive viewpoint (see [56], pp. 97-99, [13], pp. 404-408, and [100], pp. 8-11).

Interestingly, through the work of Gentzen [50], it is known that the natural deduction system for intuitionistic logic can be obtained by removing just the principle of the excluded middle from the natural deduction system for classical logic.

[1]For an introduction to intuitionism, see [56], and for a concise but more comprehensive survey to various principal constructivist schools, see [100].

The resulting negation, which we shall call *intuitionistic negation*, is stronger than classical negation.

Along with the principle of the excluded middle, the other classical principles of double negation, de Morgan, etc have also to be partially dismissed from intuitionistic logic. For example, it is known that intuitionistic logic has the disjunction property (see [102]):

$$p \vee q \text{ is provable iff } p \text{ is provable or } q \text{ is provable.}$$

But the following does not hold for intuitionistic logic:

$$\neg(p \wedge q) \text{ is provable iff } \neg p \text{ is provable or } \neg q \text{ is provable,}$$

which seems necessary for a truly constructive negation. Because of this unsymmetric characteristic, we may say that intuitionistic negation does not completely adhere to constructivity despite its constructive demand for negatively asserted mathematical propositions. This shortcoming and other undesirable features of intuitionistic negation have been overcome by another kind of negation, called strong negation which we discuss in the next subsection.

1.1.3 Negation in Constructive Logics

In the logic community, *strong negation* was introduced by Nelson [76], based on a distinction between two proof methods for the negation of a sentence: a sentence can be refuted either by *reductio ad absurdum* or by construction of a counter-example. Essentially, strong negation expresses the notion of directly established falsity. Independently, Markov [68] also introduced strong negation from the point of view of constructive logic.

Strong negation was later incorporated into various logical systems, Nelson's propositional systems $\mathbf{N}$ and $\mathbf{N}_1$, that is, the propositional parts of Nelson's system

$\mathbf{N}_1$ of constructible falsity (see [76], and Routley [94]), and their first-order extensions are initial examples (see Almukdad and Nelson [4][2]). Two similar systems **F** and **G**, or equivalently, **HF** and **HG** (using Routley's notation), have been studied by Fitch [39]. For the difference between Fitch's systems and Nelson's systems, see [94], and see also Thomason's footnote on page 255 of [98]. There are also an intuitionistic logic with strong negation $\overline{\mathbf{H}}$ by Gurevich [54], constructive predicate logic with strong negation **S** by Akama [2], and first-order logic **CF** by Thomason [98], constructive propositional calculus with strong negation by Vorob'ev [107] and the semantics of the calculus in terms of $\mathcal{N}$-lattices in Rasiowa [89]. Furthermore, Wansing [114] has systematically investigated the whole family of substructural subsystems of Nelson's systems from the point of view of the fine-structure of information processing.[3]

We shall refer to the resulting logics loosely as constructive logics with strong negation. In comparison to intuitionistic logic, these logic systems demonstrate several satisfying features, including symmetry, and partiality. See Chapter 3 for details. Constructive logics have another important feature – persistence, which is also shared by intuitionistic logic but not by classical first-order logic. We shall discuss this feature in Chapter 2 and Chapter 3.

Compared with intuitionistic negation, it turns out that strong negation is simpler and more straightforward. Moreover, the notion of directly established falsity expressed by strong negation lends itself to the explicit expression of negative information in logic programming and deductive databases. We shall discuss this utility of strong negation in the context of logic programming in Chapter 7.

[2]In [4], Almukdad and Nelson use **N** and $\mathbf{N}^-$ for their first-order systems, where $\mathbf{N}^-$ is the proper subsystem of **N** without the axiom schema $\varphi \supset (\sim \varphi \supset \psi)$.

[3]Wansing uses $\mathbf{N}^-$ and **N** instead of **N** and $\mathbf{N}_1$ respectively. $\mathbf{N}^-$ and **N** are formulated in symmetrical sequent calculus (see pp. 24-25 of [114]).

1.2 Situation Theory and Negation

Recently, classical first-order logic has also been challenged by situation theorists (see [10] and [32]). The basic insight of situation theory is that cognitive activities such as thought, speech, communication, and inference are all situated relative to a situation or context. Situations are introduced to model limited portions of the world in which agents carry out their activities. As information sources, situations can be seen as concise representations of various kinds of context sequences in possible world semantics. Another related novel characteristic of situation theory is its emphasis on information content. More generally, situation theory concerns the development of a general theory of meaning and information, in particular, the development of an information-based theory of inference.

In [10], Barwise and Etchemendy put forward a model of information content called infon algebra in order to develop an information-based theory of inference. An infon algebra $\mathcal{I} = \langle Sit, I, \Rightarrow, \models\rangle$ consists of a non-empty collection Sit of situations, a distributive lattice $\langle I, \models\rangle$ on infons, together with the makes-factual or support relation $\models$ between situations and infons satisfying certain additional conditions.

In an infon algebra $\mathcal{I}$, infons represent pieces of information, and situations are intended to be limited portions of the world. Thus situations provide us with a kind of incomplete information; that is to say, the support relation $\models$ is essentially partial: a situation may support some infons and refute others but remain silent on many. It follows that any algebraic theory of infons is definitely not Boolean. Furthermore, they argue that the situation-theoretic model of infons is at least a complete distributive lattice, that is Heyting algebra. Therefore, they conclude that algebraic structure of infons satisfies the axioms for Heyting algebra but not all the axioms for a Boolean algebra. Thus, the logic for situation theory is at least intuitionistic but not classical.

This argument immediately poses at least two questions. One is about negation, the other about the interpretation of quantifiers. Here we only consider the question about negation, leaving the other one to Chapter 3.

We recall that in situation theory there are two kinds of basic infons: one is $\ll R, a_1, a_2, ..., a_n; 1 \gg$, the other $\ll R, a_1, a_2, ..., a_n; 0 \gg$, where R is an n-place relation, $a_1, a_2, ..., a_n$ are objects with a restriction of appropriateness. Note that $a_1, a_2, ..., a_n$ need not necessarily be individuals. 0 and 1 are the polarity of infons. For basic infons, negation is defined through a dual operation as follows:

$$\overline{\ll R, a_1, a_2, ..., a_n; 1 \gg} = \ll R, a_1, a_2, ..., a_n; 0 \gg \tag{1.1}$$

$$\overline{\ll R, a_1, a_2, ..., a_n; 0 \gg} = \ll R, a_1, a_2, ..., a_n; 1 \gg \tag{1.2}$$

So, we have

$$\overline{\overline{\ll R, a_1, a_2, ..., a_n; 1 \gg}} = \ll R, a_1, a_2, ..., a_n; 1 \gg \tag{1.3}$$

$$\overline{\overline{\overline{\ll R, a_1, a_2, ..., a_n; 1 \gg}}} = \overline{\ll R, a_1, a_2, ..., a_n; 1 \gg} \tag{1.4}$$

However, it is well-known that intuitionistic negation does not satisfy (1.3) though it satisfies (1.4).

Furthermore, the negation of compound infons in situation theory is defined by the following version of de Morgan's laws (see Barwise [8], p. 235, and Fernando [38], p. 108). Even in [10], p. 55, Barwise and Etchemendy do mention that (1.5) is sometimes assumed in situation theory. However, (1.5) does not hold though (1.6) does for intuitionistic negation.

$$\overline{\sigma \wedge \tau} = \overline{\sigma} \vee \overline{\tau} \tag{1.5}$$

$$\overline{\sigma \vee \tau} = \overline{\sigma} \wedge \overline{\tau} \tag{1.6}$$

Therefore, we conclude that the situation-theoretic negation is not intuitionistic. Moreover, the above way of treating negation by situation theorists to some extent suggests that the negation used in situation theory is in fact strong negation. More importantly, it turns out that intuitionistic negation can in fact be simulated by strong negation (see [68] and [2]).

In light of this, we are inclined to see constructive logic with strong negation as the underlying logic for situation theory rather than intuitionistic logic.

We shall argue in Chapter 3 that the intuitionistic interpretation of universal quantifiers is not satisfying either from the situation-theoretic viewpoint. In fact, universal quantifiers used in related situation theory literature are classical as in the first-order logic. This motivates the construction of a logic with strong negation and classical quantifiers. See Chapter 3 for details.

1.3 Negation in Deductive Databases

The notion of negation in logic programming and deductive databases is more complicated and involved than that in logic. This as we shall see later is mainly because logical negation has proved unsuitable to application domains where the form of reasoning is usually nonmonotonic. A different kind of negation has been introduced through semantics associated with a logic program. Roughly speaking, the main idea behind is that a piece of negative information is implicitly or "by default" assumed whenever there is no sufficient evidence to the contrary, where the exact meaning of "sufficient evidence" depends on what semantics is associated with a logic program. Whatever semantics is used, a common characteristic of this new form of negation is nonmonotonicity, that is a piece of assumed-by-default negative information could be given up later when more information is available. So, we may as well call this kind of negation nonmonotonic negation[4].

Two initial examples of nonmonotonic negation developed in logic programming languages are the *THNOT* construct in PLANNER [55] and the *not* operator in PROLOG (see [25] and its citations). Both of them express something which was later termed by Clark [25] as the *negation as failure* (NAF) inference rule. It states to the effect that a negative conclusion *not* p is inferred if any search for a proof of p fails *finitely*. Here, the finiteness condition is indispensable unless a system has some mechanism for detecting an infinite process. So the rule is sometimes also called the *negation-as-finite-failure rule.*

First formalisations of nonmonotonic negation include Reiter's *closed world assumption* (CWA) [90] and Clark's program completion semantics [25]. The CWA is essentially an inference rule, stating that if a ground atom p is not a logical consequence of a program, then infer *not* p. The completion semantics is introduced with the aim of reducing the negation-as-finite-failure rule to a derived rule of first-order logic so as to justify the use of the rule.

In Chapter 5 we shall show that both the CWA and the completion semantics have serious shortcomings. They can only be applied to a limited class of logic programs without causing any inconsistency or other problems. In the past two decades, various kinds of more fine-grained semantics for logic programs have been proposed, including the Fitting semantics [40], the stratified semantics [21], the stable semantics [47], the default semantics [16], and the well-founded semantics [105]. These semantical theories to a great extent draw on *nonmonotonic logic* developed in early 1980's[5]. For example, the default semantics is based on default

[4]In the literature, it is also called *negation as failure* [25], *negation by default* [16] or simply *negation* when there is only one kind of negation is involved [1]. There is even a further classification of this kind of negation according to specific semantics used [73]. We prefer to use nonmonotonic negation because *nonmonotonicity* captures the essential characteristic of negation in question, and is more general in comparison to other terms.

[5]The history of nonmonotonic logic traces back to the work by McCarthy [69] in the late 1950's. The initial formal theories of nonmonotonic logic were collected in an issue of the Artificial Intelligence Journal (1980), devoted exclusively to nonmonotonic reasoning. An important reference

logic [91], and the stable semantics has its root in autoepistemic logic [74].

In logic programming and deductive databases, the stable model semantics and the WFS have become two dominant semantical theories. We shall see in Chapter 5 that they are more fine-grained than the CWA and the NAF. The introduction of these two semantics is an important step towards understanding nonmonotonic negation. Nevertheless, it can be argued that they are still not fully satisfying. In particular, we shall argue that the underlying definition of stable models in the stable model semantics is to some extent inconsistent with the essential characteristics of nonmonotonic negation used in logic programs. Based on the WFS and hypothetical reasoning, we shall propose a novel semantics in Chapter 6. It turns out that the newly introduced semantics is very similar to the stable model semantics but avoids its shortcomings. So we call it the *quasi-stable semantics*.

1.4 Outline of the Thesis

In addition to this introductory chapter, the thesis includes a further seven chapters. We summarise the contents of these chapters as follows.

Chapter 2: Negation in first-order logic. We begin with the analysis of classical negation at the semantic and proof-theoretic levels, showing that classical negation at the semantic level is entirely different from the negation at the proof-theoretic level. Given the discrepancy, we had better divide classical negation into two notions: one is the semantical notion of classical negation, and the other proof-theoretical one. We also show that, more generally, there exists a sharp difference in first-order logic between the semantical and proof-theoretic levels. Although first-order logic is proof-theoretically monotonic, nevertheless, it is non-persistent at the semantical level. The monotonic inference of first-order logic is not suitable for

book on the subject is *Nonmonotonic Logic* by Marek and Truszczynski [67]. For a survey of nonmonotonic logic, see [72].

the retrieval of negative information even in the context of relational databases. Instead it is the semantic notion of classical negation that is used in relational databases. These arguments will be used in Chapter 4 to facilitate understanding of the relationships among various different kinds of negation, and to enable us to see what is needed to be done if we are going to use the deductive approach to databases.

Chapter 3: First-order Logic $\mathbf{CF}'$ We study the logic of strong negation. A first-order logic $\mathbf{CF}'$ with strong negation and bounded quantifiers is constructed. The logic is based on constructive logics, in particular, Thomason's logic **CF**. However, unlike constructive logic, quantifiers in our system as in Thomason's are static rather than dynamic. The usual Kripke formal semantics is defined for the logic $\mathbf{CF}'$, based on situations instead of conventional possible worlds. A sound and complete axiomatic system of $\mathbf{CF}'$ is established based on the axiomatic systems of constructive logics with strong negation and Thomason's completeness proof techniques. $\mathbf{CF}'$ is proposed as the underlying logic for situation theory. Thus the connection between $\mathbf{CF}'$ and infon logic is briefly discussed. The work on the logic $\mathbf{CF}'$ has been previously published in [112].

Chapter 4: From Relational to Deductive Databases In this chapter, we first consider the limited expressive power of the conventional relational model of databases and briefly discuss how the deductive approach to databases gives a straightforward solution to the expressive problem of the relational model. It is then followed by the formal definitions of logic program and deductive database. Finally, we give some arguments to show why negation is one of central problems in the deductive approach.

Chapter 5: Nonmonotonic Negation. This chapter is to review some main semantic theories for logic programs, including the semantics of definite logic programs mainly by van Emden and Kowalski [103], Clark's program completion semantics and the associated SLDNF[25], Reiter's closed world assumption [90], Fitting's semantics [40], the well-founded semantics by Gelder, Ross and Schlipf [105], stable semantics by Gelfond and Lifschitz [47], and a three-valued version of the stable model semantics by Przymusinski [85]. There is no doubt that these semantic theories give us instructive insights into understanding nonmonotonic negation. Nevertheless, it can be argued that they all to some extent have shortcomings of one sort or another. We shall argue that the stable model semantics and its three-valued version are conceptually flawed, based on the observation that the stable model semantics has no mechanism whatsoever for revising provisionally assumed negative information in light of newly discovered information while it aims to model an essentially nonmonotonic negation. The critical argument on the stable model semantics has been previously reported in [113].

Chapter 6: Quasi-stable Semantics. Based on the study of some main existing semantic theories, we proposed a novel semantics for logic programs, called quasi-stable semantics. An important characteristic of the quasi-stable semantics is the introduction of a mechanism of consistency-recovery. As a result, the quasi-stable semantics is able to revise tentatively held negative information whenever conflicts occur. It has been proved that (i) every logic program has at least one quasi-stable extension, (ii) a quasi-stable extension of a logic program $\mathbb{P}$ is a total model of $\mathbb{P}$, (iii) a quasi-stable extension of $\mathbb{P}$ is minimal in the sense that no positive literal can be replaced in the extension by a negative one without its ceasing to be a model of $\mathbb{P}$, (iv) the well-founded partial model for a logic program $\mathbb{P}$ is included in every quasi-stable extension of $\mathbb{P}$, (v) every stable model is a quasi-stable extension. The work in this chapter has been previously reported in [111].

Chapter 7: Quasi-stable Semantics with Strong Negation. Normal logic programs are extended with strong negation, resulting in logic programs with both nonmonotonic negation and strong negation. Such logic programs are called *extended* *logic programs*. The dominant Semantics for extended logic programs is the so-called answer set semantics [48], which is derived from the stable model semantics [47]. However, the stable model semantics is problematic as we shall show in Chapter 5, we instead define a semantics for extended logic programs, based on our quasi-stable semantics.

Chapter 8: Conclusions and Future Work Finally we give a brief summary about the our research on both strong negation and nonmonotonic negation, and discuss some potential extensions of the logic $\mathbf{CF}'$ and related issues about the quasi-stable semantics.

Chapter 2

Negation in First-order Logic

In this chapter, we are going to analyse the notion of negation at the semantic and proof-theoretic levels. The analysis will reveal that classical negation at the semantic level is entirely different from the negation at the proof-theoretic level. Given the difference, we had better divide classical negation into two notions: the semantical notion of classical negation, and the proof-theoretical one. More generally, the analysis will show that, partly as a consequence of the difference, there exists a sharp discrepancy in first order logic between the semantical and proof-theoretic levels.

The main purposes of the analysis are to facilitate the understanding of relationship among various different kinds of negation, and to enable us to see what needs to be done if we are going to use the deductive approach to databases.

2.1 Two levels of first-order logic

It is well-known that first-order predicate calculus is semantically complete, as shown by Gödel's semantic completeness theorem [51]. That is to say logical consequence in first-order logic has a neat correspondence with theoremhood: a sentence is a theorem of a theory if and only if it is the logical consequence of the theory.

But logical consequence is a very strong notion. We have to consider all possible models. For a formula φ is a logical consequence of a theory T if and only if whenever T is true of a model, so is φ. Here, specific semantic elements are abstracted away. In practical applications such as the field of databases, however, we are usually more concerned with what is true of specific databases than what is true of all the possible databases. The above neat correspondence given by the Gödel completeness theorem no longer holds when specific semantic elements come into focus. Instead, what we see is a sharp discrepancy between the semantic level and the proof-theoretic level in first-order logic:

(T1) The meaning of negation at the semantic level is different from that at the proof-theoretic one.

(T2) Semantically, it is two-valued: a sentence is either true or false in a given model. In contrast, any formal system of first-order logic is "three-valued" in the sense that a sentence may be provable or refutable or neither from a given theory relative to the system.

(T3) Semantically, first-order logic is not persistent whereas proof-theoretically it is monotonic[1].

In order to appreciate (T1), I shall show that in first-order logic, negation expresses a kind of "failure" at the semantic level whereas it expresses inconsistency in the formal system. The notion of negation as failure is much weaker than that of inconsistency. It is from this difference that (T2) follows as we shall show below[2]. It also partially contributes to (T3). We mentioned in Chapter 1 that negation as

[1]See Section 2.4 for the definition of persistence and monotonicity.

[2]Another factor contributing to (T2) is due to classical universal quantifiers. But we only consider negation here.

failure was originally introduced by Clark in [25] as a procedural rule for deriving negative facts in logic programming. We shall come back to this topic in Chapter 5 when we review some main semantic theories of logic programs. Here, by an abuse of the phrase, I am using it for negation as failure to support.

The above difference is reminiscent of another famous theorem by Gödel, usually called the First Incompleteness Theorem (see the related historical remarks in [13]). It says to the effect that "in a formal system satisfying certain precise conditions there is an undecidable proposition, that is, a proposition such that neither the proposition itself nor its negation is provable in the system." (cited from the editor's comment [52], p. 592). The incompleteness theorem also highlights the gulf between the semantic level and the theorem-proving level of first-order logic. But our emphasis here is that there exists crucial difference even at the conceptual level.

2.2 Failure vs Inconsistency

Fix a first-order language $\mathcal{L}$. Let $\models$ be the semantic satisfaction or support relation between interpretations of $\mathcal{L}$ and $\mathcal{L}$-sentences of first-order logic, and $\vdash$ the deduction relation between sets of $\mathcal{L}$-sentences and $\mathcal{L}$-sentences. An interpretation as usual consists of some domain of discourse over which all constant and predicate symbols of $\mathcal{L}$ are assigned a meaning. In first-order logic an interpretation is artificial, and arbitrary in the sense that its domain can be any set of objects, and all constant symbols can be assigned any elements of the domain. In the database context, we need not consider arbitrary interpretations. Instead, much more restricted interpretations called *Herbrand interpretations*, are used. In a Herbrand interpretation, the domain of discourse consists of the constant symbols of $\mathcal{L}$, and the constant symbols are interpreted *literally*.

Usually, a Herbrand interpretation is identified with a set of positive atomic sentences [64]. An outstanding characteristic of a Herbrand interpretation is its double status: it can be taken as a theory as well as a particular model. When restricted to finite domains, a Herbrand model can also be taken as a relational database, and vice versa in a straightforward way. We shall make use of the double status characteristic of Herbrand interpretations in the following discussion.

2.2.1 Negation as Failure to Support

At the semantic level, the meaning of negation is determined by the well-known Tarski truth definition (see [13]) as follows[3]:

(M1) $I \models \neg\varphi$ if and only if $I \not\models \varphi$.

where I is a Herbrand interpretation, φ is a sentence. When $I \models \varphi$, we say that I supports φ, and when $I \not\models \varphi$, we say that I fails to support φ. According to M1, we may say that the meaning of negation at the model-theoretic level is *failure to support*: $\neg\varphi$ holds relative to I if and only if I fails to support φ.

For an atomic sentence $R(a_1, a_2, ..., a_n)$, I supports $R(a_1, a_2, ..., a_n)$ if and only if $R(a_1, a_2, ..., a_n)$ is in I by the Tarski's truth definition again and the definition of Herbrand interpretation (see [64]). It then follows that I supports $\neg R(a_1, a_2, ..., a_n)$ if and only if $I \not\models R(a_1, a_2, ..., a_n)$ if and only if $R(a_1, a_2, ..., a_n)$ is not in I. So we have

(M2) $I \models \neg R(a_1, a_2, ..., a_n)$ if and only if $R(a_1, a_2, ..., a_n)$ is not in I.

Since $R(a_1, a_2, ..., a_n)$ is either in I or not in I but cannot be in both, we have I either supports $R(a_1, a_2, ..., a_n)$ or supports $\neg R(a_1, a_2, ..., a_n)$, but cannot support both. In general, M2 does not hold for any sentence. However, it can be shown that for any sentence φ,

[3]Our notation is slightly different from that used in [13] though.

(M3) $I \models \varphi$ or $I \models \neg\varphi$.

(M4) Not both $I \models \varphi$ and $I \models \neg\varphi$.

The proof of M3 can be obtained easily from M1, which at bottom is based on M2. To show M4, an induction on the complexity of φ is enough.

The above is a basic scenario about negation from the semantic viewpoint. We shall refer to negation characterised by M1 - M4 as the semantic notion of classical negation. In particular, we shall take M2 as a semantic inference rule for inferring negative information given the form of M2. We now turn to the formal system to see how classical negation is dealt with proof-theoretically, and see what deductive inference mechanism is available for inferring negative information.

2.2.2 Negation as Inconsistency

There are many formal systems of first-order logic, including Hilbert-style axiom system, the tableau method (see [13] and relevant historical and bibliographical remarks there), and Gentzen's natural deduction system [50]. They have all been proved equivalent. So, without losing any generality, we consider Gentzen's calculus NK [50] for simplicity. In this system, rules relevant to negation in one way or another are listed as follows, where $\perp$ stands for the false proposition.

$$(R1)\quad \begin{array}{c}\varphi\\ \vdots\\ \underline{\perp}\\ \neg\varphi\end{array} \qquad\qquad (R2)\quad \frac{\perp}{\varphi}$$

$$(R3)\quad \varphi \vee \neg\varphi \qquad\qquad (R4)\quad \frac{\varphi, \neg\varphi}{\perp}$$

We have seen that negation at the semantic level is negation as failure to support.

Given the above rules about negation, what does classical negation mean at the theorem-proving level? Have these rules fully characterised the semantic notion of classical negation? Let us begin with the first question.

According to R1, only when the given set ? of sentences is not consistent with $R(a_1, a_2, ..., a_n)$ can we conclude that $\neg R(a_1, a_2, ..., a_n)$ from ?. In fact, it can be proved that, in a natural deduction system,

$$? \vdash \neg\varphi \text{ if and only if } ?, \varphi \vdash \bot. \qquad (N)$$

So, we can derive $\neg\varphi$ from ? if and only if ? and φ are not consistent. If ? and φ are consistent, then we cannot deduce the negative conclusion $\neg\varphi$.

By (N) we may say that classical negation expresses *inconsistency* at the proof-theoretical level. This conclusion can be justified using a more formal argument given by Gabbay. In [43], he investigates how to characterise negation in a logical system. The basic idea in his definition of negation in a system is that $? \vdash \neg\varphi$ holds if and only if ? and φ together lead to some undesirable result, say a contradiction. So we may take (N) as defining what classical negation means at the theorem-proving level. For the detailed exposition, readers are invited to refer to [43].

Now we come to the second question about whether rules R1-R4 characterise the semantic notion of classical negation. It is not very difficult to show that rules R3 and R4 correspond with properties M3 and M4 of negation with semantic elements being abstracted away. To see this, we note that M3 follows from R3 in a straightforward way, and conversely, R3 also follows from M3 using the Gödel's semantic completeness theorem. A similar correspondence between M4 and R4 can be established using the same argument.

However, the correspondence stops there. Unlike M3 and M4, we find that M1 and M2 are not represented by the corresponding rules R1 and R2. Worse still, M1 and M2 in fact are not characterised by the whole deductive system of first-order logic.

Given a set T of atomic sentences and an atomic sentence $R(a_1, a_2, ..., a_n)$, if $R(a_1, a_2, ..., a_n)$ is not in T, then T as an Herbrand interpretation will fail to support $R(a_1, a_2, ..., a_n)$, and thus will support $\neg R(a_1, a_2, ..., a_n)$. However, T as a theory cannot be used to derive the negative fact at the proof-theoretical level since T is in fact consistent with $R(a_1, a_2, ..., a_n)$.

Therefore, we conclude that the formal system of first-order logic provides us with only a very weak inference mechanism for inferring negative information. The weak mechanism only partially reflects the semantic notion of negation. In particular, rules R1 - R4 together only partially characterise the semantic notion of negation, and it does not capture the semantic inference mechanism of M2 at all.

Before moving on to the next difference of first-order logic between the semantic and proof-theoretic levels, we give some comments about rules R1 and R2. It is natural to ask what role R1 and R2 play in the deductive system since that they do not characterise the corresponding semantic rules M1 and M2?

Although rule R1, according to its form, can be used to derive a negative conclusion, the resulting notion of negation is that of inconsistency as we pointed out before. In other words, its function in essential is to characterise classical negation as inconsistency at the proof-theoretical level, a notion stronger than that of failure to support.

The rule R2 is relevant to negation since φ may be any sentence. In particular φ may be a negative sentence. It says to the effect that, from absurdity, we can infer everything. In reality, it is certainly not the case. The discovery of contradictions, say in a database, does not enable us to derive everything but only signals that there is the violation of integrity constraints. So, it is hardly acceptable from the perspective of information retrieval. Moreover, we would not lose anything without the rule. Then, we may wonder why the rule is there in the deductive system of first-order logic? Briefly, it has a great deal to do with the semantic interpretation of material implication and the ideal requirement of semantic completeness. An implication proposition with the false antecedent is vacuously valid, and thus for the sake of semantic completeness, a similar rule is required for the proof of such implication propositions in a deductive system. So, the purpose of rule R2 is only to reflect the semantic constraint of material implication.

2.3 Two-valued vs Three-valued

2.3.1 Two-valuedness

At first glance, it seems that the two-valuedness of first-order logic at the semantic level is a straightforward consequence of the semantic property M3. However, in fact it is the other way around. That is to say, it is on two-valuedness that the simple semantic concept of classical negation is built. As to the two-valuedness itself, it is connected with the two basic assumptions as we mentioned in Chapter 1: the denotation assumption and the completeness assumption.

The denotation assumption originally comes from Frege's requirement of a logically perfect formal language [42]. The assumption alone is no more than an expedient to avoid unnecessary complications that may arise otherwise. Nevertheless, the assumption has the important ontological commitment that every singular term denotes an *existing object* when combined with Quine's famous dogma [86]: to be is to be a value of a bound variable, endowing the quantifiers with existential import. It is objection to these commitments that leads to the development of free logics (see [14]). Our concern here is not the related philosophical arguments but the use of logic in the database area. From the database viewpoint, it suffices to consider Herbrand interpretations. For such interpretations, denotation failures are automatically avoided since every constant symbol denotes itself. That is to say, as far as the application of logic to databases is concerned, it will not make any difference whether the assumption is made or not.

In contrast, the completeness assumption seems doubtful from the database viewpoint. Given a relational database, taken as a Herbrand interpretation, there is no reason to assume that the interpretation represents a complete scenario of states of the whole world. So, a relation r in the database is not necessarily complete; it may be *partial*, that is, for a certain individual a, it may be the case that neither $r(a)$ nor $\neg r(a)$ holds. Various logical theories have been developed to accommodate partiality, including partial logic and constructive logics. We shall consider how to use strong negation to deal with partiality in Chapter 3 and Chapter 7.

Nevertheless, the introduction of strong negation does not mean that we have to give up classical negation, in particular, its semantical component. Although strong negation as we shall see has proved very useful in some application domains, the semantical notion of classical negation also has its own advantages which is closely associated with the way in which negative information is represented. It is well-known that, in a relational database, only positive facts are explicitly expressed. A negative fact is implicitly assumed provided its positive counterpart is not explicitly available. The implicit representation certainly has one great advantage. As pointed out by Reiter [90], the number of negative facts about a given domain is in general much greater than the number of positive ones. So, it is not practical to include negative facts in a database explicitly.

How can we still have the semantical notion of classical negation without the completeness assumption? An alternative to the assumption is to borrow an idea from nonmonotonic logic. When a complete knowledge about a given domain is not available, we simply *jump* to the negative conclusion in any case whenever its positive counterpart is not present. The consequence is that negative information concluded in this way is defeasible. When more information is available, the conclusion may have to be given up. In other words, rule M2 is now taken as a *non-persistent* rule[4] rather than a semantic condition based on the completeness assumption. We shall give another argument to support our view on rule M2 in the next section.

Given a Herbrand model I and an atomic sentence φ, I supports either φ or not φ. If I supports φ then φ is true relative to I; otherwise we simply jump to the conclusion that φ is false relative to I even without the completeness assumption. So, any atomic sentence is either true or false relative to a Herbrand model. It follows that any sentence will also be either true or false relative a Herbrand model. As a result, we still have the simple semantical notion of classical negation while giving up the completeness assumption.

2.3.2 Three-valuedness

Now we consider the "three-valuedness" of first-order logic from the proof-theoretic viewpoint. We have seen that to infer the negation of a sentence φ from a theory T is equivalent to proving that φ is not consistent with T. From which it follows that to infer $\neg\neg\varphi$ from T is equivalent to proving that $\neg\varphi$ is not consistent with T. By the principle of double negation, we have that to infer φ from T if and only if $\neg\varphi$ is not consistent with T. So, given a theory, we can neither infer $\neg\varphi$ if φ is consistent with the theory, nor infer φ if $\neg\varphi$ is consistent with the theory. Since it is possible not only that a sentence is consistent with a theory but also that its negation is consistent with the theory, we conclude that a sentence may neither be proved nor refuted from a theory.

So, though it is always the case that either φ or $\neg\varphi$ is true relative to any model, it may be the case that we can prove neither of them. To prove φ (or $\neg\varphi$) from a theory requires that φ ($\neg\varphi$ respectively) be true across all the models of the theory

[4]We use non-persistent instead of nonmonotonic to emphasise the semantic aspect.

by the Gödel's completeness theorem. The uniformity condition is very strong from the database viewpoint. In the context of databases, we need not consider any interpretation beyond Herbrand interpretations (see [1]). So it is not surprising to see that the uniformity requirement makes the inference mechanism of first-order logic not suitable for the retrieval of negative information in the area of databases[5]. Let us consider an example by way of illustration.

Example 2.3.1 Consider a situation s, consisting of a database DB of an imaginary mathematics department. Suppose that DB contains the following facts:

$$
\begin{aligned}
&teach(frege, \mathit{first\text{-}order\text{-}logic}).\\
&teach(cantor, \mathit{set\text{-}theory}).\\
&teach(tarski, \mathit{model\text{-}theory}).\\
&teach(turing, \mathit{recursion\text{-}theory}).\\
&teach(godel, \mathit{proof\text{-}theory}).
\end{aligned}
$$

Take DB as a theory with only atomic sentences. We can add to DB $teach(turing, \mathit{first\text{-}order\text{-}logic})$ or its negation without causing any contradiction, so we have neither $teach(turing, \mathit{first\text{-}order\text{-}logic})$ nor its negation. In other words, it is impossible from the theory to infer either $teach(turing, \mathit{first\text{-}order\text{-}logic})$ or its negation.

But from the semantic viewpoint, DB can also be taken as a Herbrand interpretation. Then from DB we may reach the conclusion that $\neg teach(turing, \mathit{first\text{-}order\text{-}logic})$ using M2. Moreover, this conclusion is what we want when we make a query to the database DB about whether or not Turing teaches first-order logic. □

[5]As far as positive information is concerned, it does not make any difference whether we consider all the interpretations or just Herbrand interpretations. See [64] for related discussion in the context of definite logic programs.

The example underlines the weakness of first-order logic as a deductive mechanism for inferring negative information. The weak deductive mechanism has to be strengthened somehow if we are going to take a deductive approach to databases.

2.4 Non-persistence vs Monotonicity

In this section, we look at another pair of properties of first-order logic: persistence and monotonicity. We shall see that classical first-order logic is monotonic but it is not persistent.

Persistence is a model-theoretic property. It means whatever is supported by a model is still supported by a larger one. Formally, persistence can be formulated as follows:

$$\text{if } I \models \varphi, I \subseteq I' \text{ then } I' \models \varphi.$$

where I and I' are Herbrand interpretations, φ is a $\mathcal{L}$-sentence, and $\subseteq$ is the subset relation. In the general context, $\subseteq$ should be replaced with the sub-model relation $\leq$.

In comparison to persistence, monotonicity is a proof-theoretic property. It expresses the cumulativity of deductive consequences: what is proved is still proved when more information is added. Formally, monotonicity can be formulated as follows:

$$\text{if } ? \vdash \varphi \text{ and } ? \subseteq ?' \text{ then } ?' \vdash \varphi$$

where $?, \Delta$ are sets of $\mathcal{L}$-sentences, and φ is a $\mathcal{L}$-sentence, and $? \vdash \varphi$ means there is a deduction of φ from $?$.

From the informational viewpoint, both persistence and monotonicity express a kind of the preservation of information. The discussion in the previous sections suggests that, in first-order logic, the way in which information is dealt with at the semantic level is different from that at the proof-theoretic level. As a result, we have another pair of differences between the semantics of first-order logic and its formal system. Let us consider non-persistence first.

2.4.1 Non-persistence

We have shown that negation at the semantic level means failure to support. For atomic sentences, the failure can be reduced to *absence* according to the semantic rule M2. In the previous section, we argued that we had better take M2 as a non-persistent rule rather than as a rule based on the completeness assumption. It is the non-persistence of the rule that contributes to the non-persistence of first-order logic: with the addition of new facts, what is originally absent may become present, and thus what is originally not true may become true. More generally, by induction on the complexity of φ, it is not difficult to show that $\neg\varphi$ is not usually persistent either.

Consider the departmental database DB in the last section again. We know that s supports that Turing does not teach first-order logic since *teach*(*turing*, *first-order-logic*) is not in DB. But when we move to a larger situation, say a situation s_1 of the university which contains another database DB_2 of a philosophy department in addition to DB_1. If Turing happens to be a joint professor in the two departments, and DB_2 does have the fact that Turing teaches first-order logic, then s_1 will support this fact, and as a result, s_1 no longer supports that Turing does not teach first-order logic. Thus, with the addition of the new facts to s, what is originally not true now becomes true.

It is interesting to note that the non-persistence of first-order logic explains the non-persistence[6] of first-order queries to relational databases (see [1]). Conversely,

[6]In [1], term *monotonic* is used. But monotonic is usually used to describe the property of the formal system of first-order logic. Since the property in question is semantic, we prefer to use term persistent.

the fact that there exist non-persistent queries gives us another argument to support our view on rule M2. Had we insisted that M2 be based on the completeness assumption, then it would be meaningless to talk about non-persistent queries. Since our knowledge about the given domain is complete, there would be no space for the addition of new information, and thus there would be no non-persistent queries.

