Representing Indefinite Information using Abductive Logic Programs

Thesis submitted for the degree of Ph.D.

to Imperial College, University of London

by

Christopher William Preist

BIBL.
LONDIN.
UNIV.

This thesis is dedicated to my parents, for all their support and encouragement through the years.

Thanks to Bob Kowalski and Fariba Sadri for their guidance, to Francesca Toni for her comments, and to Lin Jones for all her work in producing the final version.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

ABSTRACT

In this thesis, we investigate how to represent indefinite information, in particular disjunctive information, within logic programs.

We identify different forms of disjunction which can be used in knowledge representation, and present an abductive approach to representing this knowledge. We argue that this approach to knowledge representation is preferable to using disjunctive logic programs, because it is more general and more straightforward.

We prove that both the possible model semantics and Indefinite Closed World Assumption semantics of disjunctive logic programming can be represented within this approach. We also discuss its relationship with the disjunctive stable model semantics.

We argue that, in many cases, it is possible to simplify the representation of disjunctive knowledge still further, as most disjunctive knowledge is exclusive. We define a new class of disjunctive logic programs, *exclusive disjunctive logic programs*, and show that their semantics is straightforward in comparison with that of disjunctive logic programs, and can also be represented using the abductive approach. Furthermore, we show that other forms of disjunctive knowledge can be transformed in such a way that only exclusive disjunction is used to represent them. We then show that exclusive disjunctive logic programs can be transformed into normal logic programs with integrity constraints, which are interpreted using the stable model semantics. Hence, we show that much disjunctive knowledge can be represented within normal logic programs with integrity constraints.

Finally, we present a critical survey of semantics of disjunctive logic programs and their ability to represent different forms of disjunctive information. We compare disjunctive logic programs with abductive and normal logic programs, and present in which circumstances each is the best approach to adopt.

TABLE OF CONTENTS

Title pages

Abstract..... *3*

Contents..... *4*

Chapter 1 ..... *5*

Introduction and Overview

Chapter 2 ..... *23*

The Tools of Logic Programming

Chapter 3..... *45*

Using an Abductive Framework to Represent Indefinite Information

Chapter 4..... *71*

The Relationship between the Framework and Existing Semantics of Disjunctive Logic Programs

Chapter 5..... *95*

Exclusive Disjuctive Logic Programs

Chapter 6..... *117*

Using Normal Logic Programs to Represent Indefinite Information

Chapter 7..... *127*

A Critical Survey of Semantics of Disjunctive Logic Programs

Chapter 8..... *157*

Conclusions and Future Directions

Appendix..... *165*

References..... *169*

Chapter 1 - Introduction and Overview

Work on logic programming originally focused on programs consisting of Horn clauses, and later Horn clauses were extended to incorporate a form of default negation, negation-as-failure. However, many researchers in the area have argued that this is an inadequate knowledge representation language for certain tasks, tasks which require the representation of indefinite or uncertain information. In this thesis, we will look at three different approaches to representing indefinite information by extending the expressivity of logic programs; disjunctive logic programming, abductive logic programming, and even loop constructs in normal logic programs.

1.1 Indefinite information

An *indefinite statement* is one which does not allow the conclusion of a single fact from its premises, but rather allows various alternative conclusions. There are different kinds of indefinite statements, as the following examples illustrate.

- A region on a map can be coloured red, green or blue.

- A bag contains 3 numbered balls. If we pull a ball from the bag and don't replace it, it will have the number 1, 2 or 3 on it.

- A dirty sample injected into a gas chromatograph will result in a blockage in the inlet nozzle, the dispersion tube or the detector.

- An employee of NASA will be qualified either as a rocket scientist or as an astronaut.

- The symptom of facial spots can be caused either by acne or by measles.

- An electrical device contains a bulb. A short circuit in the device may result in the bulb blowing.

All these statements have in common the fact that it is not possible to conclude something with certainty from them. We cannot be sure of the colour of the map's region. We cannot be sure that the bulb will blow if a short circuit occurs. However, despite this commonality, these statements cover quite different situations, and any logic programming representation of them must be able to reflect these differences.

Firstly, most of these statements express a collection of different possibilities; a disjunction. However, the last example of the bulb does not. It simply states that a certain event may take place, but we can't be certain that it will. An adequate approach to representing indefinite information should be able to handle both kinds of statement - ones which lists different possibilities, and ones which simply say that something may or may not be the case. Furthermore, all the examples which explicitly express disjunction could be weakened to express that one or several of the disjuncts *may* hold. Thus, an adequate approach should be able to make it possible to move smoothly from one representation (explicit disjunction) to the other (possible outcome), or vice versa.

When a statement gives a collection of different possibilities, they may or may not be mutually exclusive alternatives. A given region of a map can be at most one colour; it cannot be both red and blue. An employee of NASA, on the other hand, may be trained both as a rocket scientist and as an astronaut. Again, an adequate approach to knowledge representation should be able to handle both such situations.

In some cases, selecting one of the alternative possibilities will have no effect on other uses of the statement. If one region of a map is chosen to be green, it does not prevent another region on the map from being green. However, in other cases, selecting one alternative will 'use up' that alternative. If ball number 1 is chosen from the bag on the first pick, then the next time such a choice is made it cannot be chosen again. The second pick can choose only ball 2 or ball 3. An adequate approach to representation of indefinite knowledge should be able to handle both cases.

Finally, some of the statements have the notion of causality behind them. The fact that spots can be caused by either measles or acne can be treated as indefinite information - 'If someone has spots, then they have measles or acne'. However, it can also be treated as definite information - 'Measles causes spots, and acne causes spots'. A good knowledge representation language will allow such causality to be explicitly expressed where it exists.

We will now look at the different approaches to representing indefinite information which have been explored by various researchers.

1.2 Disjunctive Logic Programs

Minker [Min 82] first proposed disjunctive logic programming as an extension to logic programming, to increase its expressivity and allow indefinite information to be represented. In a *disjunctive logic program*, the language of logic programming is extended to allow the heads of clauses to contain a disjunction of atoms, rather than a single atom. Hence, it combines the expressivity of first-order logic with the advantages of the default negation of logic programming. It allows indefinite information about a situation to be represented explicitly. For example, we can represent the statement above that a map region can be either red, green or blue with the disjunctive clause:

$$colour(X, red) \vee colour(X, green) \vee colour(X, blue) \leftarrow region(X)$$

A disjunctive logic program is able to express such indefinite information, and draw conclusions from it.

Minker [Min 82] initially focused on positive disjunctive logic programs, and what positive and negative literals can be concluded from them. He proposed that positive literals be considered true if they can be concluded if the program is treated as a classical theory - a proposal which is straightforward, and has achieved widespread acceptance. He also considered when to conclude a negative literal from a positive disjunctive logic program. He did this by extending the Closed World Assumption of Reiter [Rei 78] to positive disjunctive logic programs.

The *Closed World Assumption* is applied to positive databases not containing disjunctions, and allows a negative literal $\neg a$ to be concluded if the positive literal a is not a logical consequence of the program. Minker [Min 82] considered whether a similar approach could be applied to positive disjunctive logic programs and determined that it could lead to inconsistencies if the negation is treated as classical. For example, the program $\{a \vee b\}$ allows both $\neg a$ and $\neg b$ to be concluded, giving the inconsistent theory $\{a \vee b, \neg a, \neg b\}$. In his view, this is unacceptable and the Closed World Assumption can not be applied to disjunctive logic programs. However, if negation is read as 'not provable', then such a theory can be considered consistent.

Based on this observation, he defined the *Generalised Closed World Assumption* (GCWA) [Min82] on positive disjunctive logic programs. It concludes $\neg a$ only if a is false in every minimal model of a program. The GCWA would not be able to conclude either $\neg a$ or $\neg b$ from the above program, but could conclude $\neg c$, etc. for any other atom in the Herbrand Base.

Work has been carried out on extensions of the GCWA, most notably on the *Extended Generalised Closed World Assumption* [YH 85] which concludes disjunctions of negative literals if the disjunction is true in all minimal models of the program. In the above example, the negative disjunction $\neg a \vee \neg b$ would be concluded.

The Generalised Closed World Assumption and its extensions interpret disjunction in a 'strong' sense; the clause $a \vee b$ cannot be used to conclude both a and b simultaneously. This is often referred to as an 'exclusive' interpretation in the literature. However, this terminology is mistaken. If disjunction is treated exclusively, then the theory $\{a \vee b, a, b\}$ is inconsistent. However, under the Generalised Closed World Assumption and its extensions, this is consistent.

The GCWA is concerned with minimal models of the theory, allowing more than one atom within a disjunction to be true if another part of the theory requires this. Exclusive disjunction places the stronger restriction that only one of the atoms within a disjunction can be true.

An alternative method of interpreting disjunction in a weaker form was developed by [RT 88] and [RLM 89]. The *Weak Generalised Closed World Assumption* (WGCWA) deems the negation of an atom to hold if the atom cannot be concluded in the Horn clause theory formed by replacing each disjunctive clause, $a_1 \vee .. \vee a_n \leftarrow body$ with the n Horn clauses, $\{a_1 \leftarrow body, ..., a_n \leftarrow body\}$.

The effect of this is to prevent the conclusion of certain negative literals, even though their positive counterpart is indeed false in all minimal models. To see this, consider the example program $\{a \vee b, a\}$. This has the single minimal model, $\{a\}$, and hence, under the GCWA, $\neg b$ can be assumed. However, under the Weak Generalised Closed World Assumption, the disjunction $a \vee b$ prevents the conclusion of $\neg b$. The WGCWA is referred to in the literature as treating disjunction as 'inclusive'. By this, they mean that the WGCWA does not consider only minimal models. Disjunctions act as a weaker upper bound on the models to be considered; the WGCWA considers models which minimise atoms not appearing in disjunctions, and allow atoms appearing in disjunctions to range freely. Hence, 'inclusive' is a misnomer; as we have seen, the GCWA in some cases interprets disjunction as inclusive. However, as there is no better terminology available, we will say that a semantics treats a disjunction as 'inclusive' (in quotations) if it considers models which allow atoms appearing in the disjunction to range freely in this way.

Most work on the semantics of disjunctive logic programs extends an existing semantics of normal logic programs to define negation in disjunctive logic programs in a way which, when restricted to positive disjunctive logic programs, is equivalent to one of these variants of the Closed World Assumption. For example, Minker and Rajasekar [MR 90] extend the semantics of stratified logic programs defined in [ABW 88] to give a semantics for disjunctive logic programs compatible with the GCWA.

Przymusinski [Prz 91] extends the stable model semantics [GL 88] to give a semantics compatible with the GCWA, and also defines a three valued version. However, how exactly to extend such a semantics is not always clear.

[Ross 89], [Prz 95] and [BD 95] each propose different extensions of the well-founded semantics [VGRS 88] to handle disjunctive logic programs, all of which reduce to the Generalised Closed World Assumption in the positive case.

Other semantics which are not based on such extensions have also been proposed. These include the Generalised Disjunctive Well Founded Semantics [BLM 90a], [BLM 90b] the possible model semantics [Cha 89], the Indefinite Closed World Assumption [KWC 94], the causal semantics [DGM 94] and theory curbing [EGG 93].

The variety of proposals and the lack of agreement among researchers about which are best demonstrates that disjunctive logic programs are far from being clearly understood. This suggests we should be wary of adopting them as tools for knowledge representation, without considering alternative approaches which are more straightforward.

Minker gives a full list of semantics of disjunctive logic programs, and their relationship with semantics of normal logic programs in his overview paper [Min 94]. Semantics which extend the GCWA tend to be favoured over those which extend the WGCWA by many researchers. In some cases, such as the optimal well-founded semantics [Ross 89], some disjunctions in a theory are interpreted using the GCWA, while others are interpreted using the WGCWA.

As with semantics of normal logic programs, we can divide semantics of disjunctive logic programs into those which are credulous and those which are sceptical. Credulous semantics, such as the stable model semantics [GL 88], give several alternative models to a single program. Sceptical semantics, such as the well-founded semantics [VGRS 88], give a single three-valued model which may assign 'unknown' to certain atoms in the program.

Informally, we can say that credulous semantics give information about alternative possibilities, while sceptical semantics give information about what is definitely the case.

It is possible to construct a sceptical semantics from a credulous semantics, by assigning true to atoms true in all models, false to those false in all models, and unknown to the remaining atoms (see, for example, [Dix 91]). Bondarenko, Dung, Kowalski and Toni [BDKT 97] have proposed that all sceptical semantics of normal logic programs can be viewed as being generated in this way from an appropriate credulous semantics.

In this thesis, we will focus primarily on the various credulous semantics of disjunctive logic programs which have been proposed. Here, we give an intuitive overview of three of them, and how they differ.

Przymusinski [Prz 91] extended the definition of stable models [GL 88] to apply to disjunctive logic programs. This resulted in a semantics which, like the Generalised Closed World Assumption, minimises conclusions from a disjunctive clause. Hence, the simple disjunctive logic program $\{a \vee b, a \vee c\}$ will have two stable models; $\{a\}$ and $\{b, c\}$. If the additional clause $a \vee b \vee d$ is added, it will have no effect on the stable models.

The possible model semantics [Cha 89], [Sak 90], however, does not minimise in the same way. It treats disjunctive clauses as 'inclusive' in the way that the WGCWA does, and so considers all models which minimise atoms not appearing in disjunctions, and allow atoms appearing in disjunctions to range freely. The program $\{a \vee b, a \vee c\}$ has five possible models: $\{a\}$, $\{b, c\}$, $\{a, b\}$, $\{a, c\}$ and $\{a, b, c\}$. Adding the clause $a \vee b \vee d$, even though it is logically redundant classically, results in an additional five possible models, containing d, as d is no longer minimised.

There is a proven relationship between the WGCWA and the possible model semantics. Sakama [Sak 90] shows that given a disjunctive logic program without negation, P, the WGCWA will conclude a negative literal $\neg p$ iff p is false in every possible model of P. This shows that the possible model semantics is a credulous extension of the WGCWA to programs with negation.

The Indefinite Closed World Assumption [KWC 94] is a variant of the possible model semantics, but extends neither the GCWA nor the WGCWA. It is most easily understood constructively. The possible model semantics can be viewed as constructing a possible model by allowing each disjunction to contribute any number of disjuncts to the model. The Indefinite Closed World Assumption places the restriction that each disjunction can only contribute one disjunct. However, it does <u>not</u> require minimisation.

Consider $\{a \vee b, a \vee c\}$. This has four indefinite closed worlds (referred to as *I*-models): $\{a\}$, $\{b, c\}$, $\{a, b\}$ and $\{a, c\}$. To construct the model $\{a\}$, both disjunctions contribute a to the model. To construct $\{a, b\}$, $a \vee c$ contributes a, and $a \vee b$ contributes b. The resulting model is non-minimal.

However, this semantic does not treat disjunction as 'inclusive', in the way that the WGCWA and possible model semantics do. $\{a, b, c\}$ is not an *I*-model, as it can only be constructed by allowing some disjunction to contribute more than one disjunct to the model.

As we have seen, there are many different semantics for disjunctive logic programming, which interpret both negation and disjunction in a variety of different ways. The various researchers responsible for proposing these semantics often motivate their work by considering how the semantics interprets certain examples, usually abstract. No consensus has been reached on which semantics is the most appropriate for disjunctive logic programs. Indeed, proponents of disjunctive logic programming argue that it is necessary to choose from among the various proposed semantics based on the characteristics of a given problem. Hence, the choice of semantics is one of the knowledge representation decisions which must be made. Therefore, the users must understand all the semantics to make an educated choice. But once a semantics is chosen, it must be used for all disjunctions in the given application.

To see this, let us consider some of the examples above.

Say we wish to represent the statement, 'An employee of NASA is trained either as a rocket scientist or as an astronaut'. We can do this with the disjunctive clause:

$$astronaut(X) \vee rocket_engineer(X) \leftarrow works_at_nasa(X)$$

We would wish to interpret this as meaning that someone who works at NASA is trained in at least one of these, and possibly both. Hence, we would want to adopt a semantics which treats disjunctions as 'inclusive', such as the possible model semantics described above.

If we wish to represent the statement 'a region on the map is coloured red, green, or blue', with the disjunctive clause, $colour(X, red) \vee colour(X, blue) \vee colour(X, green) \leftarrow region(X)$, we would not wish to adopt the possible model semantics. Such a semantics, which treats disjunctions as 'inclusive', would allow models in which a region of the map is coloured more than one colour at the same time. We would rather treat these different possibilities as mutually exclusive. In such a situation, proponents of disjunctive logic programming adopt semantics which minimise the number of conclusions of a disjunctive clause; hence the clause will be able to conclude that a region is coloured only one of the colours in a given model. As we argued above, this is only an approximation to true exclusivity.

In both these cases, the choice of semantics was dictated by the different ways we wished to interpret disjunction, to represent the different forms of indefinite information we had. This approach to knowledge representation has several disadvantages:

- Choosing an appropriate semantics from among the plethora available can be quite a daunting task.

- The choice of semantics simultaneously makes decisions about how to handle negation, and about how to handle disjunction. It is not possible to separate out these two decisions in disjunctive logic programming.

- The semantics is applied to the entire program. Hence it is not possible to treat different disjunctions in different ways, making it hard to represent a variety of different kinds of indefinite information in the same program.

- The choice of semantics reflects characteristics of the problem, such as whether disjunction should be considered exclusive or 'inclusive'. It would be desirable to have these characteristics made explicit in the knowledge representation language, where they are open to inspection and critique by others.

- Different semantics require different inference procedures. The disjunctive well founded semantics can be found using a bottom up procedure[BD 95], the stable model semantics can be found using the SLP procedure [FL 93] and the general disjunctive well founded semantics can be found using the SLIS procedure [BLM 90b]. Other semantics, such as the static semantics [Prz 95], currently have no associated procedure. Hence a knowledge engineer needs to have a variety of different procedures available.

1.3 Abductive Logic Programs

In modern times, the philosopher Peirce [Peirce 58] first introduced the concept of *abduction*. He characterised it as a form of synthetic reasoning where from a rule, $a \leftarrow b$, the body of the rule, b, is inferred if the head of the rule, a, is observed to hold. The body of the rule can be considered as an explanation of the head which has been observed. However, the body cannot be concluded with certainty; there may be other possible explanations. For that reason, Peirce described abduction as the 'probational adoption of a hypothesis'.

Abductive logic programs extend normal logic programs to allow reasoning of this form. They designate certain atoms as *abducible* and allow these to be assumed as hypotheses when required. In addition, they allow the imposition of *integrity constraints*.

Integrity constraints are used in databases to describe properties the data must satisfy. For example, a payroll database may have an integrity constraint which ensures that every employee in the database has an employee number. It is quite natural for such integrity constraints to contain negation and disjunction, even though the database itself is definite. In abductive logic programming the program, together with a set of hypotheses, must satisfy the integrity constraints. The program itself must consist of Horn clauses, extended to allow default negation to appear in their bodies. This is for reasons of efficiency, and also because default negation allows information to be expressed succinctly. The integrity constraints, however, do not need to be constrained in this way. They are used as a filter on the program - a 'test' to the program's 'generate'. For this reason, they often use classical negation rather than default negation, and can be arbitrary first order logic. Hence, they can contain disjunction and existential and universal qualification.

Abductive logic programming, by extending normal logic programming to allow hypotheses to be assumed, is able to represent certain forms of indefinite information. For example, the statement 'a short circuit in the device may cause the bulb to blow' can be represented as the clause;

$$blown_bulb \leftarrow short_circuit, \alpha^*$$

If the abducible, α^*, is assumed, then the rule can be used to derive *blown_bulb* from *short_circuit*. If α^* is not assumed, this derivation cannot take place. Hence, the statement is faithfully represented. Poole [Poole 89a] first used this approach to represent rules which are uncertain.

Console and Torasso [CT 90] exploit it to model 'may-cause' links in the diagnosis of car faults: A may-cause link $e <\sim c$ represents knowledge that a cause c may result in an effect e occurring. We can generalise this notion to give indefinite implication. An indefinite implication, $a <\sim body$, states that if *body* is true, then a may be true. This can be represented abductively by the clause $a \leftarrow body, \alpha^*$ where α^* is a new abducible.

Abduction can also be used to represent the causality underlying such statements as 'if someone has spots, then they have measles or acne'. Rather than representing it as a disjunction, as would happen in a disjunctive logic program, we represent the causal statements that acne causes spots, and that measles causes spots. These are represented as two normal clauses, each containing abducibles representing the underlying cause:

$spots \leftarrow measles^*$
$spots \leftarrow acne^*$

The atoms $measles^*$ and $acne^*$ abducible. If either is abduced, it is possible to conclude *spots*. Hence, either of these abducibles is a possible explanation of the observation *spots*.

The disjunctive statement carries the additional information that measles and acne are the only causes of spots. This can be treated in two ways. It can be considered as a meta-statement about the program, stating that the program contains complete information about this predicate, and hence one of the abductive explanations must be correct. Alternatively, it can be treated as an integrity constraint which states that one of the abducibles $measles^*$ or $acne^*$ must hold if the atom *spots* holds.

By combining the may-cause approach used above with appropriate integrity constraints, abduction can also be used to represent statements such as 'a region of the map is coloured red, green or blue'.

We do this by introducing three normal clauses containing abducibles:

$colour(zone1, red) \leftarrow region(zone1), \alpha_1^*$
$colour(zone1, blue) \leftarrow region(zone1), \alpha_2^*$
$colour(zone1, green) \leftarrow region(zone1), \alpha_3^*$

By assuming different abducibles, we can conclude that the given region, *zone*1, is any of these colours. However, if we are allowed to assume any combination of abducibles, then it is possible to conclude that *zone*1 is two colours at once, or that it has no colour at all. To prevent this situation, we use integrity constraints.

Firstly, we use an integrity constraint to ensure that at least one abducible is assumed; $\alpha_1^* \vee \alpha_2^* \vee \alpha_3^*$. This ensures that, if *region*(*zone*1) holds, then at least one clause will have a body which holds, and so *zone*1 will have at least one colour. To prevent it having more than one colour, we use integrity constraints directly on the conclusions of the rules:

$\neg(colour(zone1, red) \wedge colour(zone1, blue))$
$\neg(colour(zone1, red) \wedge colour(zone1, green))$
$\neg(colour(zone1, green) \wedge colour(zone1, blue))$

Note that, unlike disjunctive logic programs, the use of integrity constraints makes the decisions about how to treat an indefinite statement explicit and inspectable.

An extensive survey of abductive logic programming is given in [KKT 93]. We here give a brief description of alternative formal semantics for abductive logic programming.

Eshghi and Kowalski [EK 88] consider a logic program, P, and a set, A, of predicates which appear in P, termed the abducibles. They also consider a set of integrity constraints, IC, which are sentences in first order logic. They consider the *extensions* of the abductive logic program, A, to be all logic programs, $P \cup \Delta$, such that $\Delta \subseteq A$ and $P \cup \Delta$ satisfies IC. They define an abductive explanation for G to be some set $\Delta \subseteq$ A such that some extension $P \cup \Delta$ logically implies G.

Kakas and Mancarella [KM 90] define an abductive explanation for G to be some set $\Delta \subseteq A$ such that G is true in any stable model of $P \cup \Delta$, and the stable model satisfies the integrity constraints *IC*. As the stable model semantics is credulous, this means that $P \cup \Delta$ may not necessarily logically imply G, as is required by Eshghi and Kowalski.

In both the above semantics, an extension must satisfy the integrity constraints. However, we have not specified what this means. Different researchers have proposed alternative definitions of what it means for integrity constraints to be satisfied. Lloyd and Topor [LT 85] have proposed the theoremhood view, whereby a program is said to satisfy the integrity constraints if it implies them. Kowalski and Sadri [KS 87] propose an alternative, weaker, requirement that the integrity constraints be consistent with the logic program, known as the consistency view.

Eshghi and Kowalski [EK89] treat such integrity constraints as metalevel statements which must be true of an extension $P \cup \Delta$. Kakas and Mancarella [KM90], however, view them as statements which must be true in the stable models of the abductive logic program.

In chapter 2, we will give a formal presentation of Kakas and Mancarella's semantics of abduction.

1.4 Even loops in Normal Logic Programs

Saccà and Zaniolo [SZ 90] have proposed using normal logic programming to handle indefinite information, without extending it in any way. They do this by exploiting the even loop construct, and using semantics of normal logic programming which are able to handle it.

An even loop is a set of clauses of the form $\{p \leftarrow \neg q, q \leftarrow \neg p\}$. Some semantics of logic programming, such as the perfect model semantics, [Prz 88b], are not able to handle programs containing such clauses.

Other semantics, such as the well-founded semantics, [VGRS 88], treat p and q as undefined, or unknown. The stable model semantics [GL 88] and its variants, however, consider this to have two alternative models. In one, p is true and q is false, while in the other, q is true and p is false. The stable model semantics therefore allows the representation of different possibilities within a program. Hence, it can be used to represent statements of indefinite information.

For example, Saccà and Zaniolo [SZ90] use this approach for solving a map colouring problem. They would represent our statement that a region in the map is coloured either red, green or blue by using statements in an even loop:

$$colour(X, red) \leftarrow region(X), \neg colour(X, green), \neg colour(X, blue)$$
$$colour(X, green) \leftarrow region(X), \neg colour(X, red), \neg colour(X, blue)$$
$$colour(X, blue) \leftarrow region(X), \neg colour(X, green), \neg colour(X, red)$$

For a given region, the stable model semantics (and other credulous semantics) would have three alternative models, each corresponding to the region being coloured a different colour. Hence, the statement is faithfully represented, and this could be used within a program to explore different possibilities.

In this way, normal logic programs can be used to represent indefinite information of a certain kind. It is less straightforward to represent indefinite information of the other kinds we have introduced above.

1.5 Aims of the Thesis

We have introduced three alternative ways of representing indefinite information within logic programs.

- This thesis will argue that, in many specific cases, disjunctive logic programs are not the most appropriate way of representing indefinite knowledge. Abduction can be used more effectively.

- This thesis will show that disjunctive logic programs can be understood by transforming them to other formalisms, rather than by developing ad-hoc semantics for them in their own right. We will focus in particular on abduction, and show that we can use it to give semantics to disjunctive logic programs which correspond to some of the existing semantics for disjunctive logic programming.

- In certain cases, there is a natural transformation of disjunctive logic programs to normal logic programs containing even loop constructs, allowing a disjunctive logic program to be represented as a normal logic program.

Overview of thesis:

In chapter 2, we give the core concepts and definitions of normal and abductive logic programming, which will be used later in the thesis.

In chapter 3, we show how an abductive framework can be used to represent indefinite information. We use this approach to represent five different kinds of disjunctive information, corresponding to the different examples given in section 1.1. We also show that this approach can be used to represent indefinite information which does not naturally take a disjunctive form. We discuss how this approach compares with the multiple semantics approach used in disjunctive logic programming to handle different forms of disjunctive information.

In chapter 4, we show how the abductive approach is related to three semantics of disjunctive logic programming: The possible world semantics, the disjunctive stable model semantics, and the Indefinite Closed World semantics. In each case, we define the semantics, we present an abductive formulation using the techniques introduced in chapter 3, and show how the abductive formulation relates to the original disjunctive logic programming semantics.

In chapter 5, we introduce exclusive disjunctive logic programs - logic programs with associated integrity constraints which enforce an exclusive interpretation on all disjunctions appearing within them. We discuss the semantics of such logic programs, and show how they relate to the abductive representation of exclusive disjunction, introduced in chapter 3. We then show that a disjunctive logic program which is to be interpreted using the possible world semantics or the Indefinite Closed World Assumption can be transformed into an equivalent exclusive disjunctive logic program.

In chapter 6, we consider the representation of indefinite information using even loops in normal logic programs, as advocated by Saccà and Zaniolo [SZ 90]. Dung has used this approach to transform acyclic disjunctive logic programs into normal logic programs [Dung 92]. We prove that, even if the disjunctive logic program is not acyclic, we can still use this transformation if we are interested in a strictly exclusive interpretation of disjunction. Hence, if the stable model semantics or one of its variants is used as semantics for negation, then exclusive disjunctive logic programs can be transformed into normal logic programs.

Very little work has taken place which compares different semantics of disjunctive logic programming as tools for knowledge representation. In chapter 7, we do this by assessing how well different semantics are able to represent the examples given in section 1.1, and hence the different forms of disjunctive and indefinite information introduced in chapter 3. In chapter 8, we provisionally conclude that abductive and normal logic programs are adequate tools for representing indefinite information, and discuss what further work would be necessary to prove this conclusively.

We summarise the most important results of the thesis in the appendix, for use as reference when reading it.

Contributions of this thesis:

- We argue against much of the current work on semantics of disjunctive logic programs. This argument has two aspects:

  Firstly, we argue against having different semantics for representing different kinds of knowledge (chapters 1 and 3).

  Secondly, we argue that many of the semantics proposed are not able to represent the forms of indefinite knowledge which are used to motivate them (chapter 7).

- We propose an abductive logic programming framework for representing disjunctive and other indefinite forms of knowledge (chapter 3).

- We prove that both the possible model semantics and Indefinite Closed World semantics of disjunctive logic programs can be represented in this framework. We also discuss its relationship with the disjunctive stable model semantics (chapter 4).

- We define exclusive disjunctive logic programs and their semantics. We prove that they can be represented within the abductive logic programming framework (chapter 5).

- We prove that a disjunctive logic program being interpreted under the Indefinite Closed World Assumption semantics can be transformed into an exclusive disjunctive logic program (chapter 5).

- We prove that, if the stable model semantics of negation is adopted, then exclusive disjunctive logic programs can be transformed into normal logic programs (chapter 6).

Chapter 2 - The Tools of Logic Programming

In this chapter we will look formally at some of the concepts we have introduced previously and give definitions and results which will play a role subsequently in the thesis.

Firstly, we will give an introduction to the syntax and model theoretic semantics of first order predicate logic, the formal system upon which logic programming is founded. We will then introduce logic programs themselves, and show how model theory provides a semantics for definite logic programs - the class of logic programs which do not contain negation. Next, we will introduce default negation, as used in logic programming and show how, for a large class of programs, this can be used to assign a single model to a given program by using the *perfect model* semantics. However, for programs not in this class, the perfect model semantics is not defined. We will give an overview of the different semantics which have been used to provide meaning to such programs, focusing in particular on the *stable model* semantics. We will show how the stable model semantics can be used to allow logic programs to represent indefinite information.