2.4.2 Monotonicity

Now consider the monotonicity of first-order logic. Given a formal system of first-order logic, conceptually, the monotonicity follows directly from the characterisation of the deduction relation $\vdash$ among sentences relative to the system. Alternatively, the monotonic property has been used in the defining characteristic of a formal system such Gentzen's natural deductive system NK. So, the monotonicity holds automatically.

Specifically, any deductive consequence of a theory relative to a formal system is determined by the theory and inference rules, and also the axioms of the system if it is a logistic calculus. Applications of an inference rule solely depend on the *explicit presence* of information both in the theory and derived from the theory. The use of inference rules is *not sensitive* to the *absence* of any information as the use of M2 is. The point is that all inference rules in the system are monotonic: with the addition of new information to the theory, more information will be inferred, and previously inferred information cannot be retracted or revised in any way. A logic with only monotonic inference rules can only be monotonic.

The discrepancy between the monotonicity of first-order logic and its non-persistence once again indicates that the first-order formal system only partially characterises its semantics.

2.5 Concluding Remark

We have shown that classical negation in fact should be divided into two different notions: one is semantic, and the other proof-theoretic. The semantic notion of classical negation expresses failure to support whereas the proof-theoretic one expresses inconsistency. The notion of negation as failure to support is much weaker than that of negation as inconsistency. More generally, we have shown that, in first-order logic, there exists a big discrepancy between the semantic and proof-theoretic levels. One the one hand, first-order logic is semantically two-valued and non-persistent. On the other hand, it is proof-theoretically three-valued and monotonic.

In the area of relational databases, it is the semantic not proof-theoretic notion of classical negation that is used to deal with the retrieval of negative information. First-order logic only provides us with a monotonic deductive mechanism, which is not adequate for dealing with negative information even in the area of relational databases. This implies that if we are going to take a deductive approach to databases, then we have to extend the inference mechanism of first-order logic so as to at least accommodate non-persistent inferences in one way or another. This is a topic we shall cover in later chapters.

Chapter 3

First-Order Logic $\mathbf{CF}'$

The purpose of this chapter[1] is to study strong negation in formal logic systems. We shall construct a first-order logic $\mathbf{CF}'$ with strong negation and bounded quantifiers. The logic is based on constructive logics, in particular, Thomason's logic **CF**. However, unlike constructive logics, quantifiers in our system as in Thomason's are static rather than dynamic.

For the logic $\mathbf{CF}'$, the usual Kripke formal semantics is defined but based on situations instead of conventional possible worlds. Situations as limited portions of the world seem more suitable than possible worlds for characterising the partiality of constructive logics with strong negation. A sound and complete axiomatic system of $\mathbf{CF}'$ is established based on the axiomatic systems of constructive logics with strong negation and Thomason's completeness proof techniques. With the use of bounded quantifiers, $\mathbf{CF}'$ allows the domain of quantification to be empty and allows for non-denoting constants. $\mathbf{CF}'$ is proposed as the underlying logic for situation theory. Thus the connection between $\mathbf{CF}'$ and infon logic is briefly discussed.

[1]This chapter mainly is based on the work published in [112].

3.1 Constructive Logics with Strong Negation

In the first chapter, we noted that strong negation was introduced independently by Nelson [76] and Markov [68] from the constructive perspective. Various constructive logics with strong negation have been developed. In comparison to intuitionistic logics, these logic systems demonstrate some satisfying features, including symmetry and partiality.

First of all, since negative information is treated as of equal importance with positive information, such logics are more symmetrical than intuitionistic logics and satisfy very natural duality laws. In particular, strong negation avoids non-constructive features possessed by intuitionistic negation (see [54], and [114]). Secondly, constructive logics with strong negation can be provided with a more satisfying interpretation than the well-known Brouwer-Heyting-Kolmogorov(BHK) interpretation for intuitionistic logics (see [114], and Lopez-Escobar [65]). Moreover, they permit a sentence to be undetermined and thus can accommodate the partiality of information (see [98], and [114]).

Another desirable characteristic of constructive logics with strong negation is the heredity or persistence of information,[2] to the effect that what is true at a state of information is still true at all later states. This is also true of intuitionistic logic.

3.2 Motivations behind Logic **CF**$'$

Since that we have already various constructive logics with strong negation, it is natural to ask why we need to construct another one. In this section, we give some arguments to show why the existing constructive logics are not satisfying. Two is-

[2]The terminology of the heredity of information is used in [114] whereas the persistence of information is the situationists' parlance. Note that [114] is only concerned with propositional logics. For the property of predicate logic, see lemma 3.1 on page 53 of [54]. In intuitionistic logic, the property is called monotonicity (See the lemma on page 78 of [100]).

sues are to be addressed briefly. One is about the the semantic interpretation of quantification, and the other about domains of quantification. The basic philosophical stance here is taken from Situation Theory. Indeed, as we pointed out in the beginning, **CF**$'$ originally is intended as an underlying logic for Situation Theory. See section 3.5 for the discussion about the connection between **CF**$'$ and infon logic.

3.2.1 Dynamic vs. Static Quantifiers

There are two main semantic interpretations of universal quantifiers: *dynamic* and *static*. In the dynamic interpretation, a sentence $\forall x\varphi(x)$ is true at a state of information s only when $\varphi(\underline{a})$[3] is true at all states of information $t \geq s$ for all individuals a in the domain of t (where $\geq$ orders states by increasing information). In the static interpretation, a sentence $\forall x\varphi(x)$ is true at a state of information s provided that $\varphi(\underline{a})$ is true for all individuals a in the domain of s. The dynamic interpretation of universal quantifiers at a state of information s requires us to look at all the states beyond s and their domains. In contrast, the static interpretation of a sentence $\forall x\varphi(x)$ at a state of information s only involves the state s and the individuals in the domain of s. So, the satisfaction condition on universal quantifiers by the dynamic interpretation is much stronger than that by the static interpretation.

Which interpretation is more natural from the situation-theoretic viewpoint? Consider the situation s of a room full of people. The sentence "All men here are hungry" will be true at s provided that all the men in the room are hungry. Here the quantifier is taken as restricted to the men in that room. We do not look at wider situations and (possibly) wider extensions of "men". So if we take a point s in a Kripke model as a situation rather than a state of information, then it seems we should evaluate the quantifier statically.

Unfortunately, dynamic universal quantifiers are used in most of existing first-

[3]Hereafter, we use $\underline{a}$ as a name for a.

order constructive logics with strong negation, including Almukdad and Nelson's first-order systems [4], Gurevich's $\overline{\mathbf{H}}$ [54], Akama's **S** [2]. It is not difficult to see that these logics would not be persistent had the static universal quantifiers been used. In other words, the persistence of constructive logics is bought at the cost of the very strong dynamic satisfaction condition on universal quantifiers.

Thomason's first-order logic [98] **CF** does interpret universal quantifiers statically rather than dynamically. His semantical model is a hybrid of a Kripke model for propositional intuitionistic logic (as the conditional is intuitionistic) and a classical model for predicate logic (as the universal quantifier is static).[4] Nevertheless, his semantical framework requires different stages to have the same domain. This leads us to the issue of the domains of quantifiers.

3.2.2 Expanding vs Constant Domains of Quantification

From the standpoint which treats stages as situations, it is obvious that this restriction is inappropriate. From an intuitionistic viewpoint, it is not suitable either. As is well-known, Kripke models for intuitionistic logic also require expanding domains. But the connection of intuitionistic logic with expanding domains is both more complicated and more tenuous than is the case with Situation Theory. In order to see this, let us consider the following schema which we call the Distribution Schema:

$$\text{(DS)} \qquad \forall x(\varphi \vee \psi(x)) \supset (\varphi \vee \forall x \psi(x)), \text{ where } x \text{ is not free in } \varphi.$$

If we add to intuitionistic logic all instances of (DS), we obtain a logic whose models are exactly the Kripke models with constant domain. Thus to motivate expanding domains from an intuitionistic viewpoint is to motivate the rejection of this schema.

[4]It should be pointed out that his model for propositional logic, strictly speaking, is not intuitionistic since the falsity of an atomic sentence at a stage of construction is treated as being discovered directly rather than being decided by later stages.

The BHK interpretation is of little help. According to that we need to show how a proof of $\forall x(\varphi \vee \psi(x))$ could be extended to a proof of $\varphi \vee \forall x\psi(x)$. Well, to have a proof of $\forall x(\varphi \vee \psi(x))$ is to have a construction C which transforms a proof of $a \in D$ (D the intended range of the variable x) into a proof of $\varphi \vee \psi(a)$. If the construction C transforms a proof of $a \in D$ into a proof of φ, then since x is not free in φ, we would have a proof of φ. Otherwise, it transforms a proof of $a \in D$ into a proof of $\psi(a)$ and thus from the construction C we derive a proof of $\forall x\psi(x)$. Either way we have a proof of $\varphi \vee \forall x\psi(x)$ (for the BHK interpretation, see p. 9 of Troelstra and van Dalen [100]). The informal semantics of intuitionistic logic does not, at least not obviously, show what is wrong with (DS). Why, then, is (DS) rejected at all? Very briefly, it happens that certain Brouwerian principles of continuity which are more or less self-evident from an intuitionistic standpoint are formally inconsistent in classical logic. These principles say roughly that an assertion about an infinite sequence α must be decided by a finite initial segment of α, and hence will be decided the same way for all sequences β that agree on that initial segment. Adding (DS) to intuitionistic logic will restore inconsistency with these same principles. Dummett [36] contains a treatment of the semantics of intuitionistic logic which discusses these issues in detail.

There is a further point. Kripke models are not the only semantic structures for intuitionistic logic. Beth trees may be used instead. In the Beth semantics we have a more complicated rule (see p. 106 of Troelstra [99]) for evaluating disjunctions:

$$s \models \varphi \vee \psi \text{ iff } \forall t \geq s \exists u \geq t(u \models \varphi \text{ or } u \models \psi)$$

This evaluates a disjunction true provided that however knowledge is extended eventually one or other of the disjuncts will become true. With this it is easy to find a counter-example to (DS) that makes no appeal to expanding domains.

The upshot is that expanding domains seem more an artifact of the Kripke semantics than an essential part of the interpretation of intuitionistic logic. However, they are quite central to Situation Theory, which to some extent supports our choice not to use intuitionistic logic as a basis for Situation Theory (for more, see the final part below)

If we are to allow expanding domains, there is a technical problem to overcome. Specifically, the semantical completeness proof of **CF** depends on an auxiliary lemma, that is, lemma 2 on page 250 in [98], and the proof of the lemma in turn makes use of the conditional introduction rule $\supset$I. However, it is easy to check that if different stages in the semantical models are allowed to have different domains, then the rule is generally not sound since universally quantified sentences, when interpreted statically instead of dynamically, generally are not persistent (see below 2.2 and 2.5). So, the condition of a constant domain has to be imposed on his models for the sake of **CF**'s semantic completeness, that is to say, in order to have static universal quantifiers, we are forced to adopt a model with constant domain.

Conversely, from the model theoretic standpoint, the models for **CF**$'$ are a special case of the intuitionistic models. Accordingly, the dynamic condition for quantifiers collapses into the static one. Since the dynamic condition is not suitable and expanding domains are desired as we said above from the situation theoretic viewpoint, it is natural to ask if we can have a logic for situation theory with both static quantifiers and expanding domains.

3.2.3 From Thomason's Logic CF to Our Variant CF$'$

Motivated by the above arguments, we propose a first-order logical system **CF**$'$ with strong constructive negation like Thomason's but that allows for expanding domains. Our semantical analysis is still based on Kripke frames $\langle S, \prec, D\rangle$ but we have it in mind to interpret S as a collection of situations rather than conventional possible worlds. Accordingly, $\prec$ is a pre-order on situations, and D is a function assigning a set of individuals to each situation. Situations are limited parts of the world. Thus, generally, situations provide us with only incomplete information. The partiality of situations to some extent also justifies the use of situations in our semantical framework since, as we pointed out before, constructive logics with strong negation are partial. In addition, we note that another source of the partiality of the logic is from the use of inexact predicates (see [4], Wagner [109], and related citations there). We treat universally quantified sentences statically instead of dynamically. And since static unbounded universally quantified sentences are generally not persistent, we instead consider bounded ones, say $\forall^{\beta}x\varphi(x)$ where β is a bounder. This is reminiscent of Devlin's infon logic. Devlin [32] considers $\forall x \in u\sigma$ where u is a set and σ is an infon. Such compound infons are persistent because the set u bounds the quantifier. In our framework the bounder β may itself be non-persistent in the sense that the extension of β is liable to change from situation to situation, and consequently $\forall^{\beta}x\varphi(x)$ is in general not persistent either. Thus, we further distinguish persistent bounders from non-persistent ones (see the next section).

We summarise the various approaches to the universal quantifier in the following table, where **INT** is intuitionistic predicate logic, $\overline{\mathbf{H}}$ is Gurevich's intuitionistic logic with strong negation [54], and **CF** is Thomason's first-order logic [98]. For a unifying exposition of both Kripke and Beth models, see van Dalen [102].

Logics	*Quantifiers* $\forall x$	*Models*	*Domains*
INT, $\overline{\mathbf{H}}$	dynamic	Kripke models	expanding
INT	static	Beth models	constant
CF	static	Kripke models	constant
CF$'$	static	Kripke models	expanding

In the following, we shall first introduce the logical system **CF**$'$, and then prove its soundness and completeness. Finally, we discuss its connection with situation theory, its possible extensions as well as its potential applications.

3.3 Logical System CF$'$ With Strong Negation

3.3.1 Language $\mathcal{L}$ of CF$'$.

The language[5] of our logical system **CF**$'$ consists of an infinite set $V_{\mathcal{L}}$ of individual variables (as metavariables for variables we use $x, x_0, x_1, ...$), a set $C_{\mathcal{L}}$ of individual constants (metavariables: $c, c_0, c_1, ...$), and for each $n, n \geq 0$, a set $P^n_{\mathcal{L}}$ of n-ary predicate symbols (metavariables: $R_1, R_2, R_3, ...$). In addition, $\mathcal{L}$ has a set $B_{\mathcal{L}}$ of bounders with a subset $B^P_{\mathcal{L}}$ of persistent bounders (metavariables: $\beta, \beta_0, \beta_1, ...$ with or without superscript P), and a relation symbol $\in$.

The set $T_{\mathcal{L}}$ of terms of $\mathcal{L}$ is $V_{\mathcal{L}} \cup C_{\mathcal{L}}$. We use $t, t_0, t_1, ...$ as metavariables for terms.

Atomic formulas of $\mathcal{L}$ are $R(t_1, t_2, ..., t_n)$ and $c \in \beta$, where $t_1, t_2, ..., t_n \in T_{\mathcal{L}}, c \in C_{\mathcal{L}}, R \in P^n_{\mathcal{L}}$ and $\beta \in B_{\mathcal{L}}$. The well-formed formulas of $\mathcal{L}$ are defined recursively from atomic formulas using the connectives $\vee, \supset$, and $\sim$, and for each bounder β , a bounded universal quantifier $\forall^{\beta}$ as follows:

(i) atomic formulas are formulas;

(ii) if φ is a formula, then so is $\sim \varphi$;

(iii) if φ, ψ are formulas, then so are $\varphi \vee \psi, \varphi \supset \psi$;

(iv) if φ is a formula, x is a variable, and β is a bounder , then $\forall^{\beta} x \varphi(x)$ is also a formula. For simplicity, we write $\forall x \in \beta \varphi(x)$ for $\forall^{\beta} x \varphi(x)$.

[5]Function symbols introduce nothing new. For simplicity, we avoid them here.

A formula of form $\forall x \in \beta\varphi(x)$ is called a bounded universally quantified formula. Such formulas can be used to express local generality since the bound variables thereof are to range over a subset of the individuals in the universe. In contrast, the generality expressed by unbounded universally quantified formulas is a kind of overall generality (see Frege [41]). In order to express overall generality by a variable, we only need a device for the scope of the variable whereas, in order to express local generality, we need in addition the range of the variable. So, generally speaking, in order to express generality via a variable, we need both a mechanism for the scope of the variable and a parameter for its range. In other words, a logical quantifier consists of the scope of a variable and the range of the variable. From the pragmatic point of view, it is clear that bounded formulas are more frequently used than unbounded ones. In translating natural language, restricted quantifiers are usually represented as unrestricted quantifiers over a material conditional or something equivalent. Thus, "All birds fly" is formalised as $\forall x(\sim \mathrm{Bird}(x) \vee \mathrm{Flies}(x))$ or $\forall x(\mathrm{Bird}(x) \rightarrow \mathrm{Flies}(x))$ if the material conditional $\rightarrow$ is defined. In $\mathbf{CF}'$ it is represented as $\forall x \in \beta\mathrm{Flies}(x)$, where β is a bounder for birds. We prefer our approach to the usual one. In our opinion, it is tidy, and emphasises the two aspects of local generality. More importantly, as we mentioned in the introduction, bounded universally quantified formulas, can be used to express the persistence of information (see below). That is the primary motive for our use of bounded formulas instead of unbounded ones.

Syntactically bounders[6] are flags on quantifiers. Semantically they are to be interpreted as sets, that is in the same way as predicates are in classical first-order logic. Then, it may be asked, why do we have a special syntax for bounders instead of treating them simply as unary predicates? The answer is that a predicate such as "Flies(x)" gives three possibilities: an object may fly, it may not fly, or it may

[6]It is worth pointing out that bounders are very similar to sorts in a sorted logic (see [31]) though we will not explore any connection between them in this thesis.

be undecided whether it flies or not. But a bounder supplies only two possibilities: an object is included in the bounder or it is not. The consequence is that $\forall x(\sim \psi(x) \vee \varphi(x))$ in fact says a little more than $\forall x \in \beta\varphi(x)$ (see Formal Semantics below for exact comparison). It is the latter that captures the informal reading of "All birds fly" rather than the former.

Conjunction and bounded existential quantification are defined as follows:

$$\begin{aligned} \varphi \wedge \psi \quad &=_{df} \quad \sim(\sim \varphi \vee \sim \psi) \\ \exists x \in \beta\varphi(x) \quad &=_{df} \quad \sim \forall x \in \beta \sim \varphi(x). \end{aligned}$$

The concept of free and bound variables is defined as usual. Bound variables are used as position markers only and thus $\forall x \in \beta\varphi(x)$ and $\forall y \in \beta\varphi(y)$ would be counted as the same formula. We use as above $\varphi, \psi, \chi, ...$ as metavariables for formulas, and ?, Δ (with or without subscripts) for arbitrary sets of formulas.

3.3.2 Persistent Formulas

We have met the concept of persistence in Chapter 2. Put it in terms of situations, it says that what is true in one situation is still true in a larger situation. Formally, there is the so-called persistence principle, stated as

$$\text{If } s \prec s' \text{ and } s \models \sigma, \text{ then } s' \models \sigma,$$

where s, s' are situations, σ is an infon, and $\models$ is a support relation between situations and infons. If an infon σ satisfies the persistence principle, we say that σ is persistent (see Barwise [8]). We have seen that in Chapter 2, one source of non-persistence comes from the semantic notion of classical negation, which in essence is a kind of negation as failure. In contrast, strong negation expresses a directly established negative truth, and thus is both persistent and monotonic (see sections 2.3 and 2.4 for the formal characterisation).

However there is another source of non-persistence which comes from static universal quantification. Informally speaking, universally quantified sentences in natural language are not persistent. "Everyone here is hungry" may be verified when evaluated from the situation in one poor household, but falsified when evaluated from a larger situation including comfortable ones. There is a tension between quantification and persistence. If we take it that the persistence principle is true of every infon, then it seems universally quantified sentences have to be excluded from the category of infons. And conversely, if universally quantified sentences are taken as infons then the persistence principle would only hold partially (see pp. 234–236 of [8]). However, quantified sentences are such important forms for expressing information that they can hardly be excluded from the category of infons. We also want to retain the persistence principle because, as situation theorists have argued, it captures our intuition "that what goes on in part of the world still goes on when one has a broader perspective"(see p. 236 of [8]). For the sake of both persistence and a rich algebraic structure of infons, we only consider bounded quantified formulas for which these problems do not arise. However, as we pointed out in the introduction, in our present framework, a bounder β in $\forall x \in \beta\varphi(x)$ may be non-persistent. So we introduce an auxiliary notion of persistent bounders. Syntactically, persistent bounders are treated as a primitive notion. The semantic meaning of persistent bounders will be given below (see condition (iii) on an interpretation in Formal Semantics). Pragmatically, persistent bounders can be obtained by incorporating context into bounders in universally quantified sentences. Then we can define persistent formulas of $\mathcal{L}$ recursively as follows :

(i) $R(t_1, t_2, ..., t_n)$ and $\sim R(t_1, t_2, ..., t_n)$ are persistent for any n-ary predicate R, terms $t_1, t_2, ..., t_n$, and $c \in \beta$ and $\sim c \in \beta^P$ are persistent for any bounders β and β^P;

(ii) if φ, ψ are persistent, then so are $\varphi \vee \psi$ and $\varphi \wedge \psi$;

(iii) $\varphi \supset \psi$ is persistent for any formulas φ, ψ;

(iv) if φ is persistent, then $\forall x \in \beta^P \varphi(x)$ is persistent;

(v) if φ is persistent, then $\exists x \in \beta\varphi(x)$ is persistent for any $\beta \in B_{\mathcal{L}}$.

Given a set ? of formulas, let $?^P$ be $\{\varphi \in ? : \varphi \text{ is persistent}\}$. So the persistent formulas of $\mathcal{L}$ will be $F^P_{\mathcal{L}}$, where $F_{\mathcal{L}}$ is the set of all $\mathcal{L}$-formulas. Note that, in the definition of persistent formulas, negation is restricted to only atomic formulas. Nevertheless, this will not lose any generality since the negation of a compound formula, according to related rules (see Derived Rules for **CF**$'$ below), is equivalent to another compound formula in which negation is applied to only atomic formulas.

By the definition, non-persistence of formulas is only due to the non-persistence of bounders in universally quantified formulas. So, pragmatically, the persistence of such formulas can be recovered by incorporating context into related bounders. Nevertheless, there exists indeed a kind of unrecoverable non-persistence. In fact, such non-persistence is the consequence of the partiality of situations. If a situation is silent on σ then it certainly does not preclude a larger more extensive situation settling σ. In order to express the unrecoverable non-persistence, we need to add a kind of modal operators such as "definitely" into our language. Such an extension, however, is outside the scope of this thesis (for more, see Mott [75]).

The syntactic definition of persistence will be used in Derived Rules for **CF**$'$ below.

3.3.3 Formal Semantics

Our semantical analysis is essentially similar to Thomason's, but it is based on general Kripke frames instead of particular ones, that is, we allow different points in a Kripke frame to have different domains. A Kripke frame $\mathcal{F}$ is a triple $\langle S, \prec, D\rangle$ such that

(i) S is a non-empty set;

(ii) $\prec$ is a pre-order on S, that is, $\prec$ is a reflexive and transitive binary relation on S;

(iii) D is a monotone function assigning sets of individuals to the elements of S, that is, for any $s, s' \in S$, if $s \prec s'$ then $D(s) \subseteq D(s')$.

S is to be thought of as a set of situations, $\prec$ is the containment relation among situations, and for each $s \in S, D(s)$ is the set of individuals existing at situation s.

An interpretation I of language $\mathcal{L}$ on a Kripke frame $\mathcal{F} = \langle S, \prec, D\rangle$ is a function such that: for any $s, s' \in S, c \in C_{\mathcal{L}}, R \in P^i_{\mathcal{L}}, \beta, \beta^P \in B_{\mathcal{L}}$,

(i) I_s is a partial function from $C_{\mathcal{L}}$ into $D(s)$, and (a) if $s \prec s'$ and $I_s(c)$ is defined, then $I_{s'}(c)$ is defined too and $I_s(c) = I_{s'}(c)$; and(b) for each d in $D(s)$, $I_s(\underline{d})$ is defined and $I_s(\underline{d}) = d$.[7]

(ii) $I_s(R)$ is a partial function from the Cartesian product $D(s)^i$ into {T, F}, and if $s \prec s'$, then $I_{s'}(R)$ is an extension of $I_s(R)$.

(iii) I_s is a total function from $B_{\mathcal{L}}$ into $\mathcal{P}(D(s))$ such that if $s \prec s'$, then $I_s(\beta) \subseteq I_{s'}(\beta)$ and $I_s(\beta^P) = I_{s'}(\beta^P)$.

Clause (iii) in the definition of interpretation gives us the semantic meaning of persistent sets. In other words, it is the semantic requirement for a set of individuals to be persistent. It is worth pointing out the restriction incorporated in (iii) is compatible with the situation theoretic viewpoint though it may look ad hoc. Anyway,

[7]We are assuming that every object has a name. In effect we work with the expansion of language $\mathcal{L}$ to accommodate all the objects of all the countable domains.

situations are treated as first-class citizens in situation theory. So, one possible way to ensure the persistence of universal quantified formulas would be to incorporate reference to situations into them (see p. 236 of [8]). In this paper, however, we instead adopt the device of persistent bounders.

A Kripke model $\mathcal{M}$ is a pair $\langle \mathcal{F}, I \rangle$ consisting of a Kripke frame $\mathcal{F}$ and an interpretation I on $\mathcal{F}$.

Before we continue the formulation of formal semantics, some remarks seem in order about the definition of Kripke models. First, note that, in a Kripke model $\mathcal{M} = \langle S, \prec, D, I \rangle$, $D(s)$ can be empty for any (and all) $s \in S$. The use of bounders means that the usual restriction to non-empty domains is unnecessary. Thus **CF**$'$ is inclusive in the sense that it allows the domain of quantification to be empty (see pp. 379-382 of Bencivenga [14]).

Second, note that the function $I_s \lceil C_{\mathcal{L}}$ is partial. So **CF**$'$ allows for non-denoting constants as a free logic does (see [14]). In a free logic, an extra unary predicate E or something equivalent is introduced to deal with reference failure. Nevertheless, in **CF**$'$, we do not need such a special predicate. Bounders of quantifiers can play the role of the predicate E of free logic. It may be that bounders are preferable to an existence predicate, at least if one wishes to confine existence to a purely semantic role (as we would). Anyway, it will be no surprise that some axioms and inference rules of **CF**$'$ will correspond to axioms and inference rules of a free logic.

Next, note that the function $I_s \lceil P^n_{\mathcal{L}}$ $(n \geq 0)$ is also partial. That is to say, it may be the case that a basic sentence $R(c_1, c_2, ..., c_n)$ is neither true nor false, so **CF**$'$ allows truth value gaps. Such gaps may arise from the use of inexact predicates, but we emphasise that there is another source of truth value gaps – the partiality of situations.

When a predicate has truth value gaps, we call it a partial predicate, otherwise a total predicate. A total predicate can be interpreted as a set, that is in the same way as predicates are in classical first-order logic. With partial predicates, however, we have to associate two sets: one is for the positive assertions, the other for the strong negative assertions. So we might as well divide a partial predicate into two parts, a positive part corresponding to the positive assertions, and a negative part corresponding to the strong negative assertions. We recall that, syntactically, bounders are flags on quantifiers. Semantically, as can be seen from the clause (iii) in the definition of interpretation, bounders are interpreted as sets. What sets, then, should we associate with a bounder β? There are two natural candidates. We could say that β was assigned all the objects in the current situation. Then $\forall x \in \beta\varphi(x)$ would be supported by s provided that s made true $\varphi(\underline{a})$ for each object a in $D(s)$. In this case, bounder β is nothing more than a denotational variant of the existential predicate E of free logic (see pp. 251-252, Garson [45]). An alternative would see bounders in a more restricted way as corresponding to the positive parts of particular predicates, so that $\forall x \in \beta\varphi(x)$ would be interpreted as asserting of all the objects that were β in the current situation that they were also φ. In fact, we choose here not to restrict bounders beyond requiring that the objects a bounder β is associated with in a situation s are all objects that belong to the situation s.

Given a Kripke model $\mathcal{M} = \langle S, \prec, D, I\rangle$, we define a satisfaction relation $\models^+_{\mathcal{M}}$ (or simply $\models^+$) and a refutation relation $\models^-_{\mathcal{M}}$ (or simply $\models^-$) between situations $s \in S$ and $\mathcal{L}$-sentences φ relative to $\mathcal{M}$ as follows, by induction on the complexity of φ:

(i) $s \models^+ R(c_1, c_2, ..., c_n)$ iff $I_s(c_1), I_s(c_2), ..., I_s(c_n)$ are all defined and $I_s(R)(I_s(c_1), I_s(c_2), ..., I_s(c_n)) = \mathrm{T}$

$s \models^- R(c_1, c_2, ..., c_n)$ iff $I_s(c_1), I_s(c_2), ..., I_s(c_n)$ are all defined and $I_s(R)(I_s(c_1), I_s(c_2), ..., I_s(c_n)) = \mathrm{F}$

$s \models^+ c \in \beta$ iff $I_s(c)$ is defined and $I_s(c) \in I_s(\beta)$

$s \models^- c \in \beta$ iff either $I_s(c)$ is not defined or

$I_s(c)$ is defined and $I_s(c) \in D(s) \perp I_s(\beta)$

(ii) $s \models^+ \varphi \vee \psi$ iff $s \models^+ \varphi$ or $s \models^+ \psi$

$s \models^- \varphi \vee \psi$ iff $s \models^- \varphi$ and $s \models^- \psi$

(iii) $s \models^+ \sim \varphi$ iff $s \models^- \varphi$

$s \models^- \sim \varphi$ iff $s \models^+ \varphi$

(iv) $s \models^+ \varphi \supset \psi$ iff for all s' such that $s \prec s'$ if $s' \models^+ \varphi$ then $s' \models^+ \psi$

$s \models^- \varphi \supset \psi$ iff $s \models^+ \varphi$ and $s \models^- \psi$

(v) $s \models^+ \forall x \in \beta \varphi(x)$ iff for all $d \in D(s)$, if $s \models^+ \underline{d} \in \beta$ then $s \models^+ \varphi(\underline{d})$

$s \models^- \forall x \in \beta \varphi(x)$ iff for some $d \in D(s)$, $s \models^+ \underline{d} \in \beta$ and $s \models^- \varphi(\underline{d})$.

Basic semantic notions such as consequence, satisfiability and validity can be defined in the usual way in terms of the satisfaction relation $\models^+$. For any sentence φ and set ? of sentences, we write $\models \varphi$ to indicate that φ is valid, ? $\vDash \varphi$ to indicate that φ is a semantic consequence of ?, and ? $\vDash \Delta$ to indicate that there is a subset $\{\varphi_1, \varphi_2, ..., \varphi_n\}$ of Δ such that $\varphi_1 \vee \varphi_2 \vee ... \vee \varphi_n$ is a semantic consequence of ?.

Lemma 3.3.1 (Persistence Lemma) *Let $\mathcal{M} = \langle S, \prec, D, I \rangle$ be a Kripke model, φ a persistent formula of $\mathcal{L}$.*

(i) *if $s \prec s'$ and $s \models^+ \varphi$ then $s' \models^+ \varphi$;*

(ii) *if $s \prec s'$ and $s \models^+$? then $s' \models^+$?^{P}.*

Proof. For (i), routine induction on the complexity of φ. (ii) is a straightforward corollary of (i).

The persistence lemma (i) gives us the semantic meaning of persistence. It can be viewed as a variant of the persistence principle.

3.3.4 Axiomatic System for CF′

Our axiomatic system **CF′** is based on the axiomatic systems for constructive logics with strong negation (see [94], [54], and [2]). It takes as axioms the following list of schemas:

(A1) $\varphi^P \supset . \psi \supset \varphi^P$

(A2) $\varphi \supset (\psi \supset \chi) \supset . \varphi \supset \psi \supset . \varphi \supset \chi$

(A3) $\varphi \wedge \psi \supset \varphi$

(A4) $\varphi \wedge \psi \supset \psi$

(A5) $\varphi^P \supset . \psi \supset \varphi^P \wedge \psi$

(A6) $\varphi \supset \varphi \vee \psi$

(A7) $\psi \supset \varphi \vee \psi$

(A8) $\varphi \supset \chi \supset . \psi \supset \chi \supset . \varphi \vee \psi \supset \chi$

(A9) $\varphi \supset . \sim \varphi \supset \psi$

(A10) $c \in \beta \wedge \varphi(c) \supset \exists x \in \beta \varphi(x)$

(A11) $\forall x \in \beta \varphi(x) \supset \sim c \in \beta \vee \varphi(c)$

(A12) $\forall x \in \beta(\varphi \vee \psi(x)) \supset (\varphi \vee \forall x \in \beta \psi(x))$

(A13) $\sim (\varphi \wedge \psi) \equiv \sim \varphi \vee \sim \psi$

(A14) $\sim (\varphi \vee \psi) \equiv \sim \varphi \wedge \sim \psi$

(A15) $\sim\sim \varphi \equiv \varphi$

(A16) $\sim (\varphi \supset \psi) \equiv \varphi \wedge \sim \psi$

(A17) $\sim \forall x \in \beta\varphi(x) \equiv \exists x \in \beta \sim \varphi(x)$

(A18) $\sim \exists x \in \beta\varphi(x) \equiv \forall x \in \beta \sim \varphi(x)$

(A19) $c \in \beta \vee \sim c \in \beta$

In axioms A1 and A5, φ^P means that φ has to be persistent, which is the small price we have to pay for the relaxation of the dynamic condition on universal quantifiers to the static one. In order to see why this restriction is necessary, let us consider the following formula φ

$$\forall x \in \beta R(x) \supset \top \supset \forall x \in \beta R(x)$$

For this formula, our persistence condition on $\forall x \in \beta R(x)$ is actually to require that β is a persistent bounder. If β is not persistent, it is not difficult to find out that φ cannot be valid based on our formal semantics for **CF′**.

In axiom A12, x is required not to be free in φ. In addition, note that axiom A12 is not assumed in constructive logics (see [54], and [2]). We emphasise our situation theoretical standpoint rather than intuitionistic or constructive viewpoint. So there seems nothing preventing us from assuming the axiom.

With axiom A19, we are assuming that, at any situation, we can always decide if a constant c is in β or not. The assumption is consistent with the semantic interpretation of β given above. In addition, note that axioms A13 and A18 can in fact be derived from the other axioms and related definitions, and thus can be omitted.

CF′ has the following inference rules:

(R1) $$\frac{\varphi,\ \varphi \supset \psi}{\psi}$$

(R2) $$\frac{c \in \beta \wedge \varphi(c) \supset \psi}{\exists x \in \beta\varphi(x) \supset \psi}$$

$$\text{(R3)}\quad \frac{\psi \supset (\sim c \in \beta \vee \varphi(c))}{\psi \supset \forall x \in \beta\varphi(x)}$$

In rules R2 and R3, the constant c is required not to occur in ψ.

The axiomatic system $\mathbf{CF}'$ is a first-order modification of Almukdad and Nelson's $\mathbf{N}$ as well as Thomason's $\mathbf{CF}$.[8] If we delete axiom A9 from $\mathbf{CF}'$, denoted $\mathbf{CF}'^{-}$, then we have a system which is a modification of Almukdad and Nelson's $\mathbf{N}^{-}$. Since axiom A9 is not available in $\mathbf{CF}'^{-}$, we need another axiom to the effect that $c \in \beta$ and $\sim c \in \beta$ do not hold at the same time, say $c \in \beta \wedge \sim c \in \beta \supset \bot$. So, with logic $\mathbf{CF}'^{-}$, inconsistent situations are allowed, but the inconsistency of situations does not arise from the contradictory statements of form $c \in \beta \wedge \sim c \in \beta$.

Basic notions (relative to $\mathbf{CF}'$) such as thesishood, consequence, and consistency can be defined in the usual way. For any sentence φ, and set ? of sentences, we write $\vdash \varphi$ to indicate that φ is a thesis of $\mathbf{CF}'$, $? \vdash \varphi$ to indicate that φ is a consequence in $\mathbf{CF}'$ of ?, and $? \vdash \Delta$ to indicate that there is a subset $\{\varphi_1, \varphi_2, ..., \varphi_n\}$ of Δ such that $\varphi_1 \vee \varphi_2 \vee ... \vee \varphi_n$ is a consequence of ?.

From the definition of thesishood and consequence, it is easy to prove the following lemma.