Next, we will give a formal definition of *abductive logic programs,* which extend normal logic programs by allowing them to contain assumptions and integrity constraints which place restrictions on permissible models. We will show how any model theoretic semantics of normal logic programs can be extended to provide a semantics of abductive logic programs. We will also show how an abductive logic program can be used to simulate negation, and how to represent a normal logic program as a definite abductive logic program such that the stable models of the normal logic program are in 1-1 correspondence with the abductive solutions to the abductive logic program. We will show how this approach has led to the definition of argumentation theoretic semantics of negation. Finally, we will present the completion semantics of negation, and discuss it's relationship with abduction.

2.1 First Order Logic

Before we can present the formal definitions of logic programming, and particular definitions and results which are of importance to our thesis, it is necessary first to introduce the formal framework on which logic programming is built - first order predicate logic. Our presentation is based on the semi-formal approach used in [Hogger 90].

2.1.1 The Language of First Order Logic

Firstly, we introduce a language L with which we construct statements. We assume several alphabets:

- Constant symbols: $a,b,c\ldots$
- Function symbols: $f,g,h,\ldots$
- Variable symbols: $X,Y,Z,\ldots$
- Predicate symbols: $p,q,\ldots$

A *tuple* of n items is an ordered arrangement of those items, denoted $(a_1,\ldots,a_n)$.

A *term of arity n* is either a constant symbol, a variable, or a function symbol applied to a tuple of n terms. If it contains no variables, it is said to be *ground.*

Definition: An *atomic formula of arity n,* or *atom,* is a predicate symbol applied to a tuple of n terms. Atomic formulas are often called *predicates.*

Definition: An *existential quantifier* is a construct of the form $(\exists X)$, where X is a variable symbol.
Similarly, a *universal quantifier* is a construct of the form $(\forall X)$.

Definition: A *well-formed formula* is defined as follows:

(i) An atomic formula is a well-formed formula.

(ii) *true* and *false* are well-formed formulas.

(iii) If $W1$ and $W2$ are any well-formed formulas, then so are:

$(W1)$

$(W1 \vee W2)$ - A *disjunction.*

$(W1 \wedge W2)$ - A *conjunction.*

$(\neg W1)$- A *negation.*

$(W1 \leftarrow W2)$ - A *conditional.*

$(W1 \leftrightarrow W2)$ - A *biconditional.*

(iv) If W is any well-formed formula, and X any variable, then so are $(\forall X)W$ and $(\exists X)W$.

Definition: A *literal* is either an atomic formula, A, or a negated atomic formula, $\neg A$.

Definition: In each of the well-formed formulas $(\forall X)W$ and $(\exists X)W$, the scope of the quantifier is defined as W.

Definition: An occurrence of a variable symbol, X, in a well-formed formula W, is said to be *bound* if it lies within the scope of some quantifier which occurs in W, or occurs in the quantifier itself. An occurrence of a variable symbol which is not bound is said to be *unbound*, or *free*.

Definition: A *sentence* is a well-formed formula which contains only bound occurrences of variable symbols.

Definition: A *ground formula* is a sentence which does not contain any variables.

Definition: A set, T, of sentences is called a *theory*.

2.1.2 Semantics of First Order Predicate Calculus

Having introduced the language of first order predicate calculus, we now show how a sentence in this language can be assigned a meaning. This is done through *model theory.* The meaning is given relative to a chosen *domain.*

Definition: An *interpretation* of a first order language consists of the domain, *D*, which is a set, together with:

(i) An association of each constant symbol in the language with some element in *D*.

(ii) An association of each *n*-ary function symbol in the language with some function from D_n to *D*.

(iii) An association of each *n*-ary predicate symbol with some relation in D_n.

An element of the set *D* is referred to as a *domain element.*

A ground atomic formula is considered true under this interpretation if the statement it makes about the interpretation is true. Otherwise, the ground atomic formula is considered false under this interpretation. The ground atomic formula *true* is always true, while *false* is always false.

More complex well-formed formulas are assigned truth values as follows:

Let W, $W1$ and $W2$ be any well-formed formulas.

W is false iff W is not true.

$\neg W$ is true iff W is false.

$W1 \wedge W2$ is true iff $W1$ is true and $W2$ is true.

$W1 \vee W2$ is true iff $W1$ is true or $W2$ is true.

$W1 \leftarrow W2$ is false iff $W1$ is false and $W2$ is true.

$W1 \leftrightarrow W2$ is true iff either both $W1$ and $W2$ are true, or both $W1$ and $W2$ are false.

(W) is true iff W is true.

$(\forall X)W$ is true iff, irrespective of which domain element we associate with those occurrences of X not bound by quantifiers within W, the resulting ground formula is always true.

$(\exists X)W$ is true iff there is some domain element which, when we associate it with all occurrences of X not bound by quantifiers within *W*, the resulting ground formula is true.

Definition: An interpretation is said to be a *model* of a sentence iff it results in a sentence being given the value *true*. It is said to be a model for a theory, *T*, iff it is a model for all sentences in *T*.

Definition: A sentence or theory is *satisfiable* iff it has at least one model. It is *valid* iff every interpretation of it is a model.

Definition: A sentence or theory, *T*, *logically implies* a sentence *S*, written $T \models S$, iff any model of *T* is also a model of *S*.

First order predicate calculus also has a system of *inference rules* associated with it, which allow further sentences S to be *derived from* a theory *T*, written $T \vdash S$. We are not concerned here with the details of the inference rules, but rather with their properties. Details of them can be found in [BJ 74].

The important properties of this derivability relation, $\vdash$, are that it is *sound* and *complete*.

Definition: A derivability relation $\vdash$ is *sound* if, for all *T* and *S*, $T \models S$ if $T \vdash S$.

Definition: A derivability relation $\vdash$ is *complete* if, for all *T* and *S*, $T \vdash S$ if $T \models S$.

Proofs of these properties are given in [BJ 74].

2.2 Normal Logic Programs and their Semantics

Definition: A *normal logic program* is a set of clauses, P, of the form, $a \leftarrow b_1,...,b_n, \neg c_1,...,\neg c_m$, where a, b_i, c_j, $(0 \le i \le n, 0 \le j \le m)$ are atoms. a is said to be the *head* of the clause, while $b_1,...,b_n,\neg c_1,...,c_m$ is said to be the *body*. If no clause in P contains a negative literal in its body, (i.e. $m = 0$), then the program is said to be a *definite logic program.*

The semantics of logic programs can be defined either model-theoretically or denotationally. For a summary of the approaches to denotational semantics of logic programs, see [Hogger 90]. In this thesis, we focus on model theoretic semantics. A model-theoretic semantics provides a meaning for a given program by defining interpretations which are models for the logic program. However, we focus on a specific domain, and interpretations using this domain. The domain we use is called the *Herbrand Domain*, and consists of all ground terms in a language L.

Definition: The Herbrand Domain, H_L, of a language L consists of the set of all ground terms which can be constructed within that language. The Herbrand Base, B_L, of L consists of all ground atoms in the language.

If we wish to provide a semantics for a given program, P, we can restrict the language L to contain only those predicate symbols which appear in the program. As [Hogger 90] observes, we cannot restrict other terms in the language in the same way, as we must allow terms which will appear in queries to the program. We represent the Herbrand base using this restricted set of predicates by B_p.

We can use the Herbrand base to reduce a program to propositional form:

Definition: A *ground instance* of a clause consists of the clause with all occurrences of each variable within an atom substituted with some ground term in the Herbrand domain.

Definition: Given a normal logic program, P, we define the *ground instantiation* of P, $G(P)$, to be the program formed by taking the set of all ground instances of all of its clauses.

Note that the ground instantiation of a program need not be finite. Also note that P and $G(P)$ are not logically equivalent in the classical sense, but are logically equivalent if models are restricted to models of H_L.

We can now define the set of interpretations over the Herbrand domain:

Definition: A Herbrand interpretation assigns a truth value to each ground atom in the Herbrand base. It uses the following associations:

(i) It associates each constant in the language with itself.

(ii) It associates an n-ary function symbol, f, in the language with the function which maps an n-tuple of terms $(t_1,...,t_n)$ to the term $f(t_1,...,t_n)$.

(iii) It associates an n-ary predicate symbol p with a relation R_p such that $R_p \subseteq \{t \mid p(t) \in B_p\}$.

There is a convenient shorthand for a Herbrand interpretation; it can be represented as the subset of the Herbrand base which are atoms true in this interpretation.

We can focus our attention specifically on the Herbrand models of a given program, because of the following satisfiability property.

A program P has a model
iff P has a Herbrand model
iff $G(P)$ has a Herbrand model.

Furthermore as there is an equivalence between Herbrand models of P and Herbrand models of $G(P)$, much work on semantics of logic programming focuses on propositional programs, and assigns to a program P the same semantics as its ground instantiation, $G(P)$. For this reason, we will assume that P is a propositional program for the remainder of this section discussing semantics.

A given program, if it is satisfiable, will therefore have a set of Herbrand models which satisfy it. We focus specifically on *minimal* Herbrand models - those models which are not a superset of any other model, when a model is represented as the set of atoms considered true under this interpretation, as described above.

If we consider a definite logic program, P, then it has a unique minimal model. The semantics of such a program is defined in terms of this model. This is because the minimal model of P contains exactly the ground atoms which are implied by P.

Van Emden & Kowalski [VEK 76] have given a constructive characterisation of the minimal Herbrand model of a definite logic program, P. They do this by associating with P a monotonic operator T_p, which transforms a Herbrand interpretation, I into another interpretation, $T_p(I)$ defined as:

$$T_p(I) = \{a \in B_p \mid a \leftarrow b_1,...,b_n \in P \text{ and } b_1,...,b_n \in I\}$$

The unique minimal Herbrand model of P can be found by taking $I = \varnothing$, and repeatedly iterating T_p until a fixed point is reached.

However, if we consider a logic program which is not definite, then it is possible for it to have multiple minimal models. For example, the logic program $\{a \leftarrow \neg b\}$ has two minimal Herbrand models; $\{a\}$ and $\{b\}$. In this situation, researchers have chosen to diverge from a classical logical interpretation of logic programs, to move towards a default interpretation of the negation which appears in the clause bodies.

The intuition behind negation in logic programs comes from the Closed World Assumption of Reiter [Rei 78]; the negation of an atom holds if the atom itself does not hold. This maps neatly onto the procedural interpretation of logic programs, to give the concept of negation as failure [Clark 78]; an atom can be assumed false if the inference procedure cannot prove it true. This is generically referred to as a *default* interpretation of negation: Assume something is false unless it is known true.

Classical logic is *monotonic*, in that if a theory P implies a sentence S, then any superset of P will also imply S. However, when negation is given a default interpretation, this property no longer holds.

Consider the simple logic program $\{a \leftarrow \neg b\}$. As the program does not imply atom b, we can assume the atom $\neg b$. Hence, the program will imply a. However, if we add the atom b to the program, we can no longer assume $\neg b$, and hence a no longer holds.

Normal logic programs thus require a semantics which formalises this intuition. For a certain class of normal logic programs, known as *stratified logic programs*, it is possible to assign a unique model which does this [ABW 88]. For example, the program $p \leftarrow \neg q$ has two minimal models, $\{p, \neg q\}$ and $\{q, \neg p\}$. Under the semantics of [ABW 88], only $\{p, \neg q\}$ is permitted.

However, for arbitrary normal logic programs, it is not possible to assign such a unique model. For example, consider the program:

$p \leftarrow \neg q$
$q \leftarrow \neg p$

Again, this has the two minimal models, $\{p, \neg q\}$ and $\{q, \neg p\}$. However, it is not possible to say that one of these models is favoured over the other under a default interpretation of negation.

There are two different approaches to handle this situation. We can permit multiple models, corresponding to different possibilities. Semantics which adopt this approach are referred to as credulous semantics.

Alternatively, we can construct a single three-valued model, in which atoms can be true, false or unknown. Semantics adopting this approach are referred to as sceptical semantics. The well-founded semantics [VGRS 88] is an example of a sceptical semantics of normal logic programs, while the stable model semantics [GL 88] is a credulous semantics. An overview of semantics of stratified and normal logic programs is given in [DLN 90].

As the stable model semantics will play an important role in this thesis, we define it here.

Definition: Given a normal logic program, P, a Herbrand interpretation, M, of P, we define the definite logic program P^M to be the program obtained from P by:

(i) deleting each clause with a negative literal $\neg c$ in its body, such that c is in M.

(ii) deleting all the negative literals, $\neg c$, in the remaining rules. (i.e deleting all negative literals $\neg c$ such that c is not in M).

M is a stable model of P iff M is the minimal model of the definite program P^M.

An alternative, logically equivalent, definition of P^M is given by Przymusinski [Prz 91]:

Definition: Let M be an interpretation of a general logic program, P. Define the definite logic program, P^M, to be the program formed by substituting each negative literal $\neg a$ with *false* if a is true in M, and with *true* if a is false in M.

We now introduce several lemmas which show how small changes can be made to a program, P, while preserving one of its stable models, M. Later in the thesis, we will use these to prove the correctness of more complex transformations.

Lemma 2.1:

Consider a general logic program, P, an interpretation M, and a clause $a \leftarrow body$ (where $body = b_1,...,b_n, \neg c_1,...,\neg c_m$) such that $body$ is false in M. Then M is a stable model for P if and only if M is a stable model for $P \cup \{a \leftarrow body\}$.

Proof:

Firstly, we consider $\{a \leftarrow body\}^M$. There are two alternative forms it can take.

If some c_i is true in M, then $\{a \leftarrow body\}^M = \varnothing$.

If all c_i are false in M, then $\{a \leftarrow body\}^M = \{a \leftarrow b_1,...,b_n\}$. As $body$ is false in M, it follows that at least one b_i must be false in M.

We now consider these two cases, and show that in each case, M is a stable model of P iff M is a stable model of $P \cup \{a \leftarrow body\}$.

Case 1:

Some c_i is true in M, and hence $\{a \leftarrow body\}^M = \varnothing$.

M is a stable model of P

iff M is the minimal model of P^M

iff M is the minimal model of $P \cup \{a \leftarrow body\}^M$

iff M is a stable model of $P \cup \{a \leftarrow body\}$.

Case 2:

$\{a \leftarrow body\}^M = \{a \leftarrow b_1,...,b_n\}$ and at least one b_i is false in M.

Let Λ be the set of atoms appearing in $P \cup \{a \leftarrow body\}$.

M is a stable model of P

iff M is the minimal model of P^M

iff $P^M \vdash q \Leftrightarrow q \in M$ for all q in Λ, because P^M is a definite logic program

iff $P^M \cup \{a \leftarrow b_1,...,b_n\} \vdash q \Leftrightarrow q \in M$ for all q in Λ, as some b_i is false in M

iff M is the minimal model of $P \cup \{a \leftarrow body\}^M$

iff M is a stable model of $P \cup \{a \leftarrow body\}$.

Hence, in each of the two cases, we have shown that M is a stable model for P if and only if M is a stable model for $P \cup \{a \leftarrow body\}$, as required.
PROVED

Lemma 2.2:

If M is a stable model for a normal logic program P, and a clause $a \leftarrow body$ is true in M, (where $body = b_1,...,b_n, \neg c_1,...,\neg c_m$) then M is a stable model for $P \cup \{a \leftarrow body\}$.

Proof:

If *body* is false in M, this follows from lemma 2.1.

If *body* is true in M, then $\{a \leftarrow body\}^M = \{a \leftarrow b_1,...,b_n\}$ has both head and body true in M. Hence M is a minimal model of $(P \cup \{a \leftarrow body\})^M$, and therefore a stable model of $P \cup \{a \leftarrow body\}$, as required.
PROVED

Lemma 2.3:[1]

Let P be a normal logic program, M be an interpretation, $a \leftarrow body$ (where $body = b_1,...,b_n, \neg c_1,...,\neg c_m$) be a clause such that *body* is true in M, and δ be an atom which does not appear in $P \cup \{a \leftarrow body\}$.

Then M is a stable model of $P \cup \{a \leftarrow body\}$
iff $M \cup \{\delta\}$ is a stable model of $P \cup \{a \leftarrow \delta, \delta \leftarrow body\}$.

Proof:

Firstly, observe that , as *body* is true in M, $\{a \leftarrow body\}^M$ is $\{a \leftarrow b_1,...,b_n\}$.

Let Λ be the set of all atoms appearing in $P \cup \{a \leftarrow body\}$.

Then:

M is a stable model of $P \cup \{a \leftarrow body\}$

[1] This and Lemma 2.4 are special cases of the far more general result that the folding and unfolding program transformation preserves stable models, and any other semantics which can be represented argumentation theoretically. Because the background necessary for these theorems is beyond the scope of the thesis, we do not present them here. The interested reader can refer to [TK 95b],[AD 95] and [PP 94].

iff M is the minimal model of $(P \cup \{a \leftarrow body\})^M$

iff M is the minimal model of $P^M \cup \{a \leftarrow b_1,...,b_n\}$, because body is true in M

iff $P^M \cup \{a \leftarrow b_1,...,b_n\} \vdash q \Leftrightarrow q \in M$ for all q in Λ, because P^M is a definite logic program.

iff $P^M \cup \{a \leftarrow \delta, \delta \leftarrow b_1,...,b_n\} \vdash q \Leftrightarrow q \in M$ for all q in Λ, by resolution.

iff $P^M \cup \{a \leftarrow \delta, \delta \leftarrow b_1,...,b_n\} \vdash q \Leftrightarrow q \in M \cup \{\delta\}$ for all q in $\Lambda \cup \{\delta\}$, as $b_1,...,b_n$ are true in M

iff $(P \cup \{a \leftarrow \delta, \delta \leftarrow body\})^M \vdash q \Leftrightarrow q \in M \cup \{\delta\}$ for all q in $\Lambda \cup \{\delta\}$

iff $(P \cup \{a \leftarrow \delta, \delta \leftarrow body\})^{M \cup \{\delta\}} \vdash q \Leftrightarrow q \in M \cup \{\delta\}$ for all q in $\Lambda \cup \{\delta\}$, as δ does not appear in P or in $body$.

iff $M \cup \{\delta\}$ is the minimal model of $(P \cup \{a \leftarrow \delta, \delta \leftarrow body\})^{M \cup \{\delta\}}$

iff $M \cup \{\delta\}$ is a stable model of $P \cup \{a \leftarrow \delta, \delta \leftarrow body\}$, as required.

PROVED

Lemma 2.4:

Let P be a normal logic program containing a clause $a \leftarrow body$, and α be a literal not appearing in any clause in P. Let P' be the program formed by replacing $a \leftarrow body$ with $a \leftarrow body, \alpha$, and adding the fact α to P.

Then M is a stable model of P if and only if $M \cup \{\alpha\}$ is a stable model of P'.

Proof:

We firstly show that, if $M \cup \{\alpha\}$ is a stable model of P', then M is a stable model of P.

This follows because P is formed from P' by resolving the atom α with the clause $a \leftarrow body, \alpha$.

Now we show that if M is a stable model of P then $M \cup \{\alpha\}$ is a stable model of P'.

M is a stable model of P, therefore a minimal model of P^M. $M \cup \{\alpha\}$ is therefore a model of $(P')^M$. We must show that it is a minimal such model, and therefore a stable model of P'.

Consider $N \subseteq M$ such that $N \cup \{\alpha\}$ is a minimal model of $(P')^M$ and therefore a stable model of P'.

By the first part of the theorem, it follows that N is a stable model of P. Hence $N = M$, and M is a minimal model of $(P')^M$ as required.
PROVED

Lemma 2.5:

Let P be a normal logic program, and $a \leftarrow body$ be a clause in P. Let P' be a program formed from P by substituting $a \leftarrow body$ with $a \leftarrow body, \neg b_1, ..., \neg b_n$. Let M be an interpretation such that, $\neg b_1, ..., \neg b_n$ are true in M.

Then M is a stable model of P iff M is a stable model of P'.

Proof:

Firstly, we show that $(a \leftarrow body, \neg b_1, ..., \neg b_n)^M = (a \leftarrow body)^M$.

Case 1: *body* contains negative literals which are false in M. Hence $(a \leftarrow body)^M = (a \leftarrow body, \neg b_1, ..., \neg b_n)^M = \varnothing$.

Case 2: All negative literals in *body* are true in M. Hence $(a \leftarrow body, \neg b_1, ..., \neg b_n)^M$ is formed by deleting all negative literals which are true in M. As $\neg b_1, ..., \neg b_n$ are all true in M, this corresponds to $(a \leftarrow body)^M$.

As $(a \leftarrow body, \neg b_1, ..., \neg b_n)^M = (a \leftarrow body)^M$, it follows that $P^M = (P')^M$, and hence that M is a stable model of P iff M is a stable model of P', as required.
PROVED

We have informally introduced disjunctive logic programs and their semantics in chapter 1. We now introduce their formal syntax.

Definition: A *disjunctive logic program* is a set of clauses, P, of the form,

$a_1 \vee ... \vee a_n \leftarrow b_1,...,b_m, \neg c_1,...,\neg c_p$, where $a_i, b_j, c_k, (1 \le i \le n, 0 \le j \le m, 0 \le k \le p)$ are atoms. $a_1 \vee ... \vee a_n$ is said to be the *head* of the clause, while $b_1,...,b_m,\neg c_1,...,\neg c_p$ is said to be the *body*. If no clause in P contains a negative literal in its body, (i.e. $p = 0$), then the program is said to be a *positive disjunctive logic program.*

In this thesis, we will focus primarily on semantics of disjunctive logic programs which are related to the stable model semantics of normal logic programs in some way. We will discuss these in chapter 4. However, it is important to note that in certain circumstances, the stable model semantics is inadequate, because of its requirement that every atom be either true or false. This means it is unable to assign models to certain examples, such as $p \leftarrow \neg p$. For that reason, variants of the stable model semantics have been proposed by [SZ 90], [Dung 91] and [Prz 91] which allow p to be undefined in such circumstances.

2.3 Abductive Logic Programs

2.3.1 Semantics of Abductive Logic Programs

Abductive logic programs extend normal logic programs in two ways. Firstly, they allow certain atoms in the language of the logic program to be assumable. Secondly, they allow the imposition of integrity constraints, to restrict the set of possible models to those which satisfy the integrity constraints.

We adopt the formal presentation of abductive logic programs used in [KM 90].

Definition: An *abductive framework* $< P, A, IC >$ consists of P - a normal logic program, A - a set of atoms in the language of P called abducibles, and IC - a set of integrity constraints which are sentences in arbitrary first order logic.

Kakas and Mancarella [KM 90] originally chose for the set A to be a set of predicate symbols, allowing all atoms with these predicate symbols to be abduced. However, we prefer to use the more general approach in which A is a set of ground atoms.

Given a set of predicate symbols, we let A be all the ground instantiations of the corresponding predicates.

They also placed the restriction that no clause in the program P can have an atom in A in its head. This restriction can be made without loss of generality, as Kakas et al. [KKT 92] observe. For any abducible atom, a, which appears in the head of a clause in P, we introduce a new abducible atom δa, and add the clause $a \leftarrow \delta a$ to P. The atom a is no longer abducible. We also adopt this restriction.

Kakas and Mancarella [KM 90] extended the stable model semantics to give a formal semantics of abduction. They considered integrity constraints as sentences which must be true in any acceptable model.

Definition: Let Δ be a set of atoms in A, and M a set of atoms not in A. Then $M \cup \Delta$ is a *generalised stable model* for the abductive framework $< P, A, IC >$ iff $M \cup \Delta$ is a stable model for the normal logic program, $P \cup \Delta$, where $\Delta \subseteq A$, and IC is true in $M \cup \Delta$.

If we wish to find an abductive explanation for a query, g, then we need to find a set of abducibles, Δ, such that $M \cup \Delta$ is a generalised stable model of $< P, A, IC >$, and g is true in $M \cup \Delta$.

The same approach used by Kakas and Mancarella can be used to extend any model theoretic semantics of normal logic programs to abductive logic programs.

Definition: Given a model-theoretic semantics of normal logic programs, *SEM*, then we say that $M \cup \Delta$ is a *generalised SEM model* of the abductive framework $< P, A, IC >$ iff $M \cup \Delta$ is a *SEM* model for the normal logic program, $P \cup \Delta$, where $\Delta \subseteq A$, and IC is true in $M \cup \Delta$.[2]

[2] *SEM* may be a semantics which assigns partial or three valued models to a program, such as the three valued stable model semantics [Prz 91]. In this case, M and Δ are not sets of atoms, but instead are sets of pairs, representing assignments of truth values to atoms in P and A respectively.

In this thesis, we will use this approach to represent disjunction and other forms of indefinite information within an abductive framework while making no commitment as to what semantics of negation is to be used.

We can also place a requirement on Δ to be minimal or maximal:

Definition: A generalised *SEM* model $M \cup \Delta$ of an abductive framework $< P, A, IC >$ is *minimised with respect to A* if there is no other generalised *SEM* model $M \cup \Delta'$ of $< P, A, IC >$ such that Δ' is a subset of Δ.

A generalised *SEM* model $M \cup \Delta$ of an abductive framework $< P, A, IC >$ is *maximised with respect to A* if there is no other generalised *SEM* model $M \cup \Delta'$ of $< P, A, IC >$ such that Δ is a subset of Δ'.

As A is normally understood in context, we abbreviate this to *minimised generalised SEM model* and *maximised generalised SEM model* respectively.

2.3.2 Representing Negation using Abductive Logic Programs

As we explained in section 2.2, logic programs differ from classical logic in their interpretation of negation. The negation adopted is default negation. Negative literals can be assumed if their positive equivalent cannot be proven. As abductive logic programs provide a mechanism for allowing literals to be assumed, it is possible to represent negative literals as abductive hypotheses and provide a semantics of negation by transforming a normal logic program into an abductive logic program which does not contain negation. This approach was presented in [EK 88] and [EK 89]. Here, we present their work using the terminology introduced in the previous section.

Given a normal logic program, P, we define a corresponding abductive framework $< P, A, IC >$ by:

Each negative literal, $\neg a$, in P, is substituted with a new positive abducible literal, $\overline{a}$, to form P^*.

A is the set of all the abducible literals introduced in this way.

IC contains the integrity constraints $a \vee \overline{a}$ and $\neg(a \wedge \overline{a})$ for each abducible literal $\overline{a} \in A$.

The integrity constraints, *IC*, capture the meaning of negation as failure. The integrity constraint $\neg(a \wedge \overline{a})$ prevents an assumption $\overline{a}$ from holding in $P^* \cup \Delta$ if a holds, ensuring that $\overline{a}$ is the negation of a. The integrity constraint $a \vee \overline{a}$ ensures that $\overline{a}$ holds in $P^* \cup \Delta$ if a does not. Hence, in any extension $P^* \cup \Delta$, either a holds or $\overline{a} \in \Delta$, but not both.

Eshghi and Kowalski showed that there is a one-to-one correspondence between extensions of $< P, A, IC >$ and stable models of P.

2.3.3 Argumentation Semantics of Normal Logic Programs

The work presented above considers negative literals as hypotheses which can be assumed in certain circumstances. These circumstances are defined by the integrity constraints associated with the theory. Dung [Dung 91] replaced the requirement that a set of hypotheses must satisfy these integrity constraints, with a more subtle notion based on argumentation.

This work led to the development of an argumentation theoretic semantics for logic programming which acts as a unifying framework for various different semantics of negation [Dung 93]. Furthermore, the applicability of this approach extends beyond logic programming, to other default reasoning techniques [KT94] [DBKT97]. Our presentation of this work is based on that in [KKT 93] and [KT94].

Given a logic program, P, we substitute all negative atomic sentences $\neg a$ with new literals, $\overline{a}$. We define the candidate set of assumptions, A, to be the set of all such literals.

Given two sets of assumptions, Δ, $E \subseteq A$, we say that E *attacks* Δ if $P \cup E \vdash a$ for some $\overline{a} \in \Delta$.

A set of assumptions, Δ, is *acceptable* iff it does not attack itself, and it attacks any other set of assumptions which attack it.

Definition [Dung 91]: A set Δ is a *preferred extension* iff it is a maximal acceptable set. Under the preferred extension semantics, a literal a is considered false if $\overline{a}$ appears in Δ. It is considered true if $P \cup E \vdash a$. Otherwise, it is considered undefined.

Note that this is a weaker requirement than that imposed on stable models. Consider the program, (from [EK89]):

$r \leftarrow \neg r$

$r \leftarrow q$

$p \leftarrow \neg q$

$q \leftarrow \neg p$

This has the single stable model, $\{q, r\}$, in which p is false. This corresponds to the preferred extension $\{\neg p\}$. However, the program has the additional preferred extension, $\{\neg q\}$. In this, p is true and r is undefined.

Stable models correspond to preferred extensions in which no literal is undefined. In other words, a set of assumptions Δ is stable if it is acceptable, and it attacks every assumption it does not contain.

Hence, argumentation semantics can capture the stable model semantics. Kakas and Mancarella [KM92] have also shown that the preferred extension semantics corresponds to the partial stable model semantics of Saccà and Zaniolo [SZ 90].

Dung [Dung 93] has shown that the argumentation approach is also able to capture sceptical semantics. To do this, we need to introduce the notion of defence.

A set Δ *defends* an assumption $\overline{a}$ iff, for every set E attacking $\overline{a}$, Δ attacks E.

We say that a set Δ is *complete* if it contains all literals it defends. Dung has shown that the well-founded semantics is the least such set, and that the stationary models of Przymusinski [Prz 90] correspond to the complete sets of assumptions.

Hence, an argumentation theoretic perspective is able to capture various different semantics of negation which have been proposed for logic programming. The framework it uses is a simple and intuitive extension of abductive logic programming.

2.3.4 Abduction and the Completion Semantics

One of the first, and highly influential, semantics of negation proposed for logic programming was the *completion semantics* proposed by Clark [Clark 78]. The semantics transforms a logic program, P, into a corresponding classical theory, Comp(P). This transformation has played an important role in the understanding of abduction, so we present it here. We give the definition of Comp(P) for propositional programs. Additional equality axioms are required for predicate programs; see [Clark 78] for the full definition.

For every predicate, a, appearing in the head of a rule in P, we collect together all rules in P with a as head:

$$a \leftarrow body_1$$
$$a \leftarrow body_2$$
$$\vdots$$
$$a \leftarrow body_n.$$

To obtain Comp(P) from P, we replace each set with the formula:

$$a \leftrightarrow body_1 \vee \ldots \vee body_n$$

The resulting theory, treated as a classical logic theory, allows the conclusion of more negative information than the original theory, P. $\neg a$ can be concluded in the completion semantics if Comp(P) logically entails $\neg a$. Fitting [Fitting 85] and Kunen [Kunen 87] have proposed the use of a three valued logic to interpret Comp(P).

The link between abductive problems and the iff form of rules used in the completion semantics was first touched upon by Reiter [Rei 87]. He used iff rules to model an abductive problem, (the set covering diagnosis algorithm of Reggia et. al. [RNW 83]), within his logical framework for consistency based diagnosis. It was further explored by Poole ([Poole 88], [Poole 89b]) who used this approach to represent abductive diagnosis when comparing it with consistency-based diagnosis.

Console, Theseider Dupre and Torasso [CTT 90] used the completion semantics to provide an understanding of abduction as reasoning with the completion of a definite abductive logic program, and first proposed an algorithm for computing abductive explanations based on this idea. The algorithm takes a goal, and uses if-and-only-if definitions from the completed program to unfold the goal, replacing it with an equivalent formula - a disjunction of conjunctions. This is repeated, until we are left with a formula which is in the form of a disjunction of conjunctions of abducibles. Each conjunction of abducibles corresponds to one possible abductive solution. They subsequently extended this to cover stratified logic programs and a restricted form of integrity constraints [CTT 91].

Fung and Kowalski [FK 97] have taken this approach further, to allow the handling of abductive logic programs with more general integrity constraints. Their algorithm is a hybrid of that of Console et. al. and the SLDNFA procedure of Denecker and De Schreye [DD 92]. We now use an example to illustrate the idea common to this work. Consider the simple abductive problem introduced in chapter 1.

$spots \leftarrow measles^*$

$spots \leftarrow acne^*$

If we take the completion of this, we get:

$spots \leftrightarrow measles^* \vee acne^*$

If we assert *spots*, then we can conclude $measles^* \vee acne^*$. Any set of abducibles in which this clause holds will be an abductive explanation of *spots*. If we wish to focus on minimal abductive explanations, these correspond to the minimal models of this clause; $\{measles^*\}$ and $\{acne^*\}$.

Hence the completion provides insight into the semantics of abductive logic programs, and also into algorithms for performing abduction. As we shall see in chapter 7, disjunctive logic programs are sometimes motivated with examples which are better represented as the 'only if' part of the completion.

In this chapter, we have introduced the tools of normal and abductive logic programming. We have introduced sceptical and credulous semantics for normal logic programs, in particular the stable model semantics. We have defined abductive logic programming and we have shown how any semantics of normal logic programming can be used to give it a semantics. We have presented the completion semantics, and discussed its relationship with abduction. In the next chapter, we will show how a variety of different forms of indefinite information can be represented within an abductive logic program.

Chapter 3 - Using an Abductive Framework to Represent Indefinite Information

We have shown in chapter 1 how disjunctive logic programs are used to represent indefinite information, and discussed the disadvantages of this approach to knowledge representation. In particular, we argued that choosing a semantics to interpret a disjunctive logic program from among the variety of alternatives implicitly contained several choices, which should be made explicit in the language, rather than in the semantics.

In this chapter, we show how we can represent indefinite knowledge of the kind which has been used to motivate disjunctive logic programming using an abductive framework instead of using explicit disjunctions in the program. In section 3.1, we present a general approach to representing indefinite information within an abductive framework. This uses an extension of the indefinite implication of Console and Torasso [CT 90], together with a collection of different integrity constraints. Depending on which integrity constraints are used, different forms of indefinite information can be represented. In section 3.2, we show that, with appropriate choice of integrity constraints, it is possible to represent exclusive disjunction, causally exclusive disjunction, disjunction with minimised positive consequences and 'inclusive' disjunction. We discuss the advantages of this approach over the multiple semantics approach used in disjunctive logic programming, in particular that it makes explicit the form of disjunction being represented. In section 3.3, we show that the framework allows us to represent other forms of indefinite information which disjunctive logic programs are not able to capture. Hence, the framework is both more expressive and more explicit than disjunctive logic programs. Finally, in section 3.4, we discuss how the framework can be extended to handle non-ground programs.

3.1 The Framework

The framework extends the notion of indefinite implication which we discussed in chapter 1. An indefinite implication, $a <\sim body$, states that if *body* is true then *a* may be true. We showed how Console and Torasso [CT 90] have represented such statements through abduction.

To capture the kinds of indefinite information we are interested in, we generalise the notion of indefinite implication introduced in chapter 1 to allow a clause to indefinitely imply a set of atoms; $\{a_1,...,a_n\} <\sim body$. By this we mean that, if *body* is true in an interpretation, then some of the atoms in the set $\{a_1,...,a_n\}$ may be true in that interpretation. The requirement that 'some of the atoms may be true' is very weak, and can be made more precise in a variety of ways. For example, the indefinite implication:

$$\{nozzle_blocked, tube_blocked, detector_blocked\} <\sim contaminated_sample$$

represents the statement that 'If a contaminated sample is used, then there may be a blockage in the nozzle, there may be a blockage in the dispersion tube, and there may be a blockage in the detector'. This statement could be made more precise in a variety of ways, for example as any of the following three statements.

'If a contaminated sample is used, then there will be a blockage in either the nozzle, the dispersion tube, or the detector, but the blockage will occur in one of these only'.

'If a contaminated sample is used, then there may be a blockage in either the nozzle, the dispersion tube, or the detector, but the blockage will occur in at most one of these'.

'If a contaminated sample is used, then there will be a blockage in either the nozzle, the dispersion tube, or the detector. There may be a blockage in more than one of these'.

Clearly, if the indefinite implication

$\{nozzle_blocked, tube_blocked, detector_blocked\} <\sim contaminated_sample$ is used to represent any of these three statements, then it only partly captures the knowledge to be represented. In addition, we need to be able to place restrictions on which subsets of the set in the head of the clause are to be considered possible. This, as we shall see, will be done formally through the use of integrity constraints.

Note that a disjunctive logic program can represent the first and third of these statements, but not the second, by using the disjunctive clause:

$$nozzle_blocked \vee tube_blocked \vee detector_blocked \leftarrow contaminated_sample$$

This clause is interpreted as meaning either the first or the third statement, (or possibly other alternatives) depending on which semantics of disjunctive logic programs is used.

We will now show how an abductive framework $< P, A, IC >$ can be used to capture such indefinite implications, and can impose different interpretations on them through the use of integrity constraints.

The indefinite implication $\{a_1,...,a_n\} <\sim body$ is represented by a set of clauses of the form used by [Poole 89a]:

$$a_1 \leftarrow body, \alpha_1^*$$
$$\vdots$$
$$a_n \leftarrow body, \alpha_n^*$$

$\{\alpha_1^*,..,\alpha_n^*\}$ are newly introduced abducible atoms.

We refer to this as the *clause set* corresponding to the indefinite implication $\{a_1,...,a_n\} <\sim body$. By assuming different sets of abducibles from among $\{\alpha_1^*,..,\alpha_n^*\}$ to be true, we can conclude, from *body*, different subsets of the set of atoms in the head.

We use integrity constraints to restrict the subsets of abducibles which can be considered. The choice of such integrity constraints will therefore affect which of the possible subsets of $\{a_1,...,a_n\}$ can indeed be concluded from this clause set. Hence, such a choice must be tailored to the needs of the domain.

These integrity constraints fall into two categories:

- Constraints on the lower bound of the subsets of abducibles to be considered.

- Constraints on the upper bound of the subsets of abducibles to be considered.

The lower bound constraints ensure that enough abducibles are assumed to satisfy the requirements of the problem. For example if we want the clause set to simulate disjunction we don't want the empty set of conclusions to be valid.

Options for formalising the lower bound we will consider are:

$$\alpha_1^\bullet \vee \ldots \vee \alpha_n^\bullet$$

This is the simplest form of lower bound constraint. It requires at least one of the abducibles to hold, so ensures that at least one of the clauses in the clause set will conclude some a_i if *body* is true. However, it has the disadvantage that it still requires at least one abducible to be true, even if *body* is false, and so the abducible has no effect on the clause set.

$$\alpha_1^\bullet \vee \ldots \vee \alpha_n^\bullet \leftarrow body$$

This integrity constraint overcomes the problem just described above. As it requires an abducible to be assumed only if the clause set body is true, it allows none to be assumed if the clause set is false, and so not contributing to the model. However, there are still circumstances in which there is not a one-to-one correspondence between sets of abducibles and sets of conclusions of a clause set. If we want such a correspondence (as we will in chapter 4 for representing disjunctive logic programming semantics in our framework), we must add additional integrity constraints.

To see this, consider the following example:

$a \leftarrow \alpha_1^*$
$b \leftarrow \alpha_2^*$
$c \leftarrow \alpha_3^*$
a
b

If we use the integrity constraint $\alpha_1^* \vee \alpha_2^* \vee \alpha_3^*$ alone, then the abductive framework has the generalised stable models $\{\alpha_1^*, a, b\}$, $\{\alpha_2^*, a, b\}$, $\{\alpha_1^*, \alpha_2^*, a, b\}$ and $\{\alpha_1^*, \alpha_2^*, \alpha_3^*, a, b, c\}$. The first three of these differ only with respect to the abducibles they contain. If we wish to have only one set of abducibles corresponding to a given set of conclusions of a clause set being true, we must use additional integrity constraints.

The constraints we use are of the form $\{\alpha_i^* \leftarrow a_i, body \mid i = 1..n\}$, and are included in the lower bound constraint set. These have the effect of ensuring that an abducible α_i^* is abduced iff a_i and *body* are true in the model.

In the previous example, we would add the integrity constraints $\alpha_1^* \leftarrow a$, $\alpha_2^* \leftarrow b$ and $\alpha_3^* \leftarrow c$. Abducing α_1^* or α_2^* alone no longer result in programs with models satisfying the integrity constraints. Of the generalised stable models listed above, only $\{\alpha_1^*, \alpha_2^*, a, b\}$ and $\{\alpha_1^*, \alpha_2^*, \alpha_3^*, a, b, c\}$ satisfy the additional integrity constraints.

As we will discuss in section 3.2.3, these integrity constraints can also be combined with minimisation of abducibles to ensure that a minimal set of atoms a_i in the head of a disjunctive clause holds true.

We may also use the integrity constraint $a_1 \vee ... \vee a_n \leftarrow body$, as Sakama and Inoue [SI94] do in work related to ours. However, as we shall discuss in section 4.1, this integrity constraint does not give a one-to-one correspondence between sets of abducibles and conclusions of a clause set.

BIBL.
LONDIN.
UNIV.

We also have the option of placing no lower bound, by using no constraints. Alternatively, we could define a relative lower bound, by considering only sets of abducibles which are maximal.

The upper bound constraints ensure that we do not assume too much. For example, if we want the clause set to model exclusive disjunction, we use integrity constraints to ensure that at most one of the conclusions is true.

$\{\neg(a_i \wedge a_j) \leftarrow body \mid \forall i \neq j\}$ - This constraint set enforces exclusivity on the possible conclusions of the clause set: If the body of the clause holds, then at most one of the atoms in its head can hold.

$\{\neg(a_i \wedge a_j) \mid \forall i \neq j\}$ - This constraint set enforces a stronger form of exclusivity: At most one of the atoms in the clause head can hold, irrespective of whether the body holds.

$\{\neg(\alpha_i^{\bullet} \wedge \alpha_j^{\bullet}) \mid \forall i \neq j\}$ - This constraint set ensures that at most one of the abducibles in a clause set are assumed, and hence at most one of the clauses can be used to draw a conclusion.

$\{\neg(\alpha_i^{\bullet} \wedge a_j) \mid \forall i \neq j\}$ - This constraint set ensures that an abducible can be assumed only if no other conclusion of the clause set holds. This is stronger than the previous set of constraints, in that if some a_j holds because of another clause set, it prevents any other $\alpha_i^{\bullet}$ being assumed.

$body \leftarrow \alpha_1^{\bullet} \vee ... \vee \alpha_n^{\bullet}$ - This integrity constraint ensures that abducibles can only be assumed if the body of the clause set is true. This combines neatly with the lower bound integrity constraint $\alpha_1^{\bullet} \vee ... \vee \alpha_n^{\bullet} \leftarrow body$ to give $\alpha_1^{\bullet} \vee ... \vee \alpha_n^{\bullet} \leftrightarrow body$.

This combined integrity constraint ensures that a clause set has abducibles associated with it if and only if its body is true. This avoids the situation that a program containing a clause set with a false body may have several generalised models, simply corresponding to different choices of abducibles in that clause set, but otherwise identical to each other.

Again, we also have the option of placing no upper bound by using no constraints in this category, or a relative upper bound by considering only sets of abducibles which are minimal. It is sometimes appropriate to use several upper bound integrity constraints, or to combine upper bound integrity constraints with minimisation. For example, consider the upper bound integrity constraint $body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$. We introduced this constraint earlier to ensure that if *body* is false in a model, no abducibles appear in that model. However, if *body* is true then this integrity constraint places no restrictions on sets of abducibles. We can use other integrity constraints, or minimisation, to constrain the abducibles when *body* is true.

Hence, a knowledge engineer must make the following decisions when representing indefinite information:

- How should the lower bound on acceptable conclusions from a clause set be represented?

- How should the upper bound on acceptable conclusions from a clause set be represented?

These decisions provide the integrity constraints for the abductive framework $< P, A, IC >$, and whether abducibles are to be maximised, minimised or neither.

In addition, two further decisions are required:

- How is negation to be interpreted?

As we discussed in section 2.3, we can extend any semantics of negation for normal logic programs to assign a semantics to the abductive framework. In this thesis, we will primarily use minimal models of definite logic programs and stable models of normal logic programs, as these semantics will allow us to prove relationships between our framework and existing semantics of disjunctive logic programs. However, other semantics of negation may be preferable for different purposes. Given a choice of semantics, *SEM*, we can apply it to our abductive framework $< P, A, IC >$ and consider generalised *SEM* models, as defined in chapter 2.

- Do we require credulous or sceptical conclusions?

The abductive framework $< P, A, IC >$ may have several generalised *SEM* models, corresponding to different sets of abducibles which can satisfy the integrity constraints. We say an atom holds credulously in the framework if it is true in any of these generalised *SEM* models. We say it holds sceptically if it is true in all of the generalised *SEM* models.

Note that this approach uses abduction to simulate the effects of modal operators, 'possible' and 'necessary'. Thus, to some extent, abductive logic programming is used here to give the functionality of 'possibility' and 'necessity' in modal logic. Possibility corresponds to being true in some generalised model, while necessity corresponds to being true in all generalised models. However, this simulation is weak, in that sentences of the language do not include such modalities. Here, they are external to the language. [Gelfond 94] explores the use of these modalities explicitly within disjunctive logic programming clauses.

3.2 Representing Disjunctive Information in the Framework

We will now show in greater detail how this knowledge representation framework can be applied to represent a variety of different interpretations of disjunction, depending on what is appropriate in a particular case.

3.2.1 Representing exclusive disjunctive information.

In the majority of cases where it is necessary to represent different alternatives, only one of the alternatives can be true. The different alternatives are mutually exclusive of each other. For example, a test result could be positive or negative. A pool ball could be spot, stripe or black. A loose connection in a circuit can result in no current flowing, or a very low current flowing, but not both at the same time. However, despite the fact that this kind of situation occurs very commonly, it is not adequately captured by disjunctive logic programming alone. We can capture a lower bound on what is possible by using a disjunctive clause such as $no_current \vee low_current \leftarrow loose_connection$. However, such a clause does not capture the fact that at most one of the different alternatives can be true, unless the chosen semantics can force this to be the case.

Some semantics of disjunctive logic programming are described as 'exclusive', and try to ensure that at most one of the different atoms in the head of a clause holds in a given model by minimising the atoms which hold. However, as we discussed in chapter 1, this does not accurately interpret disjunction as exclusive. Sakama and Inoue [SI92] propose adding integrity constraints which explicitly state that the atoms in the head of a disjunctive clause are mutually exclusive of each other. In the above example, the integrity constraints would be $\neg(no_current \wedge low_current)$.

We now consider how to represent disjunctive knowledge where the alternatives are mutually exclusive, in an abductive framework $< P, A, IC >$ using the representation techniques we have presented above.

Consider a disjunctive statement $a_1 \vee ... \vee a_n \leftarrow body$ where we wish to represent the atoms appearing in the head as being mutually exclusive,

1. We add the clause set:

$$\begin{aligned} a_1 &\leftarrow body, \alpha_1^* \\ &\vdots \\ a_n &\leftarrow body, \alpha_n^* \end{aligned}$$

to our logic program, P.

2. We take the lower bound integrity constraint for this clause set to be

   $\alpha_1^\bullet \vee \ldots \vee \alpha_n^\bullet \leftarrow body$ and combine it with the upper bound constraint

   $body \leftarrow \alpha_1^\bullet \vee \ldots \vee \alpha_n^\bullet$ to give the constraint $\alpha_1^\bullet \vee \ldots \vee \alpha_n^\bullet \leftrightarrow body$.

3. We take the additional upper bound integrity constraints to be the set

   $\{\neg(a_i \wedge a_j) | \forall i \neq j\}$, which enforces mutual exclusivity on the conclusions.

These ensure that a set of abducibles Δ is acceptable only if it contains at least one abducible for the clause set if *body* is true. Hence, if *body* is true in a given model $M \cup \Delta$ of $P \cup \Delta$, then at least one of the clauses in the clause set will have a true *body*, and so at least one a_i will be true in $M \cup \Delta$. Furthermore, because of the upper bound integrity constraints, $M \cup \Delta$ will be acceptable as a generalised model of the framework only if at most one of the a_i in the clause set is true in $M \cup \Delta$.

Consider the example used earlier in this section. The disjunctive statement

no_current $\vee$ *low_current* $\leftarrow$ *loose_connection* is represented by:

1. Adding the clause set to the logic program, P:

   no_current $\leftarrow$ *loose_connection*, $\alpha_1^\bullet$.
   low_current $\leftarrow$ *loose_connection*, $\alpha_2^\bullet$.

2. Adding the abducibles $\alpha_1^\bullet$ and $\alpha_2^\bullet$ to the set A.

3. Adding the integrity constraints $\alpha_1^\bullet \vee \alpha_2^\bullet \leftrightarrow$ *loose_connection* and $\neg$(*no_current* $\wedge$ *low_current*) to the set IC.

Hence, we can represent information where the alternative outcomes are mutually exclusive.

The integrity constraints $\{\neg(a_i \wedge a_j) | \forall i \neq j\}$ proposed by Sakama and Inoue are stronger than is appropriate in many cases.

Sometimes additional information allows such constraints to be used, but in the general case a clause containing exclusive disjunction does not justify them. Consider the statement, " a pool ball can be striped, spotted or black." If a ball is a pool ball, it can be only one of these. However, these alternatives are not intrinsically mutually exclusive. A ball which is both black and striped can exist (though it wouldn't be a pool ball). Hence, the integrity constraints proposed by Sakama and Inoue are too strong to truly represent exclusive disjunction. Instead, we use constraints of the form $\neg(a_i \wedge a_j) \leftarrow body$ as the upper bound.

This time, the disjunctive statement $spot(b) \vee stripe(b) \vee black(b) \leftarrow pool_ball(b)$ is represented by:

1. Adding the clause set to the logic program, P:

   $spot(b) \leftarrow pool_ball(b), \alpha_1^*$
   $stripe(b) \leftarrow pool_ball(b), \alpha_2^*$
   $black(b) \leftarrow pool_ball(b), \alpha_3^*$

2. Adding the abducibles $\{\alpha_1^*, \alpha_2^*, \alpha_3^*\}$ to the set A.

3. Adding to IC the integrity constraints $\alpha_1^* \vee \alpha_2^* \vee \alpha_3^* \leftrightarrow body$ and

   $\neg(spot(b) \wedge stripe(b)) \leftarrow pool_ball(b)$
   $\neg(spot(b) \wedge black(b)) \leftarrow pool_ball(b)$
   $\neg(stripe(b) \wedge black(b)) \leftarrow pool_ball(b)$

These integrity constraints prevent both $stripe(b)$ and $black(b)$ from being true in the same model only if $pool_ball(b)$ is also true in that model. The general form of this is as follows:

Given the disjunctive statement $a_1 \vee ... \vee a_n \leftarrow body$ which we wish to represent as being an exclusive disjunctive clause.

1. We add the clause set:

$$a_1 \leftarrow body, \alpha_1^\bullet$$
$$\vdots$$
$$a_n \leftarrow body, \alpha_n^\bullet$$

to our logic program, P.

2. We take the lower bound integrity constraint for this clause set to be $\alpha_1^\bullet \vee ... \vee \alpha_n^\bullet \leftarrow body$ and combine it with the upper bound constraint $body \leftarrow \alpha_1^\bullet \vee ... \vee \alpha_n^\bullet$ to give the constraint $\alpha_1^\bullet \vee ... \vee \alpha_n^\bullet \leftrightarrow body$.

3. We take the additional upper bound integrity constraints to be the set $\{\neg(a_i \wedge a_j) \leftarrow body \mid i \neq j\}$, which enforces mutual exclusivity on the conclusions.

These integrity constraints are weaker than the strict mutual exclusivity constraints used above. Depending on the knowledge to be represented a knowledge engineer may choose either of these approaches within our framework. The second approach gives an exclusive interpretation of the disjunction appearing in the head of the original clause, whilst the first approach enforces the stricter requirement that all atoms appearing in the clause head must be mutually exclusive.

3.2.2 Representing causally exclusive disjunction

As we discussed earlier, in the majority of cases where an underlying cause can result in one of several possible alternatives, these alternatives are mutually exclusive. However, in some cases, even though the cause can result in only one of the alternatives, the alternatives themselves are not mutually exclusive. To see this, consider the following example.

A contaminated sample in a gas chromatograph will result in a blockage either in the entry nozzle, or in the dispersion tube, or in the detector. However, it will result in a blockage in only one of these places. Furthermore, there are other causes of blockages, such as a pinching of the tube.

In this example, the cause can have three possible outcomes. However, it will result in only one of these three. As a result of this particular cause, a blockage will appear in exactly one of three places, and not more. However, the three outcomes are not mutually exclusive. It is possible that some other event, such as a collapsed column, could result in another blockage. Because of this, it is not appropriate to use the integrity constraints discussed in the previous section.

We now consider how to represent causally exclusive disjunctive knowledge in an abductive framework $< P, A, IC >$.

Given the disjunctive statement $a_1 \vee ... \vee a_n \leftarrow body$, which we wish to understand as causally exclusive.

1. We add the clause set:

$$a_1 \leftarrow body, \alpha_1^*$$
$$\vdots$$
$$a_n \leftarrow body, \alpha_n^*$$

to our logic program, P.

2. We add the abducibles $\{\alpha_1^*,...,\alpha_n^*\}$ to the set A.

3. We take the lower bound integrity constraint for this clause set to be $\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$, and combine it with the constraint $body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$ to give the constraint $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$.

4. We take the additional upper bound integrity constraints to be the set $\{\neg(\alpha_i^* \wedge \alpha_j^*) \mid \forall i \neq j\}$

The lower bound and upper bound integrity constraints together ensure that any acceptable set of abducibles, Δ, will contain exactly one of the abducibles corresponding to this clause set if *body* is true. Hence, if *body* is true in a model $M \cup \Delta$ of $P \cup \Delta$, then exactly one of the clauses will have a true body, and so the corresponding head a_i will be true in the model. However, unlike the case of the exclusive integrity constraints discussed above, this does not place the restriction that no other head a_j be true in the model.

Returning to our example, we can see how this is represented in our framework.

P will contain the clause set:

nozzle_blocked ← *contaminated_sample*, $\alpha_1^\bullet$
tube_blocked ← *contaminated_sample*, $\alpha_2^\bullet$
detector_blocked ← *contaminated_sample*, $\alpha_3^\bullet$

$\alpha_1^\bullet, \alpha_2^\bullet, \alpha_3^\bullet \in A$ are abducibles.

IC will contain the integrity constraint $\alpha_1^\bullet \vee \alpha_2^\bullet \vee \alpha_3^\bullet \leftrightarrow$ *contaminated_sample*, and the additional upper bound integrity constraints $\{\neg(\alpha_1^\bullet \wedge \alpha_2^\bullet), \neg(\alpha_1^\bullet \wedge \alpha_3^\bullet), \neg(\alpha_2^\bullet \wedge \alpha_3^\bullet)\}$.

If we also add the fact *contaminated_sample* to *P*, then *P* has 3 generalised minimal models, each containing one of the atoms *nozzle_blocked*, *tube_blocked* and *detector_blocked*. This is identical to what we would get if we treated the disjunction as exclusive. However, if we add an additional clause, *tube_blocked* ←*pinched_tube*, representing the information that the tube will be blocked if it gets pinched, and the atom *pinched_tube*, then this is no longer the case. The abductive framework still has three generalised minimal models, but in all of them, *tube_blocked* is true. In one, *nozzle_blocked* is also true, in the second, *detector_blocked* is also true, while in the third, neither is also true. The first two of these models would not be models if the disjunction were treated as exclusive.

We will see in chapter 4 that the Indefinite Closed World Assumption semantics [KWC 95] interprets disjunction in a causally exclusive fashion. Consequently, the Indefinite Closed World Assumption semantics can be represented within our framework by transforming all disjunctive clauses into clause sets with integrity constraints enforcing causal exclusivity. Furthermore, our representation is neutral with respect to negation, allowing a choice of how to interpret it, and so can be used to extend the indefinite closed world assumption to interpret negation in a variety of ways.

3.2.3 Minimising positive consequences of disjunction

As we discussed in chapter 1, the Generalised Closed World Assumption and the various semantics of disjunctive logic programs which extend it all act to minimise the set of atoms true in the head of a disjunctive clause.

This approach can be used to represent statements such as 'If someone has *spots*, then they have acne or measles'. We can represent this as a disjunctive clause:

$acne \vee measles \leftarrow spots$

Given *spots*, we do not wish to consider models in which both *acne* and *measles* hold, as this is a non-parsimonious solution we only wish to consider the case that someone has two illnesses only if one illness cannot explain the observed symptoms. However, we cannot say that *acne* and *measles* are mutually exclusive and so we can't treat this as exclusive disjunction. Hence, if the statement is represented as a disjunctive clause, it is appropriate to consider models in which a minimal set of atoms in the clause head are true.

As we will discuss in chapter 7, we believe that information of this form is better represented using the completion of a program, or as causal statements in an abductive program. However, as disjunction of this form is used to motivate work on disjunctive logic programming (for example [Ross 89] p387), we show that it can be represented in our framework.

Firstly, we define formally what it means to minimise the positive consequences of disjunction:

Definition: Let M be a model of a disjunctive logic program, and let $d(M)$ be those atoms in M which appear in the head of some disjunctive clause. We say that M *minimises positive consequences of disjunction* iff there is no model N such that $d(N)$ is a proper subset of $d(M)$.

Within our framework, $< P, A, IC >$, we can choose to interpret a disjunctive statement in this way. Given the disjunctive statement $a_1 \vee ... \vee a_n \leftarrow body$, whose positive consequences we wish to minimise.

1. We add the clause set:

$$a_1 \leftarrow body, \alpha_1^*$$
$$\vdots$$
$$a_n \leftarrow body, \alpha_n^*$$

to the logic program, P. The newly introduced abducibles, $\{\alpha_1^*,...,\alpha_n^*\}$, are in A.

2. We take the lower bound integrity constraint for this clause set to be $\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$, and combine it with the constraint $body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$ to give the constraint $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$.

We have the additional lower bound integrity constraints, $\{\alpha_i^* \leftarrow a_i, body \mid \forall i = 1..n\}$.

We take no additional upper bound integrity constraints, but instead place the restriction on acceptable sets of abducibles Δ, that the set $\{\alpha_1^*,...,\alpha_n^*\}$ be minimised.

The lower bound integrity constraints ensure that any set of abducibles will contain at least one of the abducibles corresponding to some clause in this clause set, if the body of that clause set is true.

Furthermore, if a_i is true in $M \cup \Delta$, then Δ must also contain α_i^* if IC is to be true in $M \cup \Delta$. By placing the requirement that Δ contain a minimal set of abducibles α_i^*, we ensure that Δ will contain only one α_i^* unless some clauses in P not in this clause set result in more a_i being true.

Hence the disjunctive statement $acne \vee measles \leftarrow spots$ becomes the abductive logic program defined as follows:

P:

$acne \leftarrow spots, \alpha_1^*$

$measles \leftarrow spots, \alpha_2^*$

$spots$

$A = \{\alpha_1^*, \alpha_2^*\}$

IC:

$\alpha_1^* \vee \alpha_2^* \leftrightarrow spots$

$\alpha_1^* \leftarrow acne, spots$

$\alpha_2^* \leftarrow measles, spots$

We consider A-minimised generalised SEM models of $< P, A, IC >$.

As the program does not contain any negation, we have no need to adopt a semantics of negation, so can use the minimal model semantics of definite logic programs as our underlying semantics.

Hence we consider the A-minimised generalised minimal models of our framework. These are $\{acne, spots, \alpha_1^*\}$ and $\{measles, spots, \alpha_2^*\}$.

Note that IC is true in the model $\{acne, measles, spots, \alpha_1^*, \alpha_2^*\}$ but this is not an acceptable solution, as $\Delta = \{\alpha_1^*, \alpha_2^*\}$ is not A-minimised - If we take $\Delta' = \{\alpha_1^*\}$, then $P \cup \Delta'$ also has a model in which IC are true.

Hence, in this way, we can use our framework to represent disjunctive information which we wish to minimise. In chapter 4, we will explore how this approach relates to the stable model semantics [Prz 91] of disjunctive programs.

3.2.4 Representing 'inclusive' disjunction.

Another very common use of disjunction in knowledge representation is to represent a statement in which one or more possible conclusions follow from a condition. For example, [LMR 92], we may choose to represent the statement 'if someone works for NASA, then they are either an astronaut, a rocket engineer, or possibly both', with the disjunctive clause: $astronaut \vee rocket_engineer \leftarrow works_at_nasa$.