Lemma 3.3.2 *Let* $?, \Delta$ *be sets of* $\mathcal{L}$*-sentences. If* $? \vdash \Delta$*, then* $?' \vdash \Delta'$ *for some finite subsets* $?'$ *and* Δ' *of* ? *and* Δ*, respectively.*

3.3.5 Derived Rules for $\mathbf{CF}'$.

In this section, we list some rules for the deducibility-relation $\vdash$ of $\mathbf{CF}'$ between sets of $\mathcal{L}$-sentences that are needed in the proof of semantical completeness. It is not difficult to derive them from the axioms and rules of $\mathbf{CF}'$ given before. We divide these rules into three groups. Group I consists of two structural rules, and group II

[8]Note that neither **N** nor **CF** is formulated in axiomatic formalism.

some operational rules. For $\mathbf{CF}'^{-}$, rule $\sim$ E is to be replaced by a rule equivalent to $c \in \beta \wedge \sim c \in \beta \supset \bot$. Group III is about connection between strong negation and other connectives. Lacking the $\sim$-introduction rule, we have to use numerous negation rules to connect negation and other connectives by driving strong negation back and forth across them. Note that, because there is no rule of $\sim$-introduction, we are able to use multiple-conclusion rules without in general being able to derive the Law of Excluded Middle (see the related remarks on p. 82 by Gentzen [50], and the example about the derivation of the law on p. 85 of [50])

Group I.

R: If ? and Δ are not disjoint, then $? \vdash \Delta$.

$$\text{T:}\quad \frac{? \vdash \Delta}{?, \Theta \vdash \Xi, \Delta}$$

Group II.

$$\vee\text{I:}\quad \frac{? \vdash \varphi, \psi, \Delta}{? \vdash \varphi \vee \psi, \Delta} \qquad \vee\text{E:}\quad \frac{?, \varphi \vdash \Delta;\ ?, \psi \vdash \Delta;\ ? \vdash \varphi \vee \psi, \Delta}{? \vdash \Delta}$$

$$\supset\text{I:}\quad \frac{?^{P}, \varphi \vdash \psi}{?^{P} \vdash \varphi \supset \psi} \qquad \supset\text{E:}\quad \frac{? \vdash \varphi \supset \psi, \Delta;\ ? \vdash \varphi, \Delta}{? \vdash \psi, \Delta}$$

$$\sim\text{E:}\quad \frac{? \vdash \varphi, \Delta;\ ? \vdash \sim \varphi, \Delta}{? \vdash \Delta}$$

$$\forall\text{I:}\quad \frac{? \vdash \sim c \in \beta \vee \varphi(c), \Delta}{? \vdash \forall x \in \beta \varphi(x), \Delta} \qquad \forall\text{E:}\quad \frac{? \vdash \forall x \in \beta \varphi(x), \Delta}{? \vdash \sim c \in \beta \vee \varphi(c), \Delta}$$

In $\forall$I, c has no occurrence in $\varphi(x)$, or in any member of ? or of Δ;

Group III.

$$\sim\vee\text{I:}\qquad \frac{?\vdash\sim\varphi,\Delta;\ ?\vdash\sim\psi,\Delta}{?\vdash\sim(\varphi\vee\psi),\Delta}$$

$$\sim\vee\text{E:}\qquad \frac{?\vdash\sim(\varphi\vee\psi),\Delta}{?\vdash\sim\varphi,\Delta}\qquad\qquad \frac{?\vdash\sim(\varphi\vee\psi),\Delta}{?\vdash\sim\psi,\Delta}$$

$$\sim\sim\text{I:}\qquad \frac{?\vdash\varphi,\Delta}{?\vdash\sim\sim\varphi,\Delta}$$

$$\sim\sim\text{E:}\qquad \frac{?\vdash\sim\sim\varphi,\Delta}{?\vdash\varphi,\Delta}$$

$$\sim\supset\text{I:}\qquad \frac{?\vdash\varphi,\Delta;\ ?\vdash\sim\psi,\Delta}{?\vdash\sim(\varphi\supset\psi),\Delta}$$

$$\sim\supset\text{E:}\qquad \frac{?\vdash\sim(\varphi\supset\psi),\Delta}{?\vdash\varphi,\Delta}\qquad\qquad \frac{?\vdash\sim(\varphi\supset\psi),\Delta}{?\vdash\sim\psi,\Delta}$$

$$\sim\forall\text{I:}\qquad \frac{?\vdash c\in\beta\wedge\sim\varphi(c),\Delta}{?\vdash\sim\forall x\in\beta\varphi(x),\Delta}$$

$$\sim\forall\text{E:}\qquad \frac{?\vdash\sim\forall x\in\beta\varphi(x),\Delta;\ ?,c\in\beta\wedge\sim\varphi(c)\vdash\Delta}{?\vdash\Delta}$$

In $\sim\forall$E, c does not occur in $\varphi(x)$, or in any member of ? or of Δ.

Theorem 3.3.1 (Soundness of CF′) *Let ? be a set of $\mathcal{L}$-sentences, and φ a $\mathcal{L}$-sentence. and $\mathcal{M}=\langle S,\prec,D,I\rangle$ a model of $\mathcal{L}$, s a situation in $\mathcal{M}$. If $?\vdash\varphi$, and $s\models^{+}?$, then $s\models^{+}\varphi$.*

Proof. Proof is routine and thus omitted.

Note that the soundness of **CF**′ would fail if we included a rule of $\sim$-introduction ($\sim$-I) to the effect that from $?,\varphi\vdash\Delta$ we can infer $?\vdash\sim\varphi,\Delta$. To see this, observe that, by derived rule R of **CF**′, $\varphi\vdash\varphi$. By $\sim$-I it then follows that $\vdash\sim\varphi,\varphi$. And so

$\vdash\sim \varphi \vee \varphi$ by rule $\vee$-I. But it is not difficult to see that $\sim \varphi \vee \varphi$ is not valid in the current semantic framework. This shows that $\sim$-I is not sound in **CF**$'$.

3.4 Completeness of CF$'$

In this section, we show that first-order logic **CF**$'$ is semantically complete, based on Thomason's completeness proof techniques in [98]. We first introduce related definitions and prove some auxiliary lemmas.

Definition 3.4.1 A set ? of $\mathcal{L}$-sentences is $\mathcal{L}$-ω-complete if for all $\mathcal{L}$-formulas $\varphi(x)$, we have ? $\vdash \forall x \in \beta\varphi(x)$ if ? $\vdash \sim c \in \beta \vee \varphi(c)$ for all $c \in C_{\mathcal{L}}$. And ? is $\mathcal{L}$-saturated if it meets the following five conditions: for any $\mathcal{L}$-sentences φ, ψ,

(i) ? is consistent;

(ii) ? is deductively closed, that is, if ? $\vdash \varphi$, then $\varphi \in$?;

(iii) if ? $\vdash \varphi \vee \psi$, then ? $\vdash \varphi$ or ? $\vdash \psi$;

(iv) if $\sim \forall x \in \beta\varphi(x) \in$?, then for some constant $c \in C_{\mathcal{L}}$, $c \in \beta \wedge \sim \varphi(c) \in$?;

(v) ? is $\mathcal{L}$-ω-complete.

Lemma 3.4.1 (Saturation Lemma I) *Let ? be a set of $\mathcal{L}$-sentences, and φ a $\mathcal{L}$-sentence. Suppose ? $\nvdash \varphi$. Let $C = \{c_0, c_1, c_2, ...\}$ be a countable set of constants foreign to $\mathcal{L}$, B a set of bounders of $\mathcal{L} \cup C$, and $\mathcal{L}' = \mathcal{L} \cup C \cup B$. Then there is a $\mathcal{L}'$-saturated set ?$_\omega$ such that ? $\subseteq$?$_\omega$ and ?$_\omega \nvdash \varphi$.*

Proof. In order to obtain required ?$_\omega$, we define two sequences $\langle ?_i\rangle_i$ and $\langle\Delta_i\rangle_i$ by induction as follows. Let $\langle\varphi_i\rangle_i$ enumerate all $\mathcal{L}'$-sentences, and $\langle\varphi_{i,1} \vee \varphi_{i,2}\rangle_i$, $\langle\forall x \in \beta_i\varphi_i(x)\rangle_i$ and $\langle\sim \forall x \in \beta_i\varphi_i(x)\rangle_i$ enumerate with infinite repetition all disjunctive, bounded universal and bounded existential sentences of $\mathcal{L}'$ respectively.

Let $?_0 = ?$ and $\Delta_0 = \{\varphi\}$. Suppose that $?_k$ and Δ_k have been defined. To define $?_{k+1}$ and Δ_{k+1}, we distinguish the following five cases.

Case 1. $k = 4n$, $?_k \vdash \varphi_{n,1} \vee \varphi_{n,2}$, and $\varphi_{n,1} \notin ?_k$ and $\varphi_{n,2} \notin ?_k$. Put

$$\begin{aligned} ?_{k+1} &= ?_k \cup \{\varphi_{n,i}\}, \\ \Delta_{k+1} &= \Delta_k, \end{aligned}$$

where i is the least of $\{1, 2\}$ such that $?_k \cup \{\varphi_{n,i}\} \nvdash \Delta_k$.

Case 2. $k = 4n + 1$. $?_k \vdash \sim \forall x \in \beta_n \varphi_n(x), \Delta_k$ and for all constants $c \in C_{\mathcal{L}'}$, $(c \in \beta_n \wedge \sim \varphi_n(c)) \notin ?_k$. Put

$$\begin{aligned} ?_{k+1} &= ?_k \cup \{c_k \in \beta_n \wedge \sim \varphi_n(c_k)\}, \\ \Delta_{k+1} &= \Delta_k, \end{aligned}$$

where c_k is the first member of $C_{\mathcal{L}'}$ not to occur in $\varphi_n(x)$ or in any member of $?_k$ or of Δ_k.

Case 3. $k = 4n + 2$, there are two subcases.

Case 3.1. $?_k, \varphi_n \vdash \Delta_k$. Put

$$\begin{aligned} ?_{k+1} &= ?_k, \\ \Delta_{k+1} &= \Delta_k \cup \{\varphi_n\}; \end{aligned}$$

Case3.2. $\varphi_n \notin ?_k$ and $?_k, \varphi_n \nvdash \Delta_k$. Put

$$\begin{aligned} ?_{k+1} &= ?_k \cup \{\varphi_n\}, \\ \Delta_{k+1} &= \Delta_k. \end{aligned}$$

Case 4. $k = 4n + 3$. $?_k, \forall x \in \beta_n \varphi_n(x) \vdash \Delta_k$, and for all constants $c \in C_{\mathcal{L}'}$, $(\sim c \in \beta_n \vee \varphi_n(c)) \notin \Delta_k$. Put

$$\begin{aligned} ?_{k+1} &= ?_k, \\ \Delta_{k+1} &= \Delta_k \cup \{\sim c_k \in \beta_n \vee \varphi_n(c_k)\}, \end{aligned}$$

where c_k is the first member of $C_{\mathcal{L}'}$ not to occur in $\varphi_n(x)$ or in any member of $?_k$ or of Δ_k.

Case 5. None of the cases above applies, put

$$\begin{aligned} ?_{k+1} &= ?_k, \\ \Delta_{k+1} &= \Delta_k. \end{aligned}$$

It is then not difficult to check by induction that for any $k \in \omega$, $?_k \nvdash \Delta_k$ using the derived rules for **CF′**. To illustrate, let us consider case 3.1. We need to show that if $?_k, \varphi_n \vdash \Delta_k$, then $?_k \nvdash \Delta_k \cup \{\varphi_n\}$. Suppose $?_k \vdash \Delta_k \cup \{\varphi_n\}$. We assume that φ_n, Δ_k and $\Delta_k \cup \{\varphi_n\}$ are the same set of formulas. By rule T and rule $\vee$I, we have $?_k \vdash \varphi_n \vee \varphi_n, \Delta_k$. Since we are assuming that $?_k, \varphi_n \vdash \Delta_k$, it follows that $?_k \vdash \Delta_k$ by rule $\vee$E. But this contradicts the induction hypothesis. So we have $?_k \nvdash \Delta_k \cup \{\varphi_n\}$.

Now let $?_\omega = \cup\{?_k : k \in \omega\}$ and $\Delta_\omega = \cup\{\Delta_k : k \in \omega\}$. We can show that $?_\omega \nvdash \Delta_\omega$, $?_\omega = F_{\mathcal{L}'} \perp \Delta_\omega$ and $?_\omega$ is $\mathcal{L}'$-saturated as desired. The details of verification are omitted.

Lemma 3.4.2 (Saturation Lemma II) *Let $?$ be a set of $\mathcal{L}$-sentences, and φ and ψ $\mathcal{L}$-sentences, and $B^P_{\mathcal{L}}$ all the persistent bounders in $\mathcal{L}$. Suppose $\varphi \supset \psi \notin ?$. Let $C = \{c_0, c_1, c_2, ...\}$ be a countable set of constants foreign to $\mathcal{L}$, B a set of bounders of $\mathcal{L} \cup C$, and $\mathcal{L}' = \mathcal{L} \cup C \cup B$. Then there is a $\mathcal{L}'$-saturated set $?_\omega$ such that $?^P \subseteq ?_\omega$,*

$\varphi \in ?_\omega$ but $\psi \notin ?_\omega$ and $(\sim c_j \in \beta_i^P) \in ?_\omega$ for any $c_j \in C, \beta_i^P \in B_{\mathcal{L}}^P$.

Proof. The proof is similar to that of Saturation Lemma I except that this time we let $?_0 = ?^P \cup \{\varphi\} \cup \{\sim c_j \in \beta_i^P : c_j \in C \ \&\ \beta_i^P \in B_{\mathcal{L}}^P\}$ and $\Delta_0 = \{\psi\}$.

Definition 3.4.2 (Canonical Model Construction) Let $C_1, C_2, C_3, ...$ be a countable sequence of disjoint countable sets of constants foreign to $\mathcal{L}$. Let C_n^* be $C_1 \cup C_2 \cup ... \cup C_n$, and B_n a set of bounders of $\mathcal{L} \cup C_n^*$ such that $B_l \subseteq B_m$ for any $l \leq m \leq n$. Then for language $\mathcal{L}_\omega = \mathcal{L} \cup (\cup C_n) \cup (\cup B_n)$, we can define a Kripke model $\mathcal{M} = \langle S, \prec, D, I \rangle$ as follows:

(i) S consists of all ? such that for some n, $\mathcal{L}_\Gamma = \mathcal{L} \cup C_n^* \cup B_n$, and ? is $\mathcal{L}_\Gamma$-saturated.

(ii) for any $\mathcal{L}_\Gamma$-saturated set ? and $\mathcal{L}_\Delta$-saturated set Δ with $\mathcal{L}_\Gamma = \mathcal{L} \cup C_m^* \cup B_m$ and $\mathcal{L}_\Delta = \mathcal{L} \cup C_n^* \cup B_n$ $(m < n)$, $? \prec \Delta$ if and only if $?^P \subseteq \Delta$ and for any $c \in C_n^* \perp C_m^*$ and $\beta^P \in B_{\mathcal{L}_\Gamma}$, $(\sim c \in \beta^P) \in \Delta$.

(iii) if ? is $\mathcal{L}_\Gamma$-saturated and $\mathcal{L}_\Gamma = \mathcal{L} \cup C_n^* \cup B_n$ then $D(?) = C_{\mathcal{L}} \cup C_n^*$.

(iv) $I_\Gamma(c) = \begin{cases} c & \text{if } c \in C_{\mathcal{L}} \cup C_n^*; \\ \text{undefined} & \text{otherwise.} \end{cases}$

(v) $I_\Gamma(\beta) = \{c \in C_{\mathcal{L}} \cup C_n^* : (c \in \beta) \in ?\}$.

(vi) $I_\Gamma(R)(c_1, c_2, ..., c_n) = \begin{cases} \text{T} & \text{if } R(c_1, c_2, ..., c_n) \in ?; \\ \text{F} & \text{if } \sim R(c_1, c_2, ..., c_n) \in ?; \\ \text{undefined} & \text{otherwise.} \end{cases}$

Lemma 3.4.3 (Truth Lemma) *Suppose $\mathcal{M} = \langle S, \prec, D, I \rangle$ is a canonical Kripke model associated with $\mathcal{L}$. Then for all $? \in S$, and all $\mathcal{L}_\Gamma$-sentences χ, we have*

$$? \models^+ \chi \textit{ if and only if } \chi \in ?.$$

Proof. By induction on the complexity of χ.

Case 1. χ is an atomic sentence $R(c_1, c_2, ..., c_n)$ or $c \in \beta$, the lemma holds by the definition of a canonical Kripke model.

Case 2. χ is an atomic sentence $\sim R(c_1, c_2, ..., c_n)$, the lemma holds again by the definition of a canonical Kripke model. If χ is $\sim c \in \beta$, suppose that $? \models^{+} \sim c \in \beta$, that is $? \models^{-} c \in \beta$. By definition, either $I_\Gamma(c)$ is not defined or $I_\Gamma(c)$ is defined and $I_\Gamma(c) \notin I_\Gamma(\beta)$. In either case, $(c \in \beta) \notin ?$. By axiom A19 and saturatedness of $?$, we get $(\sim c \in \beta) \in ?$. For converse, let $(\sim c \in \beta) \in ?$. By axiom A9 and the consistency of $?$, we get $(c \in \beta) \notin ?$. From this it follows that $? \models^{+} \sim c \in \beta$.

Case 3. χ is $\varphi \vee \psi$. Straightforward and thus omitted.

Case 4. χ is $\sim (\varphi \vee \psi)$. Straightforward and thus omitted.

Case 5. χ is $\varphi \supset \psi$. Suppose $\varphi \supset \psi \in ?$. We show $? \models \varphi \supset \psi$. For any Δ such that $? \prec \Delta$, we have $?^P \subseteq \Delta$. Since $\varphi \supset \psi \in ?$ and $\varphi \supset \psi$ is persistent, we get $\varphi \supset \psi \in ?^P \subseteq \Delta$. It follows that if $\varphi \in \Delta$, then $\psi \in \Delta$ by rule $\supset$ E. By the hypothesis of induction, then, for all such Δ, if $\Delta \models \varphi$, then $\Delta \models \psi$; and therefore $? \models \varphi \supset \psi$.

Conversely, suppose $\varphi \supset \psi \notin ?$, then $\varphi \supset \psi \notin ?^P$, so $?^P \cup \{\varphi\} \nvdash \psi$ by rule $\supset$ I. Using the Saturation Lemma II, we can get a saturated set $\Delta \in S$ such that $? \prec \Delta, \varphi \in \Delta$, but $\psi \notin \Delta$. By the hypothesis of induction, we get $\Delta \models \varphi$ but $\Delta \not\models \psi$. Thus $? \not\models \varphi \supset \psi$.

Case 6. χ is $\sim (\varphi \supset \psi)$. $? \models^{+} \sim (\varphi \supset \psi)$ if and only if $? \models^{-} \varphi \supset \psi$ if and only if $? \models^{+} \varphi$ and $? \models^{-} \psi$ if and only if $? \models^{+} \varphi$ and $? \models^{+} \sim \psi$, and this if and only if $\varphi \in ?$ and $\sim \psi \in ?$ by the hypothesis of induction. But $\varphi \in ?$ and $\sim \psi \in ?$ if and only if $\sim (\varphi \supset \psi) \in ?$ by rules $\sim\supset$ I and $\sim\supset$ E.

Case 7. χ is $\sim\sim \varphi$. Straightforward and thus omitted.

Case 8. χ is $\forall x \in \beta \varphi(x)$. Suppose $? \models^{+} \forall x \in \beta \varphi(x)$, then for all $c \in D(?)$, if $? \models^{+} c \in \beta$, then $? \models^{+} \varphi(c)$. But $? \models^{+} c \in \beta \vee \sim c \in \beta$. It follows that for all $c \in D(?)$, $? \models^{+} \sim c \in \beta \vee \varphi(c)$, so $(\sim c \in \beta \vee \varphi(c)) \in ?$ by the hypothesis of induction. Thus $\forall x \in \beta\varphi(x) \in ?$ by the $\mathcal{L}_\Gamma$-ω-completeness of ?. Conversely, suppose $\forall x \in \beta\varphi(x) \in ?$, then for any $c \in D(?)$, $(\sim c \in \beta \vee \varphi(c)) \in ?$ by rule $\forall$E, so $\sim c \in \beta \in ?$ or $\varphi(c) \in ?$ by the saturatedness of ?. Since ? is consistent, if $c \in \beta \in ?$, then $\sim c \in \beta \notin ?$, so $\varphi(c) \in ?$. That is, for any $c \in D(?)$, if $? \models^{+} c \in \beta$ then $? \models^{+} \varphi(c)$ by the hypothesis of induction, so $? \models^{+} \forall x \in \beta\varphi(x)$.

Case 9. χ is $\sim \forall x \in \beta\varphi(x)$. The proof is similar to that for case 8 except that we use condition (iv) of $\mathcal{L}_\Gamma$-saturatedness of ? and rule $\sim \forall$I, completing the proof.

Theorem 3.4.1 (Strong Completeness for CF′) *Let φ be a $\mathcal{L}$-sentence and ? a set of $\mathcal{L}$-sentences. If $? \vDash \varphi$ then $? \vdash \varphi$.*

Proof. Suppose $? \nvdash \varphi$. By canonical model construction, we can associate $\mathcal{L}_\Gamma$ with a canonical Kripke model $\mathcal{M} = \langle S, \prec, D, I\rangle$. Saturation Lemma I then guarantees us that there is a $\Delta \in S$ such that $? \subseteq \Delta$ and $\varphi \notin \Delta$. By Truth Lemma, $\Delta \models ?$ but $\Delta \not\models \varphi$. Therefore, $? \nvDash \varphi$.

3.5 First-Order Logic CF′ and Infon Logic

The main result of this chapter is the proposal of a first-order logic **CF**′ with strong negation and bounded static quantifiers, which is a variant of Thomason's logic **CF**. Different from most constructive logics, quantifiers in our system as in Thomason's are static rather than dynamic. Our intention is to develop **CF**′ further so that it can serve as a logic for situation theory.

Originally, situation theorists were not much concerned with developing their own logical systems. Their semantic theory of consequence emphasised the external significance of language and the role of non-linguistic contexts. Consequence is for them no longer a relation between syntactic elements. There is no exact correspondence between the information conveyed by an utterance and the sentence used to convey. In fact "... there can be no syntactic counterpart, of the kind traditionally sought in proof theory and theories of logical form, to the [situation] semantic theory of consequence." (see pp. 44-45 of Barwise and Perry [11]). However the desire to use situation theory and situation semantics to give an account of inference eventually led Barwise and Etchemendy to construct a situation theoretical model of inference, emphasising information content. They called this *infon logic*; that is a logic whose elementary formulas represent items of information and whose compounds correspond to ways of compounding those items (see Barwise and Etchemendy [10], [32]).

In [10], Barwise and Etchemendy argue that infon logic is at least intuitionistic but not classical. This argument is problematic. There are at least two issues associated with the argument as we pointed out in the first chapter. One is about what kind of negation is used in infon logic. Is intuitionistic or strong negation used in infon logic? The other is about the form of universal quantifiers. Are dynamic or static quantifiers used in infon logic? We have argued in the first chapter that negation in infon logic should be strong negation rather than intuitionistic one. As to universal quantifiers, from the situation-theoretic viewpoint, static universal quantifiers are more desirable than dynamic ones for the reasons we discussed in this chapter. In fact, quantifiers in related situation theoretical literature are interpreted in one way or another statically rather than dynamically (see p. 271 of [8], pp. 134-136 of [32], and p. 109 of [38]) though they are not treated in Barwise and Etchemendy's infon logic [10]. Since it is intuitionistic negation and dynamic universal quantifiers that are used in intuitionistic logic, we conclude that infon logic cannot be intuitionistic either.

Instead, we are inclined to use constructive negation, more generally, to use constructive logic with strong negation as the underlying logic for situation theory but to interpret quantifiers statically rather than dynamically. That is the way we arrive at the logic **CF**$'$ from situation theorists' work on infon logic. However, we do not claim that our logic is fully-fledged. For one thing, the components in a basic formula $R(a_1, a_2, ..., a_n)$, or using the notation of infon logic, $\ll R, a_1, a_2, ..., a_n; i \gg$ are still individuals whereas infon logic allows them to be any objects. Nevertheless, we do intend to claim that our logic preserves many features of infon logic since (i) **CF**$'$ is partial in the sense that a formula can be neither true nor false; (ii) it has a rich algebraic structure of persistent formulas; (iii) with strong negation available, **CF**$'$ has in fact two kinds of basic formula very similar to the two kinds of basic infons of situation theory; (iv) the negation of compound formulas satisfies DeMorgan's laws which are assumed to hold in situation theory; (iv) quantifiers in **CF**$'$ are static, as is consistent with the situation theoretical interpretation of quantifiers.

Chapter 4

Deductive Databases

From this chapter onwards, we are going to discuss both nonmonotonic negation and strong negation in deductive databases. The discussion will be in the more general context of logic programming. Our emphasis, however, is on the database viewpoint. In the present chapter, we first of all briefly consider one of limitations in expressive power under the conventional relational model. This should motivate the deductive approach to databases. Then we give the formal definitions of a logic program and a deductive database. Finally, we consider one of central problems in the deductive approach, that is the problem of negation, which is closely related to nonmonotonic reasoning.

4.1 From Relational to Deductive Databases

In the area of database theory, the relational model, introduced by Codd in 1970s in a series of papers [26], [27], [29], [28], [30], has become dominant. As its name suggests, the model has relations as its data structures. Based upon relational algebra or the equivalent relational calculus, manipulation of data in a relational database is a model-theoretic process (see [77] and [92]): the database is treated as an interpretation of the query language, and queries and integrity constraints are logical formulas that are to be evaluated over the interpretation using the well-known Tarski's truth definition (see [13]). A typical characteristic of a relational system is its declarativeness: the user may query or update a database in a *declarative* way by saying *what* is wanted, rather than in a *procedural* way by saying *how* the operation is computed. With its simplicity and declarativeness, the relational database model has enjoyed widespread success. The leading database vendors produce relational systems[1], including IBM's DB2, Oracle, Informix, INGRES, Sybase, and Microsoft Access.

However, it has long been recognised that the relational model is not perfect and has one or another limitation (see [61] and [92]). One conspicuous deficiency is its expressive power[2]. In order to see this, we recall that non-recursive datalog with negation is one of three syntactically different but equivalent relational query languages, the other two being the relational algebra and relational calculus [1]. In the non-recursive datalog with negation, recursion is explicitly forbidden, that is to say the relational model itself does not provide any facilities for expressing general rules. As a result, the transitive closure of a relation cannot be defined in a relational query language without interfacing with a host procedural language. Indeed, practical relational query languages like Structured Query Language (SQL) are usually embedded in full programming languages such as C programming language to express recursion (see [61] and [1]).

Although with SQL being embedded in a full programming language, SQL statements are allowed to be coupled with host language programs, and thus recursion can be provided by "while" loops in the host language; nevertheless, the declarativeness of the relational systems is thereby compromised.

[1]These systems may be object-oriented as well.

[2]For the study of the expressive power of a relational query language in a formal framework, see [1] and citations there.

There has been much effort devoted to extending the relational model instead of coupling a relational query language with a host programming language. Deductive database systems are one such extension (see [44], [92], [71], [15], [87]). In comparison to relational systems, deductive systems adopt a proof-theoretic paradigm rather than the model-theoretic one[3]. Accordingly, a database is not taken as an interpretation but as a theory consisting of a set of first-order sentences, and executing a query or satisfying an integrity constraint is regarded as proving that some specified formula is a logical consequence of the theory (see [77] and [92]).

Another important thrust to the development of deductive databases is owed to logic programming, more generally to automatic theorem proving. See [71] for a historical introduction of the development of deductive databases, and the relationship between deductive databases and logic programming. Indeed, deductive databases can be seen as the integration of relational databases with logic programs, with relational query languages being extended to logic programming languages. For non-recursive datalog with negation, this means that it is extended to (recursive) datalog with negation (see [1]).

With the presence of general rules in deductive databases, the transitive closure of a relation can be recursively defined in a straightforward way, and thus it becomes possible to query a deductive database about recursively defined relations. So, deductive systems provide a simple solution to the weak expressive power of the relational model. The increased expressive power of deductive database systems, as pointed out in [87], "is important in a variety of application domains, including decision support, financial analysis, scientific modeling, various applications of transitive closure (e.g. bill-of-materials, path problems), language analysis, and parsing." (see also [88]).

In addition to the stronger power, the proof-theoretic approach to database the-

[3]In [92], it is shown that how the model-theoretic perspective on databases can be reinterpreted in purely proof-theoretic terms.

ory against the model-theoretic one has logic as a single uniform formalism for describing facts and rules in databases, queries and integrity constraints, and for a variety of other apparently different problems, including verification of integrity constraints, program correctness proofs and many others (see [44]). More importantly, with logic programming languages as declarative database query languages, deductive database systems preserve the important property of being declarative.

The proof-theoretic approach to databases despite its merits does have its own difficulties to overcome. Before we move on to discuss the problem of negation in deductive databases, we give the main concepts and notations.

4.2 Logic Programs and Deductive Databases

From the database viewpoint, deductive databases are an extension of the traditional relational databases. In a deductive database, there are not only ground atoms, which correspond to tuples of the relations in relational databases, but also general rules, which constitute the extended part.

From the logic programming viewpoint, deductive databases can also be equally seen as logic programs. A major difference between deductive databases and logic programs is that deductive databases are usually restricted to function-free languages. Another difference between deductive databases and logic programs is that databases usually have more facts than rules whereas logic programs are other way around. See [1] for a brief summary of the differences between two fields. From a theoretical viewpoint, these differences are inessential and can be ignored. So deductive databases and logic programs are often used interchangeably.

Although there is no great difference between deductive databases and logic programs at the theoretical level, we shall nonetheless define logic programs and deductive databases in a slightly different way. This as we shall see mainly is concerned with the representation of negative information.

Both deductive databases and logic programs use the language of first-order logic as their underlying language. But, the terminology and notations used are to great extent different from that in first-order logic. Following the convention of logic programming, we shall use comma "," for conjunction, use "not" for non-monotonic negation, and use uppercase letters $X, Y, Z, ...$ for variables, and lowercase letters for predicates, constants and functions if there are any. The word *ground* is used to mean variable-free.

Definition 4.2.1 A *normal logic program* or simply a *program* , consists of a finite set of rules of the form

$$p :\perp q_1, q_2, ..., q_m, not\ q_{m+1}, \ldots, not\ q_n.$$

where $m, n \geq 0$, and p and q_i are atoms. p is called the head of the rule, the q_i make up the body of the rule. An *atom* is of form $p(t_1, t_2, ..., t_n)$, where p is an n-ary predicate symbol and $t_1, t_2, ..., t_n$ are terms which are defined as usual.

If p is an atom, then p is a *positive literal*, $not\,p$ is its *negative literal*, and they are complements of each other. For any literal l, we use $\bar{l}$ for its complement. For any rule r in a logic program, we use $head(r)$ for its head, $body(r)$ for its body. A literal $l \in body(r)$ is called a subgoal. For a set L of literals, we use L^+ for positive atoms in L, L^- for atoms whose negative literals are in L. For a set Δ of atoms, we use $not \cdot \Delta$ for the set $\{not\,p : p \in \Delta\}$. Thus, we have $body(r) = body(r)^+ \cup not \cdot body(r)^-$.

If a rule has no negative literals in its body, then it is a *definite rule*. A normal logic program is *definite* provided that the program consists of only definite rules.

Definition 4.2.2 The *Herbrand universe* of a program $\mathbb{P}$, denoted $\mathcal{H}U_P$, is the set of all ground terms constructed from only the constant symbols and function symbols in the program. In the case the program $\mathbb{P}$ does not contain a constant symbol, we add an arbitrary one to the universe $\mathcal{H}U_P$ so that $\mathcal{H}U_P$ is not empty.

The *Herbrand base* of $\mathbb{P}$, denoted $\mathcal{H}B_P$, is the set of all ground atoms constructed from predicates in $\mathbb{P}$ whose arguments are in $\mathcal{H}U_P$.

We shall assume that all rules in a logic program are ground. That is to say, we assume all variables in logic program $\mathbb{P}$ are already instantiated relative to its Herbrand universe.

Definition 4.2.3 Given a normal logic program $\mathbb{P}$, and a set L of ground literals whose atoms are in the Herbrand base $\mathcal{H}B_P$ of $\mathbb{P}$. If there is no ground atom p such that both p and $not\, p$ are in L, then L is said *conflict-free*; otherwise, it is *conflicting.*

For a conflict-free set L of literals, if L contains every atom of the Herbrand base or its negation, then L can be taken as a Herbrand interpretation (see [64]) with the understanding that missing atoms from a Herbrand interpretation are now *explicitly* represented as the negative literals in L.

A literal l is said to be *true* in a set L of literals if the literal l is in L; and *false* in L if its complement $\bar{l}$ is in L. For a Herbrand interpretation L, a rule of $\mathbb{P}$ is satisfied in L if whenever all subgoals are true in L, the head is also true in L. If every rule of $\mathbb{P}$ is satisfied in a Herbrand interpretation L, then L is a *model* of $\mathbb{P}$.

In the above definition of logic program, facts are integrated with rules; facts are taken as special rules with the empty body. In the following definition of deductive databases, we shall have facts separate from rules, dividing information into two categories: facts and rules (see [87]).

Definition 4.2.4 A *deductive database* DDB is a tuple $\langle DB, \mathbb{P}\rangle$ of a set DB of facts (also called data), and a set of rules, that is a normal program $\mathbb{P}$.

Facts are represented by ground atoms. The data DB are referred to as the *extensional database* and the normal logic program $\mathbb{P}$ as the *intensional database.* Given a deductive database $\langle DB, \mathbb{P} \rangle$, its predicates are accordingly divided into two categories: *extensional predicates* and *intensional predicates.* An extensional predicate is a predicate occurring either in DB or only in the body of a rule. An intensional predicate is a predicate occurring as the head of some rule of $\mathbb{P}$. Furthermore, there is a similar division among atoms. An extensional atom is an atom whose predicate is extensional. An intensional atom is an atom whose predicate is intensional.

An extensional predicate may occur in the body of a rule but not the head of any rule. In contrast, an intensional predicate may occur both in the head of a rule and in the body of the same rule, but if an intensional predicate occurs only in the head of a rule, we call it a *simply intensional* predicate. A simply intensional atom is an atom whose predicate is simply intensional. The notion of simply intensional atom will be used in the definition of quasi-stable semantics in Chapter 6.

Given a deductive database $\langle DB, \mathbb{P} \rangle$, one of main tasks is to determine the extensions of intensional predicates, that is the semantics of its intensional component $\mathbb{P}$ since the the meaning of its extensional component DB is clear cut: all extensional atoms are taken as true if they are in DB, and false otherwise. Based on this observation, we may as well assume that $\mathbb{P}$ is purely intensional in the sense that there are not any extensional atoms occurring in the body of rules of $\mathbb{P}$. If there is any such extensional atom p, we can always reduce $\mathbb{P}$ based upon DB to a purely intensional program in the following way:

(1) if p occurs in the body of some rule of $\mathbb{P}$,

- if p is in DB then remove it from the rule;
- p is not in DB then remove the rule since then the rule will never be fired

to infer any information;

(2) if *not* p occurs in the body of some rule of $\mathbb{P}$,

- if p is not in DB, then remove *not* p from the rule;
- if p is in DB, then remove the rule since then the rule will never be fired to infer any information.

For a purely intensional component $\mathbb{P}$ of a deductive database $\langle DB, \mathbb{P}\rangle$, it is not difficult to see that its meaning is independent of the corresponding extensional component DB. So we may restrict our discussions to logic programs.

The point of separating facts from rules in a deductive database as we mentioned earlier is concerned with the representation of negative information. This is a good place to offer an explanation of this. In the extensional component of a deductive database, positive facts are *explicitly* represented whereas negative facts are *implicitly* represented. In contrast, the same strategy of implicitly representing negative information as we shall see in the next chapter is not always adequate for the intensional component in the sense that the computation of extensions of intensional predicates may involve explicit representation of negative information. With facts and rules separated, we may still use the strategy of implicitly representing negative facts for extensional databases.

4.3 Why Negation?

Given a database, we are concerned with what can be drawn from it from the informational viewpoint. In the context of relational databases, information retrieval, as we mentioned in the beginning of this chapter, is a model-theoretic process. A relational database is taken as an interpretation. The truth of atomic queries, that is atomic queries of form p against a relational database is determined by its presence or absence in the database. The truth of more complex queries is then evaluated using Tarski's truth definition. In contrast, the information retrieval in the context of logic programs is a proof-theoretic process. A logic program is taken as a theory. The truth of a query against a deductive database is determined by its *provability* from the database. Unfortunately, provability is a very strong condition which is not suitable from the database viewpoint.

In the second chapter, with the analysis of classical negation at the semantic and proof-theoretic levels, we have shown that on the one hand, a monotonic deductive inference mechanism is not adequate for dealing with negative information *even* in the area of relational databases. This is because classical negation at the proof-theoretic level expresses inconsistency which is monotonic whereas negative information concerned in the area of databases is usually nonmonotonic. On the other hand, the semantic rule M2 does provide us with a mechanism for deriving desired negative information in the context of relational databases.

Unfortunately, the semantic rule M2 is no longer valid in the context of deductive databases. Consider the following logic program $\mathbb{P}$, which is an extension of the relational database we discussed in the second chapter:

$$
\begin{array}{l}
teach(frege, \textit{first-order-logic}). \\
teach(cantor, \textit{set-theory}). \\
teach(tarski, \textit{model-theory}). \\
teach(turing, \textit{recursion-theory}). \\
teach(godel, \textit{proof-theory}). \\
teach(tarski, \textit{proof-theory}). \\
teach(turing, \textit{proof-theory}). \\
teach(X, \textit{set-theory}) :\perp\ teach(X, \textit{model-theory}). \\
teach(X, \textit{set-theory}) :\perp\ teach(X, \textit{recursion-theory}). \\
teach(X, \textit{set-theory}) :\perp\ teach(X, \textit{proof-theory}).
\end{array}
$$

and two queries to it,

$$
\begin{array}{ll}
q_1 : & \text{Who does not teach } \textit{proof-theory}? \\
q_2 : & \text{Who does not teach } \textit{set-theory}?
\end{array}
$$

Intuitively, we want $\{frege, cantor\}$ to be the answer to q_1 since $\mathbb{P}$ contains neither $teach(cantor, \textit{proof-theory})$ nor $teach(frege, \textit{proof-theory})$. But the final rule of $\mathbb{P}$ is only concerned with *set-theory* instead of *proof-theory*, so it cannot be used to infer any information about $teach(cantor, \textit{proof-theory})$ or $teach(frege, \textit{proof-theory})$. However, first-order logic itself does not enable us to reach the intended answers to the query q_1. To logically infer that $\neg teach(cantor, \textit{proof-theory})$ and $\neg teach(frege, \textit{proof-theory})$, we have to show that $teach(frege, \textit{proof-theory}) \vee teach(cantor, \textit{proof-theory})$ is not consistent with $\mathbb{P}$. From the given information, however, this is impossible since we can find a model for $teach(frege, \textit{proof-theory}) \vee teach(cantor, \textit{proof-theory})$ and $\mathbb{P}$. So we cannot logically reach the conclusion.

As to the second query q_2, we want $\{frege\}$ to be the answer to it. Using the same argument as for q_1, we know that this conclusion cannot be logically reached either. But this time, we do not want to include *tarski*, or *turing* or *godel* in the answer. Although $\mathbb{P}$ does not directly contain *teach*(*tarski*, *set-theory*), *teach*(*turing*, *set-theory*), or *teach*(*godel*, *set-theory*), nevertheless, they can be inferred logically from $\mathbb{P}$ using the final rule. That is to say, in the context of logic programs, we cannot obtain desired negative information by simply employing a simple rule like M2. In order to obtain intended answers to q_1 and q_2, we have to seek a different inference mechanism.

The logic program we just discussed is only an example of datalog. As we shall see, the problem of negation in datalog is not very difficult. The real challenge comes from logic programs with negation in their bodies, especially when negation is involved in recursion. In order to see such logic programs do exist in practical applications, let us consider an example, which is based on similar examples from [105] and [1], It is worth pointing out that the example is one of the motivational examples for both the stable model semantics [47] and the well-founded semantics [105], and also closed related to a game described by Kolaitis [60].

Example 4.3.1 Consider a game between two players. The game consists of a series of states. The possible moves of the game are held in an extensional relation *moves*, where *moves*(a, b) means that a player, when in state a, may choose to move to b.

A player wins if the opponent has no moves. Let *winning*(X) denote X is a winning state; that is a state has a winning strategy for a player. Then the winning rule can be expressed by the following program $\mathbb{P}_w$:

$$winning(X) :\perp move(X, Y),\ \mathit{not}\ winning(Y).$$

Suppose the game has three states a, b, c, with the following sample instance of relation *moves*:

$$moves = \{(a, b), (b, a), (b, c)\}$$

which is graphically represented as

$$\begin{tikzcd}
a \arrow[r, bend left] & b \arrow[l, bend left] \arrow[r] & c
\end{tikzcd}$$

The Herbrand instantiation of the program is as follows:

$$winning(a) :\perp move(a, a),\ not\ winning(a).$$
$$winning(a) :\perp move(a, b),\ not\ winning(b).$$
$$winning(a) :\perp move(a, c),\ not\ winning(c).$$
$$winning(b) :\perp move(b, a),\ not\ winning(a).$$
$$winning(b) :\perp move(b, b),\ not\ winning(b).$$
$$winning(b) :\perp move(b, c),\ not\ winning(c).$$
$$winning(c) :\perp move(c, a),\ not\ winning(a).$$
$$winning(c) :\perp move(c, b),\ not\ winning(b).$$
$$winning(c) :\perp move(c, c),\ not\ winning(c).$$

where there are three rules in which *winning* depends negatively on itself. According to the winning rule, it is not difficult to see that $winning(b)$ should be true but not $winning(a)$ and $winning(c)$. But, how shall we compute the set of winning states for any instance of the relation *moves*? What we need is a general mechanism to determine winning states. It turns out that in this case, the well-founded semantics [105] does provide us with such a mechanism to find the intended answer. □

The treatment of negative information in deductive databases, more generally in logic programming, turns out to be extremely difficult, especially when negation is involved in recursion. In the past two decades, much effort has been devoted to the research about negation in deductive databases and in logic programming by proposing various kinds of semantic theories for logic programs. Indeed, as pointed out in [87], a very important thrust to the study of deductive databases and logic programming, "has been the problem of coping with negation or non-monotonic reasoning, where classical logic does not offer, through the conventional means of logical deduction, an adequate definition of what some very natural logical statements 'mean' to the programmer."

4.3.1 Nonmonotonicity of *not*

Since logic does not help, it is inevitable for us to devise some non-logical mechanism to deal with the use and retrieval of negative information in the context of logic programs. More generally, we have to define what can be inferred from a logic program. From the semantic perspective, this is the same as determining what is true and what is false relative to a given logic program. Consequently it will also provide an interpretation of *not*. Different semantics may have different interpretations of *not* in one way or another (see the next chapter for more details). This roughly explains why we said in the first chapter that the exact meaning of a piece of negative information represented by *not p* depends on the specific semantics used. Whatever semantics is used, however, nonmonotonicity is a common and essential characteristic of this kind of negation.

It is interesting to note that the nonmonotonicity can in fact be envisaged in advance independently of any semantics. Indeed, from the database viewpoint, the nonmonotonicity of *not* can be argued quite directly. In relational databases, it is the semantic notion of classical negation that we use to deal with negative information in evaluating a query expressed in non-recursive datalog with negation. But we have seen that classical negation at the semantic level is nonmonotonic, so is the negation *not* used in non-recursive datalog. Equivalently, in the relational algebra[4]

[4]Recall the relational algebra and the non-recursive datalog with negation are two of three different but equivalent relational query languages [1].

negation is expressed by exploiting the set difference operation. The R's that are not S's are just $R \perp S$. This notion of negation is clearly non-monotonic, for initially x may be in $R \perp S$ but when more is learned of x we may add $x \in S$ and so lose the previous negative item of information. When we extend non-recursive datalog with negation to allow recursion we should preserve this non-monotonicity so that, whenever the semantics is applied within a non-recursive datalog program, we still have the same result as we would from a relational database using the conventional relational theory. Thus *not* has to remain nonmonotonic.

Combining the examples in the previous section with the above argument, we can now formulate the relationship between nonmonotonic negation and classical negation. The examples in the previous section shows that nonmonotonic negation *not* should be different from classical negation at both the semantical and proof-theoretical levels. On the one hand, compared with classical negation at the semantical level, nonmonotonic negation *not* should be more general so that *not* will be suitable in the context of logic program, a wider context than that of relational databases. Moreover, the nonmonotonicity argument above shows that nonmonotonic negation should be in fact an extension of classical negation at the semantic level. On the other hand, compared with classical negation at the proof-theoretical level, nonmonotonic negation *not* should be much weaker so that the negative information required from the database viewpoint can be adequately inferred and expressed by *not*. As a result, nonmonotonic negation should also to some extent extend classical negation at the proof-theoretic level in the context of logic programs. We shall see in the next chapter, such a proof-theoretic extension of classical negation is achieved through a kind of nonmonotonic reasoning introduced in a semantics of logic programs.

4.3.2 Strong Negation in Logic Programs

From the knowledge representation viewpoint, non-monotonic negation can only represent indefinite negative knowledge. A piece of nonmonotonically negative information might be given up later when more information is available. But there are situations where such a negation is not adequate. When definitely negative knowledge is required, a stronger negation seems indispensable.

Let us consider an example from [3]:

$$convicted(X) : \perp charged(X), \sim innocent(X).$$

The intended interpretation to the above formula is that a person charged with a crime should be convicted if there is a direct evidence against him or her. Here we cannot replace $\sim$ with non-monotonic negation. Otherwise, it would mean that a person charged with a crime should be convicted when there is no evidence showing that he or she is innocent. This is certainly not consistent with judicial practice.

Since non-monotonic negation is not suitable in the above example, we have to use a different one. Obviously, this different kind of negation has to be at least monotonic so that when more information is added, what is originally negated should not be overturned[5]. Classical negation at the proof-theoretical level is a candidate. But the semantic principle of bivalence behind the classical negation makes it not suitable. In addition, there are also computational difficulties associated it (see [63]).

Without the restriction of bivalence, the classical negation becomes intuitionistic. So, another choice may as well be intuitionistic negation. But in chapter 3 we have shown that intuitionistic negation is less desirable than the strong negation of constructive logics. In logic programming and deductive databases, the application of strong negation for explicit representation of negative information is originally proposed in [79]. See also [108], [109], [110]. Gelfond and Lifschitz in [48], [49]

[5]We are assuming that the evidence supporting the negation correct and reliable.

also proposed to extend logic programs with classical negation from the knowledge representation viewpoint. But, as it is pointed out in [109], what is named classical negation in [48] is in fact strong negation.

Back to our example, it turns out that strong negation can perfectly capture the negation in question. A direct evidence against a person p will enable us to explicitly assert that he or she is not innocent, that is $\sim innocent(p)$. One might suggest to replace $\sim innocent(X)$ with $guilty(X)$ to eliminate the use of strong negation at all. The problem with the replacement, as it is pointed out by Wagner [109], is that we will lose reasoning capabilities based on inconsistency.

Augmented with strong negation, logic programming can represent both definitely positive and definitely negative knowledge. Without the restriction of bivalence, logic programs with strong negation may also deal with incomplete information in the sense that the answer to a ground query need not be yes or no. It may be neither. Motivated by the above, we shall also study extended logic programs, that is logic programs extended by strong negation, after we study logic programs.

Chapter 5

Non-monotonic Negation

A great number of interesting semantic theories for logic programs with nonmonotonic negation have been proposed in the past two decades. The purpose of this chapter is to review several main semantic theories for logic programs, showing how non-monotonic negation is dealt with in them. The choice of reviewed semantic theories is mainly based on their relevance to our proposed semantics in the next chapter. So it is by no means intended to be a comprehensive overview of the area, and does not imply any prejudice to other theories either. Readers are invited to refer to [64] for an elementary introduction to logic programming, and [15], [72], [33], and [34] for detailed surveys on negation in logic programming.

The semantics reviewed here include the semantics of definite logic programs mainly by van Emden and Kowalski [103], Clark's program completion semantics and the associated SLDNF[25], Reiter's closed world assumption [90], Fitting's semantics [40], the well-founded semantics by Gelder, Ross and Schlipf [105], stable semantics by Gelfond and Lifschitz [47], and a three-valued version of the stable model semantics by Przymusinski [85]. The review is discussed in the context of logic programs. We shall show that these semantics in spite of their shortcomings in one way or another do give us an instructive insight towards understanding nonmonotonic negation.

5.1 Introduction

Semantics of Logic programs can be dealt with using many different approaches. Here we concentrate on two different but related interpretations: the model-theoretic interpretation and the deductive one. The model-theoretic interpretation, declarative in nature, is to take a logic program as a first-order theory. To define the semantics for the program is to choose some models of this theory and use them to assign a meaning to the program.

The deductive interpretation, more procedural in nature, is to take a logic program as a set of inference rules. To define semantics for a logic program is to find out what can be inferred from the program; here inference is not necessarily logical reasoning: it may be non-monotonic reasoning or other forms. Using the terminology of default logic[1], the deductive approach is to take logic program rules as defaults. That is a logic program is taken as a default theory. Then we use concepts like extensions or weak extensions associated with the theory to provide the program with a meaning (see [66]).

Note that there are various classifications of different approaches to semantic theories for logic programs with nonmonotonic negation. In [105], semantic theories are classified into the "program completion" approach and the "canonical model" approach. In [33] they are divided into logic-programming-semantics, which is based on the idea of negation-as-finite-failure [25], and non-monotonic reasoning-semantics, which is inspired by logics of common sense reasoning such as default logic [91], and autoepistemic logic [74].

We are not attempting to give any classification of existing semantics. Instead

[1]In [16], default logic has been used a uniform formalism for defining semantics of logic programs.

we only consider the model-theoretic approach and the deductive approach with the emphasis on the latter one. We shall make use of the concept of extensions rather than that of models as much as we can. We take classical two-valued logic as our underlying logic. As we shall see in this chapter, the Fitting semantics and the well-founded semantics are in general partial in the sense that some atoms in a logic program may be neither true nor false. From the model-theoretic perspective, fixpoints in the Fitting semantics and the well-founded semantics, when taken as models, have to be viewed in the framework of three-valued logic. From the deductive perspective, they can also taken as incomplete extensions as in default logic [91]. This way, classical logic is augmented with logic programming rather than being replaced by a three-valued logic.

5.2 Semantics for Definite Logic Programs

Semantics for definite logic programs is now well established. There are three different but equivalent approaches to defining the semantics, resulting in model-theoretic semantics, fixpoint semantics, and procedural semantics[2]. Since procedural semantics involves introducing some kind of refutation procedure based on the resolution inference rule such as SLD-resolution [6], which we will not cover later in the discussion of normal logic programs, we shall only have a brief look at the model-theoretic and fixpoint semantics. See [64] for a detailed introduction to the three approaches.

The model-theoretic semantics of logic programs is based on the semantics of first-order logic. The truth value of an atom relative to a logic program $\mathbb{P}$ is determined by the logical consequences of $\mathbb{P}$, that is $\{p(t_1, \ldots, t_n) : \mathbb{P} \models p(t_1, \ldots, t_n)\}$, where $\mathbb{P} \models p(t_1, \ldots, t_n)$ means that $\mathbb{P}$ logically implies $p(t_1, \ldots, t_n)$. It is shown in [103] that the logical consequences of $\mathbb{P}$ are equal to the intersection of all Herbrand

[2]In [1], it is also called proof-theoretic semantics since it is based on obtaining proofs of facts from a logic program.

models of $\mathbb{P}$, which is itself a Herbrand model, and thus is called the *least Herbrand model* for $\mathbb{P}$. So we may reduce the logical consequences of a definite logic program to its unique least Herbrand model, and determine the truth of an atom using the least Herbrand model as follows: if an atom is in the model then it is taken as true; otherwise taken as false. So it is natural to assign the least model as the declarative meaning of a definite program.

Note that when we say an atom is taken to be false if it is not in the least model, we are moving out of the realm of logical reasoning into that of nonmonotonic reasoning. An atom is not in the least model of a definite program only means that the atom is not a logical consequence of the program, which is different from that the negation of the atom is a logical consequence of the program. When we assign the truth-value false to all the atoms not in the least model, we are implicitly using the CWA, which is nonmonotonic as we shall see later in this chapter.

The fixpoint semantics is also based upon a theorem by van Emden and Kowalski [103] (see also Apt and van Emden [6]). The theorem shows that the least Herbrand model is equal to a least fixpoint, which in turn can be characterised in a constructive way, using fixpoint theory, that is using Knaster-Tarski's and Kleene's theorems (see [6] and [64] for details). To define the fixpoint characterisation, an operator T_P on $\mathcal{H}B_P$, now usually called the immediate consequence operator [1], is introduced as follows:

$$T_P(L) = \{p \in \mathcal{H}B_P : \exists r \in \mathbb{P}(head(r) = p \wedge \forall q \in body(r) q \in L\}.$$

It is easy to prove that T_P is monotonic. In fact it can be proved more strongly that T_P is continuous. See [64] for relevant details. It then follows that T_P has a least fixpoint $lfp(T_P)$, which can be computed in a constructive way. Since we shall use the computation for other monotonic operators, we state it here as a lemma in a general form for reference later.

Lemma 5.2.1 *Assume the underlying complete lattice is either $\mathcal{H}B_P$ or ($\mathcal{H}B_P \cup$ $not \cdot \mathcal{H}B_P$) with the containment $\subseteq$ as its natural and default ordering*[3]*, where $\mathbb{P}$ is a logic program. For a monotonic operator ? on the complete lattice, the least fixpoint $lfp(?)$ can be characterised by the following:*

$$lfp(?) = \bigcup ? \uparrow \alpha$$

and there exists an ordinal β such that for any $\alpha \geqslant \beta$

$$lfp(?) = ? \uparrow \alpha$$

where

$$\begin{aligned} ? \uparrow 0 &= \emptyset. \\ ? \uparrow \alpha &= ?(? \uparrow (\alpha \perp 1)),\ \textit{if } \alpha \textit{ is a successor ordinal.} \\ ? \uparrow \alpha &= \bigcup \{? \uparrow \beta : \beta < \alpha\},\ \textit{if } \alpha \textit{ is a limit ordinal.} \end{aligned}$$

Definition 5.2.1 The *closure ordinal* for the sequence $\langle ? \uparrow \alpha \rangle_\alpha$ is the least ordinal α such that $? \uparrow \alpha = lfp(?)$.

Although it is universally accepted that for any definite logic program $\mathbb{P}$, its least Herbrand model can be used to define the meaning of $\mathbb{P}$, it is much more controversial about how to assign an appropriate meaning to a normal logic program. In comparison to a definite logic program, a normal logic program may have more than one minimal Herbrand model. For example, $P = \{p : \perp \textit{not } q\}$ has $\{p\}$ and $\{q\}$ as its minimal models.

The operator T_P is usually not monotonic and may have no fixpoint at all. For example, let $\mathbb{P} = \{p : \perp \textit{not } p\}$. T_P is not monotonic. Indeed, $T_P \uparrow \alpha$ alternates between $\emptyset$ and $\{p\}$. The program also gives us an example of T_P having no fixpoint at all. For some programs, even if T_P has a least fixpoint, the fixpoint may not

[3]Later in the discussion of 3-valued stable models, we shall see a different ordering.

be approximated by the sequence $T_P \uparrow \alpha$. See [1] for some more counterexample programs.

So, various equivalent forms of semantics for definite logic programs cannot be simply generalised to normal logic programs. In the following sections, we present some main contributions to defining semantics of normal logic programs.

5.3 Program Completion and Negation as Failure

The program completion semantics and its associated SLDNF-resolution were initiated by Clark in [25] and further developed by many others (see [64], [105] and its citations there).

The notion of a program completion[4], as its name may suggest, is to complete a program by making explicit what is implicit in the program. Specifically, the basic point is to take a rule as a partial definition for the predicate in the rule head, and use all rules with same head predicate to make a complete definition for the predicate. But this viewpoint about a program is not explicitly represented in a logic program due to the conditional form of rules in a program. The process of completing a program is to make explicit the point that a program is taken as a set of definitions.

So, given a program $\mathbb{P}$, a completed program $Comp(\mathbb{P})$ is defined as follow: replace all rules with the same predicate in the head with a single bi-conditional formula whose left operand is the head of the original rules and whose right operand is a disjunction of the original rule bodies; for predicates not in the head of any rule, a negative literal for the predicate is added. In addition, a theory of equality is also required to impose restrictions on the possible interpretations of =. For the formal details of the definition, see [25] and [64].

[4]In [25] this was called the completion of a data base.

The introduction of the completion of a program was used to provide a validation of negation as failure rule, a rule for augmenting SLD-resolution, resulting in SLDNF-resolution. It is well-known that SLD-resolution is a particular refutation procedure, but can only deal with positive literals (see [64] for details). For negative literals, some additional mechanism is required. Negation as failure is such a one. The basic idea is to define a "proof" of a negative subgoal as finite failure to obtain a proof of corresponding positive subgoal[5]. Such a way to deal with negation was not novel. As it was pointed out by Clark in [25], it had already been used in both PLANNER and PROLOG (see [25] and related citations there). Clark observed that a negated fact inferred from a program by negation as failure can also be reached through deduction by completing the original program. So he proposed to justify the use of negation as failure by reducing the rule to a derived rule of first-order logic.

In [25], it is proved that for consistent completed programs, SLDNF is sound, that is if all SLD derivations starting from p are finite and none of them produces an SLD-refutation, then *not* p is a logical consequence of the completed program. For an inconsistent completed program, the soundness is trivial. The completeness of the SLDNF, as it is pointed in [25], does not hold in general since a query may be a logical consequence of $Comp(\mathbb{P})$, and at the same time neither succeed nor fail but have an infinite derivation tree. A limited form of completeness was proved in [25] by imposing constraints on logic program and its queries. See also [64] for related discussion about completeness of SLDNF.

It has been argued that neither program completion nor SLDNF-resolution is satisfying. In comparison to the original program, the logical consequences of a completed program do indeed include negative literals as well as positive literals. But the program completion approach is completely based upon logical deduction.

[5]This implies non-determinism when infinite failure occurs.

So it is not surprising to discover that it is still a very weak inference mechanism for dealing with negative information. In addition, a completed program may be either inconsistent or unintuitive. Moreover, SLDNF is also problematic. It does not always terminate, and may run into infinite looping. And the matching with completion semantics, as we mentioned above, is not perfect.

We shall not involve SLDNF any more in our further discussion. Our interest is mainly about the completion of a program. Various existing semantics are related one way or another with completed programs. These connections will be pointed out at suitable places later.

5.4 Fitting Semantics

The Fitting semantics [40] is taken as an important result in the program-completion approach in [105]. We have seen in chapter 2 that classical logic is three-valued from the deductive viewpoint. Since Clark's program completion semantics is based upon logical deduction, it is then easy to see that this approach implicitly defines a 3-valued interpretation[6] for a consistent completed program: truth value *true* is assigned to those atom that are logical consequence of the completed program, and truth value *false* to those atoms whose negations are logical consequences, and (unknown) to all other atoms. Obviously, a 3-valued interpretation is equivalent to a set of literals in a natural correspondence.

The 3-valued interpretations were made explicitly in Fitting's semantics [40]. Based upon a 3-valued constructive logic, it is shown in [40] that the completion of every program has a unique minimum 3-valued Herbrand model, that is a set of literals. Technically, the central point behind Fitting semantics is the introduction of the following operator N_P on sets of literals, which is used to generate negative

[6]Given the fact that the truth of each literal is based on traditional 2-valued logic, a 3-valued interpretation is also awkwardly called the 2-valued program completion interpretation in [105].

facts:

Definition 5.4.1 $N_P(E) = \{p \in \mathcal{H}B_P : \forall r \in \mathbb{P}(head(r) = p \to \exists l \in body(r)\overline{l} \in E)\}$

Combining the operator N_P with the operator T_P defined in section 5.2, another operator, denoted F_P, on sets of literals can be defined as follows.

Definition 5.4.2 $F_P(E) = T_P(E) \cup not \cdot N_P(E)$

We shall call F_P the Fitting operator. The operator T_P is used to draw only positive facts whereas the operator N_P produces only negative facts. Combined together, the operator F_P can be used to conclude both negative and positive facts. It is an easy exercise to show that F_P is monotonic, and thus has a least fixpoint $lfp(F_P)$, which is also a 3-valued model of the completed program of $\mathbb{P}$ according to the following theorem.

Theorem 5.4.1 (Fitting [40]) *Given a logic program $\mathbb{P}$. A 3-valued interpretation E is a 3-valued model of the completed program of $\mathbb{P}$ if and only if $E = F_P(E)$.*

Although the original definition of the Fitting semantics was carried out in a 3-valued logic, the use of a 3-valued logic is inessential. Instead of taking the least fixpoint $lfp(F_P)$ as a 3-valued model of the completion of a program $\mathbb{P}$, and working in a 3-valued logic, Fitting's semantics can be taken as extending the inference mechanism of classical logic for deriving negative information in 2-valued logic framework, by complementing the operator T_P with the operator N_P. Such a view is no more than to take a logic program as a set of inference rules, or using the terminology from default logic [91], a set of defaults. And then the least fixpoint $lfp(F_P)$ is used to define the extension associated with the default theory corresponding to the logic program (see [16], and [67]).

Some comments are in order. First of all, the inference based upon $lfp(F_P)$ is non-monotonic. In particular, $lfp(F_P)$ extends the semantic rule M2 proof-theoretically. For any extensional atom p, if p is not in the extensional database, then the condition of N_P will hold vacuously and thus p will be included in N_P. So *not* p in $lfp(F_P)$. Consider a simple example program $\mathbb{P}$ with following two rules

$$\begin{aligned} p &\ :\perp \textit{not}\ q. \\ r &\ :\perp q. \end{aligned}$$

Obviously, $lfp(F_P) = \{\textit{not}\ q, p, \textit{not}\ r\}$. But when $\mathbb{P}$ is augmented with q, we have $lfp(F_{P\cup\{q\}}) = \{q, r, \textit{not}\ p\}$. So what is originally not true now becomes true when more information is added. The example also illustrates that the non-monotonicity applies not only to extensional atoms but intensional atoms as well, such as the atom r in the present example. Moreover, what is originally true is affected too. This is because of the use of the rule $p : \perp \textit{not}\ q$, by which p is infected with the non-monotonicity of *not* q. We shall come back to this point later when we discuss supportedness in the next chapter.

Secondly, the fact that the original definition of the Fitting semantics was carried out in a 3-valued logic itself suggests that the inference mechanism of the Fitting semantics is very weak since some atoms may have no truth value at all. As pointed out by Bidoit in [15], the Fitting semantics does not generalise the fixpoint semantics for definite programs in the sense that what is not true in the fixpoint semantics may be undefined in the Fitting semantics.

Thirdly, for finite logic programs, the closure ordinal of F_P is finite. Otherwise, the closure ordinal of F_P may be beyond ω; that is the computation of $lfp(F_P)$ may go beyond ω because the operator F_P is in general not continuous. A counterexample clarifies the point.

Example 5.4.1 Let p and q be two unary predicates, c a constant, and s a function symbol. Let $\mathbb{P}$ be a logic program with the following three rules:

$$\begin{array}{lll} q(c) & :\perp & . \\ q(s(X)) & :\perp & q(X). \\ p(c) & :\perp & not\ q(X). \end{array}$$

The Herbrand universe of $\mathbb{P}$ $\mathcal{H}U_P = \{s^0(c), s^1(c), s^2(c), \ldots, s^n(c), \ldots\}$, where $s^0(c)$ is for c, $s^1(c)$ for $s(c)$, $s^2(c)$ for $s(s(c))$, and so on. Let $\mathbb{P}'$ be the instantiated program of $\mathbb{P}$ relative to $\mathcal{H}U_P$. Then $\mathbb{P}'$ is not finite, and we have

$$\begin{array}{lll} F_{P'} \uparrow n & = & \{q(s^m(c)) : m \leqslant n\}; \\ F_{P'} \uparrow \omega & = & \{q(s^m(c)) : m < \omega\}; \\ F_{P'} \uparrow \omega + 1 & = & \{notp(c)\} \cup \{q(s^m(c)) : m < \omega\}. \\ lfp(F_{P'}) & = & F_{P'} \uparrow \omega + 1. \end{array}$$

□

Last but not least, Kunen's work should be mentioned. In [62], Kunen describes a variant, which does have at most ω iteration. The resulting 3-valued interpretation is recursively enumerable but may not be a 3-valued model. What is significant is probably that Kunen's 3-valued interpretation characterises the 3-valued logical consequences of the completed program. We emphasise the underlying two-valued logic instead of 3-valued logic, so we shall not use Kunen's 3-valued interpretation in the further discussion.

5.5 Closed World Assumption

The Closed World Assumption was introduced by Reiter in [90] in the the context of deductive databases. As we mentioned earlier, the deductive approach to databases has to deal with negative information retrieval. In order to obtain a piece of negative information, one straightforward way is to include various negative facts in addition to positive ones in EDB. Unfortunately, this is not feasible since the number of negative facts about a given domain generally far exceeds the number of positive ones (see [90]). Instead, Reiter proposes to merely explicitly represent positive facts of a database DB, and treat negative facts as implicitly present provided that their positive counterparts are not provable from the database DB. This is the so-called Closed World Assumption (CWA), or more preferably the Closed World Rule given the form of its formulation [64]. It is a rule used to infer negative information from a databases in proof-theoretic contexts.

Definition 5.5.1 Given a normal logic program $\mathbb{P}$, the CWA states that one can infer a negated atom *not* p from $\mathbb{P}$ if $\mathbb{P} \not\vdash p$:

$$CWA(\mathbb{P}) = \{not\ p : \mathbb{P} \not\vdash p\}$$

Note that, if a normal logic program $\mathbb{P}$ consists of a purely extensional database EDB, that is $\mathbb{P}$ has an empty intensional database IDB and thus there is no deduction involved, the CWA is no more than a complement operation in a relational database: $\mathbb{P}$ implicitly contains a negative fact provided its positive counterpart is not explicitly present in $\mathbb{P}$. For a general case, $\mathbb{P}$ with both EDB and IDB, a positive fact not explicitly present in EDB may still be derived by rules in IDB, so "not provable from $\mathbb{P}$" in the CWA is crucial: a negative fact is taken as implicitly present in $\mathbb{P}$ if its positive counterpart cannot be *provable* from $\mathbb{P}$.

Another important point to note is that the CWA can lead to inconsistency. For example, let $\mathbb{P}$ be a program with a single rule $p : \perp not\ q$. Then $CWA(\mathbb{P}) = \{notp, notq\}$ since we have both $\mathbb{P} \not\vdash p$ and $\mathbb{P} \not\vdash q$. But $\mathbb{P} \bigcup CWA(\mathbb{P})$ is not consistent. Since the CWA is not consistent with all programs, the problem of consistency has to be taken into account. This motivates the following definition.

Definition 5.5.2 A logic program $\mathbb{P}$ is consistent with CWA if $\mathbb{P} \bigcup CWA(\mathbb{P})$ is consistent.

A natural question is which programs are consistent with the CWA. Reiter has shown that, if a logic program $\mathbb{P}$ is definite, then $\mathbb{P}$ is consistent with CWA. More generally, it can be shown that various kinds of stratifiable programs are consistent with the CWA (see [1]).

Although the CWA can be applied to a large natural class of logic programs, such as definite logic programs and stratifiable programs, it can be argued that the CWA is not satisfactory. There are many unstratifiable programs. In fact, any programs with a recursive application of negation is not stratifiable since the stratification condition prohibits such structure. A trivial example is $\mathbb{P} = \{p \rightarrow not\, q, q \rightarrow not\, p\}$.

Instead of restricting logic programs to those consistent with the CWA, we can restrict the CWA itself so that it is consistent for all programs. In [70], Minker proposed a restriction of the CWA, called the Generalised Closed World Assumption (GCWA), which can be applied to any normal logic program[7]. To understand the GCWA, we need to know that the issues of logic program consistency with the CWA can also be addressed from a semantic point of view. This was initiated by van Emden (see [70] and its citations). He defines the notion of a "minimal model" for a program as the intersection of all its models. If this minimal model is itself a model of the program, then the program is consistent with the CWA. In [103] van Emden and Kowalski show that definite programs have a minimal model and thus are consistent with the CWA. However, the notion of minimal model is very strong in the sense that, for non-definite logic programs, the minimal model may not be a model and thus a non-definite logic program may not be consistent with the CWA. In [70], Minker generalises the notion of minimal model, defining it with respect to

[7]In [70] the GCWA is applied to both definite and indefinite logic programs. Since we are not going to discuss indefinite logic programs in this thesis, we apply the GCWA to normal logic programs.

set inclusion instead of using the intersection of models.

Definition 5.5.3 Given a logic program, a model M of P is *minimal* if no smaller subset M' of M is a model of $\mathbb{P}$.

Definition 5.5.4 Given a normal program $\mathbb{P}$, the GCWA states that *not p* may be inferred from $\mathbb{P}$ if p is not in any minimal model of $\mathbb{P}$:

$$GCWA(P) = \{not\ p : \text{for any minimal model } M \text{ of } P, p \notin M\}$$

For any atom p, if it is not in any minimal model of $\mathbb{P}$, then it is safe to assign any truth value to the atom without any inconsistency. The GCWA chooses to assign false to such atoms. As a result, the GCWA can be consistently applied to any normal logic program. Although consistency is guaranteed, the price paid by the GCWA is high. Indeed, it turns out that the GCWA is very restricted. Let us consider an example from [105].

Example 5.5.1 Let P be the following program:

$p(a) :\perp p(c), \text{not } p(b).$

$p(b) :\perp \text{not } p(a).$

$p(e) :\perp \text{not } p(d).$

$p(c).$

$p(d) :\perp q(a), \text{not } q(b).$

$p(d) :\perp q(b), \text{not } q(c).$

$q(a) :\perp p(d).$

$q(b) :\perp q(a).$

It is not difficult to see that $\mathbb{P}$ has four minimal models:

$$M_1 = \{p(a), p(c), p(e)\}$$
$$M_2 = \{p(b), p(c), p(e)\}$$
$$M_3 = \{p(a), p(c), p(d), q(a), q(b)\}$$
$$M_4 = \{p(b), p(c), p(d), q(a), q(b)\}$$

It can be seen that $q(c)$ is not in any minimal model above. According to the GCWA, $q(c)$ is false. Obviously, $p(c)$ is a logical consequence of $\mathbb{P}$, so it is a common member of all minimal models of $\mathbb{P}$ and is taken as true by the GCWA. However, the GCWA is silent on all other atoms though it seems reasonable to treat $\{p(d), q(a), q(b)\}$ as false. So, though the GCWA can be consistently applied to all the programs, its weak reasoning ability is hardly satisfactory. A more fine-grained mechanism is needed for inferring negative information. $\square$

5.6 Well-founded Semantics

The WFS [105] can be seen as another approach to avoiding the inconsistency caused by the application of the CWA to an arbitrary logic program. Instead of inferring all the negated atoms from a given program according to the unprovability of their positive counterparts, the WFS only accepts those negated atoms whose positive counterparts satisfy a stricter and more specific condition than unprovability. Since we shall use the WFS later, let us have a closer look at the definition of the WFS to see how the WFS derives negative information.

The central notion of the WFS is that of unfounded sets, which nicely combines together Fitting's approach to the derivation of negative literals and the notion of closed sets (see [105] and citations there).

Definition 5.6.1 Given a program $\mathbb{P}$, its associated Herbrand base $\mathcal{H}B_P$, and a set

L of literals[8], a set $X \subseteq \mathcal{H}B_P$ is said to be an *unfounded set* of $\mathbb{P}$ with respect to L if it satisfies :

$$X \subseteq ?_{P,L}(X),$$

where $?_{P,L}$ is defined by

$$?_{P,L}(X) = \{p \in \mathcal{H}B_P : \\ \forall r \in \mathbb{P}(head(r) = p \rightarrow (\exists q \in body(r)\overline{q} \in L \vee \exists q \in body(r)^+ q \in X))\}.$$

Informally, $?_{P,L}(X)$ expresses all the atoms p satisfying one of the following conditions:

(1) There is no rule r in $\mathbb{P}$ such that $head(r) = p$;

(2) The head of some rule r in $\mathbb{P}$ is p, and for each such rule,

(2.1) either some subgoal q of the body is false in L;

(2.2) or some positive subgoal of the body occurs in X.