This kind of disjunction is often referred to in the literature as 'inclusive', in that we wish to consider the possibility that both atoms in the head are true. However, this terminology is misleading. A minimised disjunctive clause is also inclusive, in that it is possible for more than one atom in its head to be true. For simplicity, we will use the conventional, if somewhat misleading, term 'inclusive' for this kind of disjunction.

It is straightforward to represent this form of disjunction within our framework. Consider the disjunctive statement $a_1 \vee ... \vee a_n \leftarrow body$, which we wish to represent as 'inclusive'.

1. We add the clause set:

$$a_1 \leftarrow body, \alpha_1^\bullet$$
$$\vdots$$
$$a_n \leftarrow body, \alpha_n^\bullet$$

to the logic program, P.

2. We take the lower bound integrity constraint for this clause set to be $\alpha_1^\bullet \vee ... \vee \alpha_n^\bullet \leftarrow body$, and combine it with the upper bound constraint $body \leftarrow \alpha_1^\bullet \vee ... \vee \alpha_n^\bullet$ to give the constraint $\alpha_1^\bullet \vee ... \vee \alpha_n^\bullet \leftrightarrow body$.

3. We take no additional upper bound constraint set.

With this choice of constraints, we ensure that a set of abducibles Δ which results in a generalised *SEM* model $M \cup \Delta$ will contain at least one α_i^* for the clause set. However, we place no upper bound on the number of abducibles corresponding to the clause set. Hence, if *body* is true, then different generalised *SEM* models will correspond to different sets of atoms in the head of the disjunction being true.

For our example, we represent the disjunctive clause:

$$astronaut \vee rocket_engineer \leftarrow works_at_nasa$$

within the abductive framework $< P, A, IC >$ as follows:

The logic program P contains the clause set:

$$astronaut \leftarrow works_at_nasa, \alpha_1^*$$
$$rocket_engineer \leftarrow works_at_nasa, \alpha_2^*$$

where $\{\alpha_1^*, \alpha_2^*\} \in A$ are new abducibles.

IC contains the integrity constraint. $\alpha_1^* \vee \alpha_2^* \leftrightarrow works_at_nasa$.

If we add the fact *works_at_nasa*, then the program has three generalised minimal models. In one *astronaut* is true, in another *rocket_engineer* is true, and in the third, both these literals are true.

We will see in chapter 4 that the abductive representation we employ in our framework can be used to represent the possible model semantics [Sak 89] of disjunctive logic programs abductively. Hence, our framework is also able to capture this semantics of disjunction.

3.2.5 Advantages of this approach

Rather than using various different semantics to handle disjunction in different ways, we propose a single framework within which different approaches to disjunction can be unified. Furthermore, the framework can represent other forms of indefinite information. Knowledge engineers make a series of four choices, based on the needs of their problem; which lower bound integrity constraints to adopt, which upper bound constraints to adopt, which semantics of negation to adopt, and whether to use the resulting framework credulously or sceptically.

This approach has the following advantages:

- The choices that are made about the approach to disjunction adopted are explicitly represented, rather than being implicit in the choice of disjunctive semantics adopted. Hence they are inspectable and open to verification by others more easily.

- While the choice of a semantics must be applied globally to the entire program, integrity constraints can be applied locally to specific clause sets. Hence, different approaches to disjunction can easily be used in the same program.

- Decisions about which semantics of negation to adopt are kept separate from decisions about how to treat disjunction. Semantics of disjunctive logic programming do not allow this. However, unlike the decision of how to handle each disjunction, the decision of how to handle negation must be taken globally.

- The decision about how to treat the indefinite information is divided into two; how to ensure that enough is concluded, and how to ensure that too much is not concluded. By separating this out, we allow a greater richness of possibilities than is available through adopting a single semantics of disjunctive logic programming in which both these decisions are combined in a single semantics.

- A single inference procedure such as that of Fung and Kowalski [FK 97], able to handle abductive logic programs with integrity constraints, can be used to reason with all the different forms of disjunctive and indefinite information we have considered, provided a specific semantics of negation is adopted.

3.3 Representing other forms of indefinite information within our framework.

In the previous sections, we looked at how to use the framework to represent different interpretations of disjunctive information. However, our framework is not restricted to representing disjunctive information. Within our framework, the different forms of disjunction all share a common integrity constraint, $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$, ensuring that if the body of a clause set is true in a generalised model, at least one of the atoms in the head of a clause set will be true in it. It is possible, and indeed, at times, appropriate, to use the framework with alternatives to this lower bound integrity constraint. In particular we may decide to have no lower bound, allowing the possibility that, even though the body of a clause set is true in a generalised model, the model need not contain any atom appearing in the head of the clause set.

For example, consider an alternative version of the gas chromatograph example used above. Rather than assuming that a contaminated sample will definitely result in a blockage in one of the three places, we may wish to say that a contaminated sample could result in a blockage, or it may result in no blockage, as the contaminant manages to pass right through the chromatograph.

We can represent this in the framework as:

P will contain the clause set:

$nozzle_blocked \leftarrow contaminated_sample, \alpha_1^*$

$tube_blocked \leftarrow contaminated_sample, \alpha_2^*$

$detector_blocked \leftarrow contaminated_sample, \alpha_3^*$

$\alpha_1^*, \alpha_2^*, \alpha_3^* \in A$ are abducibles.

IC will contain the upper bound integrity constraints

$$\{\neg(\alpha_1^\bullet \wedge \alpha_2^\bullet), \neg(\alpha_1^\bullet \wedge \alpha_3^\bullet), \neg(\alpha_2^\bullet \wedge \alpha_3^\bullet)\}.$$

Unlike in section 3.2.2, *IC* will contain no lower bound integrity constraints.

If we represent the knowledge in this way, and add the fact *contaminated_sample*, then we will get the same generalised models as in section 3.2.2, plus the additional model in which none of the literals *nozzle_blocked*, *tube_blocked* and *detector_blocked* are true. This form of indefinite information is not disjunctive, so cannot be captured by disjunctive logic programs. Nonetheless, it clearly has a value in knowledge representation.

Finally, we can also consider adding integrity constraints which link abducibles within one clause set with abducibles in another. We can use this to ensure that two clause sets can't have the same outcome.

Consider a clause set $\{a_1 \leftarrow body1, \alpha_1^\bullet, \ldots, a_n \leftarrow body1, \alpha_n^\bullet\}$, and a second clause set $\{a_1 \leftarrow body2, \beta_1^\bullet, \ldots, a_n \leftarrow body2, \beta_n^\bullet\}$, where both clause sets have the same atoms in their head. We can enforce these two clause sets to have different outcomes by adding the integrity constraints $\{\neg(\alpha_i^\bullet \wedge \beta_i^\bullet) \mid i = 1..n\}$ to *IC*.

Care must be taken to ensure that these new integrity constraints do not conflict with others in *IC*. In particular, if the integrity constraints $\{\alpha_i^\bullet \leftarrow a_i, \beta_i^\bullet \leftarrow a_i \mid \forall i = 1..n\}$ are also included in *IC*, then any set of abducibles containing some $\alpha_i^\bullet$ will be inconsistent with the integrity constraints if both *body*1 and *body*2 hold. Hence, this combination of integrity constraints should not be adopted.

While situations requiring the use of the framework in this way are not common, they do occur. For example, consider picking a numbered ball out of a bag, and not replacing it. The first pick will remove one number. The second pick will be unable to remove the same number again.

Each of these events is causally exclusive, in that each will result in the removal of only one ball, but more than one ball can (and will) be removed. Hence, they can be represented in our framework using the approach described in section 3.2.2. However, they have the additional constraints described above, representing the fact that one event cannot result in the same outcome as the other event.

So, in the case of three balls and two selections, we would get two clause sets in P:

$ball1 \leftarrow first_pick, \alpha_1^*$

$ball2 \leftarrow first_pick, \alpha_2^*$

$ball3 \leftarrow first_pick, \alpha_3^*$

$ball1 \leftarrow second_pick, \beta_1^*$

$ball2 \leftarrow second_pick, \beta_2^*$

$ball3 \leftarrow second_pick, \beta_3^*$

A would contain the abducibles $\alpha_1^*, \alpha_2^*, \alpha_3^*, \beta_1^*, \beta_2^*$ and β_3^*.

IC would contain the integrity constraints $\{\alpha_1^* \vee \alpha_2^* \vee \alpha_3^*\}$ and $\{\neg(\alpha_1^* \wedge \alpha_2^*), \neg(\alpha_1^* \wedge \alpha_3^*), \neg(\alpha_2^* \wedge \alpha_3^*)\}$ ensuring that the first clause set is treated as causally exclusive disjunction, and would contain similar constraints for the second clause set. In addition, it would contain the constraints $\{\neg(\alpha_1^* \wedge \beta_1^*), \neg(\alpha_2^* \wedge \beta_2^*), \neg(\alpha_3^* \wedge \beta_3^*)\}$, ensuring that the two clauses cannot result in the same conclusions.

If the atoms *first_pick* and *second_pick* are added, then the abductive framework would have six generalised models. In two, the literals *ball*1 and *ball*2 would both be true, and *ball*3 would be false. One of these models, generated by the set $\{\alpha_1^*, \beta_2^*\}$,corresponds to the first pick being *ball*1, and the second being *ball*2, while the other, generated by $\{\alpha_2^*, \beta_1^*\}$ corresponds to the first pick being *ball*2, and the second being *ball*1. Similarly, in the other four generalised models, two have *ball*2 and *ball*3 true, and two have *ball*1 and *ball*3 true.

3.4 Extending the Framework to handle non-ground programs

So far, we have restricted our attention to theories which contain only ground terms, and have assumed that the corresponding integrity constraints are also ground. If we relax this assumption, it is necessary to modify the integrity constraints used.

Consider a clause in the program, P, with variables in the body, $a_1 \vee ... \vee a_n \leftarrow body(X_1,...,X_m)$. (It may also have variables in the head, but we can disregard these). In the ground instantiation of P, $G(P)$, there will be many instantiations of this clause, corresponding to different instantiations of each X_i. What integrity constraints should we use to give results equivalent to the ground integrity constraints we have introduced above? We will consider the constraint $\alpha_1^* \vee ... \vee \alpha_n^*$ as an example case.

The constraint $\alpha_1^* \vee ... \vee \alpha_n^*$ in the ground instantiation can be obtained by using the integrity constraint $\forall(X_1,...,X_m)[\alpha_1^*(X_1,...,X_m) \vee ... \vee \alpha_n^*(X_1,...,X_m)]$. The universal quantification is important to ensure that a version of the integrity constraint exists for every ground instantiation of the corresponding clause. If existential quantification were used instead, then the integrity constraint could simply be satisfied by abducing a single $\alpha_i^*(x_1,...,x_m)$, and hence imposing the disjunctive integrity constraint on only one of the ground instantiations of the clause.

However, the universal integrity constraint has the disadvantage that, to satisfy it, some $\alpha_i^*(x_1,...,x_m)$ must be abduced for every possible instantiation of the clause in the Herbrand base, the majority of which will be irrelevant from a practical point of view. While this is acceptable as a theoretical tool, it is not much use in practice, and would cause extreme inefficiency. A more elegant solution is to use the integrity constraint $\forall(X_1,...,X_m)[\alpha_1^*(X_1,...,X_m) \vee ... \vee \alpha_n^*(X_1,...,X_m) \leftarrow body(X_1,...,X_m)]$. In this case, it is necessary to abduce only some $\alpha_i^*(x_1,...,x_m)$ if $body(x_1,...,x_m)$ holds. Hence, only the relevant abductions are made.

This provides further justification for our decision to use the integrity constraint $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$ rather than $\alpha_1^* \vee ... \vee \alpha_n^*$ when representing disjunctive information in section 3.2.

We can also apply the same techniques, described above with respect to the integrity constraint $\alpha_1^* \vee ... \vee \alpha_n^*$, to the other constraints we have used.

The move to a non-ground theory also has advantages in the representation of the integrity constraints; it is possible to eliminate the use of disjunction within the integrity constraints, replacing it with existential quantification. This is done by parameterising the abducibles which are introduced. Rather than introducing n new abducibles $\alpha_1^*(X_1,...,X_m),...,\alpha_n^*(X_1,...,X_m)$ for a given disjunctive clause in P, we instead introduce a single abducible, $\alpha^*(1, X_1,...,X_m)$. The clause set corresponding to the disjunctive statement becomes:

$$a_1 \leftarrow body(X_1,...,X_m), \alpha^*(1, X_1,...,X_m).$$
$$\vdots$$
$$a_n \leftarrow body(X_1,...,X_m), \alpha^*(n, X_1,...,X_m).$$

Instead of the disjunctive integrity constraint,
$\forall(X_1,...,X_m)[\alpha_1^*(X_1,...,X_m) \vee ... \vee \alpha_n^*(X_1,...,X_m)] \leftarrow body(X_1,...,X_m)$, we can use an existentially quantified integrity constraint
$\forall(X_1,...,X_m)\exists I\,[\alpha^*(I, X_1,...,X_m) \leftarrow body(X_1,...,X_m)]$. The disjunction has been completely eliminated. Hence, in an abductive framework using predicate logic, it is possible to represent disjunctive information without the use of disjunction either in the program, *P*, or in the integrity constraints, *IC*. Similarly, denial integrity constraints can be represented more compactly.

In this chapter we have introduced our framework for representing indefinite information through the use of abduction and showed how it can be used in a variety of different ways. In particular, we have shown that it can be used to interpret disjunctive information in different ways, depending on the needs of the problem. In the next chapter, we will consider how these approaches to disjunction relate to existing semantics of disjunctive logic programs, and show equivalence relations in several cases.

Chapter 4 - The Relationship between the Framework and existing semantics of Disjunctive Logic Programs

In the previous chapter, we presented our framework for the representation of indefinite information, and showed how it could be used to represent a variety of different types of disjunctive information, as well as indefinite information which is not disjunctive. In this chapter, we show that the framework accurately captures and generalises several existing semantics of disjunctive logic programs. In section 4.1 we present the possible model semantics of Chan [Cha 89] and Sakama [Sak 89] and show how it can be represented abductively within our framework. The representation is a special case of that used to represent 'inclusive' disjunction in chapter 3. In section 4.2 we present the disjunctive stable model semantics of Przymusinski [Prz 91] and discuss its relationship with our framework. We show that the representation of disjunction with minimised sets of positive consequences used in chapter 3 does not correspond in general to disjunctive stable models. We conjecture that the correspondence does hold if we restrict the program to be stratified. Finally, in section 4.3 we show that the Indefinite Closed World Assumption semantics of Kong, Williams and Chen [KWC 94] can be represented in our framework, using a special case of the representation of causally exclusive disjunction presented in chapter 3.

4.1 The Possible Model Semantics

The possible model semantics for positive disjunctive logic programs was developed independently by Chan [Cha 89] and Sakama [Sak 89]. Sakama and Inoue [SI 93] extended the definition to handle disjunctive logic programs with negation. We will show that our framework is able to simulate this extended definition of the possible model semantics.

Firstly, we present the definition of the possible model semantics taken from [SI 93]:

Definition: Given a disjunctive clause $C = a_1 \vee ... \vee a_n \leftarrow body$, we define a *split of C* to be any non-empty subset of the set of non-disjunctive clauses:

$\{a_1 \leftarrow body, ..., a_n \leftarrow body\}$.

Given a disjunctive logic program, P, we define $GLP(P)$ to be the set of all normal logic programs which can be formed by replacing each disjunctive clause in P with one of its splits.
The set of *possible models* of P are defined as the set of stable models of programs in $GLP(P)$.

Sakama and Inoue [SI 94] subsequently explored the relationship between the possible model semantics and abduction. They showed that, by using a transformation into an abductive logic program with appropriate integrity constraints, the possible model semantics of disjunctive logic programs can be understood abductively.

To do this, they consider an extension of abductive logic programming, which allows clauses to be abducible, not just atoms, and extend the definition of generalised stable models accordingly. The transformation is defined as follows:

Given a disjunctive logic program, P, they define an abductive framework $< P', A, IC >$ as follows:

- The normal logic program, P', is formed by deleting all disjunctive clauses from P.
- For each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, the clauses $a_1 \leftarrow body, ..., a_n \leftarrow body$ are in the set of abducible clauses, A.
- The integrity constraints IC consist of all disjunctive clauses in P.

They prove that M is a possible model of P iff M is a generalised stable model of $< P', A, IC >$.

Their approach suffers from certain disadvantages:

1. It does not use the standard definition of an abductive framework, but instead requires clauses to be abducible.

2. For a given possible model, M, there may be several sets, Δ, of abducible clauses such that M is a stable model of $P' \cup \Delta$ and IC are true in M.

3. Under their transformation, it is possible for an atom to appear both in the head of a clause in P' and in the set A [KM 90] and others have argued that this situation should be avoided.

For example, consider the program P: $\{a, b, a \vee b\}$, which has the unique possible model, $M = \{a, b\}$.

The transformation gives:

P':
a
b

A:
a
b

IC:
$a \vee b$

Any subset Δ of A yields the generalised stable model, $\{a, b\}$.

Sakama and Inoue [SI 93] propose overcoming the first of these disadvantages by labelling these default rules with abducible atoms [Poole 89a], as we described in chapter 2. However, doing this means the assumptions of their theorem are no longer valid, and does not overcome the problem that multiple sets of abducibles in the transformed framework correspond to a single possible model of the original program.

We propose overcoming these problems through the use of stricter integrity constraints. We use those introduced in section 3.2.4 to model 'inclusive' disjunction, and the additional integrity constraints, $\{\alpha_i^* \leftarrow a_i, body \mid i = 1 \ldots n\}$, to ensure a one to one correspondence between sets of abducibles and possible models.

We propose the following transformation:

Given a disjunctive logic program, P, which is to be interpreted using the possible model semantics, we define a corresponding abductive framework $< P', A, IC >$ as follows:

1. Each non-disjunctive clause, p, in P is also in P'.

2. For each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, the corresponding clause set

$$a_1 \leftarrow body, \alpha_1^*$$
$$\vdots$$
$$a_n \leftarrow body, \alpha_n^* \quad \text{is in } P'$$

$\alpha_1^*, ..., \alpha_n^*$ are in A.

We take the lower bound integrity constraints for this clause set to be:

$$\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$$
$$\{\alpha_i^* \leftarrow a_i, body \mid i = 1...n\}$$

3. We take the upper bound integrity constraints to be:

$$body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$$

As in chapter 3, we combine the integrity constraints $\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$ and $body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$ to give $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$.

For example, consider the simple program:

P:

$a \vee b$
$a \vee d$
b

This has three possible models: $\{a, b\}$, $\{b, d\}$ and $\{a, b, d\}$.

The transformation gives the abductive logic program:

P':

$a \leftarrow \alpha_1^*$

$b \leftarrow \alpha_2^*$

$a \leftarrow \beta_1^*$

$d \leftarrow \beta_2^*$

b

IC:

$\alpha_1^* \vee \alpha_2^*$

$\beta_1^* \vee \beta_2^*$

$\alpha_1^* \leftarrow a$

$\alpha_2^* \leftarrow b$

$\beta_1^* \leftarrow a$

$\beta_2^* \leftarrow d$

This has three generalised stable models; $\{a, b, \alpha_1^*, \alpha_2^*, \beta_1^*\}, \{b, d, \alpha_2^*, \beta_2^*\}$ and $\{a, b, d, \alpha_1^*, \alpha_2^*, \beta_1^*, \beta_2^*\}$ corresponding to the possible models of the original program.

For a given possible model, M, we can construct a corresponding set of abducibles, Δ:

For a given clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, α_i^* is in Δ iff α_i and $body$ are true in M.

In the example above, consider the possible model, $M = \{b, d\}$. As the body of the clause $a \vee b$ is trivially true in M, and b is true in M, then α_2^* is true in Δ. Similarly, considering $a \vee d$ we conclude that β_2^* is in Δ. Hence $\Delta = \{\alpha_2^*, \beta_2^*\}$.

$P' \cup \Delta$ has the stable model $\{b, d, \alpha_2^*, \beta_2^*\}$, and the integrity constraints IC are true in this model. Hence, Δ generates the generalised stable model corresponding to $\{b, d\}$, as we wished.

We can formalise this relationship as follows:

Theorem 4.1:
Consider a disjunctive logic program, P, with a corresponding abductive framework, $< P', A, IC >$ as defined above.

(i) If $M \cup \Delta$ is a generalised stable model of the abductive framework $< P', A, IC >$, then M is a possible model of P.

(ii) If M is a possible model of P, then there exists a unique $\Delta \subseteq A$ s.t. $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$.

Proof:
(i) Firstly, we show that if $M \cup \Delta$ is a generalised stable model of the abductive framework $< P', A, IC >$, then M is a possible model of P. We do this by constructing a normal logic program, Q, for the given set of abducibles, Δ, and showing that it is a member of $GLP(P)$. We then use the lemmas of chapter 2 to show that M is a stable model of Q, and hence a possible model of the original disjunctive logic program, P.

We define Q to be the normal logic program formed from P and Δ as follows:

- Each non-disjunctive clause in P is also in Q.

- For each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, the clause $a_i \leftarrow body$ will be in Q if the clause $a_i \leftarrow body, \alpha_i^*$ is in P', and the abducible α_i^* is in Δ. If no such abducible is in Δ, then any one clause, $a_1 \leftarrow body$ say, is in Q.

There will be at least one clause in Q for each disjunctive clause. Hence Q is a member of $GLP(P)$.

We now show that M is a stable model of Q.

We form Q_1 from $P' \cup \Delta$ by deleting every clause $a_j \leftarrow body, \alpha_j^*$ such that $\alpha_j^* \notin \Delta$. By lemma 2.1, $M \cup \Delta$ is a stable model of Q_1.

We form Q_2 from Q_1 by replacing each clause $a_i \leftarrow body, \alpha_i^*$ and abducible α_i^* with the clause $a_i \leftarrow body$. By lemma 2.4, M is a stable model of Q_2.

We form Q from Q_2 by adding any clause $a_i \leftarrow body$ which appears in Q but not in Q_2. By construction of Q_2, all such clauses have *body* false in M, and so are true in M. Hence, by lemma 2.1, M is a stable model of Q.

As M is a stable model of Q, it is a possible model of P as required.

(ii) We now show that if M is a possible model of P, then there exists a unique subset Δ of A s.t. $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$.

We construct a set of abducibles, Δ: For a given clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, α_i^* is in Δ iff a_i and *body* are true in M. We show that Δ satisfies the integrity constraints and that $M \cup \Delta$ is a stable model of the program $P' \cup \Delta$. Finally we show that there exists no other subset Δ' of A such that $M \cup \Delta'$ is a generalised stable model of $< P', A, IC >$.

IC is true in $M \cup \Delta$:

By construction of Δ, all constraints of the form $\alpha_i^* \leftarrow body, a_i$ are trivially true in $M \cup \Delta$.

Consider constraints of the form $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$.

If *body* is false in M, then, by construction of Δ, Δ will contain no α_i^*. If body is true in M, then as the clause $a_1 \vee ... \vee a_n \leftarrow body$ is true in M, at least one a_i is true in M.

Hence at least one α_i^* is in Δ. Hence, *body* is true in M iff Δ contains at least one α_i^*.

Hence $\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$ is true in $M \cup \Delta$, as required.

$M \cup \Delta$ is a stable model of the program $P' \cup \Delta$:

As M is a possible model of P, there exists $Q \in GLP(P)$ such that M is a stable model of Q.

We construct Q_1 from Q by replacing each clause $a_i \leftarrow body$ such that *body* is true in M with the clause $a_i \leftarrow body, \alpha_i^*$ and the abducible α_i^*. As *body* and a_i are true in M iff $\alpha_i^* \in \Delta$, then Δ is the set of abducibles introduced in this way.

Hence, by lemma 2.4, $M \cup \Delta$ is a stable model of Q_1.

We form $P' \cup \Delta$ from Q_1 by adding the clauses $\{a_j \leftarrow body, \alpha_j^* \mid \alpha_j^* \notin \Delta\}$. As α_j^* is false in $M \cup \Delta$, then $M \cup \Delta$ is a stable model of $P' \cup \Delta$ by lemma 2.1.

Hence $M \cup \Delta$ is a stable model of $P' \cup \Delta$, and *IC* is true in $M \cup \Delta$, so $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$ as required.

It remains to show that such a Δ is unique. Consider $\Delta' \neq \Delta$ such that $M \cup \Delta'$ is a generalised stable model of $< P', A, IC >$. $\Delta \subseteq \Delta'$, otherwise some constraint of the form $\alpha_i^* \leftarrow body, a_i$ would be false in $M \cup \Delta'$. Consider $\alpha_i^* \in \Delta' \setminus \Delta$. It follows by construction of Δ that either *body* or a_i must be false in M. If *body* is false in M, then the integrity constraint of $\alpha_1^* \vee ... \vee \alpha_n^* \rightarrow body$ is false in $M \cup \Delta'$. If *body* is true in M, then a_i must be true in M, as $\alpha_i^* \in \Delta'$, and $a_i \leftarrow body, \alpha_i^*$ is true in $M \cup \Delta'$. This contradicts the assumption that either *body* or a_i is false.

Hence $\Delta' = \Delta$, as required.

PROVED

Hence, we have shown that the possible model semantics can be represented within our framework, in such a way that there is a one-to-one correspondence between possible models and abductive models.

4.2 The Stable Model Semantics of Disjunctive Logic Programs

The stable model semantics [GL88] was expanded by Przymusinski [Prz 91] in two ways. He defined it to cover disjunctive logic programs, and he extended it to use a three-valued logic. This allows some atoms in a model to remain undefined. For the purposes of this section, we will focus on the two-valued case only.

Stable models are defined for disjunctive logic programs as follows:

Definition: Let M be an interpretation of a disjunctive logic program, P. Define the positive disjunctive logic program, P^M, to be the program formed by substituting each negative literal $\neg a$ with 'false' if a is true in M, and with 'true' if a is false in M.

We say that M is a *disjunctive stable model* of P iff it is a minimal model of the positive disjunctive program, P^M.

As with normal logic programs, if a disjunctive logic program does not contain any negative literals, then the stable models of the program will coincide with its minimal models. This semantics can therefore be viewed as a combination of the stable model semantics with Minker's Generalised Closed World Assumption.

The disjunctive stable model semantics differs from the possible model semantics in that it minimises the set of atoms true in the head of disjunctive clauses. For example, consider again the program:

P:

$a \vee b$
$a \vee d$

This has four possible models; $\{a\}$, $\{a, d\}$, $\{a, b\}$ and $\{b, d\}$. However, it has only two stable models; $\{a\}$ and $\{b, d\}$. In this example, these correspond to the possible

models which are minimal; i.e. those which are not supersets of some other possible model.

In the general case, we can show that every disjunctive stable model is a minimal possible model.

Theorem 4.2.1:
Let P be a disjunctive logic program with a disjunctive stable model M. Then M is a minimal possible model of P.

Proof:
Let M be a disjunctive stable model of P. We form Q from P by substituting each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$ with the split $\{a_i \leftarrow body \mid a_i \leftarrow body \text{ true in } M\}$. If $body$ is true in M, then the split for this disjunctive clause will be $\{a_i \leftarrow body \mid a_i \text{ true in } M\}$. If $body$ is false in M, then the split will be $\{a_i \leftarrow body \mid i = 1..n\}$.

As each split set has at least one clause in it, Q is in $GLP(P)$. We now show that M is a stable model of Q, and therefore a possible model of P. We use the following lemma:

Lemma 4.2.2:
Let P be a disjunctive logic program with a minimal model M. Let R be the program formed by substituting each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$, with the set $\{a_i \leftarrow body \mid a_i \leftarrow body \text{ true in } M\}$. Then M is the minimal model of R.

Proof of lemma:
M is a minimal model of P, and the clauses $\{a_i \leftarrow body \mid a_i \leftarrow body \text{ true in } M\}$ are true in M. Hence M is a minimal model of $R^+ = P \cup \{a_i \leftarrow body \mid a_i \leftarrow body \text{ true in } M\}$.
Note that R^+ consists of R and the disjunctive clauses $a_1 \vee ... \vee a_n \leftarrow body$.
For every disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in R^+, there is at least one clause $a_i \leftarrow body$ in R^+ which logically subsumes it. Hence R is logically equivalent to R^+, as

it consists of R together with a set of logically redundant disjunctive clauses. Hence, M is a minimal model of R. As R contains no disjunctive clauses, M is the unique minimal model.

LEMMA PROVED

We now apply this lemma to the positive disjunctive logic program, P^M. For any clause $a_1 \vee ... \vee a_n \leftarrow body^+$ appearing in P^M, there is a clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, such that $body^+$ consists of $body$ with all negative literals removed. As M is a disjunctive stable model of P, it is a minimal model of P^M. Hence M is the minimal model of R, consisting of P^M with all the disjunctive clauses $a_1 \vee ... \vee a_n \leftarrow body^+$ substituted with the clauses $\{a_i \leftarrow body^+ \mid a_i \leftarrow body^+ \text{ true in } M\}$. As $R = Q^M$, M is the minimal model of Q^M. Hence M is a stable model of Q, and so a possible model of P.

As M is a minimal model of P, and possible models of P are models of P, it follows that M is a minimal possible model, as required.

PROVED

However, the converse does not hold in general, as the following example shows:

$a \vee b$
$c \leftarrow \neg a$
$c \leftarrow \neg b$

This has the three possible models, $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$. All of these are clearly minimal. However, only $\{a, c\}$ and $\{b, c\}$ are disjunctive stable models. Note that $\{a, b\}$, though minimal, does not minimise the set of atoms true in the disjunction $a \vee b$.

Hence, minimal possible models impose a weaker minimality requirement than disjunctive stable models do.

In chapter 3, we introduced an abductive approach to minimising positive consequences of disjunctive clauses. We do this by requiring that the set of abducibles in the

abductive framework be minimal. We now consider the semantics that results from transforming a disjunctive logic program into this framework. As in section 4.1, we add the additional integrity constraints, $\{\alpha_i^* \leftarrow a_i, body \mid i = 1...n\}$, to eliminate multiple sets of abducibles for a single model.

Given a disjunctive logic program, P, which is to be interpreted with disjunctions having minimal sets of consequences, we define a corresponding abductive framework $< P', A, IC >$ as follows:

- Each non-disjunctive clause in P is also in P'.

- For each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, the corresponding clause set

  $a_1 \leftarrow body, \alpha_1^*$
  $\vdots$
  $a_n \leftarrow body, \alpha_n^*$

  is in P'.

  The newly introduced abducibles, $\{\alpha_1^*,...,\alpha_n^*\}$, are in A.

- We take the lower bound integrity constraints for this clause set to be:

  $\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$
  $\{\alpha_i^* \leftarrow a_i, body \mid i = 1...n\}$

- We take the upper bound integrity constraints to be:

  $body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$.

The integrity constraints $\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$ and $body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$ again can be combined to give $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$.

Note that this is identical to the transformation used in section 4.1. However, this time we do not consider all generalised stable models of the framework, but instead restrict ourselves to the A-minimised generalised stable models. We will explore how these relate to the disjunctive stable models of the original program.

Consider applying this to the example above:

$a \vee b$
$c \leftarrow \neg a$
$c \leftarrow \neg b$

The transformation gives:

P':

$a \leftarrow \alpha_1^*$
$b \leftarrow \alpha_2^*$
$c \leftarrow \neg a$
$c \leftarrow \neg b$

IC:

$\alpha_1^* \vee \alpha_2^*$
$\alpha_1^* \leftarrow a$
$\alpha_2^* \leftarrow b$

This has the three generalised stable models, $\{a, c, \alpha_1^*\}$, $\{b, c, \alpha_2^*\}$ and $\{a, b, \alpha_1^*, \alpha_2^*\}$, of the original disjunctive program, corresponding to three possible models. However, only two of these are A-minimal; these two correspond to the disjunctive stable models $\{a, c\}$ and $\{b, c\}$ of the original disjunctive program.

Hence, A-minimisation of the abductive framework is a stricter minimality requirement than minimality of possible models, and hence closer in spirit to disjunctive stable models.

However, A-minimisation of the abductive framework is not in general equivalent to the disjunctive stable models of the original program, as the following example shows:

P:

$a \vee b \vee c$
$d \leftarrow \neg e$
$e \leftarrow \neg d$
$a \leftarrow d$
$b \leftarrow d$

This program has four disjunctive stable models; $\{e, a\}$, $\{e, b\}$, $\{e, c\}$ and $\{d, a, b\}$.

Note that the disjunctive stable model $\{d, a, b\}$ does not minimise the positive consequences of disjunctive clauses as the model $\{e, a\}$ contains a proper subset of atoms appearing in the head of disjunctive clauses.

The transformation gives:

P':

$a \leftarrow \alpha_1^*$
$b \leftarrow \alpha_2^*$
$c \leftarrow \alpha_3^*$
$d \leftarrow \neg e$
$e \leftarrow \neg d$
$a \leftarrow d$
$b \leftarrow d$

IC:

$\alpha_1^* \vee \alpha_2^* \vee \alpha_3^*$
$\alpha_1^* \leftarrow a$
$\alpha_2^* \leftarrow b$
$\alpha_3^* \leftarrow c$

The generalised stable model corresponding to $\{d, a, b\}$ is $\{d, a, b, \alpha_1^*, \alpha_2^*\}$. However, this is not A-minimised, as $\{e, a, \alpha_1^*\}$ is also a generalised stable model. The double loop in the program means that, in some generalised stable models, both a and b will be true, while in others, only one of them will be true. A-minimisation looks for minimal sets among all stable models, and so misses $\{d, a, b, \alpha_1^*, \alpha_2^*\}$.
A-minimisation accurately captures the notion of minimisation of positive consequences of disjunctive clauses, but this notion of minimisation is stronger than that used by disjunctive stable models.

For a restricted class of programs, it may be possible to show that disjunctive stable models and A-minimised generalised stable models correspond to each other.

Conjecture 4.2.3:
Consider a disjunctive logic program, P, with a corresponding abductive framework, $< P', A, IC >$ as defined above, such that any program $P' \cup \Delta$ has a unique stable model, then

(i) If M is a disjunctive stable model of P, then there is a unique Δ such that $M \cup \Delta$ is an A-minimised generalised stable model of $< P', A, IC >$.

(ii) If $M \cup \Delta$ is an A-minimised generalised stable model of $< P', A, IC >$, then M is a disjunctive stable model of P, where M is defined as $M \cup \Delta$ with all abducibles removed.

If this conjecture proves to be true, then we can also conclude that our framework can capture the perfect semantics [Prz 88b] for disjunctive logic programs. The stable model of a stratified normal logic program corresponds to its perfect model. Similarly, the disjunctive stable models of a stratified disjunctive logic program correspond to its perfect models.

Corollary 4.2.4:
Consider a stratified disjunctive logic program, P, with a corresponding abductive framework, $< P', A, IC >$ as defined above, then

(i) If M is a perfect model of P, then there is a unique Δ such that $M \cup \Delta$ is an A-minimised generalised perfect model of $< P', A, IC >$.

(ii) If $M \cup \Delta$ is an A-minimised generalised perfect model of $< P', A, IC >$, then M is a perfect model of P, where M is defined as $M \cup \Delta$ with all abducibles removed.

This follows from the fact that the stratification of P induces a stratification on P', and hence on any $P' \cup \Delta$. Hence, each $P' \cup \Delta$ will have a unique stable model, corresponding to its perfect model.

Hence, we have shown that A-minimisation as currently defined in our framework does not capture the disjunctive stable model semantics, though we believe it does in restricted circumstances. Our notion of A-minimisation is stronger than the minimisation used in disjunctive stable models. It is an open question which of these two forms of minimisation is the better. As we will discuss in chapter 7, there is no strong evidence that minimisation of heads of distinctive clauses is a useful tool for knowledge representation. It is usually motivated with examples which are better understood as exclusive disjunction, or as the completion of a normal logic program. Without better motivation for using minimisation in disjunction, we cannot come to conclusions as to which specific form of minimisation is more practically useful.

We have not yet successfully represented the disjunctive stable model semantics of disjunctive logic programs within our abductive framework. In addition to the approach described above, we have tried alternative integrity constraints and other forms of minimisation. This is somewhat disappointing, though we do believe that further work with different forms of minimisation may yield a more positive result. However, in chapter 5 we will show that if we consider exclusive disjunctive logic programs (disjunctive logic programs restricted so that at most one atom in the head of a clause is true) then we can prove that the disjunctive stable model semantics of the resulting program can be accurately represented within our framework.

4.3 The Indefinite Closed World Assumption Semantics

We will now consider the Indefinite Closed World Assumption [KWC 94] and show that it too can be represented abductively, by a different choice of integrity constraints.

The Indefinite Closed World Assumption is a variant of the possible model semantics. It is defined for positive disjunctive logic programs only.

Definition [KWC 94]: Given a disjunctive logic program not containing negation, P, we define $ICWA(P)$ to be the set of all non-disjunctive logic programs which can be formed from P by replacing each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$, with exactly one of the non-disjunctive clauses $\{a_i \leftarrow body \mid i = 1..n\}$.

M is an *I-model* of P if M is the minimal model of some logic program P^* in $ICWA(P)$.

In a given P^*, we say that the non-disjunctive clause $a_i \leftarrow body$ which has replaced the disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P is the *representative* of $a_1 \vee ... \vee a_n \leftarrow body$ in P^*.

Notice that, while the possible model semantics allows the substitution of the disjunctive clause with any non-empty subset of the corresponding definite clauses, the Indefinite Closed World Assumption allows substitution by only a single clause from the set.

The Indefinite Closed World Assumption can be easily extended to cover general disjunctive logic programs by using the same approach as used by Sakama and Inoue [SI92] to extend the possible model semantics. The new definition is as follows:

Definition: Given a general disjunctive logic program, P, we define $ICWA(P)$ to be the set of all non-disjunctive logic programs which can be formed from P by replacing each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$, with exactly one of the non-disjunctive clauses $\{a_i \leftarrow body \mid i = 1...n\}$.

M is an I-model of P iff M is the stable model of some logic program P^* in $ICWA(P)$.

As the stable model of a definite logic program corresponds to its minimal model, these two definitions are equivalent for positive disjunctive logic programs.

Subsequently in this paper, when we refer to the Indefinite Closed World Assumption semantics, this will mean the second extended definition unless otherwise stated.

To demonstrate the differences between this, the possible model semantics, and disjunctive stable model semantics, consider the following example:

$$\{b, b \vee c \vee d\}$$

$$ICIVA(P) = \{\{b, b\}, \{b, c\}, \{b, d\}\}.$$

This has the unique disjunctive stable model, $\{b\}$, which, as the program does not contain negation, corresponds to the minimal model. It has the following possible models: $\{b\}$, $\{b, c\}$, $\{b, d\}$, $\{b, c, d\}$. This semantics interprets disjunction as 'inclusive'.

The *ICIVA* does not require that a model be a minimal model of the disjunctive logic program, P, as the stable model semantics does. However, it does not allow disjunctions to be used to conclude any number of disjuncts, as in the possible model semantics. Each disjunctive rule can be used to derive exactly one consequence, but there is no requirement that the resulting model be minimal. In this example, $\{b\}$, $\{b, c\}$ and $\{b, d\}$ are I-models, but $\{b, c, d\}$ is not. (It requires the single rule $b \vee c \vee d$ to conclude both c and d).

Because the *ICIVA* is not based on the minimality of the Generalised Closed World Assumption and its variants, and because it is restricted (in its original form) to positive programs, it tends to be ignored by the disjunctive logic programming community. However, there are knowledge representation problems where, if disjunctive logic programming is to be used as the representation language, then the *ICWA* does indeed provide the appropriate semantics; those where disjunction should be treated in a causally exclusive fashion , as introduced in section 3.2. Let us revisit the example used there to motivate causally exclusive disjunction, and consider how the Indefinite Closed World Assumption would interpret the corresponding disjunctive logic program.

A contaminated sample in a gas chromatograph will result in a blockage either in the entry nozzle, or in the dispersion tube, or in the detector.
However, it will result in a blockage in only one of these places. Also, a pinch in the dispersion tube can result in it being blocked.

This can be represented as the following disjunctive logic program:

nozzle _blocked ∨ *tube _blocked* ∨ *detector_blocked* ← *contaminated_sample*
tube _blocked ← *pinched _tube*

If this logic program is adopted as a representation of the knowledge described above, then the *ICWA* is indeed the appropriate semantics for disjunction. From the facts *contaminated_sample* and *pinched_tube*, we would wish to consider non-minimal models , such as that containing *tube_blocked* and *detector_blocked*. Such models would be excluded by using a semantics such as the disjunctive stable model semantics. However, we would not want to consider the model containing {*nozzle_blocked, tube_blocked, detector_blocked*}, which the possible model semantics would give.

The *I*-models are a subset of the possible models of a positive disjunctive logic program. Because of this, the same transformation that is used to simulate the possible models semantics abductively can be modified to simulate *I*-models abductively. We consider the example above.

We represent the disjunctive clause as a clause set:

nozzle_blocked ← *contaminated_sample*, α_1^*
tube _blocked ← *contaminated_sample*, α_2^*
detector_blocked ← *contaminated_sample*, α_3^*

where $\alpha_1^*, \alpha_2^*, \alpha_3^* \in A$ are abducibles.

The possible models transformation of section 4.1 would use the integrity constraint $\alpha_1^* \vee \alpha_2^*, \alpha_3^* \leftrightarrow$ *contaminat ed_sample* to ensure that, if the clause body is true in a generalised stable model, the model will also contain at least one abducible.

To simulate *I*-models, we also use this integrity constraint. However, we need to impose additional integrity constraints that at most one abducible can be included in a generalised stable model.

We do by adding the set $\{\neg(\alpha_1^* \wedge \alpha_2^*), \neg(\alpha_1^* \wedge \alpha_3^*), \neg(\alpha_2^* \wedge \alpha_3^*)\}$ to the integrity constraints. This will ensure that any valid set of abducibles will contain exactly one abducible for this clause set if the body of the clause set is true in the corresponding generalised stable model. This corresponds to a member of *ICWA*(*P*), which contains exactly one definite clause as a representative of the disjunctive clause

nozzle_blocked $\vee$ *tube_blocked* $\vee$ *detector_blocked* $\leftarrow$ *contaminated_sample* .

This approach can be generalised to generate an abductive framework for an arbitrary disjunctive logic program, *P*, with generalised stable models of the framework corresponding to *I*-models of the program *P*.

Given a disjunctive logic program, *P*, which is to be interpreted using the Indefinite Closed World Assumption semantics, we define a corresponding abductive framework $< P', A, IC >$ as follows:

- Each non-disjunctive clause in *P* is also in *P'*.

- For each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in *P*, the corresponding clause set

  $a_1 \leftarrow body, \alpha_1^*$
  $\vdots$
  $a_n \leftarrow body, \alpha_n^*$

  is in *P'*.

- We take the lower bound integrity constraints for this clause set to be:

  $\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$

- We take the upper bound integrity constraints to be:

  $body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$
  $\{\neg(\alpha_i^* \wedge \alpha_j^*) \mid \forall i \neq j\}$

We adopt the stable model semantics as our underlying semantics.

The two disjunctive integrity constraints once again can be combined to give:

$$\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$$

Hence, each disjunctive clause is substituted with its corresponding clause set, and integrity constraints which impose a causally exclusive interpretation on the clause set.

Given this transformation, we prove the following:

Theorem 4.3:
Consider a disjunctive logic program, P, with a corresponding abductive framework, $< P', A, IC >$ as defined above.

(i) If M is an I-model of P, then there exists a $\Delta \subseteq A$ s.t. $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$.

(ii) If $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$, then M is an I-model of P.

Hence there is a correspondence between the I-models of P and the generalised stable models of $< P', A, IC >$.

Proof:
(i) We firstly show that, if M is an I-model of P, there exists a set of abducibles, Δ, such that $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$. We do this by selecting a normal logic program, Q, in $ICWA(P)$, with stable model, M, and generating a corresponding set of abducibles, Δ. We then use the lemmas of chapter 2 to transform Q in steps to $P' \cup \Delta$.

As M is an I-model of P, there is a normal logic program, Q, in $ICWA(P)$ s.t. M is a stable model of Q.

From Q and M, we construct Δ:

α_i^* is in Δ iff $a_i \leftarrow body$ is in Q, and $body$ is true in M.

The integrity constraints are true in $M \cup \Delta$:

If *body* in the clause set is false, then Δ contains no abducibles corresponding to the clause set, and hence all integrity constraints, $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$ and $\neg(\alpha_i^* \& \alpha_j^*)$ are true in $M \cup \Delta$.

If *body* is true, then Δ contains the one abducible α_i^* corresponding to the one clause $a_i \leftarrow body$ in Q. Again, the integrity constraints $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$ and $\neg(\alpha_i^* \& \alpha_j^*)$ are true in $M \cup \Delta$.

We form $Q_1 \cup \Delta$ from Q by substituting each clause $a_i \leftarrow body$ such that *body* is true in M with $a_i \leftarrow body, \alpha_i^*$ and adding Δ. By lemma 2.4, $M \cup \Delta$ is a stable model for $Q_1 \cup \Delta$. Next we form $P' \cup \Delta$ from $Q_1 \cup \Delta$ by adding the clauses $a_j \leftarrow body, \alpha_j^*$, for all $\alpha_j^* \in \Delta$. By lemma 2.1, $M \cup \Delta$ is a stable model for $P' \cup \Delta$. As IC are true in $M \cup \Delta$ it follows that $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$.

(ii) Conversely, consider $P' \cup \Delta$ such that $M \cup \Delta$ is a stable model, and *IC* are true in $M \cup \Delta$.

As *IC* are true in $M \cup \Delta$, Δ has at most one α_i^* for each clause set in P'.

Resolve each α_i^* in Δ with $a_j \leftarrow body, \alpha_j^*$ to give Q_1. M is the stable model for Q_1, by lemma 2.4. As the remaining clauses, $a_j \leftarrow body, \alpha_j^*$, have α_j^* false in M, they can be deleted to form Q_2. By lemma 2.1, M is a stable model of Q_2. If a clause set has no $\alpha_i^* \notin \Delta$, then *body* is false in M, as $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$ is true in M. Hence, for each such clause set, add $a_i \leftarrow body$ to Q_2 to form Q. By lemma 2.1, M is a stable model of Q.

As Q has exactly one clause, $a_j \leftarrow body$, for each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$, in P, Q is a member of $ICWA(P)$. Hence, M is an I-model of P, as required.

PROVED

The theorem shows that there is a direct correspondence between generalised stable models of the framework $< P', A, IC >$ and *I*-models of *P*. However, this correspondence is not 1-1. There are cases where one *I*-model has several generalised stable models which correspond to it. To see this, consider the simple example:

$a \vee b \vee c$
a
b

This has two *I*-models, $\{a, b\}$, and $\{a, b, c\}$.

However, consider the corresponding abductive framework:

P':

$a \leftarrow \alpha_1^*$
$b \leftarrow \alpha_2^*$
$c \leftarrow \alpha_3^*$
a
b

A: $\{\alpha_1^*, \alpha_2^*, \alpha_3^*\}$

IC: $\alpha_1^* \vee \alpha_2^* \vee \alpha_3^*, \neg(\alpha_1^* \wedge \alpha_2^*), \neg(\alpha_1^* \wedge \alpha_3^*), \neg(\alpha_2^* \wedge \alpha_3^*)$

This has three generalised stable models: $\{a, b, \alpha_1^*\}$, $\{a, b, \alpha_2^*\}$ and $\{a, b, c, \alpha_3^*\}$. Hence there is not a 1-1 correspondence between *I*-models and generalised stable models.

In the transformation used in section 4.1, to represent the possible model semantics in this framework, we avoid cases like this by adding integrity constraints of the form $\alpha_i^* \leftarrow a_i, body$. In the above example, this would constrain all acceptable sets of abducibles to contain both α_1^* and α_2^*.

However, we cannot apply this technique to this transformation, as it would result in a violation of the integrity constraint $\neg(\alpha_1^* \wedge \alpha_2^*)$ and the framework would therefore have no generalised stable models.

Hence, we have shown that it is possible to represent the Indefinite Closed World Assumption semantics within our framework.

However, unlike the possible model semantics, we have not been able to give a 1-1 correspondence between *I*-models and generalised stable models.

In this chapter, we have shown that the possible model semantics and the Indefinite Closed World Assumption semantics can both be represented within our framework. However, using the equivalent abductive framework is more flexible than using a disjunctive logic program interpreted under one of these semantics; the knowledge engineer has a choice of which semantics to use to interpret the negation. Furthermore, the program can contain other forms of disjunction elsewhere in it; it is not constrained to interpret all disjunctions in the same way. We have also shown that an approximation to the disjunctive stable model semantics can be represented within our framework, but that the correspondence does not hold in general.

Next, we propose a new subclass of disjunctive logic programs; those which are restricted to have at most one atom true in the head of a disjunct. We show that such programs are useful for knowledge representation, and that they can be represented within our framework.

Chapter 5 - Exclusive Disjunctive Logic Programs

In many real-world applications, disjunction is often used to represent several mutually exclusive alternatives. In section 3.2.1, we gave some examples; "A pool ball is spotted, striped, or black". "A loose connection results in no current flowing, or a low current flowing". "A map region can be coloured red, blue or green". Despite the importance of exclusive disjunction to knowledge representation, the currently proposed semantics of disjunctive logic programs cannot adequately handle it. In this chapter, we firstly demonstrate that existing semantics do not adequately represent exclusive disjunction. Next, we define exclusive disjunctive logic programs, which consist of disjunctive logic programs together with appropriate integrity constraints to ensure that the conclusions of each disjunctive clause are mutually exclusive of each other. In section 5.1 we extend the disjunctive stable model semantics, the possible model semantics and the Indefinite Closed World Assumption semantics to apply to exclusive disjunctive logic programs, and show that in this case they are all equivalent. We call this semantics the exclusive disjunctive semantics. We also show that this semantics is equivalent to the representation of exclusive disjunction presented in chapter 3. In section 5.2, we present a transformation which can be used to transform a disjunctive logic program interpreted under the Indefinite Closed World Assumption semantics into an equivalent exclusive disjunctive logic program, and prove that the *I*-models of the original program correspond to the exclusive disjunctive models of the transformed version. In section 5.3, we present a rather more cumbersome transformation, which can be used to transform a disjunctive logic program interpreted under the possible model semantics into an equivalent exclusive disjunctive logic program. Finally, in section 5.4, we show that other forms of indefinite knowledge presented in chapter 3 can also be represented within an exclusive disjunctive logic program.

The Generalised Closed World Assumption [Min 82] and the semantics of disjunctive logic programming which are based on it (such as disjunctive stable models) are often referred to as interpreting disjunction exclusively.

However, as we argued in chapter 1, this approach gives only an approximation to exclusive disjunction.

A program $\{red \vee blue\}$ has two minimal (and hence disjunctive stable) models, $\{red\}$ and $\{blue\}$. This corresponds with an exclusive interpretation of the disjunction. However, if we add the atoms *red* and *blue* to the program, it has the minimal model $\{red, blue\}$. If the disjunction were to be interpreted exclusively, the program should be considered inconsistent.

Decker [Deck 96] gives another example of this. The program $\{p \vee q, q \vee r, r \vee s\}$ has minimal (and hence disjunctive stable) models $\{p, r\}$, $\{q, r\}$ and $\{q, s\}$. The second of these models does not interpret the disjunction $q \vee r$ exclusively, while the other models interpret all three disjunctions as exclusive. Decker concludes that "it is doubtful to speak of an inclusive or exclusive semantics of disjunction at all".

We draw a different conclusion from these examples. It is clearly important from a knowledge representation perspective to be able to represent exclusive information, but existing semantics of disjunctive logic programming do not allow us to do so adequately. We propose instead to combine certain semantics of disjunctive logic programs with appropriate integrity constraints to enforce exclusivity.

Sakama and Inoue [SI 93] have proposed augmenting the possible model semantics with denial integrity constraints for those disjunctions which are to be treated exclusively. Hence, for each clause $a_1 \vee ... \vee a_n \leftarrow body$ to be treated exclusively, the integrity constraints $\{\neg(a_i \wedge a_j) | i \neq j\}$ are used.

However, as we argued in section 3.2.1, this approach is overly strong. Treating the clause $spot(b) \vee stripe(b) \vee black(b) \leftarrow pool_ball(b)$ in this way would prevent $spot(b)$ and $black(b)$ both being true in a model, even if $pool_ball(b)$ were false in that model. We propose instead to use the conditional denial integrity constraints. Hence, for each clause $a_1 \vee ... \vee a_n \leftarrow body$ in a disjunctive logic program to be treated exclusively, we use the integrity constraints $\{\neg(a_i \wedge a_j) \leftarrow body | i \neq j\}$. Let *IC* be the set of all such integrity constraints. We can apply a credulous semantics of disjunctive logic programming to the program, and consider the models in which *IC* are true.

This will interpret the disjunctive clauses with integrity constraints as exclusive disjunction, and the other clauses according to the chosen semantics. As we will discuss in chapter 7, this approach cannot be used with a sceptical semantics.

We now consider the case where all disjunctions in a logic program are interpreted exclusively. We refer to such programs as exclusive disjunctive logic programs.

Definition: An *exclusive disjunctive logic program* $< P, IC_X >$ consists of a disjunctive logic program, P, together with a set of integrity constraints, IC_X, where:

$$IC_X = \left\{ \neg(a_i \wedge a_j) \leftarrow body \middle| a_1 \vee \ldots \vee a_n \leftarrow body \in P, i \neq j \right\}$$

Given a credulous semantics of disjunctive logic programs, *SEM*, we can define the semantics of exclusive disjunctive logic programs as follows:

Definition: An exclusive disjunctive logic program $< P, IC_X >$ has a *SEM* model M iff M is a *SEM* model of P, and IC_X is true in M.

Exclusive disjunctive logic programs are significant for several reasons:

- In many real-world knowledge representation problems, the only disjunctive information which appears is in this exclusive form.
- The semantics of exclusive disjunctive logic programs is more straightforward than that of non-exclusive disjunctive logic programs.
- Because of this, as we will show in section 5.1, they can be represented straightforwardly in an abductive framework.
- Furthermore, as we will show in chapter 6, they can be represented as normal logic programs without explicit disjunction.
- Other forms of disjunctive information can be transformed into exclusive disjunctive form, as we will show in sections 5.2 to 5.4.

5.1 Representing exclusive disjunctive logic programs using abduction.

In section 3.2.1, we showed how an abductive framework can be used to represent knowledge involving exclusive disjunction. We now show that this approach can be used to capture the disjunctive stable model semantics. We show firstly that the disjunctive stable model semantics, when restricted to exclusive disjunctive logic programs, is equivalent to both the possible model semantics and the Indefinite Closed World Assumption semantics. Having done this, we are able to use the results of chapter 4 which map these semantics onto our abductive framework.

Theorem 5.1.1:

Let $< P, IC_x >$ be an exclusive disjunctive logic program. Then

M is a possible model of $< P, IC_x >$

iff M is an I-model of $< P, IC_x >$

iff M is a disjunctive stable model of $< P, IC_x >$.

Proof:

By theorem 4.2.1, if M is a disjunctive stable model of P, it is a minimal possible model of P, and therefore a possible model. Hence it suffices to show that (a) if M is a possible model of $< P, IC_x >$, it is an I-model, and (b) if M is an I-model of $< P, IC_x >$, it is a disjunctive stable model. We show the former first.

(a) If M is a possible model of $< P, IC_x >$, then M is a model for some $Q \in GLP(P)$. We choose a minimal such Q, i.e. we choose Q so that it has no proper subprogram, Q', in $GLP(P)$ such that M is a stable model of Q'.

We show that Q contains exactly one clause, $a_i \leftarrow body$, in the split set of each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$, in P, and therefore $Q \in ICWA(P)$, and so M is an I-model of P.

As Q is in $GLP(P)$, it contains at least one clause in the split set of each disjunctive clause. Assume that for some clause in P, it contains more than one.

Hence $a_i \leftarrow body \in Q$ and $a_j \leftarrow body \in Q$.

If *body* is true in *M*, then a_i and a_j are true in *M*. However, $\neg(a_i \wedge a_j) \leftarrow body$ is in IC_X. As *M* is a possible model of $< P, IC_X >$, IC_X are true in *M*, and hence we have a contradiction.

If *body* is false in *M*, then consider the program Q' formed by removing the clause $a_j \leftarrow body$ from *Q*. As Q' still contains at least one clause in the split set, it follows that $Q' \in GLP(P)$. By lemma 2.1, *M* is a stable model of Q', which contradicts the requirement that *Q* be a minimal program in *GLP*(*P*) with *M* as a stable model.

Hence *Q* contains exactly one clause in the split set of each disjunctive clause in *P*, and so $Q \in ICWA(P)$, and *M* is an *I*-model of *P*. As *M* is a possible model of $< P, IC_X >$, IC_X are true in *M*. Hence *M* is an *I*-model of $< P, IC_X >$ as required.

(b) Next, we show that, if *M* is an *I*-model of $< P, IC_X >$, then *M* is a disjunctive stable model of $< P, IC_X >$.

As *M* is an *I*-model of *P* there exists some normal logic program *Q* in *ICWA*(*P*) such that *M* is a stable model of *Q*. *Q* has exactly one clause, $a_i \leftarrow body$, for each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in *P*. We now prove a lemma showing that *M* is also a disjunctive stable model of $< P, IC_X >$.

Lemma 5.1.2:

Let $< P, IC_X >$ be an exclusive disjunctive logic program. Let *Q* be a program formed from *P* by substituting each clause $a_1 \vee ... \vee a_n \leftarrow body$ with one clause $a_i \leftarrow body$. If *M* is a stable model of *Q*, and *M* satisfies IC_X, then *M* is a disjunctive stable model of $< P, IC_X >$.

Proof of lemma:

We show that M is a disjunctive stable model of P by showing that M is a minimal model of P^M. We do this by firstly showing that $Q^M \Rightarrow P^M$, and so M is a model for P^M, and then by showing that there is no $N \subset M$ which is also a model for P^M.

For any clause $a_1 \vee ... \vee a_n \leftarrow body^+$ appearing in P^M, there is a clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, such that $body^+$ consists of $body$ with all negative literals removed. Every non-disjunctive clause $a \leftarrow body^+$ in P^M also appears in Q^M. For every disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body^+$ in P^M there is a corresponding clause $a_i \leftarrow body^+$ in Q^M. Hence $Q^M \Rightarrow P^M$, and so M is a model of P^M, as it is a minimal model of Q^M.

Now consider $N \subseteq M$ such that N is a minimal model of P^M. We show that N is also a model for Q^M:

N is a model for all the clauses in Q^M which also appear in P^M.

Consider a clause $a_i \leftarrow body^+$ which appears in Q^M but not in P^M. There is a corresponding clause $a_1 \vee ... \vee a_n \leftarrow body^+$ in P^M, by definition of Q.

We show that N is a model for $a_i \leftarrow body^+$. If $body^+$ is false in N, then N is trivially a model. If $body^+$ is true in N, then some $a_j \in \{a_1,...,a_n\}$ must be true in N, as $a_1 \vee ... \vee a_n \leftarrow body^+$ is. As $N \subseteq M$, a_j is also true in M. However, $a_i \leftarrow body^+$ is true in M, and $body^+$ is true in $N \subseteq M$, so a_i is true in M. As IC_X are true in M, there is no a_j, $j \neq i$, true in M. Hence, $a_j = a_i$, and a_i is true in N. So $a_i \leftarrow body^+$ is true in N, as required.

So we have shown that $N \subseteq M$ is a model for Q^M. However, as M is a stable model for Q, it is the minimal model of Q^M. Hence $N = M$, and M is therefore a minimal model of P^M. Hence, M is a disjunctive stable model of P.

As IC_X are true in M, M is a disjunctive stable model of $< P, IC_X >$, as required.

LEMMA PROVED

Hence we have shown that an I-model of $< P, IC_X >$ is also a disjunctive stable model of $< P, IC_X >$, and we are done.

THEOREM PROVED

Because the exclusive disjunctive stable model semantics, the exclusive I-model semantics and the exclusive possible model semantics are all equivalent, we shall refer to this semantics as the *exclusive disjunctive semantics*. Models under this semantics are *exclusive disjunctive models.* We now show that our abductive framework adequately captures this semantics.