The WFS uses unfounded sets to draw negative conclusions, inferring simultaneously all atoms in any unfounded set to be false. It is worth pointing out that conditions (1) and (2.1) above are used by Fitting in his semantics [40] to draw negative conclusions. And condition (2.2) is used by Ross and Topor in [93] to define closed sets. The WFS integrates them together. Consequently, the WFS is able to infer more negative information than either Fitting semantics or closed sets.

In order to capture all the atoms in every unfounded set of $\mathbb{P}$ with respect to L, an operator $U_P(L)$ is introduced. It is the greatest unfounded set of $\mathbb{P}$ with respect to L. Since the union of arbitrary unfounded sets is an unfounded set too, the greatest unfounded set does exist and is equal to the union of all sets that are unfounded with respect to $\mathbb{P}$ and L.

[8]In [105], L is called an interpretation if it does not contain both an atom and its complement. This is from the model-theoretic viewpoint. Since we shall emphasise the deductive viewpoint, we use the general notion of set rather than interpretation.

Definition 5.6.2 Given a logic program $\mathbb{P}$, a set L of literals, $U_P(L)$ is defined as the greatest set satisfying the condition:

$$X \subseteq ?_{P,L}(X).$$

Some comments about $U_P(I)$ are in order. First of all, we note that the above definition of $U_P(L)$ has the form of a coinduction rather than the form of an induction (see [9]).

Secondly, note that the authors of [105] did not explicitly tell us how to obtain $U_P(L)$ constructively. Subsequently, some equivalent constructions of well-founded partial models were proposed. In [104], van Gelder defined the alternating fixpoint construction. In [16], Bidoit and Froidevaux used the notion of a potentially founded set, the dual of the notion of unfounded sets. However, it is worth noting a construction is in fact implicitly there in [105]. Since $?_{P,L}$ is a monotonic operator on the Herbrand base $\mathcal{H}B_P$ and $U_P(L)$ is the greatest fixpoint of $?_{P,L}$ (see Lloyd [64], p. 26), this can be constructed using the following well-known characterisation of the greatest fixpoint (see [64], pp. 27-28):

$$\begin{array}{lcl} ?_{P,L} \downarrow 0 & = & \mathcal{H}B_P \\ ?_{P,L} \downarrow \alpha & = & ?_{P,L}(?_{P,L} \downarrow (\alpha \perp 1)), \text{ if } \alpha \text{ is a successor ordinal} \\ ?_{P,L} \downarrow \alpha & = & \bigcap\{?_{P,L} \downarrow \beta : \beta < \alpha\}, \text{ if } \alpha \text{ is a limit ordinal} \end{array}$$

Generally speaking, the least ordinal α such that $?_{P,I} \downarrow \alpha = U_P(I)$ can be beyond ω (see [64], p. 31). But for the cases in which programs are function-free and the EDB is finite, the least ordinal is finite.

Finally, note that the main strength of an unfounded set, and thus the main strength of the WFS, in comparison to Fitting semantics, lies its ability to deal with circular atoms in a program $\mathbb{P}$. This is owing to the use of both the condition (2.2) above and the coinductive definition of $U_P(I)$. Then, what atoms are circular?

Definition 5.6.3 Let $p \prec^+ q$ if and only if p is the head of a rule in $\mathbb{P}$ and q is a positive literal in the body of the same rule; and $p \prec^- q$ if and only if p is the head of a rule in $\mathbb{P}$ and q is a negative literal in the body of the same rule.

A path beginning with p in $\mathbb{P}$ ending with q is a sequence $\langle p, q_1, \ldots, q_n, q\rangle$ such that

$$p \prec^* q_1 \prec^* \ldots \prec^* q_n \prec^* p,$$

where $\prec^*$ is either $\prec^+$ or $\prec^-$.

A circular atom is an atom p such that all the paths beginning with p in $\mathbb{P}$ lead to itself positively:

$$p \prec^+ q_1 \prec^+ \ldots \prec^+ q_n \prec^+ p,$$

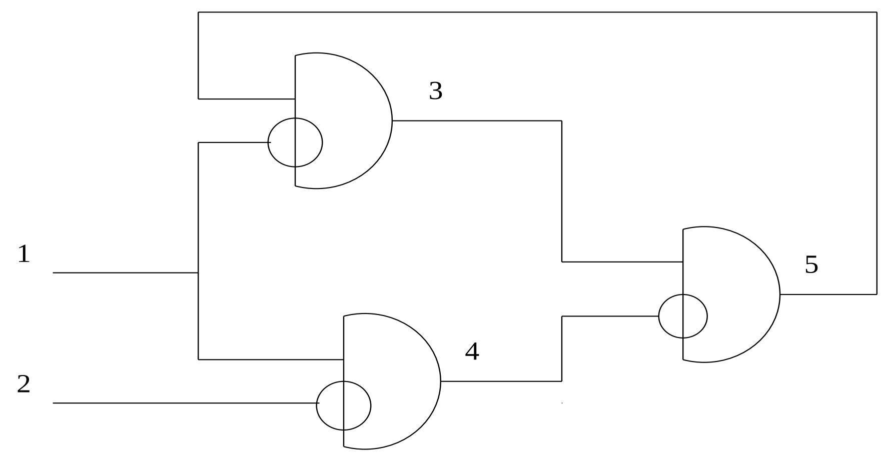
The following example about circuits is from [101], showing how circular atoms may arise in real-life applications.

Example 5.6.1 Consider a circuit of an unusual sort of logic gates $g(X, Y, Z)$, where $g(X, Y, Z)$ means that a gate has positive input X, negative input Y and positive output Z. These inputs and outputs can be taken as terminals or wire nets. Let $t0(X)$ represent that input terminal X is externally set be 1 ("true"). Let $t(X)$ represent that input terminal X is "internally" set to be 1.

Given a gate $g(X, Y, Z)$, the circuit value of Z is 1 if and only if X is 1 and Y is 0 ("false"). Then the following rules define the operation of the gates:

$$t(Z) :\perp t0(Z).$$
$$t(Z) :\perp g(X, Y, Z), t(X), not\ t(Y).$$

Suppose that the EDB consist of the following facts: $t0(2)$, $g(5, 1, 3)$, $g(1, 2, 4)$, $g(3, 4, 5)$ which represents the following circuit



$$\begin{array}{lll} t(2) & :\perp & t0(2). \\ t(3) & :\perp & g(5,1,3), t(5), not\ t(1). \\ t(4) & :\perp & g(1,2,4), t(1), not\ t(2). \\ t(5) & :\perp & g(3,4,5), t(3), not\ t(4). \end{array}$$

It is easy to see that atoms $t(3)$ and $t(5)$ are circular. □

As Barwise and Etchemendy pointed out in [9], when working with circular phenomena (in this case finding out all circular atoms) it is the coinductive definition that is more natural to use, suggesting the above definition of $U_P(I)$ and explaining why the greatest fixpoint of $?_{P,I}$ is needed (see Barwise and Etchemendy [9], pp. 53-54). In the same book, Barwise and Etchemendy gave an elegant informal explanation about coinductive definition: "Instead of working from the bottom up [since there is no bottom base when dealing with circular phenomena], asking which objects are *forced into* the defined class, we work from the top down, asking which objects are legitimately *excluded*. This feature guarantees that circular members ... are not excluded." (see [9], pp. 62-63).

In order to define the WFS, another operator is needed to decide what positive conclusions can be drawn from $\mathbb{P}$. The operator used in [105] is the same as the immediate consequence operator T_P except that it is defined on sets of literals rather than sets of atoms. By abuse of notation, we still use T_P to denote the operator:

Definition 5.6.4 $T_P(L) = \{p \in \mathcal{H}B_P : \exists r \in \mathbb{P}(head(r) = p \wedge \forall q \in body(r) q \in L)\}$.

With the operators U_P and T_P available, we can now define another operator W_P.

Definition 5.6.5 $W_P(L) = T_P(L) \cup not \cdot U_P(L)$

The three operators U_P, T_P, W_P are all monotonic on $\mathcal{H}B_P \cup not \cdot \mathcal{H}B_P$. Therefore, there is a least fixpoint $lfp(W_P)$. It is the least fixpoint that is used to define the *well-founded semantics* of a logic program $\mathbb{P}$ in [105].

In [105], it is pointed out that T_P treated positive and negative subgoals symmetrically in the sense that, in deciding whether a negative subgoal *not* q is true, the presence of *not* q rather than the absence of q is required. The point can be misleading.[9] We need to bear in mind that this symmetry is different from that between positive and strong negative information. A piece of negative information inferred through U_P is still nonmonotonic whereas a piece of strong negative information established through direct observation is monotonic. So it would have been better to say we need to explicitly keep all negative information inferred through U_P in the process of computing $lfp(W_P)$.

In [105], the least fixpoint $lfp(W_P)$ is called the *well-founded partial model.* Because we adopt a deductive rather than a model-theoretic viewpoint, we shall call it the *well-founded extension* (WFE) instead. An advantage with the deductive viewpoint is that we can still use two-valued logic as our underlying logic. It is well-known that the well-founded semantics is in general partial in the sense that some atoms in a logic program may be neither true nor false. As a result, the well-founded partial model of a logic program has to be viewed as a model in three-valued logic. From the deductive viewpoint, however, we can still work in two-valued logic.

[9]Note that in [1], p. 385, there is a similar misleading comment about the superficial symmetry between positive and negative information.

The WFS demonstrates many desirable features. It is defined for all logic programs. For any logic program $\mathbb{P}$, there exists a unique well-founded partial extension, defined as the least fixpoint $lfp(W_P)$ of W_P. Using the characterisation of a least fixpoint, the well-founded partial extension of a logic program can then be defined in a constructive way although the construction might be a process of possibly transfinite iteration.

As we mentioned before, the WFS extends the Fitting semantics, inferring more negative information and thus possibly more positive information than the Fitting semantics. The WFS also naturally extends the semantics for a large class of logic programs, including definite logic programs, stratified programs, and locally stratified programs. (see [83] and [105]).

Despite its merits, it can be argued that the WFS is too "sceptical". The inference mechanism provided by WFS is still not strong enough. Indeed, the extension given by the WFS is usually partial, leaving some atoms undefined. For many programs it even gives the empty set as their intended meaning. Incompleteness itself is not very problematic given that the information in a program is itself not complete. So, it is understandable that we may still not be able to reach any conclusion about some atoms even though we are allowed to use non-monotonic inference mechanism. The problem with WFS is that, in many programs, it seems reasonable to expect to derive some information but WFS is silent on them. The following example from [105] illustrates the problem.

Example 5.6.2 Let $\mathbb{P}$ be the logic program with the following four rules:

$$\begin{array}{lll} a & :\perp & not\ b. \\ b & :\perp & not\ a. \\ p & :\perp & a. \\ p & :\perp & b. \end{array}$$

The program $\mathbb{P}$ has the empty set as its well-founded partial extension. But it is reasonable to expect that p should hold since it is a logical consequence of $\mathbb{P}$. This example indicates that the WFS does not allow case-based reasoning. $\Box$

There are many proposals for extending the WFS, such as the generalised well-founded semantics GWFS [7], the well-founded-by-case-semantics WFS$_C$ [97], the extended well-founded semantics WFS$_E$ [57], the strong well-founded semantics WFS$_S$ [22], and the O-semantics [81]. The relationship between these semantic theories for logic programming is also studied in [33] and [34].

5.7 Stable Semantics

The stable model semantics was introduced in Gelfond [46], further developed by Gelfond and Lifschitz in [47], and also by Marek and Truszczynski in [66]. The original stable model semantics is based on a two-valued framework. Subsequently, various different but equivalent three-valued versions of stable model semantics were proposed, including 3-stable models [85], partial stable models [95] and preferred extensions [37]. In this section, we only consider the two-valued stable model semantics by Gelfond and Lifschitz [47] and its three-valued version by Przymusinski [85].

5.7.1 Two-valued Stable Semantics

Stable model semantics has its root in Moore's autoepistemic logic [74]. In [46], Gelfond observed that rules of logic programs can be naturally translated into formulas of autoepistemic logic. For example, the rule $p : \perp q, \sim r$ can be expressed as $q \wedge \neg Lr \rightarrow p$ in autoepistemic logic, where L is the belief operator of the logic. Through this kind of translation, the declarative semantics of a logic program can then be characterised in terms of expansions of autoepistemic theory associated with the program (see [46] and [66]). The basic idea is to define a *stable model* of a logic program $\mathbb{P}$ as one that is able to reproduce itself from $\mathbb{P}$ in a certain sense.

Alternatively, stable model semantics can also be defined without reference to autoepistemic logic as it is shown by Gelfond and Lifschitz [47]. This is achieved by using a different transformation, called Gelfond-Lifschitz transformation, based on the same idea that a *stable model* of a logic program $\mathbb{P}$ is a set of atoms that is able to reproduce itself from $\mathbb{P}$ (see the defining equation of stable model below). In the following, our discussion will be based on this transformation.

Definition 5.7.1 Given a logic program $\mathbb{P}$, and a set Δ of atoms in $\mathcal{H}B_P$. The stability transformation of $\mathbb{P}$ with respect to Δ, denoted by $S_{P,\Delta}$, is the definite logic program obtained from $\mathbb{P}$ by deleting:

(i) any rule that has a negative literal *not* q in its body with $q \in \Delta$;

(ii) all negative literals *not* q in the bodies of the remaining rules.

The transformation above makes use of the information from Δ only to decide negative subgoals[10] in rules of $\mathbb{P}$. If a negative *not* q occurs in a rule body and $q \in \Delta$ then we eliminate the rule as it depends on an item of negative information when Δ supplies the positive. If a negative *not* q occurs in a rule body and $q \notin \Delta$ then the negative subgoal is considered satisfied because failure to be in Δ commits to falsehood.

For any set Δ of atoms, the transformed program $S_{P,\Delta}$ is a definite logic program, so it has a unique minimal model [103], which coincides with the least fixed point of the immediate consequence operator T_P [103]. Recall that the least fixpoint $lfp(T_P)$ of T_P can be computed via the so-called ordinal powers of T_P. The computation is

[10]It is worth pointing out that a stable model could not be guaranteed to be a minimal Herbrand model had we also made use of the information from Δ to decide positive subgoals in rules of $\mathbb{P}$ though otherwise there seems no reason to prevent us from doing so.

a process of successively adding information. We shall use this observation shortly in our criticism of the stable model semantics.

A set Δ of atoms is a *stable* model of a logic program $\mathbb{P}$ provided it is the minimal model of the stability transformation of $\mathbb{P}$ [47]:

Definition 5.7.2 Given a logic program $\mathbb{P}$. A set Δ of atoms is called a stable model of $\mathbb{P}$ if and only if Δ is the minimal model of $S_{P,\Delta}$, that is Δ satisfies the following stable equation:

$$lfp(T_{S_{P,\Delta}}) = \Delta.$$

In the above stable equation, Δ represents an initial global decision as to what is false (everything not in Δ). However, the process of computing the lfp of $S_{P,\Delta}$ consists in adding truths. This process of adding new positive information should result at times in a reduction of the existing negative information, just as increase in the size of S results in decrease in the size of $R \perp S$ in the relational algebra model of negation. But the structure of the stable model semantics is such that this reduction cannot occur. The negative facts are fixed once and for all at the outset. Despite the increase of information, what is not true cannot be revised. Yet we aim to model an essentially nonmonotonic negation. This tension is, we suggest, at the heart of the existence problem(s) and anomalies of the stable semantics. Let us consider some examples by way of illustration.

Example 5.7.1 Consider a trivial but typical example program $\mathbb{P}_1$ with the following rules:

$$\begin{aligned} q \;&:\perp\quad \mathit{not}\; p. \\ p \;&:\perp\quad q. \end{aligned}$$

It is not difficult to check that $\mathbb{P}_1$ has no stable model. We need to consider the four complete assignments to p and q to see this. In each case we start with an initial assignment (left column), compute the stability transformation for this assignment (centre column), and then the least fixed point of the transformed program (right column) which represents the final assignment. The initial assignment is a stable model provided it is the same as the final assignment. The table below shows that no assignment satisfies this so that there is no (two-valued) stable model for this program.

Δ	*Transform*	*Fixpoint*
$\emptyset$	$q : \perp .$ $p : \perp\ q.$	$\{p, q\}$
$\{p\}$	$p : \perp\ q.$	$\emptyset$
$\{q\}$	$q : \perp .$ $p : \perp\ q.$	$\{p, q\}$
$\{p, q\}$	$p : \perp\ q.$	$\emptyset$

Our point is not that none of these proposed models turn out to be stable. It is that the computational process over the transformed program never manages to generate the correct solution $\{p\}$ so that stability or otherwise can be tested for it. This we contend is inevitable because there is no element of revision within the process, it is strictly monotonic and you cannot hope to adequately represent a nonmonotonic negation with a strictly monotonic process.

We admit that we cannot *prove* this. Nevertheless, we shall show in the next chapter that the introduction of an appropriate revising mechanism does enable us to assign the proper meaning to the program $\mathbb{P}_1$ (see also [111]). $\Box$

Example 5.7.2 Consider the program $\mathbb{P}_2$ with the following four rules, which is borrowed from [105]:

$$\begin{array}{lll} a & :\perp & \mathit{not}\ b. \\ b & :\perp & \mathit{not}\ a. \\ p & :\perp & \mathit{not}\ p. \\ p & :\perp & \mathit{not}\ b. \end{array}$$

It is easy to see that the program $\mathbb{P}_2^1$ with only the first two rules has two stable models: $\{a, \mathit{not} b\}$ and $\{\mathit{not} a, b\}$. But when the third rule is added to $\mathbb{P}_2^1$, the resulting program $\mathbb{P}_2^2$ has no stable models at all. Such an effect is certainly undesirable since the third rule is independent of the first two in the sense that atoms p and a, b are not related each other in any rule. Moreover, note that $p :\perp \mathit{not}\ p$ logically implies p, so it is reasonable to expect that $\mathbb{P}_2^2$ should have $\{a, \mathit{not}\ b, p\}$ and $\{\mathit{not}\ a, b, p\}$ as its intended meaning.

As pointed out in [105], the fourth rule also has an anomalous effect on $\mathbb{P}_2^2$ in the stable semantics. Since p is already a logical consequence of $\mathbb{P}_2^2$, the addition of the fourth rule to $\mathbb{P}_2^2$ should not have any effect. Nevertheless, it stabilises one of two minimal models of $\mathbb{P}_2^2$, giving us a unique stable model for the full program $\mathbb{P}_2$. $\square$

The second example shows a sort of oscillating behaviour, sometimes we have a stable model, sometimes not, and there seems no motivation for it to be one rather than the other. It is our belief that these anomalies arise because we are trying to compute a non-monotonic negation via a process that does not allow the retraction of previously assumed negative information in the light of new positive conclusions. A non-monotonic negation such as is required for logic programs (at least in a database context) should be computed by a non-monotonic, revision process. In the next chapter, we shall propose a model of negation that owes much to the stable semantics but allows, through a mechanism of consistency-recovery, for just this withdrawal of previously assumed negative information. We shall show that the example just given is not anomalous in this system.

5.7.2 Three-valued Stable Semantics

In the last section, a stable model was represented as a set of ground atoms. An atom is true of a model if it is in the model; otherwise, it is taken false by default. So, the original definition of the stable models is in the two-valued framework.

In [83], a three-valued version of stable model semantics is introduced. Three-valued stable semantics requires us to extend the basic notion of definite logic programs by introducing into the bodies of rules special atoms 0, 1/2, 1 to represent truth-values *false, unknown* and *true* respectively, which gives us 3-definite logic programs.

Given a 3-definite logic program, a least fixpoint can be constructed by using a variant of Fitting operator. The variant operator will not operate on the natural topology but a different one of $\mathcal{H}B_P \cup not \cdot \mathcal{H}B_P$, which is defined as follows:

Definition 5.7.3 Let E_1 and E_2 be sets of literals. Define an ordering $\preceq$ among a set of literals by

$$E_1 \preceq E_2 \qquad \text{iff} \qquad E_1^+ \subseteq E_2^+ \quad \& \quad E_1^- \supseteq E_2^-.$$

Equipped with the ordering $\preceq$, $\mathcal{H}B_P \cup not \cdot \mathcal{H}B_P$ now has as the bottom member $not \cdot \mathcal{H}B$ instead of the usual empty set $\emptyset$. The variant Fitting operator F_P will operate on $\mathcal{H}B_P \cup not \cdot \mathcal{H}B_P$ with the ordering $\preceq$. We shall use $F_{P,\preceq}$ to denote the variant operator:

Definition 5.7.4 $F_{P,\preceq}(L) = T_{P,\preceq}(L) \cup not \cdot N_{P,\preceq}(L)$

where

$$\begin{aligned} T_{P,\preceq}(L) &= \{p \in \mathcal{H}B_P : \exists r \in \mathbb{P}(head(r) = p \wedge \forall q \in body(r)(q \in L \vee q = 1))\} \\ N_{P,\preceq}(L) &= \{p \in \mathcal{H}B_P : \forall r \in \mathbb{P}(head(r) = p \rightarrow \exists q \in body(r)(\overline{q} \in L \vee q = 0))\} \end{aligned}$$

For any 3-definite logic programs, it is easy to prove that $F_{P,\preceq}$ is monotonic with respect to $\preceq$, and thus has a least fixpoint, computed iteratively:

$$lfp(F_{P,\preceq}) = \bigcup F_{P,\preceq} \uparrow \alpha$$

where

$$\begin{array}{lcl} F_{P,\preceq} \uparrow 0 & = & not \cdot \mathcal{H}B_P \\ F_{P,\preceq} \uparrow \alpha & = & F_{P,\preceq}(F_{P,\preceq} \uparrow (\alpha \perp 1)), \text{ if } \alpha \text{ is a successor ordinal} \\ F_{P,\preceq} \uparrow \alpha & = & \bigcup\{(F_{P,\preceq} \uparrow \beta)^{+} : \beta < \alpha\} \cup \bigcap\{(F_{P,\preceq} \uparrow \beta)^{-} : \beta < \alpha\}, \\ & & \qquad \text{if } \alpha \text{ is a limit ordinal} \end{array}$$

As in the two-valued case we now proceed by transforming a logic program into a definite program. But this time the transformation is with respect to a set of literals rather than a set of atoms, and the transformed program is a 3-definite logic program, that is to allow the inclusion of the truth-values 0, 1/2, 1 in the bodies of the rules of the transformed program.

Definition 5.7.5 Given a logic program $\mathbb{P}$, and a set L of literals in $\mathcal{H}B_P \cup not \cdot \mathcal{H}B_P$. The stability transformation of $\mathbb{P}$ with respect to L, denoted by $S_{P,L}$[11], is the 3-definite logic program obtained from $\mathbb{P}$ by replacing:

(i) any negative subgoal *not* q in a rule such that $q \in L^-$ with truth value 1;

(ii) any negative subgoal *not* q in a rule such that $q \in L^+$ with truth value 0;

(iii) any negative subgoal *not* q in a rule such that $q \notin L^+$ and $q \notin L^-$ with 1/2.

Definition 5.7.6 Given a normal logic program $\mathbb{P}$. A set L of literals is called a 3-valued stable model of $\mathbb{P}$ if and only if L satisfies the following stable equation:

[11]We rely on context to distinguish which transformation we are refer to.

$$lfp(F_{S_{P,L},\preceq}) = L.$$

In the previous subsection, we have argued that the underlying definition of stable models is to some extent inconsistent with the essential characteristics of negation *not* used in logic programs. It is this kind of inconsistency that underlies the existence problem(s) of the stable model semantics. It can be argued that our criticism continues to apply to the three-valued version of the stable semantics, indeed is perhaps stronger against the three-valued than the original two-valued version.

Our general contention is that a fully satisfactory account of a non-monotonic negation must allow the retraction of what was previously assumed false "by default" in the light of newly discovered information. The 2-valued stable semantics does not do this since the stable equation requires that the negative facts are effectively fixed as true for the duration of the computation. The same criticism applies to the three-valued semantics.

Let us consider the example program $\mathbb{P}_1$ in the last section again. This time, however, let $\Delta = \{not\, p\}$, that is explicitly assume that p is false. Relative to Δ, $\mathbb{P}_1$ is now transformed into:

$$\begin{array}{lll} q & :\perp & 1. \\ p & :\perp & q. \end{array}$$

Similarly three-valued stable semantics simply excludes Δ as a three-valued stable model of $\mathbb{P}_1$ instead of revising the initial assumption so as to reach the intended conclusion that p is true.

Note that although this program does not have any two-valued stable model, it does have a unique three-valued model. This is not accidental. In fact it has been proved that each normal logic program has at least one 3-stable model (see [1]). It seems that 3-valued stable model semantics solves the existence problem of 2-valued stable semantics. However, it can be argued that the solution is not fully satisfying.

Indeed, the unique three-valued stable model of $\mathbb{P}_1$ is the empty set though it is reasonable to expect that p is true. It seems that the non-existence property of two-valued stable semantics is just converted to a truth-value gap of atoms in the three-valued stable semantics. The three-valued stable semantics does not really solve the non-existence problem of two-valued stable semantics.

The program $\mathbb{P}_2$ considered in the previous subsection can also be used to show that three-valued stable semantics has similar anomalies. Moreover, the introduction of a third truth-value 1/2 is not without cost. There are at least two problems, showing that the three-valued stable semantics can be worse than the original one. One is illustrated by the following example.

Example 5.7.3 Let $\mathbb{P}_3$ be the logic program with the following two rules:

$$\begin{array}{lll} a & :\perp & not\ b. \\ b & :\perp & not\ a. \\ p & :\perp & a. \\ p & :\perp & b. \end{array}$$

The program $\mathbb{P}_3$ has $\{p, a\}$ and $\{p, b\}$ as its stable models. Note that p is a logical consequence of $\mathbb{P}_3$, so it is reasonable to expect that p should be always true. In the three-valued stable semantics, unfortunately, $\mathbb{P}_3$ has as its three-valued stable models the empty set $\emptyset$ in addition to $\{p, a, not\ b\}$ and $\{p, b, not\ a\}$. $\Box$

The other problem concerns the transformation of a logic program relative to a set of literals. In the three-valued semantics the atoms q such that neither q nor $not\ q$ are in Δ are taken to be "unknown" and are replaced by the value 1/2. The three-valued immediate consequence operator allows us to infer both positive and negative information. Thus it should allow us to infer q or $not\ q$ on occasions even when q was initially unknown. That $not\, q$ could well occur in the body of a rule, but even though q or *not* q is subsequently established the rule may not be used because it has been "frozen" with replacing *not* q by a constant 1/2 which, of course, no amount of reasoning can ever establish or refute. It is even less appropriate to fix what is "unknown" at the outset of a computation than it is to fix what is "false by default". Also to fix q as unknown at the outset and then infer q or *not*q subsequently is almost like assigning an atom incompatible truth-values.

Chapter 6

Quasi-stable Semantics

In the last chapter we have reviewed some of main semantic theories proposed in both the logic programming community and the deductive database community. In particular, we have shown that the two dominant semantic theories, the well-founded semantics and the stable model semantics, are not fully satisfying. In this chapter, we describe a new semantics for logic programs which we term the "quasi-stable semantics"[1] given its close relationship with the stable model semantics. We shall show that our new semantics maintains the desired features of both the well-founded semantics and the stable model semantics while overcoming their shortcomings.

Our general contention is that a non-monotonic negation such as is required for logic programs should be computed by a non-monotonic revision process. Only a process that allows one to withdraw by revising provisionally held negative information can hope to be adequate to model a non-monotonic negation. Motivated by this, we propose a model of negation that owes much to the stable semantics but allows, through a mechanism of consistency-recovery, for just this withdrawal of previously assumed negative information.

In the following, we shall first give an informal introduction to the quasi-stable

[1]The work on quasi-stable semantics have been previously reported in [111].

semantics, followed by its formal definition, and then prove some main results about the semantics to show the relationship among the quasi-stable semantics, the well-founded semantics and the stable model semantics. In the Example section, we consider some example programs from the literature to show how the quasi-stable semantics has overcome the shortcomings of the well-founded semantics and the stable model semantics. In the Discussion section, we argue why the so-called supportedness property cannot be justified though it has been taken as a indispensable feature of a proper semantics for logic programs in the logic programming community. Finally, we discuss some related work on the extensions of the WFS and the stable model semantics.

6.1 The Quasi-stable Semantics

6.1.1 Informal description of the Quasi-stable semantics

The basic idea behind the quasi-stable semantics is extremely simple. Essentially it is to iteratively extend the WFE using hypothetical reasoning. We may roughly formulate the quasi-stable semantics as follows:

$$\text{Quasi-stable Semantics} = WFE + \text{hypothetical reasoning.}$$

Given a logic program $\mathbb{P}$, we first of all compute its WFE. Note that, by doing this, we are using both deductive and non-monotonic reasoning based upon the operator W_P. When we find the well-founded extension of $\mathbb{P}$, we will not be able to infer any more information using only the operators T_P and U_P. Then, it is the time to apply hypothetical reasoning: choose an atom among the atoms undetermined so far, and assume that it is false.[2] We emphasise the assumption consists of only a single literal rather than a set of literals. As soon as an assumption is made, it is natural

[2]Note that we are implicitly using a variant of the closed world assumption [90].

to continue reasoning using W_P. Unfortunately, things are not so straightforward.

To begin with, we need to understand a problem that exists when we continue reasoning with W_P. The problem is that, although the unique well-founded extension is consistent, any further derivation based upon W_P may contain contradictions when an assumption is made. Further, it would not be clear whether it is the assumption or the non-monotonic reasoning based upon U_P that is causing the trouble.

Let us consider the logic program $\mathbb{P}$ with the following rules to see how the ambiguity may actually come come up:

$$\begin{array}{lll} a & :\perp & \textit{not}\ b. \\ b & :\perp & \textit{not}\ c. \\ c & :\perp & \textit{not}\ a. \end{array}$$

It is easy to check that the well-founded extension is the empty set. Choose an undefined atom, say b, and assume *not* b. Using the assumption, it follows from $\mathbb{P}$ that a holds, and thus *not* c by U_P, and thus b by T_P, a contradiction. The contradiction may be caused either by the assumption *not b* or by the non-monotonic reasoning. We cannot be sure which is causing the trouble.

Secondly, we need to make decision about how to deal with such ambiguity. We choose to avoid the ambiguity. Our strategy is to make use of a more restricted operator for non-monotonic reasoning instead of W_P based on the notion of a simply intensional atom which we introduced in Chapter 4 (see page 64). We note that, for any simply intensional atom p, if all the rules of $\mathbb{P}$ whose head is p have a false body, then p can be taken to be false without causing any contradiction. This motivates the following definition of an operator $N_{P,L}$ for non-monotonically inferring negative information, which is a variant of the negative component N_P of Fitting operator F_P. For convenience of formulation, we shall use the notion of coveredness from [78].

Definition 6.1.1 Given a logic program $\mathbb{P}$ and a set L of literals. An atom p is *covered* by L if p is in $L^+ \cup L^-$; otherwise it is *uncovered* by L.

Definition 6.1.2 Given a logic program $\mathbb{P}$, a set L of literals, we define two operators N_P and F_P on $\mathcal{H}B \cup not \cdot \mathcal{H}B$ as follows:

$$\begin{aligned} N_{P,L}(E) &= \{p \in \mathcal{H}B_P : p \text{ is simply intensional \& not covered by } L \text{ \&} \\ &\qquad \forall r \in \mathbb{P}(head(r) = p \rightarrow \exists l \in body(r)\bar{l} \in E\} \\ F_{P,L}(E) &= L \cup T_P(E) \cup not \cdot N_{P,L}(E) \end{aligned}$$

where $E \subseteq \mathcal{H}B \cup not \cdot \mathcal{H}B$, and the operator T_P is defined as in the previous section.

Theorem 6.1.1 *The operator $F_{P,L}$ defined above is monotonic and thus there exists a least fixed point $lfp(F_{P,L})$.*

Note that, with the use of the operator $N_{P,L}$, assumptions may be restricted to the atoms in the body of a rule in $\mathbb{P}$ since any simply intensional atom can then be dealt with by either T_P or $N_{P,L}$.

Generally, the least fixed point $lfp(F_{P,L})$ for any set L of literals may still contain contradictions. This time, however, we may be sure it is the assumptions in L that are to be blamed not the non-monotonic reasoning (see lemma 6.1.4), thus eliminating the ambiguity.

With the ambiguity removed, the next step of the refining process is to note that $lfp(F_{P,L})$ may contain no explicit contradiction even though $\mathbb{P}$ is not consistent with L. For example, consider the following program $\mathbb{P}$

$$\begin{array}{lll} p & :\perp & a. \\ p & :\perp & b. \\ a & :\perp & not\ b. \\ b & :\perp & not\ a. \end{array}$$

Let $L = \{not\ p\}$. Then we have $lfp(F_{P,L}) = \{not\ p\}$ with no contradiction. But $\mathbb{P}$ logically implies p and thus is not consistent with L. Note that the contradiction comes up if we add either a or $not\ a$. In order to make this explicit, we introduce the notion of conflict-freeness and that of consistency with respect to $\mathbb{P}$. We show that conflict-freeness is weaker than consistency in lemma 6.1.4.

Definition 6.1.3 Given a logic program $\mathbb{P}$, and a set L of literals.

(i) L is said to be conflict-free with respect to $\mathbb{P}$ if $lfp(F_{P,L})$ is conflict-free.

(ii) L is said to be consistent with respect to $\mathbb{P}$ if there is no atom p such that $\mathbb{P}, L \vdash p$ and $\mathbb{P}, L \vdash not\ p$.

where $\vdash$ is the deductive relation of the full propositional logic.

Finally, with the restricted operator $F_{P,L}$, the related notions of conflict-freeness and consistency, we are ready to finish the process of refining our basic idea by informally reformulating our semantics as follows. Given a logic program $\mathbb{P}$ and a sequence[3] $H = \langle *q_1, \cdots, *q_{n-1}, *q_n \rangle$ of assumptions, we set the least fixed point $lfp(F_{P,H}) = lfp(F_{P,L})$, where L is the set corresponding to the sequence H, and $*q$ is either a negative literal with q as its atom or just the atom q.

Suppose that $lfp(F_{P,H})$ contains both an atom p and $not\ p$. Since we know that it is the assumptions in L that are to be blamed, then in order to remove the conflict, we have to change some assumptions. We choose to change the latest assumption made, that is the last element $*q_n$ in the sequence H. We call the process *consistency-recovery* for H, and proceed as follows.

Case I. The last element $*q_n$ is a negative literal. In this case, we do not simply withdraw the assumption. We assume the atom instead. In this way, the sequence of assumptions may contain both negative and positive literals. But positive literals

[3]We use *sequence* instead of *set* in order to emphasise the order of assumptions made. We shall make the use of the order when we withdraw an assumption for the convenience of formulation.

only appear where negative ones may not be assumed. So negative literals have priority. When the negative assumption is replaced by the corresponding positive assumption, we then do reasoning based upon $F_{P,H'}$, and do consistency-recovery again if necessary, where $H' = \langle *q_1, \cdots, *q_{n-1}, q_n \rangle$.

Case II. The last element of H is a positive literal. Since the only way the positive literal is assumed is that its negative literal cannot be, we conclude that the subsequence $H' = \langle *q_1, \cdots, *q_{n-1} \rangle$ is not consistent even though it is conflict-free. We then drop the assumption and do consistency-recovery for H'.

When $lfp(F_{P,H})$ is conflict-free (either immediately or after consistency-recovery), we make another assumption by choosing another uncovered body atom and assuming it is false. Continue this process until no body atoms are left uncovered in the Herbrand base $\mathcal{HB}_P$. As a result, we obtain a total extension, which we call a *quasi-stable extension*.

6.1.2 Formal Definition of Quasi-stable Semantics

Although the informal formulation of the quasi-stable semantics is not difficult, it turns out that its formal definition is somehow involved. This mainly is due to the process of consistency-recovery, a process which we insist is indispensable for modeling non-monotonic negation (see [113]).

Before we give the formal definition of quasi-stable extension, we need the following definition and a series of related results.