Given an exclusive disjunctive logic program, $< P, IC_X >$ which is to be interpreted using the exclusive disjunctive semantics, we define a corresponding abductive framework $< P', A, IC >$ as follows:

- Each non-disjunctive clause in P is also in P'.
- For each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, the corresponding clause set

  $a_1 \leftarrow body, \alpha_1^*$

  $a_n \leftarrow body, \alpha_n^*$

  is in P'.
- We take the lower bound integrity constraints for this clause set to be:

  $\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$
- We take the upper bound integrity constraints to be:

  $body \leftarrow \alpha_1^* \vee ... \vee \alpha_n^*$

  $\{\neg(a_i \wedge a_j) \leftarrow body \mid i \neq j\}$

As before, the two disjunctive integrity constraints can be combined to give:

$$\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$$

Note that the transformation is close to that used in section 4.1 to represent the possible model semantics within the abductive framework. The difference is that this transformation does not have the integrity constraints $\alpha_i^* \leftarrow a_i, body$ but does have the integrity constraints $\neg(a_i \wedge a_j) \leftarrow body$. Let the set of integrity constraints of the form $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$ be IC_a, and the set of integrity constraints of the form $\alpha_i^* \leftarrow a_i, body$ be IC_i. Hence the above transformation uses $IC_a \cup IC_X$, while the transformation used in chapter 4 to represent the possible model semantics in our framework uses $IC_a \cup IC_i$.

Theorem 5.1.3:

Consider an exclusive disjunctive logic program, $< P, IC_X >$, with a corresponding abductive framework, $< P', A, IC >$ as defined above (i.e. $IC = IC_a \cup IC_X$).

(a) If M is an exclusive disjunctive model of $< P, IC_X >$, then there exists Δ such that $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$.

(b) If $M \cup \Delta$ is a generalised stable model of $< P', A, IC >$ then M is an exclusive disjunctive model of P.

Hence there is a correspondence between the exclusive disjunctive models of $< P, IC_X >$ and the generalised stable models of $< P', A, IC >$.

Proof:

M is an exclusive disjunctive model of $< P, IC_X >$ iff it is a possible model of $< P, IC_X >$, iff it is a possible model of P, and IC_X are true in M. We recall theorem 4.1, which gives an abductive characterisation of the possible model semantics:

Given a disjunctive logic program, P, with a corresponding abductive framework, $< P', A, IC_1 >$ where $IC_1 = IC_\alpha \cup IC_i$, then:

(i) If $M \cup \Delta$ is a generalised stable model of the abductive framework $< P', A, IC_1 >$, then M is a possible model of P.

(ii) If M is a possible model of P, then there exists a unique $\Delta \subseteq A$ s.t. $M \cup \Delta$ is a generalised stable model of $< P', A, IC_1 >$.

The exclusive disjunctive models of $< P, IC_X >$ correspond to the generalised stable models of this framework which also satisfy IC_X:

Given an exclusive disjunctive logic program, $< P, IC_X >$, with a corresponding abductive framework, $< P', A, IC_2 >$ where $IC_2 = IC_\alpha \cup IC_i \cup IC_X$, then:

(i) If $M \cup \Delta$ is a generalised stable model of the abductive framework $< P', A, IC_2 >$, then M is an exclusive disjunctive model of $< P, IC_X >$.

(ii) If M is an exclusive disjunctive model of $< P, IC_X >$, then there exists a unique $\Delta \subseteq A$ s.t. $M \cup \Delta$ is a generalised stable model of $< P', A, IC_2 >$.

Hence, it suffices to show that $M \cup \Delta$ is a generalised stable model of $< P', A, IC_\alpha \cup IC_X >$ iff it is a generalised stable model of $< P', A, IC_\alpha \cup IC_X \cup IC_i >$, - i.e. that when integrity constraints IC_X are imposed on the possible model transformation of section 4.1, then the integrity constraints IC_i become redundant.

Let $M \cup \Delta$ be a model of $P' \cup \Delta$ such that $IC_\alpha \cup IC_X$ are true in $M \cup \Delta$. It suffices to show that IC_i are also true in $M \cup \Delta$.

Consider the integrity constraints IC_i associated with the clause set $\{a_i \leftarrow body, \alpha_i^* \mid i = 1..n\}$. They consist of the set $\{\alpha_i^* \leftarrow a_i, body \mid i = 1..n\}$.

If *body* is false in $M \cup \Delta$ then the integrity constraints $\alpha_i^* \leftarrow a_i, body$ are trivially true in $M \cup \Delta$. If *body* is true in $M \cup \Delta$ then some α_i^* must be true in $M \cup \Delta$, as $\alpha_1^* \vee ... \vee \alpha_n^* \leftrightarrow body$ is true in $M \cup \Delta$.

Hence, the corresponding a_i is true in $M \cup \Delta$ and so is the constraint $\alpha_i^* \leftarrow a_i, body$. As $\neg(a_i \wedge a_j) \leftarrow body$ is also true in $M \cup \Delta$ for all $j \neq i$, it follows there can be only one such a_i true. As all a_j, $j \neq i$ are false in $M \cup \Delta$, the constraints $\alpha_j^* \leftarrow a_j, body$ are true in $M \cup \Delta$.

Hence IC_i are true in $M \cup \Delta$, as required.

THEOREM PROVED

We have shown that an exclusive disjunctive logic program can be transformed into an abductive logic program in such a way that the exclusive disjunctive models correspond to the generalised stable models of the abductive framework. Hence exclusive disjunctive information can be adequately represented in our framework.

5.2 Representing the Indefinite Closed World Semantics using Exclusive Disjunctive Logic Programs

Even if we restrict ourselves to using only exclusive disjunctive logic programs, this does not mean that the only form of indefinite information we can represent is exclusive. In this section, we demonstrate that causally exclusive disjunction can be represented, through transformation, by exclusive disjunctive logic programs.

To illustrate, we recall the example of causally exclusive disjunction used in chapter 3:

nozzle_blocked $\vee$ *tube_blocked* $\vee$ *detector_blocked* $\leftarrow$ *contaminated_sample*

Under a causally exclusive interpretation, such as that given by the Indefinite Closed World Assumption, we interpret this clause as meaning that the contaminated sample will result in one of the disjuncts being true, but as not preventing another also being true if there is some other cause for it.

However, intuitively, there is some mutual exclusivity which is taking place. The clause implicitly captures the information that the contaminated sample will either block the nozzle, or the tube, or the detector.

These three events are mutually exclusive of each other. Hence, if they are represented explicitly, they can form an exclusive disjunctive clause:

$$contaminant_blocks_nozzle \vee contaminant_blocks_tube \vee contaminant_blocks_detector \leftarrow contaminated_sample$$

The three disjuncts of the clause are now mutually exclusive, so we can add appropriate integrity constraints:

$$\neg(contaminant_blocks_nozzle \wedge contaminant_blocks_tube) \leftarrow contaminated_sample$$
$$\neg(contaminant_blocks_nozzle \wedge contaminant_blocks_detector) \leftarrow contaminated_sample$$
$$\neg(contaminant_blocks_detector \wedge contaminant_blocks_tube) \leftarrow contaminated_sample$$

We still need to be able to conclude the outcomes of the contaminant blockage in such a way that they are not mutually exclusive of each other. We do this by adding the clauses:

$$nozzle_blocked \leftarrow contaminant_blocks_nozzle$$
$$tube_blocked \leftarrow contaminant_blocks_tube$$
$$detector_blocked \leftarrow contaminant_blocks_detector$$

This transformed version of the original clause faithfully captures its causal exclusivity. Formally, we can say that the *I*-models of the original clause correspond to the exclusive disjunctive models of the transformed version. We demonstrate this correspondence using the example, and will then formally prove that it holds in the general case.

Recall that an *I*-model of a program is found by replacing each disjunctive clause with exactly one of its representatives to form a non-disjunctive program, and considering its stable model.

Consider a disjunctive logic program containing the clause:

$$nozzle_blocked \vee tube_blocked \vee detector_blocked \leftarrow contaminated_sample$$

Let M be an *I*-model of P, and Q be the corresponding normal logic program. Let P' be the transformed program.

We can construct an exclusive disjunctive model, M', of P'. If *contaminated_sample* is false in M, then we need add no additional literals representing this disjunctive clause to M' (though other disjunctive clauses in P may require literals to be added).

If *contaminated_sample* is true in M, then we consider the normal logic program, Q. The disjunctive clause has exactly one representative, *nozzle_blocked* ← *contaminated_sample* say, in Q. P' contains a clause with an identical head, and a newly introduced literal in the body: *nozzle_blocked* ← *contaminant_blocks_nozzle*. We add this new literal to M to form M'. As the original disjunctive clause can only have one representative in Q, only one such literal representing this disjunctive clause will be introduced.

Hence M' will satisfy the exclusivity integrity constraints. We prove below that M' is indeed an exclusive disjunctive model.

Conversely, consider an exclusive disjunctive model M of a program containing the transformed clause:

contaminant_blocks_nozzle ∨ *contaminant_blocks_tube* ∨ *contaminant_blocks_detector* ← *contaminated_sample*

We can construct a corresponding program Q as follows:

(i) If *contaminated_sample* is false in M, then the clause plays no role and we add any one representative, *nozzle_blocked* ← *contaminated_sample* say, to Q.

(ii) If *contaminated_sample* is true in M, then exactly one of the disjuncts in the clause head, *contaminant_blocks_tube* say, will be true in M.
We add to the corresponding clause, *tube_blocked* ← *contaminated_sample*, to Q.

If we remove from M the literals (*contaminant_blocks_x*) added by the transformation, then the remaining set will be a stable model of Q, and therefore an I-model of the original program. We now generalise this, and prove the correspondence formally.

Theorem 5.2:

Given a disjunctive logic program P, we define a new exclusive disjunctive logic program $< P', IC_X >$:

- Each non-disjunctive clause in P is in P'.
- For each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, P' contains the disjunctive clause $\delta_1 \vee ... \vee \delta_n \leftarrow body$ and the non-disjunctive clauses $a_1 \leftarrow \delta_1, ..., a_n \leftarrow \delta_n$ where δ_i are newly introduced atoms. Let D be the set of all newly introduced δ_i.
- IC_X contains the integrity constraints $\{\neg(\delta_i \wedge \delta_j) \leftarrow body \mid i \neq j\}$ for each newly introduced disjunctive clause.

(a) If $M \cup \Gamma$ is an exclusive disjunctive model of $< P', IC_X >$, where $\Gamma \subseteq D$, then M is an I-Model of P.

(b) If M is an I-model of P, then there exists some $\Gamma \subseteq D$ such that $M \cup \Gamma$ is an exclusive disjunctive model of $< P', IC_X >$.

Proof:

Firstly we show that $M \cup \Gamma$ is an exclusive disjunctive model of P' if M is an I-model of P. M is an I-model of P if and only if M is a stable model of some normal logic program, Q_1, in $ICWA(P)$. (Recall that $ICWA(P)$ consists of those normal logic programs which can be generated from P by replacing each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$ with a single clause, $a_i \leftarrow body$). We now transform Q_1 progressively into P', while preserving M as a stable model, by using the lemmas of chapter 2.

Form Q_2 from Q_1 by replacing each clause, $a_i \leftarrow body$ such that *body* is false in M, with the clauses $a_i \leftarrow \delta_i$, $\delta_i \leftarrow body$. As *body* and δ_i are false in M, M is a stable model of Q_2, by lemma 2.1.

Form Q_3 from Q_2 by replacing each clause $a_i \leftarrow body$ such that *body* is true in M with the clauses $a_i \leftarrow \delta_i, \delta_i \leftarrow body$. Let Γ be the set of δ_i appearing in these clauses. It follows from lemma 2.3 that $M \cup \Gamma$ is a stable model of Q_3.

Form Q_4 from Q_3 by adding $a_j \leftarrow \delta_j$ for all a_j such that $a_j \leftarrow body$ does not appear in Q_1. As δ_j is false in $M \cup \Gamma$, then by lemma 2.1, $M \cup \Gamma$ is a stable model of Q_4.

Note that Q_4 is a normal logic program which is very close to the disjunctive logic program P': Q_4 consists of P' with each disjunctive clause $\delta_1 \vee ... \vee \delta_n \leftarrow body$ replaced with some $\delta_i \leftarrow body$. $M \cup \Gamma$ is therefore an exclusive disjunctive model of P', by application of lemma 5.1.2.

Conversely, given an exclusive disjunctive stable model, $M \cup \Gamma$, of the exclusive disjunctive logic program, $< P, IC_x >$, we show that M is an I-model of the original program, P.

Firstly, we generate Q_2 from P' by replacing each disjunctive clause $\delta_1 \vee ... \vee \delta_n \leftarrow body$ with the set of clauses $\{\delta_i \leftarrow body \mid \delta_i \leftarrow body \text{ true in } M \cup \Gamma\}$.
As $M \cup \Gamma$ is an exclusive disjunctive model of P', it is a disjunctive stable model of P' by theorem 5.1.1. We now show that it is also a stable model of Q_1.
$M \cup \Gamma$ is a minimal model of $(P')^{M \cup \Gamma}$. For any clause $a_1 \vee ... \vee a_n \leftarrow body^+$ appearing in $(P')^{M \cup \Gamma}$, there is a clause $a_1 \vee ... \vee a_n \leftarrow body$ in P' such that *body*+ consists of *body* with all negative literals removed. By construction of Q_1, $(Q_1)^{M \cup \Gamma}$ consists of $(P')^{M \cup \Gamma}$ with each disjunctive clause $\delta_1 \vee ... \vee \delta_n \leftarrow body^+$ replaced with the set of clauses $\{\delta_i \leftarrow body^+ \mid \delta_i \leftarrow body^+ \text{ true in } M \cup \Gamma\}$. Hence $(P')^{M \cup \Gamma}$ and $(Q_1)^{M \cup \Gamma}$ satisfy the requirements of lemma 4.2.2, and so we can conclude that $M \cup \Gamma$ is the minimal model of $(Q_1)^{M \cup \Gamma}$, and therefore a stable model of Q_1.

We form Q_2 from Q_1 by deleting all $a_j \leftarrow \delta_j$ such that δ_j is false in $M \cup \Gamma$, and all $\delta_j \leftarrow body$ such that *body* is false in $M \cup \Gamma$. By lemma 2.2, $M \cup \Gamma$ is a stable model of Q_2.

We form Q_3 from Q_2 by replacing all clause pairs $a_i \leftarrow \delta_i, \delta_i \leftarrow body$ such that *body* is true in $M \cup \Gamma$ with $a_i \leftarrow body$. By lemma 2.3, M is a stable model of Q_3. As IC_X are true in $M \cup \Gamma$, there can be at most one such δ_i true in $M \cup \Gamma$. Hence, Q_3 contains one clause, $a_i \leftarrow body$, for each disjunction $a_1 \vee ... \vee a_n \leftarrow body$ in P with *body* true in M.

We form Q from Q_3 by adding the clause $a_i \leftarrow body$ for each disjunction $a_1 \vee ... \vee a_n \leftarrow body$ with *body* false in M. By lemma 2.1, M is a stable model of Q.

As Q consists of P with each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ replaced with a non disjunctive clause, $a_i \leftarrow body$, $Q \in ICWA(P)$. Hence M is an I-model of P, as required.

THEOREM PROVED

Hence, we have shown that a program in which disjunctive clauses are to be interpreted as causally exclusive (using the Indefinite Closed World Assumption semantics) can be transformed into a program in which disjunctive clauses are treated as strictly exclusive.

We believe that it is appropriate to use such a transformation to represent causally exclusive knowledge as true exclusive disjunctions for the following:

- Exclusive disjunction is a more well-defined and clearly understood logical concept than causally exclusive disjunction.

- The transformation is straightforward, and does not introduce significant additional complexity into the transformed program.

- The atoms which are newly introduced by this transformation often have a clear meaning in the context of a concrete example. In the example discussed above, the new atoms could be given names such as '*contaminant_blocks_valve*'. This means that the transformed theory has a structure which is intuitive to a knowledge engineer.

- If we transform causally exclusive disjunctions in this way, then we can represent both causally exclusive and exclusive disjunctive information in the same program.

- If we transform causally exclusive disjunctions into true exclusive disjunctions, then we can further transform them (as we will show in chapter 6) into normal logic programs. Hence, using this transformation, causally exclusive information can be represented using normal logic programs.

5.3 Representing 'inclusive' information using exclusive disjunction

In the previous section, we proposed a transformation which represents causally exclusive disjunction as exclusive disjunction. We proved that the *I*-models of a program are equivalent to the exclusive disjunctive models of the transformed version, and we argued that the transformation is appropriate and helpful from a practical perspective.

In this section, we consider a similar transformation which can be used to represent 'inclusive' disjunction in an exclusive disjunctive logic program. However, unlike the transformation presented in the previous section, we do not believe that this transformation is of practical use to a knowledge engineer.

To illustrate this transformation, we use the example of 'inclusive' disjunctive information given in chapter 1: "If someone works at NASA, then they have been trained as either a rocket scientist or an astronaut, possibly both." In chapter 2, we showed that it is this notion of 'inclusive' disjunction which is captured by the possible world semantics.

In the possible world semantics, this would be represented as the disjunctive statement:

$astronaut \vee rocket_scientist \leftarrow works_at_nasa$

However, if disjunction is to be treated as exclusive, then the statement cannot be represented so straightforwardly. We must separate it out into three, mutually exclusive, cases; if *works_at_nasa* is true, then it either implies *astronaut* but not *rocket_scientist*, or it implies *rocket_scientist* but not *astronaut*, or it implies both *astronaut* and *rocket_scientist*. We introduce three new predicates, *a*, *r* and *both*, representing each of these cases. These predicates are mutually exclusive, so can be represented as an exclusive disjunction:

$a \vee r \vee both \leftarrow works_at_nasa$

In addition, we must add new clauses which allow the conclusion of the original predicates from the newly introduced ones:

astronaut←*a*
rocket_scientist← *r*
astronaut←*both*
rocket_scientist←*both*

Hence, if *works_at_nasa* is true, this clause will have 3 models. In one $\{a, astronaut\}$ will be true, in another, $\{r, rocket_scientist\}$ will be true, and in the third, $\{both, rocket_scientist, astronaut\}$ will be true. These correspond to the possible worlds of the theory.

We now present the general form of the translation. Given a disjunctive logic program, *P*, within which disjunctions are interpreted 'inclusively', we form a corresponding program *P'* as follows:

Each non-disjunctive clause in *P* is also in *P'*.

Consider a disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P. For each non-empty subset S_i of $\{a_1,...,a_n\}$, we introduce a new atom, s_i. The clause $s_1 \vee ... \vee s_m \leftarrow body$ is in P', where $s_1,...,s_m$ are the 2^n-1 newly introduced atoms for this clause.

In addition, for each such S_i, the clauses $\{a_j \leftarrow s_i \mid a_j \in S_i\}$ are in P'.

Hence, the newly introduced atom s_i represents the fact that the original rule $a_1 \vee ... \vee a_n \leftarrow body$ concludes those literals in the set S_i, and no more.

Given this transformation, we conjecture that the possible models of a program P correspond to the exclusive disjunctive models of the transformed program, P'. The proof would be of similar structure to that presented in the previous section. However, because we do not advocate the use of this transformation as a knowledge representation tool, we do not present the proof here.

We do not advocate this transformation for the following reasons:

- If a disjunctive clause has n disjuncts, then the transformed form will have 2^n - 1 disjuncts, and a large number of new clauses will be introduced. Hence, the new theory will be significantly more complex than the original.

- Unlike the transformation in the previous section, the 2^n - 1 new atoms which are introduced do not have a clear meaning to a knowledge engineer. Hence the transformed theory does not have an intuitively clear meaning.

However, having said this, it is a way of representing 'inclusive' disjunctive knowledge in exclusive disjunctive logic programs. Together with the transformation to be presented in the next chapter, it can be used to represent 'inclusive' disjunctive information in normal logic programs.

Hence, though it is a rather low level transformation, it may be appropriate for a knowledge engineer to adopt it if he/she wishes to avoid using disjunctive or abductive logic programs.

5.4 Representing other forms of indefinite information using exclusive disjunction.

In section 3.3, we showed that abduction could be used to represent certain examples using indefinite information which cannot be represented straightforwardly in a disjunctive logic program. Here, we show that it is possible to represent these examples in an exclusive disjunctive logic program, by using a transformation.

In the first example in section 3.3, we considered a variant of the contaminated sample problem introduced in chapter 1. In this variant, we wish to represent the possibility that the contaminated sample may not cause any blockage.

To represent this in an exclusive disjunctive logic program, we use the same approach as in section 5.2, but add an additional disjunct, *contaminant_blocks_nothing*, to the exclusive disjunction. Hence the clauses become:

$$contaminant_blocks_nozzle \veebar contaminant_blocks_tube \veebar contaminant_blocks_detector \veebar contaminant_blocks_nothing \leftarrow contaminated_sample$$
$$nozzle_blocked \leftarrow contaminant_blocks_nozzle$$
$$tube_blocked \leftarrow contaminant_blocks_tube$$
$$detector_blocked \leftarrow contaminant_blocks_detector$$

In addition to the 3 models of the program in section 5.2, this version has the model {*contaminated_sample*, *contaminant_blocks_nothing*}, corresponding to the case where no blockage occurs. Hence, this does indeed capture the required knowledge.

The second example in section 3.3 considered two events with indefinite, but connected, consequences. A bag contains 3 balls, numbered one to three. One is picked out, and not replaced. Then a second is picked out. While each event is disjunctive, the first event has an effect on what can happen in the second.

Again, we can represent this using exclusive disjunction by introducing new atoms to represent mutually exclusive cases. In this case, we introduce atoms to represent that a given pick has selected a given ball. For example, *fp_ball1* represents the case that *ball1* was drawn on the first pick, while *sp_ball1* represents the case that *ball1* was drawn on the second pick. Unlike the previous examples, however, we need to add integrity constraints on these new atoms, to disallow the same ball being drawn twice. For example, $\neg(fp_ball1 \wedge sp_ball1)$ prevents considering that *ball1* was chosen in both the first pick and the second pick. Hence the knowledge can be represented as the following logic program with integrity constraints:

P:

$fp_ball1 \vee fp_ball2 \vee fp_ball3 \leftarrow first_pick$
$sp_ball1 \vee sp_ball2 \vee sp_ball3 \leftarrow second_pick$

$ball1 \leftarrow fp_ball1$

$ball2 \leftarrow fp_ball2$

$ball3 \leftarrow fp_ball3$

$ball1 \leftarrow sp_ball1$

$ball2 \leftarrow sp_ball2$

$ball3 \leftarrow sp_ball3$

IC:

$\neg(fp_ball1 \wedge sp_ball1)$

$\neg(fp_ball2 \wedge sp_ball2)$

$\neg(fp_ball3 \wedge sp_ball3)$

If we add the atoms *first_pick* and *second_pick* to these clauses, then the resulting program has six exclusive disjunctive models which satisfy the additional integrity constraints.

In each model, two out of the three atoms, *ball1*, *ball2* and *ball3*, are true: There are two models for each pair.

For example, the pair $\{ball1, ball2\}$ are true in the following two models:

$\{first_pick, second_pick, fp_ball2, sp_ball2, ball1, ball2\}$

$\{first_pick, second_pick, fp_ball1, sp_ball2, ball1, ball2\}$

The first of these corresponds to the case where *ball1* is chosen on the first pick, and *ball2* on the second, while the second corresponds to *ball2* being chosen first, and *ball1* second.

So again, we can use exclusive disjunctive models, together with appropriate integrity constraints, to represent this example.

So from these examples, we have shown that exclusive disjunctive logic programs can be used to represent a variety of different kinds of indefinite information. In the next chapter, we will show that these exclusive logic programs can be in turn transformed into normal logic programs, allowing the different forms of indefinite information discussed in this chapter to be represented without the use of any form of disjunction or abduction.

Chapter 6 - Using Normal Logic Programs to represent indefinite information.

As we outlined in chapter 1, some researchers have proposed using normal logic programs as a means of representing indefinite information by using the stable model semantics and its extensions. In this approach, indefinite information is represented within a non-stratified logic program. As the program is non-stratified, it may have multiple stable models. These different stable models correspond to the alternatives which are to be considered. The various alternatives are coded via a set of clauses which form an even loop. Hence a disjunction $a \vee b$ is represented by the clauses $\{a \leftarrow \neg b, b \leftarrow \neg a\}$.

As we saw in chapter 1, Saccà and Zaniolo [SZ 90] use this approach to represent the map colouring problem as a normal logic program. They have also applied this approach to the generation of Hamiltonian paths [SZ 97], and Eshghi has applied it to diagnosis [Eshghi 90].

In this chapter, we will consider how this approach to representing indefinite information is related to the disjunctive logic programming approach and to the abductive approach we introduced in chapter 3. In section 6.1, we present a result of Dung [Dung 92], which shows that acyclic disjunctive logic programs can be transformed into normal logic programs, preserving the stable model semantics. In section 6.2, we show that the same transformation can be applied to exclusive disjunctive logic programs, and the stable models of the resulting normal logic program correspond with the exclusive disjunctive models of the original. Finally, in section 6.3, we combine this result with the transformations of the previous chapter. Firstly, we show that disjunctive logic programs interpreted using the Indefinite Closed World Assumption semantics can be transformed into equivalent normal logic programs interpreted under the stable model semantics. Then we show that disjunctive logic programs interpreted using the possible model semantics can be transformed into equivalent normal logic programs interpreted under the stable model semantics.

6.1 Dung's transformation to represent disjunctive logic programs as normal logic programs.

Dung [Dung 92] has shown that the multiple stable model approach can, under certain conditions, be used to simulate the stable model semantics of disjunctive logic programming [Prz 91]. He transforms a disjunctive logic program into a non-stratified normal logic program:

Definition [Dung92]:

Given a disjunctive logic program, P, we define the corresponding normal logic program, $N(P)$, to be the program formed by substituting each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$, in P with the set of normal clauses:

$$a_1 \leftarrow body, \neg a_2, ..., \neg a_n$$
$$\vdots$$
$$a_i \leftarrow body, \neg a_1, ..., \neg a_{i-1}, \neg a_{i+1}, ..., \neg a_n$$
$$\vdots$$
$$a_n \leftarrow body, \neg a_1, ..., \neg a_{n-1}$$

Given this transformation, he has proven that, *if the initial disjunctive logic program is acyclic*[1], then the stable models of the disjunctive logic program, P, and the stable models of the transformed logic program, $N(P)$, coincide.

Unfortunately, this theorem does not hold for cyclic disjunctive logic programs. Dung gives the following simple counter-example:

P is the program:

$a \leftarrow b$
$b \leftarrow a$
$a \vee b$

This program has the unique minimal (and, therefore, stable) model $\{a, b\}$.

---

[1] A disjunctive logic program P is *acyclic* [Dung 92] iff it is possible to decompose the herbrand base of P into disjoint sets, $H_1, ..., H_i, ...$ such that for each clause $a_1 \vee ... \vee a_n \leftarrow b_1, ..., b_m, \neg c_1, ... \neg c_n$,
(i) all a_i belong to the same stratum, H_r say.
(ii) each b_j, c_k belongs to some stratum H_q such that $q<r$.

Applying the transformation, we get $N(P)$:

$a \leftarrow b$
$b \leftarrow a$
$a \leftarrow \neg b$
$b \leftarrow \neg a$

This has no stable model. Hence, Dung's equivalence does not hold in this case.

6.2 Applying Dung's transformation to Exclusive Disjunctive Logic Programs

It is interesting to consider in what other circumstances it is possible to use the transformation that Dung proposes to represent disjunctive logic programs as normal logic programs. As we observed above, Dung's transformation uses the same method for representing indefinite information in normal logic programs as that proposed directly by Saccà and Zaniolo [SZ 90]. If we look at the different problems which have been represented using this approach, we see that they all involve a choice between mutually exclusive alternatives. When applied to map colouring, the different stable models of the program correspond to alternative colourings of each region. When applied to diagnosis, the different stable models correspond to different choices of whether each component is considered to be faulty or functioning correctly. When applied to finding Hamiltonian circuits in networks, each alternative stable model corresponds to a different choice of edge from each node.

In each of these cases, the even loop structure is used to represent alternative, mutually exclusive, cases. However, the even loop does not rigourously enforce this mutual exclusivity, as the following example demonstrates:

$p \leftarrow \neg q$
$q \leftarrow \neg p$
p
q

This has the stable model $\{p, q\}$. If p and q are to represent mutually exclusive cases, such as a region being coloured *red* or *blue*, then such a result is undesirable. To enforce this exclusivity, we need to impose integrity constraints, $\neg(p \wedge q)$, on the theory. Such integrity constraints correspond to those used in the definition of exclusive disjunctive logic programs.

If we can guarantee that no stable model of the theory will violate these constraints, then they can be omitted (for example, if atoms appearing in the even loop do not appear in the head of any other clause in P.)

Instead of using even loops to represent mutually exclusive cases, we could represent the information using an exclusive disjunctive logic program. This suggests that there is a relationship between the even loop approach to representing indefinite knowledge, and exclusive disjunctive logic programs. We now show that this is indeed the case. Any exclusive disjunctive logic program can be transformed, using Dung's transformation, into an equivalent normal logic program with integrity constraints.

Theorem 6.2:

Let $<P, IC>$ be an exclusive disjunctive logic program. There is a 1-1 correspondence between those stable models of $N(P)$ which satisfy the exclusive integrity constraints, IC, and exclusive disjunctive models of $<P, IC>$.

Proof:

Firstly, we show that, if M is an exclusive disjunctive model of $<P, IC>$, then M is a stable model of $N(P)$.

If M is an exclusive disjunctive model of $<P, IC>$, then it is a possible model of P which satisfies IC. As M is a possible model of P, there is some $Q \in GLP(P)$ such that M is a stable model of Q. Using the lemmas of chapter 2, we transform Q into $N(P)$ while preserving M as a stable model.

We form Q_1 from Q by replacing any split set $\{a_p \leftarrow body, ..., a_q \leftarrow body\}$ of a disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ such that $body$ is false with the set of clauses $\{(a_1 \leftarrow body, \neg a_2, ..., \neg a_n) ... (a_n \leftarrow body, \neg a_1, ..., \neg a_{n-1})\}$. By lemma 2.2, M is a stable model of Q_1.

We form Q_2 from Q_1 by substituting any split set $\{a_i \leftarrow body\}$ where $body$ is true in M with the clause $a_i \leftarrow body, \neg a_2, ..., \neg a_{i-1}, \neg a_{i+1}, ..., \neg a_n$.

As *body* is true in M, and M is a stable model of Q, then a_i is true in M. As M satisfies IC, all a_j, $j \neq i$, are false in M. Hence, by lemma 2.5, M is a stable model of Q_2.

We form $N(P)$ from Q_2 by adding all clauses $a_j \leftarrow body, \neg a_1, ..., \neg a_{j-1}, \neg a_{j+1}, ..., \neg a_n$ which do not appear in Q_2 but do appear in $N(P)$. By the construction of Q_2, it follows that *body* is true in M, and there is a split set $\{a_i \leftarrow body\}$ in Q such that $a_i \neq a_j$. Hence a_i is true in M, and therefore $a_j \leftarrow body, \neg a_1, ..., \neg a_{j-1}, \neg a_{j+1}, ..., \neg a_n$ has a false *body* in M. Hence M is a stable model of $N(P)$ by lemma 2.1, and we are done.

Now we show that if M is a stable model of $N(P)$ which satisfies IC, then it is a possible model of P, and therefore an exclusive disjunctive model of $<P, IC>$.

We do this by transforming $N(P)$ into some $Q \in GLP(P)$ while preserving M as a stable model.

We form Q_1 from $N(P)$ by replacing all clauses $a_i \leftarrow body, \neg a_1 \ldots \neg a_{i-1}, \neg a_{i+1}, ..., \neg a_n$ such that $a_1, ..., a_{i-1}, a_{i+1}, ..., a_n$ are false in M with the clause $a_i \leftarrow body$. As M satisfies IC, there is at least one such clause for each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P. By lemma 2.5, M is a stable model of Q_1.

We form Q from Q_1 by deleting all clauses $a_i \leftarrow body, \neg a_1, ..., \neg a_{i-1}, \neg a_{i+1}, ..., \neg a_n$ such that some a_j, $j \neq i$, is true in M. By lemma 2.1, M is a stable model of Q.

Q is identical to P except that each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ has been substituted by the split set

$\{a_i \leftarrow body, \neg a_1 \ldots \neg a_{i-1}, \neg a_{i+1}, ..., \neg a_n \mid a_1 \ldots a_{i-1}, a_{i+1}, ..., a_n$ are false in $M\}$.

Hence $Q \in GLP(P)$. As M is a stable model of Q_2, it is a possible model of P. As M satisfies IC, M is an exclusive disjunctive model of $<P, IC>$, as required.

PROVEN

Hence, this result shows that Dung's transformation can safely be applied to exclusive disjunctive logic programs, to transform them into normal logic programs with integrity constraints. It also shows that we have three different alternatives for the representation of mutually exclusive cases. We can represent them as an exclusive disjunctive logic program, as an abductive logic program, or as a normal logic program which is not stratified. Of these, the abductive logic program is most general. If exclusive disjunctive information is represented in this way, then it does not restrict the choice of semantics of negation, and allows other forms of indefinite information to be represented in the same program. The normal logic program is more restrictive, in that it requires negation to be interpreted using the stable model semantics or one of its variants. It also is not able to represent many other forms of indefinite information. However, it is sometimes appropriate to use normal logic programs rather than abductive logic programs for pragmatic reasons. (e.g. it is easier to get interpreters for normal logic programs). The exclusive disjunctive logic program has the advantage that it is clear and easy to understand. However, it is less general than the abductive approach, and is not well supported practically, as normal logic programs are.

6.3 Representing different forms of exclusive knowledge as Normal Logic Programs.

In the previous section, we showed that an exclusive disjunctive logic program can be transformed into a normal logic program with integrity constraints, and the stable model semantics will be preserved. If we combine this result with the results of chapter 5, we can represent a variety of forms of disjunctive knowledge using normal logic programs.

6.3.1 The Indefinite Closed World Assumption

In section 5.2, theorem 5.2.1, we showed that a disjunctive logic program which is to be interpreted using the Indefinite Closed World Assumption semantics can be transformed into an exclusive disjunctive logic program which is interpreted using the exclusive disjunctive semantics.

Combining this result with theorem 6.2, we get:

Corollary 6.3.1:

If P is a disjunctive logic program to be interpreted using the Indefinite Closed World Assumption semantics, then it can be represented as a normal logic program with integrity constraints, $< N(Q), IC >$, where $< Q, IC >$ is the exclusive disjunctive logic program generated from P using the transformation of theorem 5.2.1, and $N(Q)$ is the normal logic program generated from Q using Dung's transformation. Stable models of the transformed program which satisfy IC correspond to I-models of P.

Proof:

By theorem 5.2.1, the exclusive disjunctive models of $< Q, IC >$ correspond to the I-models of P. Hence, theorem 6.2 can be applied, and stable models of $< N(Q), IC >$ will correspond to I-models of P, as required.

PROVEN

If we apply this approach to the gas chromatograph example used in section 3.2.2, then the clause:

$$nozzle_blocked \vee tube_blocked \vee detector_blocked \leftarrow contaminated_sample$$

is transformed into the normal logic program:

$$contaminant_blocks_detector \leftarrow contaminated_sample, \neg contaminant_blocks_nozzle, \neg contaminant_blocks_tube$$

$$contaminant_blocks_nozzle \leftarrow contaminated_sample, \neg contaminant_blocks_detector, \neg contaminant_blocks_tube$$

$$contaminant_blocks_tube \leftarrow contaminated_sample, \neg contaminant_blocks_nozzle, \neg contaminant_blocks_detector$$

$$nozzle_blocked \leftarrow contaminant_blocks_nozzle$$

$$tube_blocked \leftarrow contaminant_blocks_tube$$

$$detector_blocked \leftarrow contaminant_blocks_detector$$

The stable models of this are identical to the I-models of the original disjunctive logic program.

Technically, according to corollary 6.3.1, we need additional integrity constraints to impose exclusivity on the three newly introduced atoms, *contaminant_blocks_nozzle*, *contaminant_blocks_detector* and *contaminant_blocks_tube*.

However, the transformation from disjunctive logic program to exclusive disjunctive logic program is such that each newly introduced atom appears only once in the program; namely, in the head of the disjunctive clause. As a result of this, all stable models will necessarily satisfy the integrity constraints, and so the integrity constraints can be dropped from the final program.

Hence, in the case of the ICWA, we have 5 alternative representations for a given problem:

1. A disjunctive logic program interpreted using the ICWA semantics.
2. The abductive framework which corresponds to 1, according to theorem 4.3.
3. An exclusive disjunctive logic program which is equivalent to 1, according to theorem 5.2.
4. The abductive framework which corresponds to 4, according to theorem 5.1.3.
5. The normal logic program corresponding to 4, according to corollary 6.3.1.

As with exclusive knowledge discussed in the previous section, the abductive approach has the advantage of being general, while the normal logic programming approach has the advantage of adopting a standard, with the benefits this brings. We do not favour representing causally exclusive knowledge directly as a disjunctive logic program interpreted using the Indefinite Closed World Assumption semantics. This is because the causally exclusive interpretation of disjunction is non-standard, and so some confusion could arise in a person inspecting the program. It is likely to be clearer if the transformed exclusive disjunctive logic program is used instead, though this will depend partly on the domain, and whether the literals introduced by the transformation can be given meaning.

6.3.2 'Inclusive' Disjunction

In a similar way, we can combine the techniques of section 5.3 with theorem 6.2 to represent inclusive information using a normal logic program. For example, consider a disjunctive clause which is to be interpreted 'inclusively':

$$astronaut \vee rocket_engineer \leftarrow works_at_nasa$$

If we apply the transformation described in section 5.3.1, this can be represented as an exclusive disjunctive logic program, as shown in that section. If we then transform this disjunctive logic program into a normal logic program using the techniques described above, then the resulting program will be:

$a \leftarrow works_at_nasa, \neg r, \neg both$
$r \leftarrow works_at_nasa, \neg a, \neg both$
$both \leftarrow works_at_nasa, \neg r, \neg a$

$astronaut \leftarrow a$
$rocket_engineer \leftarrow r$
$astronaut \leftarrow both$
$rocket_engineer \leftarrow both$

If we add the clause *works_at_nasa,* the resulting program has three stable models. In one, *a* and *astronaut* are true. In the second, *r* and *rocket_scientist* are true. In the third, *both*, *astronaut* and *rocket_scientist* are true. Hence, this corresponds to an 'inclusive' interpretation of the original disjunctive clause, and if we remove the extra atoms *a, r* and *both,* the sets remaining correspond to the possible models of the original program.

As in section 6.3.1, we technically need to add integrity constraints to impose mutual exclusivity on the atoms introduced by the transformation. Again, as these atoms only appear once in the exclusive disjunctive logic program, as head of a disjunctive clause, no stable model of the original program will violate the integrity constraints, and hence they are redundant.

As we discussed in chapter 5, we do not advocate the transformation of 'inclusive' disjunction into exclusive disjunction, because of the extra complexity it introduces. However, if a knowledge engineer only has recourse to normal logic programming, then this transformation can be used as a last resort to represent 'inclusive' disjunctive information, if necessary.

In this chapter, we have shown that exclusive disjunctive logic programs can be represented as normal logic programs with integrity constraints, by applying the transformation of Dung.

Using this transformation, it is possible to represent a variety of forms of indefinite information in normal logic programs if the stable model semantics is adopted as the underlying semantics of negation. We showed that causally exclusive disjunction, in the form of the Indefinite Closed World Assumption, can be represented in this way. We also showed that 'inclusive' disjunction can be represented in this way.

This approach can be applied to any form of indefinite information which can be represented within an exclusive disjunctive logic program, such as the examples given in section 5.4. From this, we can conclude that normal logic programs, under the stable model semantics, can represent a variety of different kinds of disjunctive and indefinite information.

Chapter 7 - A Critical Survey of Semantics of Disjunctive Logic Programs

We have shown that abductive logic programs can be used effectively to represent a variety of different kinds of disjunctive and indefinite information. We have also shown that normal logic programs with integrity constraints can be used to represent two forms of such information - exclusive disjunction, and causally exclusive disjunction - if the stable model semantics is adopted as the underlying semantics. Given these results, does disjunctive logic programming have any role to play as a practical knowledge representation tool?

Proponents of disjunctive logic programming argue that the variety of semantics which have been proposed provide appropriate tools for representing different kinds of disjunctive information. In particular, they have proposed that semantics which extend the Generalised Closed World Assumption are useful for representing exclusive disjunction, and those which extend the weak Generalised Closed World Assumption are useful for representing 'inclusive' disjunction. As we argued in chapter 3, we believe that defining different semantics to represent different kinds of indefinite information is a flawed approach; it means it is not possible to represent more than one kind of indefinite information within a single program, and also means that how the indefinite information is interpreted is implicit in the choice of semantics, rather than explicit in the program itself. However, if we accept this approach, then we would expect that the various semantics of disjunctive logic programming proposed would be able to capture at least some of the different kinds of indefinite information described in chapter 3.

We will look at a variety of different semantics which have been proposed for disjunctive logic programming, and assess their effectiveness in representing knowledge. In general, when researchers propose a new semantics, they compare it with others by analysing how it treats certain abstract examples.

An alternative approach has been taken by Dix ([Dix92],[DM94]) who has studied various abstract properties semantics can have, and used these to categorise different semantics and propose new ones. However, very little work has taken place which compares different semantics as practical tools for knowledge representation. It is this approach which we will take here.

We will look at how well each semantics can represent the different kinds of disjunctive and indefinite information introduced in chapter 3, using the corresponding examples from chapter 1. Here we recall these categories, and the corresponding examples:

1. Exclusive disjunction: A region on a map is coloured red or blue, but cannot be coloured both.
2. 'Inclusive' disjunction: An employee of NASA is trained as an astronaut or a rocket scientist, possibly both.
3. Causally exclusive disjunction: A contaminated sample will result in a blockage of the inlet valve, the dispersion tube, or the detector. It will result in only one of these, but something else could cause another blockage.
4. Causal information: The symptom of facial spots can be caused by either acne or measles. We want to assume that a minimal set of causes is true.
5. 'May cause' links: An electrical device contains a bulb. A short circuit may cause the bulb to blow.

In addition to these categories, we will look at two examples of other forms of indefinite information introduced in section 3.3:

6. A contaminated sample may result in a blockage in the inlet tube, the dispersion column or the detector. It can result in at most one of these, and possibly none of them.
7. A bag contains balls numbered one to three. One ball is drawn out and not replaced. A second ball is also drawn. We wish to reason about the numbers appearing on the balls which have been chosen.

These examples give a rich variety of different kinds of disjunctive and indefinite information for assessing the different semantics.

Recall from chapter 3 that we have already shown that abductive logic programs with integrity constraints are able to capture all these forms of indefinite knowledge. In chapter 5, we showed that exclusive disjunction, and normal logic programs interpreted using stable models, are able to capture several of them.

When assessing the capabilities of the different semantics, we will use three approaches:

- How well are they able to represent the information directly exploiting the properties of the semantics?
- How well are they able to simulate exclusive disjunction , and so are able to use the techniques of chapter 5 to represent causally exclusive and 'inclusive' disjunction indirectly?
- How well are they able to simulate abduction, and so are able to use the techniques of chapter 3 to represent different forms of indefinite information?

In section 7.1, we consider disjunctive stable models, and show that they are unable to represent 'inclusive' and causally exclusive disjunction directly, but can represent exclusive disjunction when augmented with appropriate integrity constraints. We show that they can simulate abduction in two ways. In section 7.2 we consider the possible model semantics and show that, when augmented with appropriate integrity constraints, it can represent exclusive and 'inclusive' disjunction in the same program. It cannot represent causally exclusive disjunction, or disjunction with minimised positive consequences. It too can simulate abduction in two ways. In section 7.3, we consider the Indefinite Closed World Assumption semantics, which is able to represent causally exclusive disjunction directly. It can represent exclusive disjunction when augmented with appropriate integrity constraints, but cannot represent 'inclusive' disjunction or disjunction with minimised positive consequences. In section 7.4 we discuss several semantics which extend the Generalised Closed World Assumption, in particular the disjunctive well founded semantics. We show that such semantics cannot represent 'inclusive', exclusive, or causally exclusive disjunction effectively.

In section 7.5, we discuss the generalised disjunctive well-founded semantics, and show that it has the counter-intuitive property of swapping from an exclusive to an 'inclusive' interpretation of disjunction. In section 7.6, we discuss extensions of the weak Generalised Closed World Assumption, in particular the weak disjunctive well founded semantics and the WGCWA¬. We show that neither of them are able to represent exclusive or 'inclusive' disjunction.

7.1 Disjunctive Stable Models

The definition of disjunctive stable models is given in section 4.2. Disjunctive stable models are one of the semantics which are considered to give an 'exclusive' interpretation of disjunction, and so we would expect it to be effective at representing exclusive disjunction as required for example 1.

However, as we discussed in chapter 5, disjunctive stable models only provide an approximation to exclusivity. If we represent example 1 with the disjunctive clause $red \vee blue \leftarrow map_region$ and assert that *map_region* is true, then the disjunctive stable models will be $\{map_region, red\}$ and $\{map_region, blue\}$, corresponding to the two alternative colourings. This is what we would expect from an exclusive interpretation.

However, if we assert the facts *red* and *blue*, then the unique model of the program would be $\{map_region, red, blue\}$. In a true exclusive interpretation, the program should be inconsistent. Hence the disjunctive stable model semantics, as originally defined, does not represent exclusive disjunctive information.

There are two ways in which we can modify the disjunctive stable model semantics in such a way that it does faithfully capture exclusive disjunctivity.

Firstly, we can place a restriction on the program, P, that at most one atom appearing in the head of a disjunctive clause may appear in the head of another clause in P. This restriction ensures that any minimal model, and therefore any disjunctive stable model, of P will have at most one literal true in any disjunction.

Secondly, we can augment the disjunctive stable model semantics with denial integrity constraints using the approach described in chapter 5 to define exclusive disjunctive logic programs. It makes sense to add denial integrity constraints to a disjunctive logic program, because these merely filter models that do not satisfy these constraints. It would not make sense to add disjunctive integrity constraints, as in the case of abductive logic programming.

In the case described above, adding the integrity constraint $\neg(red \wedge blue) \leftarrow map_region$ would ensure an exclusive interpretation.

Hence, disjunctive stable models must be augmented in some way if they are to represent exclusive disjunction. Without this augmentation, they are too weak; they permit models which an exclusive interpretation of disjunctions would not.

'Inclusive' disjunction and causally exclusive disjunction also can't be represented directly by the disjunctive stable model semantics, but for the opposite reason. The disjunctive stable models semantics is too strong; it does not permit certain models which the 'inclusive' and causally exclusive interpretations of disjunction would permit.

To see this, consider example 2. The statement 'an employee of NASA is trained as an astronaut or a rocket scientist, possibly both' can be represented as the disjunctive clause $astronaut \vee rocket_scientist \leftarrow works_at_nasa$. The disjunction in this clause is considered to be 'inclusive'. However, if we assert *works_at_nasa*, and consider the disjunctive stable models, we get $\{rocket_scientist, works_at_nasa\}$, and $\{astronaut, works_at_nasa\}$. An 'inclusive' interpretation would permit these models, but would also permit the model $\{astronaut, rocket_scientist, works_at_nasa\}$, which is not a disjunctive stable model. Hence, disjunctive stable models do not capture 'inclusive' disjunction.

Because disjunctive stable models are weaker than an exclusive interpretation of disjunction, we were able to use denial integrity constraints to filter out the models which were not required.

However, disjunctive stable models are stronger than an 'inclusive' interpretation of disjunction, so this approach cannot be used; we cannot create new models with integrity constraints, only filter existing ones. Hence, the disjunctive stable model semantics cannot be used to directly represent 'inclusive' disjunction.

Similarly, we can show that disjunctive stable models miss models which would be acceptable under a causally exclusive interpretation.

Consider the program:

valve_blocked ∨ *tube_blocked* ← *contaminated_sample*
valve_blocked ← *perished_washer*
contaminated_sample
perished_washer

This is an adaptation of example 3. The disjunction is being used to represent the statement; 'A contaminated sample will result in a blockage in the valve or the tube, but not both'. However, something else could cause another blockage. Hence, it is to be interpreted as causally exclusive. Under this interpretation, it should have two models:
{*contaminated_sample*, *perished_washer*, *valve_blocked*} and
{*contaminated_sample*, *perished_washer*, *valve_blocked*, *tube_blocked*}.

The second of these is not minimal, so is not a disjunctive stable model. Again, integrity constraints cannot help, as the disjunctive stable model semantics is too strong. Hence, disjunctive stable models can't represent causal exclusivity.

The disjunctive stable model semantics can be used to represent example 4, as it requires minimisation of disjuncts. If the information is represented with the disjunctive clause *acne* ∨ *measles* ← *spots*, and *spots* is asserted, the stable models will correspond to the minimal sets of causes; {*measles*} and {*acne*}. If other causes are asserted which require both *measles* and *acne* to be true, then the model will allow this without inconsistency, as is required.

However, this approach does not model the true underlying causality. Rather than representing information in the form $cause \rightarrow effect$, it represents it in a heuristic form; $effect \rightarrow cause1 \vee cause2$. As a result, it has the disadvantage that it is not modular. If we wish to add a new piece of knowledge that spots can also be caused by an allergy, we will need to replace the clause $acne \vee measles \leftarrow spots$ with the new clause $acne \vee measles \vee allergy \leftarrow spots$. If the information had been in the form $cause \rightarrow effect$, it would have simply required the addition of a new clause, $spots \leftarrow allergy$. As a result of this, this form of knowledge representation is harder to maintain.

Interestingly, if we represent the knowledge in the form of a normal logic program containing clauses of the form $effect \rightarrow cause$, then the equivalent heuristic disjunctions, $cause1 \vee cause2 \leftarrow effect$, are the only-if part of the completion of this program. Some researchers use examples of this form to motivate disjunctive logic programming, when in truth they can already be represented by the only if half of a normal logic program. It is quite reasonable to reason with such disjunctions, as the algorithm of [FK 97] does. However, it is not necessary to represent them explicitly in a disjunctive logic program.

Hence, the disjunctive stable model semantics does not fare very well as a tool for representing the kinds of disjunctive knowledge we have identified. Moreover, the only form it represents directly, minimisation of alternative causes, is already better represented implicitly by the completion of a normal logic program. It can be modified to represent exclusive disjunction, but is unable to represent 'inclusive' and causally exclusive disjunction.

It is, however, possible to represent causally exclusive information indirectly. This is done by using the transformation of chapter 5 to convert a causally exclusive disjunction to a truly exclusive one. Once it is in this form, it can be interpreted using the disjunctive stable model semantics.

May-cause links such as example 5 can be represented using disjunctive stable models by adding a disjunct representing the fact that the event does not occur. Hence, example 5 can be represented as $bulb_blows \vee bulb_doesn't_blow \leftarrow short_circuit$.

This disjunction should be interpreted as exclusive, so we need to either impose the integrity constraint $\neg(bulb_blows \wedge bulb_doesn't_blow)$ on the models, or restrict the program containing this clause so that only one of the literals *bulb_blows* or *bulb_doesn't_blow* appears in any other clause head.

An alternative approach to representing indefinite information using the disjunctive stable model semantics is to use the semantics to simulate abduction, and then represent indefinite information using the abductive approach of chapter 3.

Abduction can be simulated in two ways. Firstly, we can adopt the approach of [IS 92] and [Inoue 91] and exploit the stable model semantics to represent abduction. To do this, for each abducible a, we introduce a new literal $\overline{a}$, and add an even loop:

$a \leftarrow \neg\overline{a}$

$\overline{a} \leftarrow \neg a$

In any stable model of the resulting program, either $\overline{a}$ or a will be true, corresponding to a being abduced or not.

Alternatively, we can simulate abduction by using disjunction. Again, we introduce a new literal $\overline{a}$ for each abducible a. However, instead of the even loop, we add the disjunct $a \vee \overline{a}$ to the program. As $\overline{a}$ is a new literal, it does not appear in the head of any other clause in the program, and so this disjunct will be interpreted exclusively by the disjunctive stable model semantics. Hence, any model of the program will either contain a or $\overline{a}$, corresponding to a being abduced or not.

Hence, disjunctive stable models can be used to represent knowledge using the approaches in chapter 3, if we augment them with appropriate integrity constraints. This is effectively using disjunction to simulate abduction, rather than using disjunction to represent disjunctive information, so is a bit of a cheat.

Furthermore, it forces a choice of semantics of negation on the knowledge engineer, whereas the abductive approach allows this choice to be made separately.

If we do not use disjunction to simulate abduction, then the disjunctive stable model semantics can be used only to represent exclusive disjunction, and even then only if augmented with integrity constraints. Hence, the disjunctive stable model semantics is not a very effective tool for representing the different forms of disjunctive and indefinite information we have identified.

7.2 Possible Model Semantics

The possible model semantics [Sak 89], [SI 92] has been defined and discussed in chapter 4. It allows the disjuncts in a rule's conclusion to range freely, so is an effective semantics for representing 'inclusive' disjunction such as that in example 2.

As we discussed in chapter 5, it can be augmented with denial integrity constraints to enforce exclusivity on the conclusions of disjunctive clauses, and so can represent exclusive disjunction. As Sakama and Inoue observe [SI 93] this means it can be used to represent both exclusive and 'inclusive' disjunction in the same program.

However, the possible model semantics, with or without integrity constraints, is not able to represent causally exclusive disjunction directly, as we now demonstrate. Consider the program:

$valve_blocked \vee tube_blocked \leftarrow contaminated_sample$
$valve_blocked \leftarrow perished_washer$
$contaminated_sample$

A causally exclusive interpretation of the disjunction would expect two models; $\{contaminated_sample, valve_blocked\}$ and $\{contaminated_sample, tube_blocked\}$. If we use the possible model semantics to interpret the program, we get an additional model, $\{contaminated_sample, valve_blocked, tube_blocked\}$. Hence, the possible model semantics is too weak to represent causally exclusive disjunction.

We were able to represent exclusive disjunction by restricting the possible model semantics with integrity constraints. However, this approach cannot be used to represent causally exclusive disjunction. If we add the fact *perished_valve* to the above program, then a causally exclusive interpretation would permit a model in which both *valve_blocked* and *tube_blocked* are true. Hence, an integrity constraint which simply constrains atoms in the head of the clause based on the truth value of the body cannot be adequate, as changes in the program which do not affect the truth value of the body of the clause can still have an effect on acceptable truth values of atoms in the head.

The possible model semantics is also unable to represent causal information in such a way that causes are minimised, as the disjunctive stable model semantics does. If we consider the heuristic representation of example 4 used in the previous section, $\{ acne \vee measles \leftarrow spots \}$, then from the fact *spots* we can conclude that both *acne* and *measles* could be true. Such a conclusion is non-parsimonious; it should only be adopted if there is evidence that cannot be explained by a subset of causes.

In this case, if we consider only the minimal possible models, then the model in which *acne* and *measles* are both true will be eliminated, as required. However, in general, this approach does not give the required form of minimisation. If we add clauses $a \leftarrow \neg acne$, $a \leftarrow \neg measles$ then the possible models become $\{spots, acne, a\}$, $\{spots, measles, a\}$, and $\{spots, acne, measles\}$. The latter is now a minimal possible model, but does not contain a minimal set of disjuncts, as we require.

Hence, possible models are able to represent 'inclusive' disjunction, and can represent 'exclusive' disjunction by using integrity constraints. However, they cannot represent causal exclusivity, and cannot represent disjunctions where positive consequences are minimised. We have argued in section 7.1 that minimising positive consequences of disjunctions is not practically motivated, as the motivating examples are better understood as the only-if part of the completion of a normal logic program. Hence, the inability of the semantics to capture this is not really a flaw.

Causally exclusive information cannot be represented directly in the possible model semantics. However, we can represent it indirectly by transforming it into a true exclusive disjunction, using the transformation introduced in chapter 5, and representing the exclusive disjunction using the possible model semantics augmented by appropriate integrity constraints, as discussed above. We can also represent examples 6 and 7 in this way.

May-cause links can be represented using the same approach as used for disjunctive stable models. This time, it is necessary to add appropriate integrity constraints to enforce mutual exclusivity. Example 5 can therefore be represented by the disjunctive clause $bulb_blows \vee bulb_doesn't_blow \leftarrow short_circuit$ and the integrity constraint $\neg(bulb_blows \wedge bulb_doesn't_blow)$.

If the integrity constraint isn't added, there will be three models if *short_circuit* is added as a fact. These are:

$\{short_circuit, bulb_blows\}$

$\{short_circuit, bulb_doesn't_blow\}$

$\{short_circuit, bulb_blows, bulb_doesn't_blow\}$

Clearly, the last is undesirable, so we use the integrity constraint to eliminate it.

If we wish to simulate abduction in a disjunctive program interpreted under the possible model semantics, then we have two alternative approaches, similar to those used in section 7.1.

Firstly, as in 7.1, we can simply represent abduction using an even loop, exploiting the stable model semantics underlying the possible model semantics.

Alternatively, we can use disjunction, as proposed by [SI 94]. We add the disjunct $a \vee \sigma$ for any abducible, a. In any model, either a will be true, or σ will be true, or both a and σ will be true.

Hence, there will be one model corresponding to the case where a is not abduced, and two corresponding to the case where it is abduced. Sakama and Inoue considered it acceptable to have this redundant model in which both a and σ are true. However, if we wish to eliminate this, we simply add the integrity constraint $\neg(a \wedge \sigma)$.

Hence, the possible model semantics is a reasonable semantics for representing indefinite information. Its strength is that it is able to directly represent both 'inclusive' and exclusive disjunctive information in the same program. As these two forms of disjunction are the most practically useful, this means it is a more practical semantics than the disjunctive stable model semantics.

7.3 The Indefinite Closed World Assumption Semantics

The Indefinite Closed World Assumption (ICWA) semantics [KWC 94] is a variant of the possible model semantics. The definition of this semantics is given in section 4.3. As we discussed in chapter 4, it treats disjunction as causally exclusive, so can represent example 3 directly. Furthermore, as we discussed in chapter 5, it can be augmented with integrity constraints to allow it to represent exclusive disjunction. Hence, it is particularly suited for representing information in which examples requiring both causally exclusive disjunction and exclusive disjunction occur.

However, it is too strong to represent 'inclusive' disjunction directly. It can only represent 'inclusive' disjunction by using the rather clumsy transformation of chapter 5. It is able to represent abduction in the same way as disjunctive stable models, as described in section 7.1.

Hence, the weakness of the ICWA semantics is its inability to represent 'inclusive' information without resorting to the transformation of chapter 5. Its strength is its ability to represent causally exclusive information directly. However, causally exclusive disjunction is not so common and can be transformed, using the straightforward transformation of section 5.2, into exclusive disjunction.

For these reasons, the ICWA semantics is less practically useful than the possible model semantics.

7.4 Extensions Of The Generalised Closed World Assumption

So far, we have focused on credulous semantics of disjunctive logic programs. However, the majority of work carried out on semantics of disjunctive logic programs has concentrated on sceptical semantics. We now turn our attention to these.

Several semantics have been proposed which simultaneously extend the well-founded semantics of normal logic programs, and the Generalised Closed World Assumption of positive disjunctive logic programs: The strong well-founded semantics of Ross [Ross 90], the D-WFS of Brass and Dix [BD 95], and the static [Prz 95] and stationary [Prz 90] semantics of Przymusynski. As all these share certain characteristics, we will focus primarily on the D-WFS as an example of this family, pointing out differences with other semantics where appropriate.

Brass and Dix [BD 95] present both an abstract and a constructive definition of the D-WFS, and show that they are equivalent. Here, we describe the intuitions behind the constructive form.

The approach they take is to transform a disjunctive logic program, P, into one consisting of only conditional facts ([DK 89], [Bry 89]). Conditional facts are clauses which contain only negative literals in their body. They do this by associating an immediate consequence operator T_p with the program P. This operator is a generalisation of that of Van Emden and Kowalski [VEK 76]. It operates on a set of conditional facts, rather than atoms, and allows clauses to have disjunctive heads. It uses hyper-resolution to conclude the immediate consequences of a set of conditional facts. Brass and Dix generate the set of conditional facts corresponding to P by taking the smallest fixed point of this operator.