Definition 6.1.4 Given a logic program $\mathbb{P}$, the well-founded extension WFE for $\mathbb{P}$, and some ordering $\mathcal{O}$ of the body atoms of $\mathbb{P}$, we define by induction the following sequence of sets:

Let $H_0 = \emptyset$ and $\mathcal{E}_0 = WFE$. Suppose $\mathcal{E}_n$ and H_n have been defined, and $H_n = \langle *q_1, \cdots, *q_{i-1}, *q_i, \cdots, *q_m \rangle$. If all body atoms are covered by $\mathcal{E}_n$ then $\mathcal{E}_{n+1} = \mathcal{E}_n$,

$H_{n+1} = H_n$. Otherwise, let q be the first uncovered body atom. Let H_{n+1} and $\mathcal{E}_{n+1}$ be defined by the following:

$$H_{n+1} = \begin{cases} H_n\,\hat{}\,\langle not\ q\rangle & \text{if } lfp(F_{P,\mathcal{E}_n \cup \{not\ q\}}) \text{ is conflict-free;} \\ H_n\,\hat{}\,\langle q\rangle & \text{if } lfp(F_{P,\mathcal{E}_n \cup \{not\ q\}}) \text{ is not conflict-free} \\ & \text{but } lfp(F_{P,\mathcal{E}_n \cup \{q\}}) \text{ is conflict-free;} \\ H'_n & \text{otherwise.} \end{cases}$$

$$\mathcal{E}_{n+1} = lfp(F_{P,\mathcal{E}_0 \cup H_{n+1}}).$$

where $\hat{}$ stands for the concatenation of two sequences, and $H'_n = \langle *q_1, \cdots, *q_{i-1}, q_i\rangle$ is derived from H_n in the following way: i is the *greatest* index of H_n such that $*q_i = not\ q_i$ and $\langle *q_1, \cdots, *q_{i-1}, q_i\rangle$ is conflict-free.

For example, for the following program $\mathbb{P}$

$$\begin{array}{lll} p & :\perp & a. \\ p & :\perp & b. \\ a & :\perp & not\ b. \\ b & :\perp & not\ a. \end{array}$$

we have $H_0 = \emptyset$ and $\mathcal{E}_0 = \emptyset$ since the well-founded extension WFE of $\mathbb{P}$ is $\emptyset$. So, body atoms a and b are not covered by $\mathcal{E}_0$. Take the body atom a, it is easy to see that $lfp(F_{P,\mathcal{E}_0 \cup \{not\ a\}})$ is conflict-free, so we have $H_1 = \langle not\, a\rangle$ and $\mathcal{E}_1 = lfp(F_{P,H_1}) = \langle not\, a, b, p\rangle$. Now all the body atoms have already been covered by $\mathcal{E}_1$. By the above definition, we thus have $\mathcal{E}_{n+1} = \mathcal{E}_n$, $H_{n+1} = H_n$.

Usually a semantics is given by the least fixed point of an operator or equivalently, as a union of sets. In our case this second option is not immediately available because the sequence $\langle \mathcal{E}_n\rangle$ is not increasing and the union of all its members is anyway likely to be inconsistent with $\mathbb{P}$. Instead we prove the existence of an increasing, consistent subsequence $\langle H_{n_i} \rangle$ of $\langle H_n \rangle$ and thus the existence of an increasing, conflict-free subsequence $\langle \mathcal{E}_{n_i} \rangle$ of $\langle \mathcal{E}_n \rangle$. We then use these two subsequences to define the quasi-stable extension of $\mathbb{P}$ with respect to an ordering $\mathcal{O}$ of body atoms. We must first show, however, that definition 6.1.4 is correct.

We need to show that $\mathcal{E}_n$ and H_n are properly defined for each n. For this we need the consistency of $\mathcal{E}_0$ with respect to $\mathbb{P}$ and some lemmas.

In [105], it is shown that WFE can be extended into a model of $\mathbb{P}$ by adding to WFE all atoms uncovered by WFE. It then follows:

Lemma 6.1.1 (Van Gelder et al. [105]) *Given a logic program $\mathbb{P}$. Let WFE be the well-founded partial extension of $\mathbb{P}$, and $\Delta = \mathcal{H}B_P \perp (WFE^+ \cup WFE^-)$. Then $WFE \cup \Delta$ is consistent with respect to $\mathbb{P}$. In particular, WFE is consistent with respect to $\mathbb{P}$.*

The following lemma states the relationship between literals in $lfp(F_{P,L})$ and the consequences of $\mathbb{P} \cup L$.

Lemma 6.1.2 (i) *If a literal $l \in lfp(F_{P,L})$ and l is not simply intensional then $\mathbb{P} \cup L \vdash l$.* (ii) *If an atom $p \in lfp(F_{P,L})$, then $\mathbb{P} \cup L \vdash p$.*

Proof. (i) By induction on α with $\alpha = 0$ trivial. For a limit ordinal α, the lemma is also trivial. Suppose $l \in F_{P,L} \uparrow \alpha + 1$. As l is not simply intensional, the $N_{P,L}$ operator was not used. If $l \in L$ the result is trivial. Otherwise there is a rule $l : \perp l_1, \cdots, l_m$ with $\{l_1, \cdots, l_m\} \subseteq F_{P,L} \uparrow \alpha$. Using the induction hypothesis $\mathbb{P} \cup L \vdash l_i$ for $1 \leq i \leq m$ and hence $\mathbb{P} \cup L \vdash l$.

(ii) By lemma 5.2.1, $lfp(F_{P,L}) = \bigcup F_{P,L} \uparrow \alpha$. If $p \in lfp(F_{P,L})$, then there is a ordinal α such that $p \in F_{P,L} \uparrow \alpha + 1 = L \cup T_P(F_{P,L} \uparrow \alpha) \cup N_{P,L}(F_{P,L} \uparrow \alpha)$. As p is an atom, p cannot be in $N_{P,L}(F_{P,L} \uparrow \alpha)$. If $p \in L$ the result is trivial. Otherwise there is a rule $p : \perp l_1, \cdots, l_m$ with $\{l_1, \cdots, l_m\} \subseteq F_{P,L} \uparrow \alpha$. Since none of l_i is simply intensional, by lemma 6.1.2 (i), $\mathbb{P} \cup L \vdash l_i$ for $1 \leqslant i \leqslant m$ and hence $\mathbb{P} \cup L \vdash p$. $\Box$

The following is a technical lemma needed for the proof of theorem 6.1.3.

Lemma 6.1.3 *Given a logic program $\mathbb{P}$. Let L be a set of literals and l a body literal such that $L \cup \{l\}$ is consistent with respect to $\mathbb{P}$. Let $L' = lfp(F_{P,L})$ We have*

$$lfp(F_{P,L}) \subseteq lfp(F_{P,L\cup\{l\}}) \tag{6.1}$$

$$lfp(F_{P,L\cup\{l\}}) = lfp(F_{P,L'\cup\{l\}}) \tag{6.2}$$

Proof. (i) We prove (6.1) by proving the following:

$$F_{P,L} \uparrow \alpha \subseteq F_{P,L\cup\{l\}} \uparrow \alpha \tag{6.3}$$

By induction on α with limit ordinals trivial. Obviously $L \subseteq L \cup \{l\}$. Using the induction hypothesis and the monotonicity of T_P, $T_P(F_{P,L} \uparrow \alpha) \subseteq T_P(F_{P,L\cup\{l\}} \uparrow \alpha)$. Since any simply intensional atom p uncovered by L is also uncovered by $L \cup \{l\}$, we have $N_{P,L}(F_{P,L} \uparrow \alpha) \subseteq N_{P,L\cup\{l\}}(F_{P,L\cup\{l\}} \uparrow \alpha)$ by the induction hypothesis. Thus $F_{P,L} \uparrow \alpha + 1 \subseteq F_{P,L\cup\{l\}} \uparrow \alpha + 1$.

(ii) In order to prove (6.2), we first prove $lfp(F_{P,L\cup\{l\}}) \subseteq lfp(F_{P,L'\cup\{l\}})$. For this we prove by induction the following:

$$F_{P,L\cup\{l\}} \uparrow \alpha \subseteq F_{P,L'\cup\{l\}} \uparrow \alpha \tag{6.4}$$

For limit ordinals, (6.4) holds trivially. Since $L \subseteq lfp(F_{P,L})$, $L \cup \{l\} \subseteq L' \cup \{l\}$. By the induction hypothesis and monotonicity of T_P, $T_P(F_{P,L\cup\{l\}} \uparrow \alpha) \subseteq T_P(F_{P,L'\cup\{l\}} \uparrow \alpha)$. Now for any $p \in N_{P,L\cup\{l\}}(F_{P,L\cup\{l\}} \uparrow \alpha)$, if $not\, p \in L'$ then $not\, p \in lfp(F_{P,L'\cup\{l\}}) \uparrow$

$\alpha+1$. Suppose *not* $p \notin L'$. p cannot be in L' either, otherwise $p \in L' = lfp(F_{P,L}) \subseteq lfp(F_{P,L\cup\{l\}})$ and thus $lfp(F_{P,L\cup\{l\}})$ is not conflict-free, contradicting the assumption of $L \cup \{l\}$ being consistent with respect to $\mathbb{P}$. So for any $p \in N_{P,L\cup\{l\}}(F_{P,L\cup\{l\}} \uparrow \alpha)$, if *not* $p \notin L'$, then p is not covered by $L' \cup \{l\}$ and thus $p \in N_{P,L'\cup\{l\}})(F_{P,L'\cup\{l\}} \uparrow \alpha)$ by the induction hypothesis. Thus, (6.4) holds for $\alpha+1$.

In order to prove the converse of (6.2), we prove

$$F_{P,L'\cup\{l\}} \uparrow \alpha \subseteq lfp(F_{P,L\cup\{l\}}) \qquad (6.5)$$

Once again we prove (6.5) by induction on α. Limit ordinals are trivial. By (6.1) $L' \cup \{l\} \subseteq lfp(F_{P,L\cup\{l\}})$. Using the induction hypothesis and monotonicity of T_P, $T_P(F_{P,L'\cup\{l\}} \uparrow \alpha) \subseteq T_P(lfp(F_{P,L\cup\{l\}})) \subseteq lfp(F_{P,L\cup\{l\}})$. Since $L \subseteq L'$, any simply intensional atom uncovered by $L' \cup \{l\}$ is also uncovered by $L \cup \{l\}$. Thus $not \cdot N_{P,L'\cup\{l\}}(F_{P,L'\cup\{l\}} \uparrow \alpha) \subseteq not \cdot N_{P,L\cup\{l\}}(lfp(F_{P,L\cup\{l\}})) \subseteq lfp(F_{P,L\cup\{l\}})$ by the induction hypothesis. It then follows that (6.5) holds for $\alpha+1$. $\square$

The following lemma shows that the notion of being conflict-free w.r.t. $\mathbb{P}$ is weaker than that of consistency w.r.t. to $\mathbb{P}$. Moreover, it also shows that whenever a conflict occurs in $lfp(F_{P,L})$ then L is to be blamed.

Lemma 6.1.4 *Given a logic program $\mathbb{P}$. Let L be a set of literals If L is consistent with respect to $\mathbb{P}$, then L is conflict-free with respect to $\mathbb{P}$, that is $lfp(F_{P,L})$ is conflict-free.*

Proof. $lfp(F_{P,L}) = \bigcup F_{P,L} \uparrow \alpha$ and we prove that each $F_{P,L} \uparrow \alpha$ is conflict free.

Suppose that $F_{P,L} \uparrow \alpha$ is the least α which is not conflict free. Clearly $\alpha > 0$, but $\alpha > 1$ as well because $F_{P,L} \uparrow 1 = L \cup T_P(\emptyset) \cup not \cdot N_{P,L}(\emptyset) = L \cup T_P(\emptyset)$, $P \cup L$ is assumed consistent and T_P draws only logical consequences. Now $F_{P,L} \uparrow \alpha = L \cup T_P(F_{P,L} \uparrow \alpha \perp 1) \cup not \cdot N_{P,L}(F_{P,L} \uparrow \alpha \perp 1)$. If this is not conflict free it contains

p and *not* p for some atom p. We consider three cases:

Case I. *not* $p \in L$ and $p \in T_P(F_{P,L} \uparrow \alpha - 1)$. There must be some rule $p : - l_1, \cdots, l_m \in P$ with $\{l_1, \cdots, l_m\} \subseteq F_{P,L} \uparrow \alpha - 1$. As none of l_i are simply intensional we can apply lemma 6.1.2 (i) to conclude that $\mathbb{P} \cup L \vdash l_i$ for each $1 \leqslant i \leqslant m$. Thus $\mathbb{P} \cup L \vdash p$ contradicting our assumption that $\mathbb{P} \cup L$ is consistent.

Case II. $p \in L$ and *not* $p \in not \cdot N_{P,L}(F_{P,L} \uparrow \alpha - 1)$. But then L covers p and no inference using $N_{P,L}$ can be made.

Case III. $p \in T_P(F_{P,L} \uparrow \alpha - 1)$ and $p \in N_{P,L}(F_{P,L} \uparrow \alpha - 1)$. Then, by the first of these, for some rule $p : - l_1, \cdots, l_m \in \mathbb{P}$, we find $\{l_1, \cdots, l_m\} \subseteq F_{P,L} \uparrow \alpha - 1$. But by the second, some $\overline{l_i}$ must be in $F_{P,L} \uparrow \alpha - 1$ again contradicting the leastness of α. $\square$

Theorem 6.1.2 *Given a logic program $\mathbb{P}$, H_n and $\mathcal{E}_n$ are properly defined for all n.*

Proof. Suppose that H_n and $\mathcal{E}_n$ are defined, and there remain atoms uncovered by $\mathcal{E}_n$. Let q be the first uncovered atom. If *not* q or q can be added we are done. Suppose then that neither can be added. Thus $lfp(F_{P,\mathcal{E}_0 \cup H_n \cup \{q\}})$ is not conflict-free. There must therefore be some negative literal in H_n. For otherwise $H_n \cup \{q\}$ is a set of atoms which by construction are not covered by $\mathcal{E}_0$, and hence $\mathcal{E}_0 \cup H_n \cup \{q\}$ is consistent with respect to $\mathbb{P}$ by lemma 6.1.1. By lemma 6.1.4 it follows that $lfp(F_{P,\mathcal{E}_0 \cup H_n \cup \{q\}})$ is conflict-free. Since there is some negative literal in H_n there is a least such and by the argument just given if that is replaced by its atom a conflict-free least fixed point will result. Thus we may choose the greatest j such that H_n includes $\langle *q_1, \cdots, *q_{j-1}, not\ q_j \rangle$ and where $lfp(F_{P,\mathcal{E}_0 \cup \langle *q_1, \cdots, *q_{j-1}, q_j \rangle})$ is conflict-free. Then we set $H_{n+1} = \langle *q_1, \cdots, *q_{j-1}, q_j \rangle$ and $\mathcal{E}_{n+1} = lfp(F_{P,\mathcal{E}_0 \cup H_{n+1}})$. $\square$

Once we have seen that $\langle \mathcal{E}_n \rangle$ and $\langle H_n \rangle$ are properly defined, the next task is to show how to extract two canonical subsequences respectively from $\langle \mathcal{E}_n \rangle$ and $\langle H_n \rangle$

to be used in defining the quasi-stable extension of $\mathbb{P}$. We begin with two auxiliary lemmas.

Lemma 6.1.5 *Given a logic program $\mathbb{P}$. Let $\mathcal{E}_0 = WFE$, $H = \langle *q_1, \cdots, *q_m \rangle$ be a sequence of assumptions, $\mathcal{E} = lfp(F_{P,\mathcal{E}_0 \cup H})$. If $\mathcal{E}$ is conflict-free and all the atoms $\mathcal{H}B_P$ are covered by $\mathcal{E}$, then $\mathcal{E}$ is a Herbrand model of $\mathbb{P}$ and thus consistent with respect to $\mathbb{P}$.*

Proof. Since $\mathcal{E}$ is conflict-free and all the atoms in $\mathcal{H}B_P$ are covered, $\mathcal{E}$ is a Herbrand interpretation. By $\mathcal{E} = lfp(F_{P,\mathcal{E}_0 \cup H})$, we know $\mathcal{E}$ is closed under T_P. Then it follows that every rule in $\mathbb{P}$ is satisfied by $\mathcal{E}$ and hence $\mathcal{E}$ is a Herbrand model of $\mathbb{P}$. Obviously, $\mathcal{E}$ as a set of literals is satisfied by $\mathcal{E}$ as a Herbrand model. So both $\mathbb{P}$ and $\mathcal{E}$ have $\mathcal{E}$ as their model, and thus $\mathcal{E}$ is consistent with respect to $\mathbb{P}$. $\square$

Lemma 6.1.6 *Given a logic program $\mathbb{P}$ and an ordering $\mathcal{O}$ of the body atoms of $\mathbb{P}$. Let $\langle H_n \rangle$ and $\langle \mathcal{E}_n \rangle$ be as in definition 6.1.4. Suppose that $H_n = \langle *q_1, \cdots, *q_{i-1}, not\ q_i, *q_{i+1}, \cdots, *q_m \rangle$ is a sequence of assumptions, where i is the greatest i such that $H'_n = \langle *q_1, \cdots, *q_{i-1}, not\ q_i \rangle$ is not consistent with $\mathbb{P} \cup \mathcal{E}_0$ but $\langle *q_1, \cdots, *q_{i-1}, q_i \rangle$ is conflict-free, then there is an $n_1 > n$ such that $H_{n_1} = \langle *q_1, \cdots, *q_{i-1}, q_i \rangle$.*

Proof. Since $H'_n = \langle *q_1, \cdots, *q_{i-1}, not\ q_i \rangle$ is not consistent with $\mathbb{P} \cup \mathcal{E}_0$, there must be some finite set $\mathbb{P}_{fin}$ of rules of $\mathbb{P}$ such that H'_n is not consistent with $\mathbb{P}_{fin} \cup \mathcal{E}_0$. Let $\mathcal{O}_{fin}$ be the smallest fragment of $\mathcal{O}$ containing all body atoms in $\mathbb{P}_{fin}$. We now prove the lemma by induction on the number of atoms in $\mathcal{O}_{fin}$.

If $\|\mathcal{O}_{fin}\| = 1$, let q be the uncovered atom in $\mathcal{O}_{fin}$. Since H'_n is not consistent with $\mathbb{P}_{fin} \cup \mathcal{E}_0$ and H'_n is a subsequence of H_n, neither $lfp(F_{P_{fin},\mathcal{E}_n \cup \{not\ q\}})$ nor $lfp(F_{P_{fin},\mathcal{E}_n \cup \{q\}})$ can be conflict-free by lemma 6.1.5. It then follows that neither $lfp(F_{P,\mathcal{E}_n \cup \{not\ q\}})$ nor $lfp(F_{P,\mathcal{E}_n \cup \{q\}})$ can be conflict-free So by the definition of $\langle H_n \rangle$, $H_{n+1} = \langle *q_1, \cdots, *q_{i-1}, q_i \rangle$.

Suppose that the lemma holds for $\|\mathcal{O}_{fin}\| \leqslant k$. We show that the lemma holds for $\|\mathcal{O}_{fin}\| = k+1$ too. Let q be the first uncovered body atom in $\mathcal{O}_{fin}$. There are three cases to consider.

Case I. $H_n\,\hat{}\,\langle not\ q\rangle$ is conflict-free. According to definition of $\langle H_n\rangle$, $H_{n+1} = H_n\,\hat{}\,\langle not\ q\rangle$. We further consider two sub-cases. (i) $H_n\,\hat{}\,\langle q\rangle$ is conflict-free too. Then we have $H'_{n+1} = H_{n+1}$. By applying the induction hypothesis to H_{n+1}, there is an $n_1 > n+1$ such that $H_{n_1} = H_n\,\hat{}\,\langle q\rangle$. Now by applying the induction hypothesis to H_{n_1} and $H'_{n_1} = H'_n$, there is an $n_2 > n_1$ such that $H_{n_2} = \langle *q_1, \cdots, *q_{i-1}, q_i\rangle$. (ii) $H_n\,\hat{}\,\langle q\rangle$ is not conflict-free. Then we have $H'_{n+1} = H_n$. By applying the induction hypothesis to H_{n+1}, there is an $n_1 > n+1$ such that $H_{n_1} = \langle *q_1, \cdots, *q_{i-1}, q_i\rangle$.

Case II. $H_n\,\hat{}\,\langle not\ q\rangle$ is not conflict-free but $H_n\,\hat{}\,\langle q\rangle$ is conflict-free. According to the definition of $\langle H_n\rangle$, $H_{n+1} = H_n\,\hat{}\,\langle q\rangle$. Then we have $H'_{n+1} = H'_n$. By applying the induction hypothesis to H_{n+1}, there is an $n_1 > n+1$ such that $H_{n_1} = \langle *q_1, \cdots, *q_{i-1}, q_i\rangle$.

Case III. Neither $H_n\,\hat{}\,\langle not\ q\rangle$ nor $H_n\,\hat{}\,\langle q\rangle$ is conflict-free. According to the definition of $\langle H_n\rangle$, $H_{n+1} = \langle *q_1, \cdots, *q_{i-1}, q_i\rangle$. □

Theorem 6.1.3 (Canonical Subsequences) *Given a logic program $\mathbb{P}$, there exists an increasing subsequence $\langle \mathcal{E}_{n_i}\rangle$ of the sequence $\langle \mathcal{E}_n\rangle$ and an increasing subsequence $\langle H_{n_i}\rangle$ of $\langle H_n\rangle$ such that (i) $\mathcal{E}_0 \cup H_{n_i}$ is consistent with respect to $\mathbb{P}$, (ii) and thus $\mathcal{E}_{n_i}$ is conflict-free, and (iii) $\mathcal{E}_{n_i}$ covers $q_1, \cdots, q_i$ in the ordering $\mathcal{O}$ of body atoms.*

Proof. By lemma 6.1.1 $\mathcal{E}_0$ is consistent with $\mathbb{P}$. Set $n_0 = 0$. Both (ii) and (iii) are trivial.

Now suppose that n_i be defined, Let q_j be the first body atom uncovered by $\mathcal{E}_{n_i}$. By the induction hypothesis $j \geqslant i+1$. (If there is no such atom, then $\mathcal{E}_{n_{i+1}} = \mathcal{E}_{n_i}$ and the theorem holds.) We consider two cases.

Case I. $\mathbb{P}, \mathcal{E}_0 \cup H_{n_i} \vdash q_j$. Since $\mathcal{E}_0 \cup H_{n_i}$ is consistent with respect to $\mathbb{P}$, it follows that $\mathcal{E}_0 \cup H_{n_i} \cup \{q_j\}$ is also consistent with respect to $\mathbb{P}$. By (6.2) of lemma 6.1.3 and lemma 6.1.4, $lfp(F_{P,\mathcal{E}_{n_i}\cup\{q_j\}})$ is conflict-free. We further consider two subcases. (i) $lfp(F_{P,\mathcal{E}_0\cup H_{n_i}\cup\{not\ q_j\}})$ is conflict-free. Then $H_{n_i+1} = H_{n_i}{}^\wedge\langle not\ q_j\rangle$. By lemma 6.1.6, there exists an integer $m > n_i + 1$ such that $H_m = H_{n_i}{}^\wedge\langle q_j\rangle$. Set $n_{i+1} = m$. (ii) $lfp(F_{P,\mathcal{E}_0\cup H_{n_i}\cup\{not\ q_j\}})$ is not conflict-free. Then $H_{n_i+1} = H_{n_i}{}^\wedge\langle q_j\rangle$. Set $n_{i+1} = n_i + 1$. In either subcase, $\mathcal{E}_0 \cup H_{n_{i+1}}$ is consistent with respect to $\mathbb{P}$, and $\mathcal{E}_{n_{i+1}} = lfp(F_{P,\mathcal{E}_0\cup H_{n_{i+1}}})$ is conflict-free.

By (6.1) of lemma 6.1.3, $\mathcal{E}_{n_i} \subseteq \mathcal{E}_{n_{i+1}}$. Since q_j was the first atom uncovered by $\mathcal{E}_{n_i}$, $\mathcal{E}_{n_{i+1}}$ covers $q_1, \cdots, q_j$. As $j \geqslant i+1$, $\mathcal{E}_{n_{i+1}}$ covers $q_1, \cdots, q_{i+1}$.

Case II. $\mathbb{P}, \mathcal{E}_0 \cup H_{n_i} \not\vdash q_j$. Then $\mathcal{E}_0 \cup H_{n_i}{}^\wedge\langle not\ q_j\rangle$ is consistent with $\mathbb{P}$. By lemma 6.1.4, $lfp(F_{P,\mathcal{E}_0\cup H_{n_i}{}^\wedge\langle not\ q_j\rangle})$ is conflict-free. By definition 6.1.4, $H_{n_{i+1}} = H_{n_i}{}^\wedge\langle not q_j\rangle$, and $\mathcal{E}_{n_{i+1}} = lfp(F_{P,\mathcal{E}_0\cup H_{n_i}{}^\wedge\langle not\ q_j\rangle})$. So set $n_{i+1} = n_i + 1$. $\mathcal{E}_0 \cup H_{n_{i+1}}$ is consistent with respect to $\mathbb{P}$, and $\mathcal{E}_{n_{i+1}}$ is thus conflict-free. And by the argument above $\mathcal{E}_{n_{i+1}}$ covers $q_1, \cdots, q_{i+1}$. $\square$

Finally we may define the quasi-stable extension (QSE) of a logic program $\mathbb{P}$ with respect to an ordering $\mathcal{O}$ of the body atoms of $\mathbb{P}$. Let $\langle \mathcal{E}_{n_m}\rangle$ and $\langle H_{n_m}\rangle$ be the canonical subsequences of $\mathbb{P}$. Set $H = \bigcup H_{n_m}$. Then $QSE = lfp(F_{P,\mathcal{E}_0\cup H})$.

6.2 Properties of Quasi-stable Semantics

Theorem 6.1.3 shows that the QSE always exists for arbitrary logic programs and is unique for a given ordering. We can also conclude:

Corollary 6.2.1 *The QSE of a logic program $\mathbb{P}$ is a Herbrand model of $\mathbb{P}$.*

Proof. As each $\mathcal{E}_0 \cup H_{n_i}$ is consistent with respect to $\mathbb{P}$, so is $\mathcal{E}_0 \cup H$. It then follows that QSE is conflict-free.

By (6.1) of lemma 6.1.3, $\mathcal{E}_{n_i} \subseteq QSE$. By part (iii) of theorem 6.1.3, QSE covers all the body atoms of $\mathbb{P}$. We show QSE also covers each head atom.

If p is a simply intensional atom there is a rule $p :\perp l_1, \cdots, l_n$ in $\mathbb{P}$. If for some such rule, each $l_i \in \bigcup \mathcal{E}_{n_i}$ then $p \in \bigcup \mathcal{E}_{n_i}$ by the T_P operator. Otherwise, for each such rule, some $\overline{l_i} \in \bigcup \mathcal{E}_{n_i} \subseteq QSE$ then *not* $p \in QSE$ by the $N_{P,L}$ operator. In either case QSE covers all atoms of the Herbrand base of $\mathbb{P}$. So QSE is a Herbrand interpretation.

Since QSE is a fixed point of $F_{P,\mathcal{E}_0 \cup H}$, QSE is closed under T_P. That is to say each rule of $\mathbb{P}$ is satisfied by QSE. Therefore, QSE is a Herbrand model of $\mathbb{P}$. □

Corollary 6.2.2 *If an atom p is a logical consequence of $\mathbb{P}$, then p is in any quasi-stable extension of $\mathbb{P}$.*

Proof. This is immediate from corollary 6.2.1. □

It is proved in [47] that any stable model of a logic program $\mathbb{P}$ is a minimal Herbrand model of $\mathbb{P}$. It can be shown that any quasi-stable extension also has the same property. An auxiliary result is first needed about the well-founded partial extension of a logic program.

Lemma 6.2.1 *Given a logic program $\mathbb{P}$. Let $\mathcal{E}_0$ be the well-founded partial extension of $\mathbb{P}$. For any atom $p \in \mathcal{E}_0$, there is a rule $r \in \mathbb{P}$ such that $head(r) = p$ and $body(r)$ is satisfied by $\mathcal{E}_0 \perp \{p\}$.*

Proof. By lemma 5.2.1, $\mathcal{E}_0 = \bigcup W_P \uparrow \alpha$. For any atom $p \in \mathcal{E}_0$, let α be the least ordinal such that $p \in W_P \uparrow \alpha$. Since $p \in W_P \uparrow \alpha$, there must be a rule $r \in \mathbb{P}$ such that $head(r) = p$ and $body(r) \subseteq W_P \uparrow \alpha \perp 1$. By the leastness of α, $p \notin body(r)$. It then follows that $body(r)$ is satisfied by $W_P \uparrow \alpha \perp 1 \subseteq \mathcal{E}_0 \perp \{p\}$. □

Theorem 6.2.1 *Any quasi-stable extension of $\mathbb{P}$ is a minimal Herbrand model of $\mathbb{P}$.*

Proof. Let QSE be a quasi-stable extension of $\mathbb{P}$. We show that QSE is also a minimal Herbrand model. We only have to show QSE is minimal. In order to do so, we show that, for any atom p in QSE, $(QSE \perp \{p\}) \cup \{not\, p\}$ is not model of $\mathbb{P}$.

We first note that for any atom $p \in QSE$, $\mathbb{P}, \mathcal{E}_0 \cup H \vdash p$ by lemma 6.1.2 (ii). It then follows that there exists a least n_i such that $\mathbb{P}, \mathcal{E}_0 \cup H_{n_i} \vdash p$. We show that there exists an integer j such that

$$\mathbb{P}, \mathcal{E}_0 \cup H_{n_j} \vdash p \text{ and } p \notin H_{n_j}. \tag{6.6}$$

If $p \notin H_{n_i}$, we are done. If $p \in H_{n_i}$. By the choice of n_i, it must be the case that $H_{n_i} = H_{n_{i-1}}{}^\wedge\langle p\rangle$ and $p \notin H_{n_{i-1}}$. It follows from the definition of $\langle H_n\rangle$ that $\mathbb{P}, \mathcal{E}_0 \cup H_{n_{i-1}} \vdash p$. So we have shown that (6.6) holds.

Now let $\mathcal{E}' = (\mathcal{E} \perp \{p\}) \cup \{not\ p\}$. If $p \in \mathcal{E}_0$, by lemma 6.2.1, there is a rule falsified by $\mathcal{E}_0 \perp \{p\}$ and hence by $\mathcal{E}'$. So $\mathcal{E}'$ is not a model of $\mathbb{P}$. If $p \notin \mathcal{E}_0$, then we have $\mathcal{E}'$ is a Herbrand model of $\mathcal{E}_0 \cup H_{n_j}$ but it is not a model of p, it follows from (6.6), $\mathcal{E}'$ is not a model of $\mathbb{P}$. □

The reverse is not true. For a counterexample, see the program $\mathbb{P}_4$ in the next section.

In [105], it is shown that the well-founded semantics has a very close relationship with the stable semantics. Specifically, it is shown that well-founded total models are unique stable models, and the well-founded partial model of a logic program $\mathbb{P}$ is a subset of every stable model of $\mathbb{P}$. This relationship holds between the well-founded semantics and the quasi-stable semantics, as is immediate from the definition of the quasi-stable semantics. We put the facts about the relationship in the following two theorems:

Theorem 6.2.2 *If a logic program $\mathbb{P}$ has a well-founded total model, then the model is the unique quasi-stable extension of $\mathbb{P}$.*

Theorem 6.2.3 *The well-founded partial model of $\mathbb{P}$ is a subset of every quasi-stable extension of $\mathbb{P}$.*

We have shown that the quasi-stable semantics has as close a relationship with the well-founded semantics as the stable model semantics does. Furthermore, we can show that there is a sense in which the quasi-stable semantics is a generalisation of the stable model semantics.

We recall that, in its original form, a stable model is two-valued. Any stable model is represented as a set of positive atoms, with missing atoms from the model being taken as its negative literals. If we explicitly represent its negative literals, we obtain a natural representation of a stable model as a set of literals. In the following, for any stable model Δ, we shall use Δ_1 for its representation as a set of literals. That is $\Delta_1 = \Delta \cup not \cdot \overline{\Delta}$, where $\overline{\Delta} = \mathcal{H}B_P \perp \Delta$.

In definition 5.7.2, a stable model is defined by the stability transformation. But it can also be characterised using a parametric form of the operator T_P as follows.

Lemma 6.2.2 *Given a logic program $\mathbb{P}$, and a set Δ of atoms, let $S_{P,\Delta}$ be the stability transformation of $\mathbb{P}$ relative Δ. Then Δ is a stable model iff $\Delta = lfp(T_{P,not\cdot\overline{\Delta}})$, where $T_{P,not\cdot\overline{\Delta}}(E)) = T_P(E \cup not \cdot \overline{\Delta})$*

Proof. By the definition of stable model, Δ is stable if and only if Δ is the minimal model of $S_{P,\Delta}$, which can be characterised as the least fixed-point of $T_{S_{P,\Delta}}$ [103].

By lemma 5.2.1, we have the following:

$$\begin{aligned} lfp(T_{S_{P,\Delta}}) &= \bigcup T_{S_{P,\Delta}} \uparrow \alpha \\ lfp(T_{P,not\cdot\overline{\Delta}}) &= \bigcup T_{P,not\cdot\overline{\Delta}} \uparrow \alpha \end{aligned}$$

So, in order to prove the lemma, we show that for each α,

$$T_{S_{P,\Delta}} \uparrow \alpha = T_{P,not\cdot\overline{\Delta}} \uparrow \alpha \tag{6.7}$$

For limit ordinals, (6.7) holds trivially. Suppose that (6.7) holds for α. We show that it also holds for $\alpha + 1$. For any $p \in T_{S_{P,\Delta}} \uparrow \alpha + 1$, there must exist a rule in $S_{P,\Delta}$ such that $head(r) = p$ and $body(r) \subseteq T_{S_{P,\Delta}} \uparrow \alpha$. By the induction hypothesis, we have $body(r) \subseteq T_{P,not\cdot\overline{\Delta}} \uparrow \alpha$. But for a rule r in $S_{P,\Delta}$, there must exist a rule r' in $\mathbb{P}$ such that $head(r') = head(r)$ and $body(r')^+ = body(r)$ and $body(r')^- \subseteq \overline{\Delta}$. It then follows that $p \in T_{P,not\cdot\overline{\Delta}} \uparrow \alpha + 1$. So we have $T_{S_{P,\Delta}} \uparrow \alpha \subseteq T_{P,not\cdot\overline{\Delta}} \uparrow \alpha$.

Similarly, we can show $T_{P,not\cdot\overline{\Delta}} \uparrow \alpha+1 \subseteq T_{S_{P,\Delta}} \uparrow \alpha+1$. So, (6.7) holds for $\alpha+1$. $\Box$

The following indicates a relationship between the least fixed point of $F_{P,L}$ and the representation of a stable model Δ as a set Δ_1 of literals.

Lemma 6.2.3 *Given a logic program $\mathbb{P}$. For any stable model Δ_1 of $\mathbb{P}$, If $L \subseteq \Delta_1$, then $lfp(F_{P,L}) \subseteq \Delta_1$.*

Proof. By lemma 5.2.1, $lfp(F_{P,L}) = \bigcup F_{P,L} \uparrow \alpha$. So, we have to show that for each α,

$$F_{P,L} \uparrow \alpha \subseteq \Delta_1 \tag{6.8}$$

For limit ordinals, (6.8) holds trivially. Suppose that (6.8) holds for α. We need to show (6.8) holds too for $\alpha + 1$. For any literal $l \in F_{P,L} \uparrow \alpha + 1$. We have to consider three cases.

Case I. If $l \in L$, then by the given assumption in the lemma we have $l \in \Delta_1$.

Case II. If $l \in T_P(F_{P,L} \uparrow \alpha)$, then l is a positive literal p, and there exists a rule $r \in \mathbb{P}$ such that $p = head(r)$ and $body(r) \subseteq F_{P,L} \uparrow \alpha$. By the induction hypothesis, we have $F_{P,L} \uparrow \alpha \subseteq \Delta_1$. So we have that *body*$(r) \subseteq \Delta_1$. By lemma 6.2.2, it then follows that $p \in \Delta \subseteq \Delta_1$.

Case III. If $l \in not \cdot N_{P,L}(F_{P,L} \uparrow \alpha)$, let $l = not\, p$. Then for any rule $r \in \mathbb{P}$ with $head(r) = p$, either there exists an atom q_i such that $not\, q_i \in F_{P,L} \uparrow \alpha$ or there exists a literal $not\, q_j$ such that $q_j \in F_{P,L} \uparrow \alpha$. By the induction hypothesis, we have either $q_i \notin \Delta$ or $q_j \in \Delta$. In either sub-case, we conclude by lemma 6.2.2 that the rule will not contribute to the derivation of p from $\mathbb{P}$ relative to $\Delta \cup not \cdot \overline{\Delta}$. So we have $l \in not \cdot \overline{\Delta}$. □

With the representation of a stable model as a set of literals and the above two lemmas, we can now state the relationship between the stable model semantics and the quasi-stable semantics.

Theorem 6.2.4 *Given a logic program $\mathbb{P}$. If Δ is a stable model of $\mathbb{P}$, then Δ_1 is a quasi-stable extension of $\mathbb{P}$.*

Proof. We first construct a sequence $\langle \mathcal{E}_n \rangle$ of extensions and a sequence $\langle H_n \rangle$ of sequences as follows:

Let $\mathcal{E}_0 = WFE$, and $H_0 = \emptyset$, where WFE is the well-founded partial extension of $\mathbb{P}$. Suppose $\mathcal{E}_n$ and H_n have been defined. Let $\mathcal{E}_{n+1}$ and H_{n+1} be defined by the following:

$$\begin{aligned} H_{n+1} &= H_n\,{}^\wedge\langle not\; q\rangle \\ \mathcal{E}_{n+1} &= lfp(F_{P,\mathcal{E}_0 \cup H_{n+1}}). \end{aligned}$$

where q is an atom in $\overline{\Delta}$ that is not covered by $\mathcal{E}_n$.

According to a result about the relationship between stable models and WFS partial models [105], we have $\mathcal{E}_0 \subseteq \Delta_1$. From the above construction of $\langle H_n \rangle$, $H_n \subseteq \Delta_1$. It then follows that each $\mathcal{E}_0 \cup H_n$ is consistent with respect to $\mathbb{P}$, and thus

$\mathcal{E}_n$ is conflict-free. So, H_n and $\mathcal{E}_n$ are in fact the canonical subsequences of $\mathbb{P}$.

To finish the proof, we have to prove that the construction above will end up with Δ_1. We show that $\Delta_1 = QSE$.

Since $\mathcal{E}_0 \subseteq \Delta_1$ and $H_n \subseteq \Delta_1$, by lemma 6.2.3, $QSE \subseteq \Delta_1$. We have shown that QSE covers all the atoms of $\mathbb{P}$ in corollary 6.2.1, so it must be the case that $QSE = \Delta_1$. □

Note that the converse of theorem 6.2.4 is not necessarily true. We say that the quasi-stable semantics generalises the stable model semantics. For a counterexample, see the program $\mathbb{P}_3$ in the next section.

6.3 Examples

In this section, we consider some example programs often used in literature.

Example 6.3.1 Let $\mathbb{P}$ be the logic program with the following four rules:

$$\begin{aligned} a &:\perp \quad \textit{not}\ b.\\ b &:\perp \quad \textit{not}\ a.\\ p &:\perp \quad a.\\ p &:\perp \quad b. \end{aligned}$$

We have seen in Chapter 5 that the program $\mathbb{P}$ has the empty set as its well-founded extension though it is reasonable to expect that p should hold.

In order to find its quasi-extensions, we start from the empty set $\emptyset$, choose the atom a from the uncovered body atoms a and b, and assume it false[4]; that is *not* a. Using this assumption, it follows that b and p from $\mathbb{P}$. So $\{\textit{not}\ a, b, p\}$ is a quasi-stable extension of $\mathbb{P}$. Through backtracking, we get another quasi-stable extension $\{a, \textit{not}\ b, p\}$ of $\mathbb{P}$.

[4]Recall that our assumptions can be restricted to body atoms.

Note that this program also has $\{not\ a, b, p\}$ and $\{a, not\ b, p\}$ as its stable models. This evidence can also be used to strengthen our claim that the quasi-stable semantics is similar to the stable model semantics. □

Example 6.3.2 Although the quasi-stable semantics may coincide with the stable model semantics for some logic programs, the two semantics may behave quite differently for other programs. The stable model semantics may have some unpleasant features for certain programs. One of the problems associated with the stable semantics, as we shown in Chapter 5, is the non-existence property. This seems inevitable given the conceptual flaw of the stable model semantics. An often cited example is the program $\mathbb{P}_2$ with the following single rule:

$$p \ :\perp \quad not\ p.$$

The two-valued stable semantics is not defined for the program $\mathbb{P}_2$, that is to say $\mathbb{P}_2$ has no stable model. The three-valued stable semantics is defined for $\mathbb{P}_2$. But the unique three-valued stable extension of $\mathbb{P}_2$ is the empty set. So the non-existence property of two-valued stable semantics is converted to truth-gap of atoms in the three-valued stable semantics. The three-valued stable semantics does not really solve the non-existence problem of two-valued stable semantics.

In contrast, the quasi-stable semantics has $\{p\}$ as its unique quasi-stable extension. Obviously *not* p cannot be assumed without inconsistency, one step of consistency-recovery enables us to establish p, giving us the unique quasi-stable extension $\{p\}$. □

Example 6.3.3 Consider another program $\mathbb{P}_1$ with the following rules:

$$q \ :\perp \quad not\ p.$$

$$p \ :\perp \quad q.$$

We have shown in Chapter 5 that this program has no two-valued stable model though it does have the empty set $\emptyset$ as its unique three-valued stable model. Given the fact that p is a logical consequence of $\mathbb{P}_1$, it is reasonable to require that any semantics should assign truth-value true to p.

In the quasi-stable semantics, $\mathbb{P}_1$ has $\{p\}$ or equivalently $\{p, \mathit{not}\ q\}$ as its unique quasi-stable extension. To see this, first we compute the well-founded extension of $\mathbb{P}_1$. It is the empty set $\emptyset$. Then we make a negative assumption say *not* p from the uncovered atoms p, q. It follows that we have q and thus p, a contradiction. By revising the assumption, we get p. Since we cannot infer anything from $\mathbb{P}_1$ using p, we make another assumption *not* q. There is no further contradiction and there is no atom left, so we conclude with $\{p, \mathit{not}\ q\}$ as desired. Note if our first assumption is *not* q instead of *not* p, we still get the same result. $\Box$

Example 6.3.4 Consider the program $\mathbb{P}_3$ with the following four rules:

$$\begin{aligned} a &:\perp \quad \mathit{not}\ b. \\ b &:\perp \quad \mathit{not}\ a. \\ p &:\perp \quad \mathit{not}\ p. \\ p &:\perp \quad \mathit{not}\ b. \end{aligned}$$

We have seen in Chapter 5 that the program $\mathbb{P}_3^1$ with only the first two rules has two stable models: $\{a, \mathit{not}\ b\}$ and $\{\mathit{not}\ a, b\}$. Anomalously, when the third rule is added to $\mathbb{P}_3^1$, the resulting program $\mathbb{P}_3^2$ has no stable models at all. The addition of the fourth rule to $\mathbb{P}_3^2$ also has an anomalous effect; it stabilises one of two minimal models of $\mathbb{P}_3^2$, giving us an unique stable model for the full program $\mathbb{P}_3$.

Given the fact that the third rule is independent of the first two, and the fact that that $p :\perp \mathit{not}\ p$ logically implies p, it is reasonable to expect that $\mathbb{P}_3^2$ should have $\{a, \mathit{not}\ b, p\}$ and $\{\mathit{not}\ a, b, p\}$ as its intended meaning.

Since p is already a logical consequence of $\mathbb{P}_3^2$, the addition of the fourth rule to $\mathbb{P}_3^2$, should not have any effect on its meaning; that is both $\mathbb{P}_3^2$ and $\mathbb{P}_3$ should have the same meaning.

In the quasi-stable semantics, $\mathbb{P}_3^1$ has two quasi-stable extensions: $\{a, not\, b\}$ and $\{not\ a, b\}$, both $\mathbb{P}_3^2$ and $\mathbb{P}_3$ still have two quasi-stable extensions $\{a, not\ b, p\}$ and $\{not\ a, b, p\}$ as expected. □

Example 6.3.5 Let $\mathbb{P}_4$ be the logic program with the rule

$$p \quad :\perp \quad not\ q.$$

The minimal models of $\mathbb{P}_4$, represented as sets of literals, are $\{not\, q, p\}$ and $\{not\, p, q\}$. In the community of logic programming, it is $\{not\ q, p\}$ that is accepted as the intended meaning of $\mathbb{P}_4$. The WFE of $\mathbb{P}_4$ is $\{not\ q, p\}$. So, in this case, the well-founded semantics can capture the intended meaning of $\mathbb{P}_4$. Since all the atoms in $\mathbb{P}_4$ are covered by $\{not\ q, p\}$, $\mathbb{P}_4$ has $\{not\ q, p\}$ as its unique quasi-stable extension, giving us a simple counterexample to the converse of theorem 6.2.1.

Recall that the inference mechanism behind the quasi-stable semantics is the combination of non-monotonic and hypothetical reasoning as well as deductive one. It is the use of non-monotonic reasoning that enables us to capture the intended meaning of $\mathbb{P}_4$. Without appealing to non-monotonic reasoning, some extra criteria such as admissibility [37] are needed in order to assign an appropriate meaning to $\mathbb{P}_4$. □

6.4 Discussion

6.4.1 Supportedness

In the community of logic programming and deductive databases, the notion of a supported model introduced in [5] has been claimed to be an important one. A set L of literals is supported by a logic program if for any positive literal $p \in L$, there is rule r in $\mathbb{P}$ such that $head(r)$ is p and $body(r) \subseteq L$.

In [15], supportedness is taken as a desirable property of the intended meaning of a logic program although it is recognised that the property itself is not a sufficient condition for assigning an appropriate meaning to a logic program.

The quasi-stable semantics is not supported in the sense that a quasi-stable extension is not necessarily supported. For instance, consider a program $\mathbb{P}$ with the following rules:

$$\begin{array}{lll} q & :\perp & not\ p. \\ p & :\perp & q. \end{array}$$

The program $\mathbb{P}$ has an unique quasi-stable extension $\{p, not\ q\}$. But the extension is not supported since p is not any immediate consequence of $\mathbb{P}$ and $\{p, not\ q\}$, that is there is no rule r in $\mathbb{P}$ such that $head(r)$ is p and $body(r) \subseteq L$.

However, the fact that the quasi-stable semantics is not supported does not mean that the quasi-stable semantics is not an adequate model of nonmonotonic negation. Indeed, it can in fact be argued that the supportedness cannot be justified in the context of nonmonotonic reasoning.

To begin with, we note that to insist that proper models of logic programs be supported is to restrict inference forms whereby positive information can be inferred. In particular, the requirement of supportedness implicitly prevents us from using any possible non-monotonic reasoning to infer positive information. However, this seems not consistent with the fact that non-monotonic reasoning is one of main features of logic programming.

We have said many times before that the essential characteristic of negation *not* is nonmonotonic. A piece of negative information such as *not* q may be given up later when more information is available. However, we should not neglect the fact that positive information may also be infected with the non-monotonicity of negative information, a fact which we noted in the discussion of the Fitting semantics before. This fact is still true of the stable model semantics as the same example illustrates as follows.

Let $\mathbb{P}_3$ consist of following two rules

$$\begin{aligned} p &: \perp \textit{not } q. \\ r &: \perp q. \end{aligned}$$

$\mathbb{P}_3$ has $\{p, \textit{not } q, \textit{not } r\}$ as its unique stable model. When $\mathbb{P}_3$ is augmented with q, its stable model changes to $\{q, r, \textit{not } p\}$. So what is originally not true now becomes true when more information is added. Moreover, like that in the Fitting semantics, what is originally true is also affected. Once again, this is because of the use of the rule $p : \perp \textit{not } q$, by which p is infected with the non-monotonicity of *not* q.

Since positive information can be nonmonotonic just as negative information even in the stable model semantics (which is supported), there seems no reason to forbid us from *nonmonotonically* inferring a piece of positive information. In other words, it would not be unreasonable for us to conclude p if the assumption *not* p is not consistent. Recall that the main feature of the quasi-stable semantics is the introduction of consistency-recovery mechanism, which in turn is based on the Classical Reductio ad absurdum (RAA): p is inferred whenever *not* p cannot be consistently assumed. The non-supportedness of the quasi-stable semantics in essential is the consequence of RAA. Since the inference form of RAA is not unreasonable as we just argued, there seems no reason to rule out non-supportedness.

Moreover, there is another point to consider. Remember that one of the main motivations for the use of nonmonotonic negation is that classical logic only provides us with a monotonic mechanism for inferring negative information. And further the non-persistent semantic rule M2 is not adequate in the context of logic programming. A logic program can be regarded as a default theory to extend classical logic. As such, it is natural to require that any adequate semantics for a logic program should assign the truth-value *true* to those atoms which are logical consequences of the program as we did in the quasi-stable semantics. Unfortunately, the supportedness condition may prevent us from doing so. Hence we conclude that to require models of logic programs to be supported is to violate but not extend classical logic, and thus once again cannot be justified.

6.4.2 Inference Forms in the Quasi-stable Semantics

Another characteristic of our semantics is its use of both hypothetical and non-monotonic reasoning to extend classical logic. The non-monotonic reasoning we use is in a very specific form. It is based upon the operations used in the WFS [105] and the Fitting semantics [40], that is the operator U_P and a variant of the Fitting operator F_P.

Our use of hypothetical reasoning is reminiscent of Poole [82] and Dung [37]. In [82], Poole applies hypothetical reasoning to extend classical logic for dealing with non-monotonic reasoning. He argued that if one allows hypothetical reasoning, then first-order logic itself is adequate to handle non-monotonic reasoning.

In logic programming, there is a similar proposal based upon abduction. By incorporating abducibles into the rules of a logic program, logic programming is extended to allow abductive reasoning, resulting in abductive logic programming [58]. In an abductive framework for logic programming, negative literals are interpreted as abducible hypotheses that can be assumed to hold subject to certain integrity constraints. Based upon the abductive interpretation of negative literals, an argumentation-theoretic framework for logic programming has been proposed in [37]. Furthermore, in [19] it is claimed that the framework can be further abstracted so as to provide an argumentation-theoretic approach to default reasoning in general.

However, the use of hypothetical reasoning in [82] and [37] is somewhat limited in one sense. Deductive reasoning is carried out from the given theory augmented with a set of assumptions. No non-monotonic reasoning is involved. Indeed, the motivation for the use of hypothetical reasoning is to replace non-monotonic reasoning [82]. In the context of logic programming, as a result, additional criteria such as admissibility [37] for choosing among sets of assumptions are needed in order for the argumentation-theoretic framework to properly define the semantics of logic programming. The resulting semantics called *preferred extension semantics* in [37], however, still has similar problems to the three-valued stable semantics (see the next section).

In contrast, our use of hypothetical reasoning does not have the above restriction. We shall use not only hypothetical but also non-monotonic reasoning in addition to deductive reasoning. Consequently, we do not need to appeal to any extra criteria such as admissibility for choosing among different sets of assumptions. We shall show by example that this is because of the use of non-monotonic reasoning.

In addition, the use of hypothetical reasoning for us is also different. Assumptions are made individually. That is to say, each time, we only consider a single assumption rather than a set of assumptions as in [82], [37]. If an assumption results in a contradiction, we may modify or withdraw it, depending on whether it is negative one or positive one.

For us, deductive reasoning, non-monotonic reasoning and hypothetical reasoning are interleaved, subject to the constraint that assumptions be made only when logical reasoning and non-monotonic reasoning can no longer infer any more information. In other words, we have implicitly used the following priority relation[5] among the different kinds of reasoning in the definition of quasi-stable semantics:

$$\begin{array}{rcl} \text{Logical Reasoning} & \prec & \text{Hypothetical Reasoning} \\ \text{Non-monotonic Reasoning} & \prec & \text{Hypothetical Reasoning} \end{array}$$

where the left hand side of the relation $\prec$ has priority over the right hand side.

6.5 Related Work

In the study of semantics for logic programs, a whole spectrum of semantic theories for logic programs with nonmonotonic negation have been proposed, ranging from those that may infer very little information from a logic program ("sceptical") to those that always infer a great deal ("credulous"). At the sceptical extreme there is the well-founded semantics WFS [105] while at the credulous one is the stable model semantics [47]. The WFS and the stable model semantics are usually taken as two dominant ones. In this section, we briefly discuss some related work on the extensions of the WFS and the stable model semantics.

6.5.1 WFS Extensions

On the one hand, we have seen in the last chapter that the WFS has many desirable features, and based on this we have chosen the WFS [105] as our starting point. On the other hand, the WFS despite its merits has been criticised for being too "sceptical" since, for many programs, it gives the empty set as their intended meaning. We have shown by example in Chapter 5 that the WFS may be silent on all atoms

[5]For us, logical reasoning and non-monotonic reasoning are independent of each other in the sense they cannot be applied to the same rule. So we shall not impose any priority relation on them though it is reasonable to require that logical reasoning have priority over non-monotonic reasoning.

which are logical consequences of a logic program.

It is the sceptical characteristic of the WFS that motivates its various extensions including the generalised well-founded semantics GWFS [7], the well-founded-by-case-semantics WFS_C [97], the extended well-founded semantics WFS_E [57], the strong well-founded semantics WFS_S [22], and the O-semantics [81]. The relationship between these semantic theories for logic programming is also studied in [33] and [34]. Furthermore, an abstract axiomatic framework for defining semantics of logic programs is introduced in [34]. In particular, a well-behaved semantics is proposed. All these semantic theories have to some extent extended the WFS in one way or another so that more information can be inferred from a logic program. A common feature among these theories is that they are all deterministic.

The quasi-stable semantics proposed in this chapter gives another extension of the WFS. Without exception, our extension is also motivated by the sceptical characteristic of the WFS. However our approach to extending the WFS is different. We use nonmonotonic and hypothetical reasoning in our approach, which is a main feature of the quasi-stable semantics as we discussed before. Consequently it is not surprising to see that the quasi-stable semantics differs from the semantical theories mentioned above in that it is non-deterministic. We do not know what relationship holds between the deterministic part of the quasi-stable semantics, that is the intersection of all quasi-stable extensions, and the existing deterministic extensions. It is worth further investigation.

6.5.2 Stable Model Semantics and Its Variants

The other dominant semantic model for logic programs is the stable model semantics [47]. Compared with the WFS, the stable model semantics is “credulous”: it derives much more information than the WFS though this is non-deterministic or “disjunctive” in the sense that there may be several stable models for a given logic program. In the previous chapter, we have shown that a stable model is not defined for all logic programs and the stable model semantics may give rise to anomalies in some circumstances. These shortcomings, we have argued, arise because the underlying definition of stable models is to some extent inconsistent with the essential characteristics of negation *not* used in logic programs. It is this kind of tension that underlies the existence problem(s) of the stable model semantics.

A modification of the stable model semantics is the three-valued stable semantics [84]. In this three-valued version, all logic programs have at least one model. The three-valued version seemingly solved the existence problem. But the solution is rather superficial. Although each program has at least one three-valued stable model, the model may leave all atoms undetermined for some programs, which is not fully satisfying as it seems once again to say that nothing can be concluded. The non-existence of the stable model semantics is now converted to truth gaps in the three-valued version. It is our opinion that the problem is still there though in a somewhat different form. More importantly, our criticism concerning the conceptual flaw of the stable model semantics continues to apply to the three-valued version.

There are also other variants of the stable model semantics, including partial stable model semantics [95] and preferred extension semantics [37]. Although we did not discuss these variants, we believe that our criticism is still true of them since it has been proved that they are in fact equivalent to each other. For the equivalence between partial models and three-valued models, see [96]; for the equivalence between partial stable models and preferred extensions, see [59]. In [34] and [35] two other variant of stable model semantics, STABLE' and STABLE$^+$ are proposed respectively by combining the well-founded semantics and stable models in different ways. Like the stable model semantics, neither STABLE' nor STABLE$^+$ has any mechanism for revising provisionally assumed negative information when inconsistency occurs. Moreover, both STABLE' and STABLE$^+$ are somewhat artificial.

It turned out that the quasi-stable semantics we described in this chapter has a close relationship with the stable model semantics, as was shown by theorems 6.2.2, 6.2.3 and 6.2.4 in section 6.2. It is based on this close connection that we called our semantics the *quasi-stable extension semantics* or simply the quasi-stable semantics. In comparison to the stable model semantics, our semantics avoids its difficulties. This is because of the introduction of consistency-recovery mechanism which allows us to revise tentatively-made negative assumptions if necessary in light of newly discovered information. Such a mechanism we believe is indispensable to model nonmonotonic negation.

6.6 Concluding Remarks

In this chapter, we have introduced a new semantics, called the quasi-stable semantics. It naturally extends the well-founded semantics using hypothetical reasoning. We have shown that the quasi-stable semantics solves the non-existence problem, and demonstrated by example that it also dissolves anomalies associated with the stable semantics.

In summary, the quasi-stable semantics has the following features (i) every logic program has at least one quasi-stable extension, (ii) a quasi-stable extension of a logic program $\mathbb{P}$ is a total model of $\mathbb{P}$, (iii) a quasi-stable extension of $\mathbb{P}$ is minimal in the sense that no positive literal can be replaced in the extension by a negative one without its ceasing to be a model of $\mathbb{P}$, (iv) the well-founded partial model for a logic program $\mathbb{P}$ is included in every quasi-stable extension of $\mathbb{P}$, (v) every stable model is a quasi-stable extension.

Chapter 7

Quasi-stable Semantics with Strong Negation

The normal logic programs we studied in the last two chapters have only one kind of negation: non-monotonic negation. In this chapter, we study the semantics of logic programs with two different kinds of negation: non-monotonic negation and strong negation. Our discussion shall be mainly based on the work in Gelfond and Lifschitz [48], [49], Pearce and Wagner [80], Wagner [110], and our own work in the last chapter. In the following, we shall first of all review the main motivations behind the two kinds of negation, and then give the definition of an extended logic program, followed by a review of the answer set semantics and its modification under the quasi-stable semantics. Then we consider some examples from the literature to argue why the term *classical negation* in [48] and [49] is a misnomer and to show how strong negation can be used in extended logic programs for knowledge representation. Finally we briefly discuss the relation between strong negation and non-monotonic negation.

7.1 Strong Negation in Logic Programs

Gelfond and Lifschitz in [48] and [49] point out that a ground query against a normal logic program will always return a definite *yes* or *no*. However (as we noted in Chapter 2) classical logic allows three possibilities: the query can be proved, it can be refuted or neither is the case. The third possibility should be admitted into logic programs: it should be possible to answer *unknown* to a ground query.

To achieve this they proposed to extend logic programs with what they called "classical negation". This results in *extended logic programs*.

Although their criticism is true of definite logic programs, it is in general not true of normal logic programs. Under the WFS, a ground query against a normal logic program may be undetermined. Under the stable model semantics there is indeterminacy because a logic program may have more than one stable model.

The problem with traditional logic programming, in our opinion, is not so much *whether* it is able to deal with incomplete information but *how* incomplete information is dealt with.

Normal logic programs provide negative information through one or another non-monotonic inference mechanism. Negative instances of extensional predicates are implicitly assumed through closed-world reasoning. Negative instances of intensional predicates are obtained in a much more complicated way, depending on what semantics is used. Under the quasi-stable semantics, non-monotonic inference is the combination of the operator U_P, the Fitting operator $F_{P,L}$, and hypothetical reasoning. Whatever semantics is used, information expressed by non-monotonic negation is defeasible.

In Chapter 4, we briefly argued that there are situations where non-monotonic negation is not suitable. Instead, what is needed is a kind of negation which can be used to express negative information in a monotonic (that is persistent) way. This consideration led us to the use of strong negation. The idea of using strong negation in logic programming and deductive databases is not novel. It was proposed by Pearce and Wagner in [79] and [80], and further developed by Wagner in [108], [109] and [110]. Although [48] and [49] call the second negation "classical negation", it is argued in [109], that this is in fact strong negation. See section 7.4 for the argument and related example in [109]. We shall give further arguments to show term "classical negation" is indeed a misnomer.

In contrast to non-monotonic negation, the strong negation of an atomic fact is to be established directly just as positive atomic facts are. From the database viewpoint, this requires the explicit inclusion of strong negative facts in extensional databases. In other words, strong negation can be used to express explicit negative information. One significant implication is the persistence of strong negative information; a piece of strong negative information will not be subject to change when more information is added.

Another desirable characteristics of strong negation is its capacity for handling incomplete information. We have pointed out in Chapter 3 that constructive logic with strong negation can accommodate partiality. So it is not surprising to discover that logic programs with strong negation inherit the same capability.

The introduction of strong negation does not give rise to any new computational difficulties. In comparison to classical negation, it has been proved that strong negation is simpler to implement (see [109] and [12]).

7.2 Extended Logic Programs

Extended logic programs introduced in [48] and [49] have two kinds of negation: non-monotonic negation and classical negation. We have pointed out in the last section that term *classical negation* is a misnomer, so we shall define an extended logic program as a logic program with non-monotonic negation and strong negation.

Definition 7.2.1 An extended logic program is a set of rules of the form

$$sl_0 : \perp \; sl_1, sl_2, \ldots, sl_m, not\ sl_{m+1}, \ldots, not\ sl_n.$$

where $n \geq m \geq 0$, and each sl_i is an atom p or the strong negation of an atom $\sim p$.

In [49], a literal is an atom p or the strong negation[1] of the atom $\sim p$. Since we have used the term *literal* either for an atom p or for its non-monotonic negation *not p*, we instead use the term *strong literal* for an atom or its strong negation.

In the above definition of an extended logic program, strong negation is allowed to appear both in the head and in the body of a rule. That is to say, an extended logic program may have a strong negative conclusion as well as positive one. So, positive and strong negative information in an extended logic program are treated with equal importance. It is interesting to note that such an equal importance follows directly from the symmetry of the logic of strong negation.

In contrast, positive and non-monotonic negative information are treated in different ways in normal logic programs. Non-monotonic negation may only appear in the body of a rule, and the derivation of negative facts is different from that of positive ones as we have seen in the last two chapters. Such difference still exists in extended logic programs.

Similarly, a deductive database $\langle DB, \mathbb{P} \rangle$ can also be extended by incorporating strong negation into the extensional database DB and the intensional one $\mathbb{P}$. An extended extensional database consists of strong literals rather than just atoms, and an extended intensional database is an extended logic program. Since an extended extensional database may contain both positive facts and strong negative facts, we require no atom and its strong negation to occur in it at the same time to preserve consistency. Semantics for extended databases can be reduced to semantics for

[1]They use the term classical negation instead of strong negation.

extended logic programs, so we shall concentrate on only extended logic programs in the following.

7.3 Semantics of Extended Programs

7.3.1 The Answer Set Semantics

In [49], the semantics of an extended program is defined by its answer sets, resulting in the answer set semantics. Answer sets are similar to stable models except that they consist of strong literals rather than atoms. The definition of answer sets in [49] is done in two steps.

Firstly, consider extended programs without nonmonotonic negation *not*. So we shall call them *extended definite programs*. Similar to definite logic programs, every extended definite program $\mathbb{P}$ may be associated with a unique set of strong literals, called the *answer set* of $\mathbb{P}$. Informally speaking, an answer set consists of all strong literals inferred from the rules of $\mathbb{P}$. We have seen in Chapter 5 that there are three different but equivalent approaches to assigning one set of atoms to a definite program, but we consider here only the fixpoint approach to formally defining the answer set of extended definite programs. For other approaches, see [110].

Recall that, in order to define the fixpoint characterisation of definite programs, the operator T_P on $\mathcal{H}B_P$ is used. For extended definite programs we need a similar operator $T_P^{\sim}$, which is defined on $\mathcal{S}L = \mathcal{H}B_P \cup \sim \mathcal{H}B_P$ instead of $\mathcal{H}B_P$ as follows.

$$T_P^{\sim}(X) = \{sl_0 \in \mathcal{S}L : \exists r \in \mathbb{P}(head(r) = sl_0 \wedge \forall sl \in body(r) sl \in X)\}$$

It is straightforward to prove that the operator $T_P^{\sim}$ is monotonic and thus has a least fix-point $lfp(T_P^{\sim})$. Although the least fix-point $lfp(T_P)$ of an definite logic program $\mathbb{P}$ contains only positive atoms, $lfp(T_P^{\sim})$ of an extended definite logic program, may contain both positive and strong negative literals, in particular, both q

and $\sim q$ as the following simple example shows.

Example 7.3.1 Let $\mathbb{P}$ consist of three rules

$$\begin{array}{lll} p & :\perp & . \\ q & :\perp & p. \\ \sim q & :\perp & p. \end{array}$$

$\mathbb{P}$ has $\{p, q, \sim q\}$ as its least fix-point $lfp(T_P^{\sim})$, which contains both q and $\sim q$. $\square$

In classical logic, all strong literals (among other things) can be inferred from a contradiction. Based on this fact, Gelfond and Lifschitz in [49] gave the following notion of the answer set for extended definite logic programs.

Definition 7.3.1 For any extended definite program $\mathbb{P}$, the answer set $\alpha(\mathbb{P})$ of $\mathbb{P}$ is defined as follows:

$$\alpha(\mathbb{P}) = \begin{cases} lfp(T_P^{\sim}) & \text{if } lfp(T_P^{\sim}) \text{ contains no pair of } p \text{ and } \sim p; \\ SL_P & \text{otherwise.} \end{cases}$$

where SL_P is the set of all strong literals of $\mathbb{P}$.

Before we define answer sets of arbitrary extended programs, a remark on inconsistency is in order. In the above definition, the way inconsistency is dealt with is classical. This is not the only approach to inconsistency. The so-called paraconsistent approach which is contradiction-tolerant can be taken as it is done in [17]. For us, it seems wrong from the informational viewpoint to say that we can infer everything from a contradiction. Rather, it would be more reasonable to take a contradiction as a signal that something is wrong. In the present context, a contradiction means that the extended program in question is problematic. A contradictory program needs to be modified so as to remove the contradiction. So, we shall reject any inconsistent set as an answer set on the basis of the very inconsistency, and not define any answer set for a contradictory program as we shall see in our definition of semantics for extended logic programs in the next subsection. There we shall also show another problem associated with the above definition.

Now we move on to define answer sets of arbitrary extended programs. This is done in a similar way to the definition of a stable model through reducing an extended program to a program without *not*. Given a set Δ of strong literals, let $\mathbb{P}^{\Delta}$ the extended program obtained from $\mathbb{P}$ by deleting

(1) each rule that has a formula *not sl* in its body with $sl \in \Delta$;

(2) all literals of the form *not sl* in the body of the remaining rules.

Definition 7.3.2 (Gelfond and Lifschitz [49]) Given an extended logic program $\mathbb{P}$. A set Δ of strong literals is called an answer set of $\mathbb{P}$ if and only if Δ coincides the answer set of $\mathbb{P}^{\Delta}$, that is the following equation holds:

$$\Delta = \alpha(P^{\Delta}).$$

It is easy to see that the two definitions of answer set coincide when applied to a program $\mathbb{P}$ without *not*. Moreover, the second more general definition of answer set is also a generalisation of the definition of stable model.

7.3.2 The Quasi-answer Set Semantics

In the last subsection, we reviewed the answer set semantics. Its definition is similar to that of the stable model semantics. Alternatively, the answer set semantics can be defined through firstly transforming an extended program into a normal program, and then applying the stable semantics to the resulting transformed normal program. This transformation-based approach is in fact preferable in the sense that we need not be restricted to only the stable semantics. Indeed any other semantics can be used on the transformed program. Naturally, we prefer to use the quasi-stable semantics. We first consider how to transform extended programs into normal programs.

Let $\mathbb{P}$ be an extended program. For any predicate $p(t_1, \cdots, t_n)$ occurring in $\mathbb{P}$, introduce a new predicate with the same arity, denoted $sn\_p(t_1, \cdots, t_n)$, where sn in $sn\_p(t_1, \cdots, t_n)$ is for strong negation. For any strong literal sl, define its positive form, denoted sl^+ as follows:

$$sl^+ = \begin{cases} p(t_1, \cdots, t_n) & \text{if } sl \text{ is a positive literal } p(t_1, \cdots, t_n); \\ sn\_p(t_1, \cdots, t_n) & \text{if } sl \text{ is a strong negative literal } \sim p(t_1, \cdots, t_n). \end{cases}$$

Definition 7.3.3 (Gelfond and Lifschitz [49]) To transform an extended program $\mathbb{P}$ into a normal program, denoted $\mathbb{P}^+$, is to replace any rule r in $\mathbb{P}$ of form

$$sl_0 :\perp sl_1, sl_2, \ldots, sl_m, not\ sl_{m+1}, \ldots, not\ sl_n.$$

by the following rule r^+

$$sl_0^+ :\perp sl_1^+, sl_2^+, \ldots, sl_m^+, not\ sl_{m+1}^+, \ldots, not\ sl_n^+.$$

The transformation[2] consists in replacing every occurrence of strong negation $\sim p(\vec{t})$ by a new positive atom $sn\_p(\vec{t})$. After transforming an extended program $\mathbb{P}$ into a normal program $\mathbb{P}^+$, we may apply any semantics to the resulting program $\mathbb{P}^+$ in a straightforward way. We shall consider the stable semantics and quasi-stable semantics respectively. For any set $S \subset SL$, let $S^+ = \{sl^+ : sl \in S\}$.

In [49], it is proved that for a set S of strong literals, if S contains no pair of p and $\sim p$, then S is an answer set of $\mathbb{P}$ if and only if S^+ is a stable model of $\mathbb{P}^+$. So, we may alternatively define the notion of answer set as follows.

[2]Based on the same idea, Wagner in [109] shows how to transform an arbitrary formula to a formula without any occurrence of strong negation, and also to apply the transformation to transfer an extended definite program into a definite logic program.

Definition 7.3.4 For any set $S \subset SL_P$ of strong literals, if S contains no pair of p and $\sim p$, and S^+ is a stable model of $\mathbb{P}^+$, then S is an answer set of $\mathbb{P}$.

The consistency assumption is indispensable. Without it, the correspondence between stable models and answer sets would no longer hold as the following example from [49] shows.

Example 7.3.2

$$\begin{array}{lll} p & :\perp & not \sim p. \\ q & :\perp & p. \\ \sim q & :\perp & p. \end{array}$$

This program has no answer sets whereas its transformation $\mathbb{P}^+$ does have $\{p, q, \sim q\}^+$ as its unique stable model. □

This example to some extent also suggests that the same consistency condition might be added into the original definition of answer set given in the last section. With this slight modification, we shall not only have a neat correspondence between different definitions of answer sets but also remove the annoyance that everything can be inferred from a contradiction. Motivated by these considerations, we offer the following definition of quasi-answer sets.

Definition 7.3.5 For any set $S \subset SL_P$ of strong literals, if S^+ is a quasi-stable extension of $\mathbb{P}^+$ and contains no pair of sl and sn_sl, then S is a quasi-answer set of $\mathbb{P}$.

With this definition available, it is straightforward to generalise various properties about the quasi-stable extensions we have proved in Chapter 6 to quasi-answer sets (with appropriate modifications). For example, we can show that if Δ is an answer set of $\mathbb{P}$ and contains no pair of p and $\sim p$, then it is also a quasi-answer-extension. One exception that we need to bear in mind is that an extended logic program may have no quasi-answer extension at all though any logic program does always have at least one quasi-stable extension as our simple example above indicates.

Another point worth mentioning is that, like positive atoms, the monotonicity of strong negative literals may be compromised when strong negation and non-monotonic negation are both used in an extended logic program. A piece of strong negative information when inferred from an extended logic program rather than established directly may be infected with non-monotonic negation and thus no longer monotonic. If we want to maintain the monotonicity of a piece of strong negative information, then we should be careful not have it depend on non-monotonic negation.

7.4 Why Not Classical Negation?

In this section, we have a closer look at why classical negation used in [48] and [49] is in fact strong negation. The misnomer was pointed out in [109]. Nevertheless, the term classical negation is still widely used in literature, such as [12], [20], etc. This is due to the influence of [48] and [49], which presumably in turn comes from the dominant role of stable models in semantics of logic programs.

We first of all consider the argument given by Wagner in [109]. His criticism is based on the fact that classical first-order logic is semantically two-valued. This bivalence entails the principle of the excluded middle which, as shown by the example in [109], is violated under the answer set semantics of extended logic programs given by [49]. It follows that the second negation in [48] and [49] cannot be classical. Here is the counterexample program used in [109].

Example 7.4.1 Let $\mathbb{P}$ consist of the following two rules,

$$\begin{array}{lll} p & :\perp & q. \\ p & :\perp & \sim q \end{array}$$

Had we interpreted $\sim$ in the above program as classical negation, it would have been reasonable for p to be true relative to whatever semantics is used for the program. But it is easy to see that the answer set of $\mathbb{P}$ is the empty set $\emptyset$. One the other hand, if when $\sim$ is taken as strong negation, then $\mathbb{P}$ does have $\emptyset$ as its unique answer set. $\square$

Let us now analyse an example from [49] to further support the conclusion that the use of the term "classical negation" is not appropriate.

Example 7.4.2 This example from [49] contains two programs $\mathbb{P}_1$ and $\mathbb{P}_2$, where $\mathbb{P}_1$ consists of

$$\begin{array}{lll} \sim p & :\perp & . \\ p & :\perp & \sim q. \end{array}$$

and $\mathbb{P}_2$

$$\begin{array}{lll} \sim p & :\perp & . \\ q & :\perp & \sim p. \end{array}$$

The example was used in [49] to show that the answer set semantics is not "contrapositive" with respect to $:\perp$ and $\sim$[3], in the sense that the semantics assigns different meanings to the rules $p :\perp \ \sim q$ and $q :\perp \ \sim p$. Indeed, according to the answer set semantics, $\mathbb{P}_1$ has a unique answer set $\{\sim p\}$, and $\mathbb{P}_2$ has a unique answer set $\{\sim p, q\}$. Since classical logic is contrapositive with classical implication $\rightarrow$ and classical negation, they instead claim that non-contrapositiveness is because that their semantics interprets program rules as *inference rules* rather than classical conditionals.

[3]In [49], $\rightarrow$ and $\neg$ are used instead of $:-$ and $\sim$.

We argue that even if program rules are interpreted as inference rules, any proper semantics should assign the same set to these two programs as long as in the framework of classical first-order logic. It is not controversial to assign the set $\{\sim p, q\}$ as the meaning of $\mathbb{P}_2$. So we need to show why $\mathbb{P}_1$ should also have the same set as its meaning when classical negation is used.

Let r_0 denote the program rule $p : \perp \quad \sim q$, $\supset$ the classical implication symbol, and $\vdash_{r_0}$ the deductive relation of the classical propositional logic[4] augmented with the inference rule r_0. Recall that in propositional logic there is a famous theorem called the Deduction Theorem relative to the deductive relation $\vdash$. When propositional logic is extended with the inference rule r_0, the Deduction Theorem still holds relative to $\vdash_{r_0}$. With these notes at hand, the remaining task is straightforward. First of all, we have $\sim q \vdash_{r_0} p$ using the inference rule r_0. By the Deduction Theorem, we then have $\vdash_{r_0} \sim q \supset p$. From which it follows that $\vdash_{r_0} \sim p \supset q$ by the contrapositiveness of first-order logic. By the deductive theorem again, we thus have $\sim p \vdash_{r_0} q$. That is to say, q is a logic consequence of $\mathbb{P}_1$, and it thus is reasonable to assign $\{\sim p, q\}$ as the meaning of $\mathbb{P}_1$ if classical negation is used.

On the other hand, if the negation in the programs $\mathbb{P}_1$ and $\mathbb{P}_2$ is interpreted as strong negation, then we can no longer infer q from $\mathbb{P}_1$, giving us a proper explanation why these two programs have different meaning under the answer set semantics. $\Box$

7.5 Knowledge Representation Using Strong Negation

Although non-monotonic negation has proved to be quite useful in various domains and application frameworks, it is not sufficient in some situations, and its use in

[4]Since we are only concerned of ground logic programs here, propositional logic is enough without using classical first-order logic.

logic programs can lead to undesirable results. In this section, we give some more examples from literature to show how how strong negation can be used to eliminate undesirable results caused by the use of non-monotonic negation.

Example 7.5.1 This example is from [49]. As pointed out in [49], the example is actually credited to John McCarthy. Consider the following regulation about crossing railway tracks: A school bus may cross railway tracks under the condition that there is no approaching train. How shall we express this regulation as a program rule? The key point is to interpret the negation in the condition that there is no approaching train. Had the negation been interpreted as non-monotonic negation *not*, the regulation would be expressed as

$$cross(school\_bus, railway\_tracks) : \perp \ not\ approaching(train).$$

This would mean that a school bus may cross railway tracks whenever there is no evidence that train is approaching. Although the exact meaning of no evidence depends on specific semantics used, the point is that the truth of *not approaching*$(train)$ is defeasible. When more information is available, we may find that there is in fact an approaching train. In this case, we certainly do not want the bus to cross tracks. So the above representation is not desirable.

Instead of non-monotonic negation *not*, a more satisfying representation can be obtained with the use of strong negation $\sim$:

$$cross(school\_bus, railway\_tracks) : \perp \ \sim \ approaching(train).$$

Now a school bus will not cross railway tracks unless it had the strong negative fact $\sim\ approching(train)$, which is established on the basis of direct observation and thus will not be subject to any change later. □

Example 7.5.2 This example from [49] is about the representation of terminal vertices of a directed graph, showing how they can be defined by using an extended logic program:

$$\begin{aligned} \sim terminal(X) &\ :\perp\ \ arc(X,Y). \\ terminal(X) &\ :\perp\ \ not \sim terminal(X). \end{aligned}$$

□

Example 7.5.3 This example comes from [3].

$$\begin{aligned} runs &\ :\perp\ \ not\ broken. \\ \sim runs &\ :\perp\ \ . \\ broken &\ :\perp\ \ flatTire. \\ broken &\ :\perp\ \ badBattery. \end{aligned}$$

Given a car not running, we cannot conclude that the car is not broken without inconsistency. So it must be broken, as might be caused by a flat tire or bad battery. □

Example 7.5.4 This is also from [49] though with a slight modification. The example illustrates how both non-monotonic negation and strong negation are used. Suppose that an anonymous college uses the following regulations for awarding scholarships to its students:

(1) Every student with the GPA of at least 3.8 is eligible.

(2) Every minority student with the GPA of at least 3.6 is eligible.

(3) No student with the GPA under 3.6 is eligible.

(4) The students whose eligibility is not determined by previous rules are interviewed by the scholarship committee.

The above regulations can be respectively encoded into the first four rules in the following extended program,

$$
\begin{aligned}
eligible(X) &:\perp \quad highGPA(X).\\
eligible(X) &:\perp \quad minority(X), fairGPA(X).\\
\sim eligible(X) &:\perp \quad \sim fairGPA(X).\\
interview(X) &:\perp \quad not\ eligible(X), not \sim eligible(X).\\
fairGPA(X) &:\perp \quad highGPA(X).
\end{aligned}
$$

The last rule is added to capture the relation between predicates $highGPA$ and $fairGPA$, that is, a GPA of at least 3.8 is also a GPA of at least 3.6. Its inclusion in the program can help to prevent us from putting $\sim fairGPA(ann)$ and $highGPA(ann)$ into an extended database on the basis of consistency.

$\square$

7.6 Relation between Strong Negation and Non-monotonic Negation

In this section, we make some informal remarks about the relation between strong negation and non-monotonic negation. We have seen that the main utility of strong negation is to express monotonic negative information, whereas information expressed by non-monotonic negation is not monotonic and usually subject to change when more information is available. Non-monotonic negation is weaker than strong negation, that is to say, if it is true that $\sim p$, then it is also true that $not\ p$, but the reverse is usually not true. Given this fact, we might be tempted to make this relation explicit by expressing it as

$$not\ p :\perp \sim p. \tag{7.1}$$

But this is not appropriate for two reasons. One is that, in logic programming, *not* is used only in the body but not in the head. The other is that we have nothing to lose without (7.1), that is, (7.1) is in fact redundant. Let us consider a simple example by the way of illustration.

Example 7.6.1 Let $\mathbb{P}$ consist of the following two rules.

$$\begin{array}{rcl} \sim q & :\perp & . \\ p & :\perp & \mathit{not}\ q. \end{array}$$

With (7.1), we can easily infer p from the program. However, it is not difficult to see that we can still infer p even without using (7.1). □

Both strong negation and non-monotonic negation can be applied to incomplete predicates. But there is an important difference between them. The logic of strong negation is three-valued whereas the logic of non-monotonic negation is two-valued. The three-valuedness of strong negation comes from the incompleteness of its associated predicates. However the two-valuedness of non-monotonic negation comes from the non-monotonic characteristic of inference and thus has nothing to do with its associated predicates.

There is another difference worth mentioning between strong negation and non-monotonic negation, which is in fact the consequence of the difference above. Although the answer set is identical to the stable model when the answer set semantics is applied to a normal logic program, it was pointed out in [49] that there is a crucial semantic difference: answer sets and stable models gave different meaning to those atoms not explicitly expressed in them. An atom not in a stable model is interpreted as false in the framework of the stable model semantics whereas an atom not in an answer set is interpreted as *unknown*. This is certainly an important observation. However we need to be clear that the difference between false and unknown is only true of predicates to which strong negation is applied. For all other predicates, we should continue to use the strategy of representing negative information implicitly; otherwise we may have an undesired result as illustrated by the following example.

Example 7.6.2 Let $\mathbb{P}$ consist of the following rules:

$$\begin{aligned} even(0) &:\perp \quad . \\ even(s(s(X))) &:\perp \quad even(X). \end{aligned}$$

Where the predicate even(X) means that X is an even number, and $s(X)$ denotes the successor function of Peano arithmetic. Both the answer set and quasi-answer set of $\mathbb{P}$ is

$$\{even(0), even(s(s(0))), even(s(s(s(s(0))))), \cdots, \}$$

Since strong negation is not applied to the predicate *even*, we use non-monotonic negation with it by default. Thus the answer to the query $even(s(0))$ is *false* as intended. Without using this default convention, we would have to conclude that the answer to the query $even(s(0))$ was *unknown*, contrary to the intended meaning of *even*.

In [49], it was proposed to avoid this problem by adding

$$\sim even(X) :\perp \textit{ not } even(X).$$

to the original program instead of using our default strategy. However, this seems unnecessary. Moreover it may result in overuse of strong negation. In contrast, our solution is simpler, more straightforward, and thus preferable. □

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this thesis, we have studied negation in logic and deductive databases. Among other things, two kinds of negation are discussed in detail: strong negation and nonmonotonic negation. We have built a first-order logic system $\mathbf{CF}'$ with strong negation, and proposed a novel model of nonmonotonic negation, called quasi-stable semantics.

8.1.1 Strong Negation

Motivated by Barwise and Etchemendy's work on infon logic [10], we were led to strong negation and argued that negation used in situation theory is in fact strong negation rather than intuitionistic one. But the usual logics of strong negation, that is constructive logics, have intuitionistic quantification which has a too strong dynamic satisfaction condition on universal quantifiers. We have argued the condition is not appropriate from the situation-theoretic viewpoint. Based on these arguments, we have built a first-order logic system with strong negation and bounded static quantifiers, called $\mathbf{CF}'$, that owes much to Thomason's logic $\mathbf{CF}$ [98] but allows for expanding domains.

The logic system $\mathbf{CF}'$ is intended to be used as infon logic, the underlying logic for situation theory. We admit that $\mathbf{CF}'$ needs to be extended in different ways in order to be a fully-fledged infon logic.

In addition to the foundational role for situation theory, $\mathbf{CF}'$ may have potential applications in database theory. The utility of strong negation in the community of logic programming and deductive databases is to express explicit monotonic negative information. Although logic involved in extended logic programs is only as a fragment of constructive logics without implicational operator, our logic system $\mathbf{CF}'$ does provide a general logical framework for further extensions. It may also be used as logical basis for the study of deductive databases in a more general context where more than one database may be involved at the same time.

8.1.2 Nonmonotonic Negation

The introduction of strong negation into logic programming and deductive databases is to complement but not to replace a more common kind of negation, that is nonmonotonic negation. A whole spectrum of semantic theories for logic programs with nonmonotonic negation have been proposed, ranging from those that may infer very little information from a logic program ("sceptical") to those that infer a great deal ("credulous"). In this thesis, we have reviewed and analysed just a few of various existing semantic theories, including the Fitting semantics, the well-founded semantics, the stable model semantics.

The analysis has shown that these semantics are not fully satisfying. Nevertheless, they do provide us with profound insight towards understanding of nonmonotonic negation. It is based on these semantics and the analysis of the essential characteristic of nonmonotonic negation that we have come up with the quasi-stable semantics. An important observation is that, given the nonmonotonicity of negation *not*, a model of negation *not* cannot be appropriate unless it has a mechanism to allow the retraction of tentatively assumed negative information in the light of newly discovered information. In other words, a non-monotonic negation such as is required for logic programs (at least in a database context) should be computed by a non-monotonic, revision process. Such a process has been introduced through a mechanism of consistency-recovery in the quasi-stable semantics. As a result, the quasi-stable semantics has avoided the conceptual flaw suffered by the stable model semantics. In the quasi-stable semantics, the existence problem of the stable model semantics is genuinely solved rather transformed into a different form as it is in the three-valued version of stable semantics. We have also shown by example that the quasi-stable semantics does not give rise to anomalies as the stable model semantics does. We have proved that the quasi-stable semantics has many desirable features similar to that of the stable model semantics. It is our belief that the quasi-stable semantics provides us with an adequate model of nonmonotonic negation, and thus enables us to assign an appropriate meaning to a logic program.

8.2 Further Work

8.2.1 Possible Extensions of $\mathbf{CF}'$

Although $\mathbf{CF}'$ is a full first-order logic system, it is only fragmentary from the situation theoretical viewpoint. For one thing, the components in a basic formula $R(a_1, a_2, ..., a_n)$, or using the notation of infon logic, $\ll R, a_1, a_2, ..., a_n; i \gg$ are still individuals whereas infon logic allows them to be any objects.

$\mathbf{CF}'$ can be extended in many ways. A natural extension is to replace basic formulas $R(a_1, a_2, ..., a_n)$ of $\mathbf{CF}'$ with basic infons $\ll R, a_1, a_2, ..., a_n; i \gg$, emphasising that components $a_1, a_2, ..., a_n$ in basic infons can be any objects not just individuals. Such structures lend themselves to the treatment of complex objects.

Another possible extension is to incorporate an operator into $\mathbf{CF}'$ in order to express non-persistence.[1] What is true in one situation is still true in a larger one. However what is undetermined in a situation may become true or false when more information is available. It is then natural to introduce an operator such as 'definitely' (see [75]) or, more directly, an 'undetermined' operator U. Using this operator U, the indeterminacy of both the assertion and the (strong) negation of an infon σ can be expressed by means of $U\sigma$ and $U \sim \sigma$ respectively. If an agent, querying a situation s for a decision whether σ, fails to establish both σ and $\sim \sigma$, (s)he can then thereby establish $U\sigma$. In a larger situation, however, what is originally absent in a smaller situation may become available, thus the same agent may verify σ so that $U\sigma$ is rejected. So, $U\sigma$ is not persistent. Similarly, if a query to a situation s fails to refute σ, then it rejects the claim that σ is refuted by s and thereby establishes $U \sim \sigma$. For the same reason, $U \sim \sigma$ is not persistent either. The distinction between strong negation and U is similar to Barwise and Etchemendy's distinction between negation and denial(see Barwise and Etchemendy [9]). However, our approach is different from Barwise and Etchemendy's. Among other things, the inclusion of U in our logic will lead us into nonmonotonic logic whereas Barwise and Etchemendy claim that "Closing the class of propositions under conjunction, disjunction, and denial would result in a notion of proposition whose logic is entirely classical."(see p. 169 of [9]). Full details of such an extension remain to be done.

8.2.2 Issues Relevant to the Quasi-stable Semantics

8.2.2.1 Non-WFE-based Quasi-stable Semantics

In the current formulation of the Quasi-stable semantics, the computation of a quasi-stable extension begins with the WFE, further extended using hypothetical reason-

[1]Readers are invited to refer to Veltman's paper *Defaults in Update Semantics* [106]. There he introduces operators like 'presumably' to deal with non-persistence within the framework of update semantics.

ing and operator $F_{P,L}$. As a result, the WFE is automatically contained in every quasi-stable extension. We admit that the use of the WFE is only a shortcut and we are liable to be accused of cheating, though we insist that the shortcut is not unreasonable given that it has been universally accepted that any extension of a logic program should contain at least the WFE. Alternatively, we might have not started from the WFE. We would have just used hypothetical reasoning and the operator $F_{P,L}$, and then shown that the WFE is indeed included in each quasi-stable extension. This approach would be conceptually more economical and thus may be preferred.

8.2.2.2 Non-ground Quasi-stable Semantics

The study of the quasi-stable semantics in this thesis is restricted to ground logic programs. A non-ground logic program is first instantiated relative to its Herbrand universe. Stable models, well-founded models, and quasi-stable extensions are represented as sets of ground atoms. In [53] the stable and well-founded semantics of logic programs, and the answer set semantics of extended logic programs are generalised based on non-ground interpretations; that is sets of atoms rather than sets of ground atoms are used to represent stable models, well-founded models and answer sets, resulting in non-ground semantic theories for logic programs and extended logic programs. A set of atoms usually provides a more compact representation of its ground counterpart. Consequently, the non-ground stable, well-founded semantics, and answer set semantics are more efficient than the corresponding ground versions. The key technical notion used is that of an "anticover" of a set of substitutions. Informally, an anticover of a set X of substitutions is described as "a set of substitutions, all of which are incompatible (i.e. they share no common instance) with the substitutions in X, and such that each substitution that is incompatible with all members of X is an instance of some substitution in the anticover." [53]. It would be interesting to know whether the notion of anticover can also be extended to non-ground quasi-stable extensions so as to to give a non-ground quasi-stable semantic theory.

8.2.2.3 Implementation of the Quasi-stable Semantics

There have been various implementations of deductive databases. See [87] for a survey. Implementations mentioned in [87] are mainly under the stratified, locally stratified or well-founded semantics. Recently, there have been different efforts on effective and efficient implementations for computation of stable models for logic programs.

Based on mixed integer programming, three different algorithms for computing stable models of logic programs have been proposed and implemented in a prototype compiler in [12]. It is reported in the same paper that these algorithms and implementations have also been extended to handle logic programs with both non-monotonic negation and strong negation[2]. One significant point is that deduction is performed at compile-time rather than run-time. As a result, run-time query execution can be reduced to the traditional relational database operations and thus can be performed relatively more efficiently than otherwise.

In [78] a direct and efficient implementation of the well-founded and stable model semantics has been proposed for range-restricted function-free logic programs. The computation of stable models for ground logic programs makes use of bottom-up backtracking search and a powerful pruning method based on a well-founded type approximation for stable models. The implementation also contains an algorithm for instantiating a logic program to its ground version. The instantiation algorithm only produces a subset of ground instances of the program without losing any stable models. The implementation can compute all stable models, decide whether a logic

[2]In [12], the term *classical negation* is used instead of strong negation.

program has a stable model, and decide whether a given formula is satisfied in some or all of the stable models of a program.

The above two implementations are for ground logic programs. A logic program has to be instantiated relative to its Herbrand universe before the computation of stable models for the program. In [24], a different implementation of the well-founded and stable model semantics has been proposed for non-ground logic programs. The computation of stable models, based on the so-called *assume-and-reduce* algorithm, is still relative to ground logic programs. But the computation of well-founded semantics is for non-ground logic programs. A prototype system called SLG [23] has been developed for goal-oriented query evaluation under the well-founded semantics. Given a query, SLG produces a residual program containing answers for all subgoals which are relevant to the query. In [24] SLG is extended to accommodate the stable model semantics. For a given query, SLG first produces a residual program of the query and then computes stable models relative to the residual program rather than original programs. In this way, SLG provides integrated query evaluation under both the well-founded semantics and stable model semantics. SLG itself, however, is elective about which semantics to use, that is the user may choose either semantics for finding answers to the query.

Given the close relation of the quasi-stable semantics with the stable model semantics, we believe that these implementations with appropriate modification could also be adapted to the quasi-stable semantics. Given the problems of the stable model semantics, it is our opinion that a further effort is worthwhile to implement the quasi-stable semantics.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading, MA, 1995.

[2] S. Akama. Constructive predicate logic with strong negation and model theory. *Notre Dame Journal of Formal Logic*, 29:18–27, 1988.

[3] J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. Strong and explicit negation in non-monotonic reasoning and logic programming. In J. J. Alferes, L. M. Pereira, and E. Orlowska, editors, *Logics in artificial intelligence*, Lecture Notes in Artificial Intelligence 1126, pages 143–163. Springer, 1996.

[4] A. Almukdad and D. Nelson. Constructible falsity and inexact predicates. *Journal of Symbolic Logic*, 49:231–233, 1984.

[5] K. R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan-Kaufmann, San Mateo, Calif., 1988.

[6] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of the Association for Computing Machinery*, 29(3):841–862, 1982.

[7] C. Baral, J. Lobo, and J. Minker. Generalized well-founded semantics for logic programs. In M. E. Stickel, editor, *The 10th International Conference*

on Automated Deduction, Lecture Notes in Artificial Intelligence 449, pages 102–116. Springer-Verlag, 1990.

[8] J. Barwise. *The Situation in Logic.* Number 17 in CSLI Lecture Notes. CSLI Publications, Stanford, 1989.

[9] J. Barwise and J. Etchemendy. *The Liar: An Essay on Truth and Circular Propositions.* Oxford University Press, New York, 1987.

[10] J. Barwise and J. Etchemendy. Information, infons, and inference. In K. Mukai R. Cooper and J. Perry, editors, *Situation Theory and Its Applications,* volume 1, CSLI Lecture Notes 22, pages 33–78. CSLI Publications, Stanford, 1990.

[11] J. Barwise and J. Perry. *Situations and Attitudes.* MIT Press, Cambridge, MA, 1983.

[12] C. Bell, A. Nerode, R. T. Ng, and V. S. Subrahmanian. Mixed integer programming methods for computing nonmonotonic deductive databases. *Journal of the Association for Computing Machinery*, 41(6):1178–1215, 1994.

[13] J. L. Bell and M Machover. *A Course in Mathematical Logic.* North-Holland, Amsterdam, 1977.

[14] E. Bencivenga. Free logics. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic,* Vol III: Alternatives in Classical Logic, pages 373–426. D. Reidel, Dordrecht, 1986.

[15] N. Bidoit. Negation in rule-based database languages. *Theoretical Computer Science*, 78:3–83, 1991.

[16] N. Bidoit and C. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78:85–112, 1991.

[17] H. Blair and Subrahmanian V. S. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.

[18] S. Blamey. Partial logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic,* Vol III: Alternatives in Classical Logic, pages 1–70. D. Reidel, Dordrecht, 1986.

[19] A. Bondarenko, P. M. Dung, R. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.

[20] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning*, pages 86–97. Morgan-Kaufmann, 1998.

[21] A. K. Chandra and D. Harel. Horn clause queries and generalizations. *Journal, Logic Programming*, 2(1):1–15, 1985.

[22] J. Chen and S. Kundu. The strong well-founded semantics for logic programs. In Z. W. Ras and M Zemankova, editors, *The 6th International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Artificial Intelligence 542, pages 490–499. Springer-Verlag, 1991.

[23] W. Chen, T. Swift, and D. S. Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–199, 1995.

[24] W. Chen and D. S. Warren. Computation of stable models and its integration with logical query processing. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):742–757, 1996.

[25] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic & Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[26] E. F. Codd. A relational model of data for large shared data banks. *Communications of the Association for Computing Machinery*, 13(6):377–387, 1970.

[27] E. F. Codd. A database sublanguage founded on the relational calculus. In *ACM SIGFIDET Workshop on Data Description, Access, and Control*, pages 35–61, San Diego, California, 1971.

[28] E. F. Codd. Further normalization of the data base relational model. In R Rustin, editor, *Courant Computer Science Symposium 6: Data Base Systems*, pages 33–64. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[29] E. F. Codd. Relational completeness of database sublanguages. In R Rustin, editor, *Courant Computer Science Symposium 6: Data Base Systems*, pages 65–98. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[30] E. F. Codd. Recent investigations in relational data base systems. In J. L. Rosenfeld, editor, *Information Processing 74*, pages 1017–1021. North-Holland, Amsterdam, 1974.

[31] A. G. Cohn. Order-sorted logic. In S. C. Shapiro, editor-in-chief, *Encyclopedia of Artificial Intelligence, Volume 1,* 2rd ed., pages 864–866. John Wiley & Sons, Inc., New York, 1992.

[32] K. Devlin. *Logic and Information.* Cambridge University Press, Cambridge, 1991.

[33] J. Dix. A classification theory of semantics of normal logic programs: I. strong properties. *Fundamenta Informaticae*, 12(3):227–255, 1995.

[34] J. Dix. A classification theory of semantics of normal logic programs: II. weak properties. *Fundamenta Informaticae*, 12(3):257–288, 1995.

[35] J. Dix and M. Müller. The stable semantics and its variants: a comparison of recent approaches. In L. Dreschler-Fischer and B. Nebel, editors, *Proceedings of the 18th German Annual Conference on Artificial Intelligence (KI '94)*, Lecture Notes in Artificial Intelligence 861, pages 82–93. Springer-Verlag, 1994.

[36] M. Dummett. *Elements of Intuitionism.* Oxford Logic Guides. Clarendon Press, Oxford, 1977.

[37] P. M. Dung. An argumentation-theoretic foundation for logic programming. *Journal of Logic Programming*, pages 151–177, 1995.

[38] T. Fernando. On the logic of situation theory. In K. Mukai R. Cooper and J. Perry, editors, *Situation Theory and Its Applications,* volume 1, CSLI Lecture Notes 22, pages 97–116. CSLI Publications, Stanford, 1990.

[39] F. B. Fitch. *Symbolic Logic.* Ronald Press Co., New York, 1952.

[40] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 4:295–312, 1985.

[41] G. Frege. Begriffsschrift (Chapter I). In P. Geach and M. Black, editors, *Translations from the Philosophical Writings of Gottlob Frege,* 3rd ed., pages 1–20. Blackwell, Oxford, 1980.

[42] G. Frege. On sense and meaning. In P. Geach and M. Black, editors, *Translations from the Philosophical Writings of Gottlob Frege,* 3rd ed., pages 56–78. Blackwell, Oxford, 1980.

[43] D. M. Gabbay. What is negation in a system? In F. R. Drake and J. K. Truss, editors, *Logic Colloquium '86*, pages 95–112. North-Holland, Amsterdam, 1988.

[44] H. Gallaire and J. Minker (ed.). *Logic & Data Bases.* Plenum Press, New York, 1978.

[45] J. W. Garson. Quantification in modal logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic,* Vol. II: Extensions of Classical Logic, pages 249–307. D. Reidel, Dordrecht, 1985.

[46] M. Gelfond. On stratified autoepistemic theories. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 207–211. Morgan-Kaufmann, 1987.

[47] M. Gelfond and L. Lifschitz. The stable model semantics for logic programs. In R. A. Kowalski and K. A. Bowen, editors, *Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[48] M. Gelfond and L. Lifschitz. Logic programs with classical negation. In D. Warren and P. Szeredi, editors, *Proceedings of the 7th International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.

[49] M. Gelfond and L. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–385, 1991.

[50] G. Gentzen. Investigation into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.

[51] K. Gödel. The completeness of the axioms of the functional calculus of logic. In J. van Heijenoort, editor, *From Frege to Gödel: a source book in mathematical logic, 1879-1931*, pages 582–591. Harvard University Press, Cambridge, 1967.

[52] K. Gödel. On formally undecidable propositions of Principia mathematica and related systems I. In J. van Heijenoort, editor, *From Frege to Gödel: a source book in mathematical logic, 1879-1931*, pages 596–616. Harvard University Press, Cambridge, 1967.

[53] G. Gottlob, S. Marcus, A. Nerode, and V. S. Subrahmanian. Non-ground stable and well-founded semantics. Available from LPNMR archives at http://www.cs.engr.uky.edu/ lpnmr/papers.html.

[54] Y. Gurevich. Intuitionistic logic with strong negation. *Studia Logica*, 36:49–59, 1977.

[55] C. E. Hewitt. Planner: a language for proving theorems in robots. In *First International Joint Conference on Artificial Intelligence*, pages 295–301, Washington, 1969.

[56] A. Heyting. *Intuitionism, An Introduction.* North-Holland, Amsterdam, 1956.

[57] Y. Hu and L. Y. Yuan. Extended well-founded model semantics for general logic programs. In K. Furukawa, editor, *The 8th International Conference on Logic Programming*, pages 412–425. MIT Press, 1991.

[58] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[59] T. Kakas and P. Mancarella. Preferred extensions are partial stable models. *Journal of Logic Programming*, 14:341–348, 1992.

[60] P. G. Kolaitis. The expressive power of stratified programs. *Information and Computation*, 90(1):50–66, 1991.

[61] R. Kowalski. Logic for data description. In H. Gallaire and J. Minker, editors, *Logic & Data Bases*, pages 77–103. Plenum Press, New York, 1978.

[62] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4:289–308, 1987.

[63] H. J. Levesque. Making believers out of computers. *Artificial Intelligence*, 30:81–107, 1986.

[64] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1984.

[65] E. G. K. Lopez-Escobar. Refutability and elementary number theory. *Indagationes Mathematicae*, 34:362–374, 1972.

[66] V. W. Marek and M. Truszczynski. Autoepistemic logic. *Journal of the Association for Computing Machinery*, 38(3):588–619, 1991.

[67] V. W. Marek and M. Truszczyński. *Nonmonotonic Logic: Context-Dependent Reasoning.* Springer-Verlag, 1993.

[68] A. A. Markov. Constructive logic (in russian). *Uspekhi Matematičeskih Nauk*, 5:187–188, 1950.

[69] J. McCarthy. Mechanization of thought processes. In M. Minsky, editor, *Semantic Information Processing*, pages 403–418. MIT Press, Cambridge, Mass. 1968. Originally published in *Proceedings of the Symposium of the National Physics Laboratory*, vol. 1, pages 77-84. London, 1958.

[70] J. Minker. On indefinite databases and the closed world assumption. In *Proceedings of the 6th Conference on Automated Deduction*, Lecture Notes in Computer Science 138, pages 292–308. Springer-Verlag, Berlin, 1982.

[71] J. Minker. Perspectives in deductive databases. *Journal of Logic Programming*, 5(1):33–60, 1988.

[72] J. Minker. An overview of non-monotonic reasoning and logic programming. *Journal of Logic Programming*, 17:95–126, 1993.

[73] J Minker and C. Ruiz. Mixing a default rule with stable negation. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics*, pages 122–125. Fort Lauderdale, Florida, 1996.

[74] R. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.

[75] P. L. Mott. Intuitionistic logic with a 'definitely' operator. Research Report 97.05, School of Computer Studies, University of Leeds, 1997.

[76] D. Nelson. Constructible falsity. *Journal of Symbolic Logic*, 14:16–26, 1949.

[77] J. M. Nicolas and H Gallaire. Data base: theory vs. interpretation. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 33–54. Plenum Press, New York, 1978.

[78] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. Available from http://www.uni-koblenz.de/ag-ki/DLP/#pubs, 1996.

[79] D. Pearce and G. Wagner. Reasoning with negative information I–strong negation in logic programs. *Acta Philosophica Fennica*, 49:430–453, 1990.

[80] D. Pearce and G. Wagner. Logic programming with strong negation. In P. Schroeder-Heister, editor, *Proceedings of Workshop on Extensions of Logic Programming*, pages 311–326. Springer, Berlin, 1991.

[81] L. M. Pereira, J. J. Alferes, and J. N. Aparicio. Adding closed world assumptions to well-founded semantics. *Theoretical Computer Science*, 122:49–68, 1994.

[82] D. Poole. A logic framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[83] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.

[84] T. C. Przymusinski. Extended stable semantics for normal and disjunctive logic programs. In *Proceedings of the 7th International Conference on Logic Programming*. MIT Press, 1990.

[85] T. C. Przymusinski. Well-founded semantics coincides with three-valued stable semantics. *Fundamenta Informaticae*, XIII, 1990.

[86] W. V. Quine. *From a Logical Point of View : 9 Logico-philosophical Essays.* Harvard University Press, Cambridge, Mass., 1953.

[87] R. Ramakrishnan and J. D. Ullman. A survey of deductive database systems. *Journal of Logic Programming*, 23(1):125–149, 1995.

[88] R. Ramakrishnan (ed.). *Applications of Logic Databases.* Kluwer Academic Publishers, 1994.

[89] H. Rasiowa. $\mathcal{N}$-lattices and constructive logic with strong negation. *Fundamenta Mathematicae*, 46:61–80, 1958.

[90] R. Reiter. On closed world databases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, New York, 1978.

[91] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[92] R. Reiter. Towards a logical reconstruction of relational database theory. In J. Brodie, M. L. Mylopoulos and J. W. Schmidt, editors, *On Conceptual Modelling*, pages 191–238. Springer-Verlag, New York, 1984.

[93] K. A. Ross and R. W. Topor. Inferring negative information from disjunctive databases. *Journal Of Automated Reasoning*, 4:397–424, 1988.

[94] R. Routley. Semantical analyses of propositional systems of Fitch and Nelson. *Studia Logica*, 33:283–298, 1974.

[95] D. Saccà and C. Zaniolo. Stable models and non-determinism in logic program with negation. In *Proceedings of the 9th ACM PODS Symposium*, pages 205–218. ACM Press, New York, 1990.

[96] D. Saccà and C. Zaniolo. Partial models and three-valued models in logic programs with negation. In A. Nerode, W. Marek, and V. S. Subramanian, editors, *Proceedings of the 1st International Workshop on Logic Programming and Nonmonotonic Reasoning*, pages 87–104. MIT Press, 1991.

[97] J. S. Schlipf. Formalizing a logic for logic programming. *Annals of Mathematics and Artificial Intelligence*, 5:279–302, 1992.

[98] R. H. Thomason. A semantical analysis of constructible falsity. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 15:247–257, 1969.

[99] A. S. Troelstra. *Choice Sequences: A Chapter of Intuitionistic Mathematics.* Oxford Logic Guides. Clarendon Press, Oxford, 1977.

[100] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics*, volume 1. North-Holland, Amsterdam, 1988.

[101] J. D. Ullman. Assigning an appropriate meaning to database logic with negation. In *Computers as Our Better Partners*, pages 216–225. World Scientific Press, 1994.

[102] D. van Dalen. Intuitionistic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic,* Vol III: Alternatives in Classical Logic, pages 225–339. D. Reidel, Dordrecht, 1986.

[103] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *JACM*, 23(4):733–742, 1976.

[104] A. van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47:185–221, 1993.

[105] A. van Gelder, K. A. Ross, and Schlipf J. S. The well-founded semantics for general logic programs. *Journal of the Association for Computing Machinery*, 38(3):620–650, 1991.

[106] F. Veltman. Defaults in update semantics. *Journal of Philosophical Logic*, 25:221–261, 1996.

[107] N. N. Vorob'ev. Constructive propositional calculus with strong negation (in Russian). *Doklady Akademii Nauk SSSR*, 85:465–468, 1952.

[108] G. Wagner. A database needs two kinds of negation. In B. Thalheim, Demetrovics J., and H.-D. Gerhardt, editors, *The 3rd Symposium on Mathematical Fundamentals of Database and Knowledge Bases Systems MFDBS-91*, pages 357–371. Springer, 1991.

[109] G. Wagner. Logic programming with strong negation and inexact predicates. *Journal of Logic and Computation*, 1:835–859, 1991.

[110] G. Wagner. *Vivid Logic: Knowledge-Based Reasoning with Two Kinds of Negation*. Springer-Verlag, Berlin, 1994.

[111] X. G. Wang and P. Mott. Quasi-stable semantics for logic programs. Research Report 98.14, School of Computer Studies, University of Leeds, available from http://csis1.leeds.ac.uk/pmottpub.htm, 1998.

[112] X. G. Wang and P. Mott. A variant of Thomason's first-order logic CF based on situations. *Notre Dame Journal of Formal Logic*, 39(1):74–93, 1998.

[113] X. G. Wang and P. Mott. A critical note on stable model semantics. Submitted for publication, available from http://csis1.leeds.ac.uk/pmottpub.htm, 1999.

[114] H. Wansing. *Logic of Information Structures*. Number 681 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 1993.