For example, consider the program:

$a \vee b \leftarrow \neg c$
$d \leftarrow a$
$d \leftarrow b$

$\Gamma_1 = T_p(\varnothing)$ is the set of conditional facts in the program; in this case, $\{a \vee b \leftarrow \neg c\}$.

$\Gamma_2 = T_p(\Gamma_1)$ is the set of immediate consequences of this conditional fact, under hyper-resolution: $\{d \vee b \leftarrow \neg c,\ a \vee d \leftarrow \neg c\}$.

$\Gamma_3 = T_p(\Gamma_2)$ gives $\{d \leftarrow \neg c\}$. Further iterations generate no new conditional facts.

Hence the transformed program is:

$a \vee b \leftarrow \neg c$
$d \vee b \leftarrow \neg c$
$a \vee d \leftarrow \neg c$
$d \leftarrow \neg c$

After transforming the program to a set of conditional facts, Brass and Dix then iteratively reduce this program. Each reduction consists of two stages:

(i) They remove all conditional facts which are known false, and those which are subsumed by others.

A conditional fact $a_1 \vee ... \vee a_n \leftarrow \neg b_1,...,\neg b_m$ is known false if there is some conditional fact $b_i \vee ... \vee b_j \leftarrow true$ in the program such that $b_i \vee ... \vee b_j$ is a sub-disjunction of $b_1 \vee ... \vee b_m$.

A conditional fact $a_1 \vee ... \vee a_n \leftarrow \neg b_1,...,\neg b_m$ is considered subsumed by a conditional fact $a_p \vee ... \vee a_q \leftarrow \neg b_r,...,\neg b_s$ if $\{a_p,...,a_q\}$ is a subset of $\{a_1,...,a_n\}$, and $\{b_r,...,b_s\}$ is a subset of $\{b_1,...,b_m\}$ and at least one of them is a proper subset.

(ii) They delete any negative literal, $\neg a$, from the body of all conditional facts if a no longer appears in the head of any conditional fact.

When this iteration terminates, the program that remains is called the residual program.

In the example above, stage (i) of the reduction reduces the transformed program to:

$a \vee b \leftarrow \neg c$
$d \leftarrow \neg c$

The other two conditional facts are subsumed by $d \leftarrow \neg c$, so are deleted.

In stage (ii) of the reduction, $\neg c$ is deleted from the body of all clauses, as it does not appear in the head of any clause. Hence, the residual program is:

$a \vee b$
d

According to the D-WFS, a disjunction $a_1 \vee ... \vee a_n$ is true if there is a conditional fact $a_i \vee ... \vee a_j \leftarrow true$ in the residual program such that $a_i \vee ... \vee a_j$ is a sub-disjunction of $a_1 \vee ... \vee a_n$. An atom b is false if b does not appear in the head of any conditional fact in the residual program. A term which cannot be assigned a truth value is considered unknown.

Hence, in the above example, c is false and d is true. a and b are unknown, but the disjunction $a \vee b$ is true. Note that a disjunction may be assigned true by the D-WFS, while each of its disjuncts is assigned unknown. Because of this property, the D-WFS is not a model theoretic semantics.

Semantics such as the D-WFS, which extend the Generalised Closed World Assumption, are often said to interpret disjunction exclusively. By attempting to represent a simplified form of example 1 using the D-WFS, we show that this is not the case. Like disjunctive stable models, they actually minimise truth in a disjunction, rather than require exclusivity. To demonstrate this, we will consider applying the D-WFS firstly to an exclusive disjunction, and then to an 'inclusive' disjunction.

We represent the fact that a map region is coloured red or blue with the disjunction $red \vee blue$. As the map region is to only have one colour, we wish to interpret this disjunction exclusively. Consider the program:

$red \vee blue$
$two_coloured \leftarrow red, blue$

Under an exclusive interpretation of disjunction, we would expect *two_coloured* to be false.
If we apply the reductions of Brass and Dix to this, we are left with the residual program:

$red \vee blue$

This is the only conditional fact appearing in the original program, and it has no immediate consequences. Hence, it is the residual program.

Hence, the D-WFS will conclude that *two_coloured* is false, that $red \vee blue$ is true, and that *red* and *blue* are each unknown. This coincides with an exclusive interpretation of the disjunction. However, if we add the atoms *red* and *blue* to the program, the residual program becomes:

red
blue
two_coloured

two_coloured is an immediate consequence of *red* and *blue*, so now appears as a fact in the residual program. The disjunction $red \vee blue$ no longer appears, as it is eliminated by stage (i) of the reduction process.

two_coloured is true and the program remains consistent. If the disjunction $red \vee blue$ is interpreted as exclusive, we would expect the program to become inconsistent when *red* and *blue* are added.

This problem can be overcome using an approach similar to that we used for disjunctive stable models. We can restrict the program so that at most one atom appearing in the head of a given disjunctive clause can appear in the head of any other clause. Alternatively, we can augment the D-WFS with appropriate denial integrity constraints. However, as the D-WFS is not a model theoretic semantics, it is not straightforward how the integrity constraints would be applied.

This semantics is also unable to conclude positive information from a disjunction of negative information. Consider the following:

$red \vee blue$
$not_two_coloured \leftarrow \neg red$
$not_two_coloured \leftarrow \neg blue$

An exclusive sceptical interpretation of $red \vee blue$ should be able to conclude that either *red* is false or *blue* is false, and hence *not_two_coloured* is true. However, of the semantics being discussed, only the stationary semantics can conclude this. All the others conclude that *not_two_coloured* is unknown. (Such a conclusion is compatible with an 'inclusive' interpretation of disjunction, but as we will show later, the D-WFS does not consistently interpret disjunction as 'inclusive' either.)

Adding a denial integrity constraint, $\neg red \vee \neg blue$, does not help in this situation. Firstly, as we discussed above, it is not straightforward how integrity constraints would be applied to the D-WFS. Secondly, integrity constraints act as a filter on the semantics, to determine if a truth assignment is permissable or not. They cannot change the fact that *red* and *blue* are both unknown under the D-WFS, and hence *not_two_coloured* remains unknown. Only if we treat $\neg red \vee \neg blue$ as a statement implicit within the theory, as the stationary semantics does, can we conclude that *not_two_coloured* is true.

Hence, the D-WFS (and the other extensions of the Generalised Closed World Assumption and the well founded semantics) cannot represent exclusive disjunction adequately, even with the addition of appropriate denial integrity constraints.

The exception to this is the stationary semantics, which can if appropriate integrity constraints are used.

These semantics cannot be used to represent 'inclusive' or causally exclusive disjunction for similar reasons to those given for disjunctive stable models in section 7.1. We illustrate this by applying the D-WFS to a problem requiring 'inclusive' disjunction. Consider the program:

astronaut ∨ *rocket_scientist* ← *works_at_nasa*
well_paid ← *astronaut*, *rocket_scientist*
works_at_nasa

The residual program of this is:

astronaut ∨ *rocket_scientist*
works_at_nasa

Hence *well_paid* is false under the D-WFS, rather than unknown, as an 'inclusive' interpretation of disjunction would require. Integrity constraints cannot help here, as they can only filter undesirable models, not generate new ones.

The D-WFS, like the disjunctive stable model semantics, maximises the set of atoms which are considered false and minimises those considered true or unknown. For this reason, we would expect that it could represent the causal information of example 4, using statements of the form *cause*1 ∨ *cause*2 ← *effect* as described in section 7.1. However, it has the disadvantage that, being a sceptical semantics, it loses certain important information.

To see this, consider the following example:

acne ∨ *measles* ← *spots*
measles ∨ *flu* ← *temperature*
spots
temperature

This program has two disjunctive stable models. In one, *measles* is true, while *acne* and *flu* are false. In the other, *measles* is false, while *acne* and *flu* are true.

The fact that *acne* and *flu* must be true at the same time, and that *measles* and *acne* are not true at the same time, is contained explicitly in the models. However, the D-WFS simply concludes that *acne*, *measles* and *flu* are all unknown. The additional information about which causes must occur together, and which do not, is not explicit. Hence it is less effective at representing causal information than the disjunctive stable model semantics.

It is possible to extract out some of this information by finding which disjunctions are true under the D-WFS. In the above example, $acne \vee measles$ and $measles \vee flu$ are true, and so we can determine that either *measles* occurs, or both *acne* and *flu* occur. However, we can not make similar statements about what causes do not occur together. The stationary and static semantics overcome this to a certain extent, by using a *state pair*. These semantics assign a state pair, $< T, F >$, to a disjunctive logic program where T is a set of atoms and disjunctions which are considered true, while F is a set of atoms and conjunctions which are considered false.

For the above program, the stationary semantics would assign the state pair $< T, F >$, where T is the set $\{spots, temperature, measles \vee flu, acne \vee measles\}$ and F is the set $\{\neg(measles \wedge flu), \neg(acne \wedge measles)\}$.

Hence, rather than presenting alternative models, as a credulous semantics would, a state-pair based sceptical semantics gives a series of disjunctive and denial statements which would be true of all these alternative models.

The D-WFS is able to represent example 6, using the same approach as used in previous sections. Consider the program:

dark_room ← *bulb_blows*
bulb_blows ∨ *bulb_doesn't_blow* ← *short_circuit* .
short_circuit .

The residual program is:

dark_room ∨ *bulb_doesn't_blow*
bulb_blows ∨ *bulb_doesn't_blow*
short_circuit

Hence, under the D-WFS, *bulb_blows* and its consequence *dark_room* are both considered unknown. This is what we would expect a sceptical semantics of may-cause links to give.

The D-WFS cannot represent inclusive or causally exclusive disjunction directly. However, it can use the approach of transforming such information into an exclusive disjunction, and considering the resulting model. This provides a sceptical semantics for inclusive and causally exclusive disjunction.

We illustrate how this works using example 2:

Consider the program:

astronaut ∨ *rocket_scientist* ← *works_at_nasa*
well_paid ← *astronaut*, *rocket_scientist*
works_at_nasa

Under an 'inclusive' interpretation of the disjunctive clause, we would expect *well_paid* to be unknown. However, as we showed above, the D-WFS will conclude that *astronaut* and *rocket_scientist* are unknown and that *well_paid* is false. Hence, it does not interpret the disjunction inclusively.

However, if we apply the transformation of section 5.3 to the disjunctive clause, we get:

a ∨ *r* ∨ *both* ← *works_at_nasa* .
astronaut ← *a*
rocket_scientist ← *r*
astronaut ← *both*
rocket_scientist ← *both*

well_paid ← *astronaut*, *rocket_scientist*
works_at_nasa.

This time, *well_paid* is unknown, as we require. Similarly, using the transformation of section 5.3, we can indirectly represent causally exclusive disjunction within a program interpreted using the D-WFS.

However, the transformation used in section 5.4 to represent example 7 does not work when the D-WFS is applied to the resulting program. To show this, we consider the two ball case of the problem. The program, after transformation into exclusive disjunctive form is:

first_pick
second_pick
$fp_ball1 \vee fp_ball2 \leftarrow first_pick$
$sp_ball1 \vee sp_ball2 \leftarrow second_pick$
$ball1_out \leftarrow fp_ball1$
$ball2_out \leftarrow fp_ball2$
$ball1_out \leftarrow sp_ball1$
$ball2_out \leftarrow sp_ball2$

$\neg(fp_ball1 \,\&\, sp_ball1)$
$\neg(fp_ball2 \,\&\, sp_ball2)$

Intuitively, the integrity constraints should ensure that a ball is not picked both in the first and the second pick (representing the fact that a picked ball is not returned to the bag). Hence, we would expect both *ball1_out* and *ball2_out* to be true. This is the case if we use the disjunctive stable model semantics to interpret the program; *ball1_out* and *ball2_out* are true in every disjunctive stable model which satisfies the integrity constraints. However, this is not the case with the D-WFS. It concludes that all disjuncts are unknown, and that *ball1_out* and *ball2_out* are unknown. Because it is not a traditional model-theoretic semantics it is not clear how to use integrity constraints to constrain truth values of disjunctions. If we consider only the truth values of the atoms appearing in them, the integrity constraints are satisfied. However, the semantics does not provide the interpretation we require.

The problem occurs because of the difference in how integrity constraints operate on credulous and sceptical semantics. In credulous semantics, the integrity constraints are applied to each candidate model, and used to filter out unacceptable ones. In sceptical semantics, the integrity constraints can only be applied to the single model. In the D-WFS, the situation is further complicated by the assignment of truth values to disjunctions.

It is also not possible to simulate abduction using the D-WFS. In a sceptical semantics, representing abducibles using a disjunction, $a \vee \bar{a}$, simply results in a and $\bar{a}$ being unknown.

7.5 The Generalised Disjunctive Well-Founded Semantics

The generalised disjunctive well-founded semantics of Baral, Lobo and Minker ([BLM 90a], [BLM 90b]) is also a sceptical semantics based on the GCWA. Hence many of the comments of the previous section also apply to it.

It, too, is primarily based on the use of minimisation to simulate exclusive disjunction. Like the stationary semantics, it uses state pairs to present information about disjunctions that are known true, and conjunctions which are known false. In addition to the flaws of other semantics discussed in the previous section, it has an additional rather idiosyncratic property.

Consider the disjunction $red \vee blue$. It assigns to this the state pair:

$\langle\ red \vee blue; \neg(red \& blue)\ \rangle$

corresponding to an exclusive interpretation of the disjunction, as we would wish. If we now add the clause $not_two_coloured \leftarrow \neg red$, then the state pair remains:

$\langle\ red \vee blue; \neg(red \& blue)\ \rangle$

In other words, *not_two_coloured* is interpreted as unknown, as we would wish. However, consider adding the second clause, $not_two_coloured \leftarrow \neg blue$.

As we are treating the disjunction $red \vee blue$ as having a minimal set of disjuncts true, we would expect either *red* or *blue* to be false, so would expect to be able to conclude that *not_two_coloured* is true. However, something rather undesirable occurs. The new state pair is:

$$\left\langle \begin{array}{c} red \vee blue, red \vee not_two_coloured, blue \vee not_two_coloured; \\ \neg(red \,\&\, blue \,\&\, not_two_coloured) \end{array} \right\rangle$$

$\{red, blue\}$ is a valid minimal model of the state pair. In other words, for some strange reason, the semantics has switched to give an inclusive interpretation of the disjunction $red \vee blue$!

Because of this rather counter-intuitive property, combined with factors discussed in the previous section, it appears that the GDWFS cannot be used as a straightforward knowledge representation tool.

7.6 Extensions Of The Weak Generalised Closed World Assumption

In this section we will look at sceptical semantics which have been proposed as extensions of the weak Generalised Closed World Assumption [RLM 89]. Firstly, we recall the definition of the weak Generalised Closed World Assumption:

Definition:

Given a positive disjunctive logic program, P, we define the Horn translation of P, $Horn(P)$, to be the program $\left\{ a_i \leftarrow body \mid a_1 \vee \ldots \vee a_n \leftarrow body \in P, 1 \leq i \leq n \right\}$.

Under the weak Generalised Closed World Assumption, a literal a can be concluded false if a is false in the minimal model of $Horn(P)$.

The WGCWA, as we discussed in chapter 1, is said to treat disjunction as 'inclusive'.

Several semantics have been proposed which extend the weak Generalised Closed World Assumption to handle disjunctive logic programs containing negation.

We will consider the WGCWA¬, proposed by Sakama and Inoue [S1 93], and the weak disjunctive well-founded semantics, proposed by Brass and Dix [BD 95].

Firstly, we consider the WGCWA¬. The observations made in this case also apply to the WPERFECT semantics, proposed by Dix and Muller [DM 94]. This is because, as they showed [DM 94], the WPERFECT semantics is equivalent to the WGCWA¬, but only defined for stratified disjunctive logic programs.

The WGCWA¬ is a straightforward extension of the WGCWA.

Definition [SI 93]:

The normal translation, *NP(P)*, of a general disjunctive logic program, *P*, is given by;

$$NP(P) = \{a_i \leftarrow body \mid a_1 \vee ... \vee a_n \leftarrow body \in P, 1 \leq i \leq n\}.$$

Note that this is identical to the definition of *Horn*(*P*), but applied to general disjunctive logic programs.

A literal *a* is considered false under the WGCWA¬ if it is false in all disjunctive stable models of *P*, and in all stable models of *NP*(*P*). Similarly, a literal *a* is considered true under the WGCWA¬ if it is true in all disjunctive stable models of *P*, and in all stable models of *NP*(*P*).[1] It is unknown if it is neither true nor false. Hence, this gives a three-valued model of the original program.

If we represent example 1 directly as a disjunction, we can demonstrate the difference between this semantics and the sceptical extensions of the GCWA discussed in section 7.4:

red ∨ *blue*

two_coloured ← *red*, *blue*

[1] The idea of truth under the WGCWA ¬ was not originally defined in [SI 93]. We use a definition based on that given for WPERFECT in [DM 94].

Whereas the semantics discussed in section 7.4 would conclude that *two_coloured* is false, the WGCWA, and hence the WGCWA$\neg$, conclude that *two_coloured* is unknown. Rather than minimising all literals, they consider the possibility that all literals in a disjunction may be true at the same time.

When considering the possible model semantics and the Indefinite Closed World Assumption semantics, we were able to impose an exclusive interpretation on the disjunction by adding an integrity constraint, $\neg(red \wedge blue)$.

However, as with the D-WFS in section 7.4, this approach cannot be used here. As it is a sceptical semantics, a program has a single model, so integrity constraints cannot be used to filter undesirable alternatives.

However, if we alter its definition to apply the integrity constraints to stable models of P and $N(P)$, before generating the model corresponding to the WGCWA$\neg$, then we can filter stable models corresponding to non-exclusive interpretations of a disjunction, and so we would get a model which gave an exclusive form of the WGCWA$\neg$.

Hence the WGCWA$\neg$ is not able to represent exclusive disjunction without significant modification, and so the techniques of chapter 5 for representing different kinds of information are not open to it. It is not able to simulate abduction either. If we attempt to represent an abducible a by using the disjunction $a \vee \overline{a}$, then both will be considered unknown.

The WGCWA$\neg$ is not a good semantics of 'inclusive' disjunctive information either, as the following example shows. Consider the following program, P:

bureaucrat $\vee$ *astronaut* $\vee$ *rocket_scientist* $\leftarrow$ *works_at_nasa*

works_at_nasa

exciting_job $\leftarrow$ *astronaut*, *rocket_scientist*, $\neg$*bureaucrat*

The second statement expresses that if someone is both an astronaut and a rocket engineer, but not a bureaucrat, then they have an exciting job.

If we interpret the disjunction as 'inclusive', then we would expect *exciting_job* to be concluded as unknown. However, the WGCWA¬ concludes it to be false as we now demonstrate:

The program has 3 stable models:

{*works_at_nasa*, *bureaucrat* }, {*works_at_nasa*, *astronaut* }and
{*works_at_nasa*, *rocket_scientist*}

NP(*P*) has the single stable model:

{*works_at_nasa*, *bureaucrat*, *astronaut*, *rocket_scientist*}

As *exciting_job* is false in all four of these stable models, it is considered false under the WGCWA¬.

This shows the flaw in using the WGCWA¬ to represent 'inclusive' disjunction. It does not allow the disjuncts in the head of a clause to range freely. In this case, it fails to consider a model in which *astronaut* and *rocket_scientist* are true and *bureaucrat* is false. Hence, *exciting_job* is concluded false, when an 'inclusive' interpretation of the program would consider it unknown.

Another semantics proposed as an extension of the WGCWA is the weak disjunctive well-founded semantics (WD-WFS) of Brass and Dix [BD95]. This is a variation on the D-WFS, which we discussed in section 7.4. Recall that the definition of the D-WFS involves reducing a disjunctive logic program to one containing conditional facts only, and then iteratively reducing this program to form the residual program. The reduction involves removing clauses and literals which are 'known false', and conditional facts which are subsumed by others. The WD-WFS is defined similarly, but does not remove conditional facts which are subsumed by others. The semantics is, however, more seriously flawed than the WGCWA¬ semantics.

As the following example shows, it is not an extension of the WGCWA:

$astronaut \vee rocket_scientist \leftarrow works_at_nasa$
$works_at_nasa$
$well_paid \leftarrow astronaut, rocket_scientist$

The WGCWA considers *well_paid* to be unknown, as we would expect from an 'inclusive' interpretation. However, the WD-WFS concludes that *well_paid* is false. The residual program, under the WD-WFS is:

$astronaut \vee rocket_scientist$
$works_at_nasa$

well_paid is false in all minimal models of this program, so is considered false under the WD-WFS.

Hence WD-WFS does not extend the WGCWA, and can't be used to represent 'inclusive' disjunction. Brass and Dix claimed to have proven that the WD-WFS does extend the WGCWA ([BD 95], corollary 5.4b). This is a counter-example to this result.

The WD-WFS is also a poor semantics for exclusive disjunction, as the following example shows:

$red \vee blue$
$two_coloured \leftarrow red, blue$

From this program, the WD-WFS would conclude that *red* and *blue* are unknown, and *two_coloured* is false, as we would expect from an exclusive interpretation of disjunction. However, if we add the atom *red* to the program, it now concludes that *red* is true, but still concludes that *blue*, and any consequences of *blue*, are unknown. Hence it is not giving an exclusive interpretation. Adding the integrity constraint $\neg(red \wedge blue)$ does not help; an integrity constraint can only filter undesirable models, not alter a model or generate a new one.

The WD-WFS is thus a poor semantics for both 'inclusive' and exclusive interpretations of disjunction.

The only sceptical semantics proposed which can handle 'inclusive' disjunction effectively is the PWA$\neg$ of Sakama and Inoue [SI93]. It is simply the intersection of all the possible models, and so has the characteristics discussed in section 7.2.

7.7 Conclusions

In this chapter, we have looked at different semantics of disjunctive logic programming, and considered their ability to represent different forms of indefinite information. We have found many of them lacking in their ability to do this.

- Semantics which extend the Generalised Closed World Assumption, such as the D-WFS and the GDWFS, are said to interpret disjunction as 'exclusive'. However, as we have seen, they cannot represent exclusive disjunction adequately, even with the addition of integrity constraints.

- Semantics which extend the weak Generalised Closed World Assumption, such as the WD-WFS and the WGCWA$\neg$, are said to interpret disjunction as 'inclusive'. However, as we have seen, they cannot represent 'inclusive' disjunction adequately, even with the addition of integrity constraints.

- The disjunctive stable model semantics is said to interpret disjunction as 'exclusive'. However, to do so accurately, integrity constraints must be used.

- The possible model semantics is the only semantics which is able to represent 'inclusive' disjunction accurately. Furthermore, with the addition of integrity constraints, it can also represent exclusive disjunction.

- The Indefinite Closed World Assumption semantics is the only semantics able to represent causally exclusive information directly.

It is interesting to note that the vast majority of work which has taken place on semantics of disjunctive logic programming has focused on extensions of the GCWA. These extensions are generally sceptical semantics.

The evidence we have presented in this chapter suggests that this is not the most appropriate direction for work on disjunctive knowledge representation, for two reasons:

- Concentrating on semantics which minimise the set of atoms considered true is appropriate in normal logic programs, but does not appear to give a useful semantics when applied to disjunctive logic programs. In particular, it does not give an exclusive interpretation of disjunction.

- Concentrating on sceptical semantics means that much useful information is lost. The development of state pairs are an effort to remedy this. However, they do not allow integrity constraints to be applied to the alternative models. It seems better to consider a sceptical semantics as the intersection of all credulous models.

As we have seen in this survey, the possible model semantics is the most useful as a knowledge representation tool. It is able to represent both 'inclusive' and exclusive disjunction accurately. However, despite this, it tends to be disregarded by the majority of the disjunctive logic programming community. For example, in his survey of semantics of disjunctive logic programming, Minker dismisses it as a "variant of the Generalised Closed World Assumption". [Min 94]

Chapter 8 - Conclusions and Future Directions

8.1 Conclusions

In this thesis, we have considered three different techniques for representing indefinite information within logic programming: disjunctive logic programs, abductive logic programs, and even loops in normal logic programs. We have considered how effective these techniques are at representing the different forms of indefinite information that we have identified. We now conclude by considering in which situations each technique will be most appropriate.

As we showed in chapter 3, we can represent all the different forms of disjunctive and indefinite information we have identified within an abductive logic program. Furthermore, we can easily represent several different kinds of indefinite information within the same abductive logic program. The choice of semantics of negation is not constrained by the need to represent a certain form of indefinite information, as it is in disjunctive logic programming. Hence, the abductive logic programming approach gives a knowledge engineer a significant amount of flexibility.

The abductive logic programming approach is not, however, able to capture all semantics of disjunctive logic programming. In particular, we have not yet been able to represent the disjunctive stable model semantics abductively. We also doubt that it will capture other semantics which are considered central to disjunctive logic programming, such as the generalised well founded semantics and the stationary semantics. This can be viewed in two ways: either as a failing of our framework, or as evidence that these semantics are not useful tools for knowledge representation. We tend to favour the latter view, though the evidence is not yet conclusive. The abductive approach is able to capture two semantics of disjunctive logic programming; the possible model semantics, and the Indefinite Closed World Assumption semantics. It is interesting to note that these two semantics can be used to represent forms of indefinite information which are of practical use in knowledge representation. The possible model semantics can represent 'inclusive' disjunction, while the ICWA semantics can represent causally exclusive disjunction.

From this, we conclude that the abductive logic programming approach is the best general purpose tool for representing disjunctive and indefinite information. A knowledge engineer must deal with abducibles and integrity constraints, resulting in a small amount of extra complexity compared with normal logic programs. However, this is offset by the flexibility of the abductive framework in representing indefinite information, and the simplicity of the semantics of the resulting program in comparison with disjunctive logic programming semantics.

Disjunctive logic programs do not provide the same flexibility and simplicity. There are many different semantics of disjunctive logic programs, and they tend to be complex. Furthermore, the choice of a semantics dictates both how negation is to be treated in the program, and how all disjunctions are to be treated. As we argued in chapters 1 and 3, this is overly constraining, and also leads to a program which is less open to inspection by others.

As we mentioned above, there are some semantics of disjunctive logic programs which cannot be captured in the abductive framework. However, these do not appear to have practical applicability to the different forms of indefinite information we have considered. It is important to consider in the future whether there are any knowledge representation problems where these semantics are appropriate. This is an open question, though the current evidence suggests that there are not. The examples given to motivate these semantics are either abstract, or are examples of exclusive or 'inclusive' disjunction. While the semantics may interpret the specific example correctly, we have shown in chapter 7 that these semantics cannot be used to represent exclusive and 'inclusive' disjunction in general.

For these reasons, we do not believe that the majority of semantics proposed for disjunctive logic programs are of practical use as knowledge representation tools. There are two notable exceptions to this; the possible model semantics and the ICWA semantics.

The abductive approach provides flexibility. If the only forms of indefinite information to be represented are 'inclusive' disjunction and exclusive disjunction, then we can sacrifice this flexibility for increased simplicity. Rather than using an abductive logic program, we can use a disjunctive logic program, labelling those rules to be interpreted as exclusive with appropriate integrity constraints as described in chapter 7. We can then apply the possible model semantics to the resultant program. As we saw in chapter 7, this will interpret disjunctions in rules without integrity constraints as 'inclusive' and those with integrity constraints as exclusive. Hence, using the possible model semantics, disjunctive logic programming can be used as an effective knowledge representation tool for indefinite information in these restricted circumstances. In such circumstances, disjunctive logic programs can represent the knowledge more straightforwardly than abductive logic programs.

As Sakama and Inoue [SI94] observe and as we have discussed in chapter 4, there is a close relationship between the possible model semantics and abduction. Because of this, we can view a disjunctive logic program which is interpreted using the possible model semantics simply as shorthand for the equivalent abductive logic program. Treated in this way, disjunctive logic programming can be considered simply as a notational convenience for a restricted form of the abductive framework. This has the additional benefit that proof procedures for abductive logic programming, such as that presented in [FK 97], can be used.

We can consider the Indefinite Closed World Assumption semantics in the same way. We can use it if we wish to consider only exclusive and causally exclusive disjunction, and no other forms of indefinite information. We can represent the knowledge as a disjunctive logic program with integrity constraints. If a disjunction is to be treated as exclusive, it is labelled with appropriate integrity constraints as presented in chapter 7. We can then apply the ICWA semantics to the resulting program. This will interpret labelled disjunctive rules as exclusive, and other rules as causally exclusive. Again, such a program can be viewed as a shorthand for the equivalent abductive program, as described in chapter 4.

If we place further restrictions on the kinds of indefinite information we choose to represent, then we can simplify even more. If we wish to represent only exclusive disjunction and are prepared to adopt the stable model semantics or one of its variants, then we can represent the information entirely within a normal logic program. We can do this by transforming exclusive disjunctions into even loops, using the transformation described in chapter 5. The transformation also introduces integrity constraints. We can either adopt these, or instead impose a restriction on disjunctions that at most one disjunct can appear in the head of any other clause.

Saccà and Zaniolo [SZ 90] use the even loop approach to represent indefinite information within normal logic programs. In the examples they give, this approach is used to represent exclusive disjunctive information. This is a valid approach, but we must be aware of potential pitfalls. Even loops without integrity constraints are, in the general case, not equivalent to exclusive disjunction, as we discussed in chapter 6. Hence we must be careful that we only use them in the restricted form.

Hence, the decision of what knowledge representation to use depends on what knowledge is to be represented.

- The abductive framework is the most general and flexible. It is able to represent all the different kinds of indefinite information that we have considered, and can be used with any semantics of negation.

- If only 'inclusive' and exclusive disjunction is needed, then disjunctive logic programs can be used. They are interpreted under the possible model semantics. This imposes the stable model semantics of negation on the program.

- If only exclusive disjunction is needed, and the stable model semantics of negation is chosen, then normal logic programs with integrity constraints can be used.

8.2 *Future Directions*

In this thesis, we have argued that an abductive framework is a more effective general tool for reasoning with indefinite information than a disjunctive logic program. We have done this in two ways:

- We have showed than an abductive framework, with appropriate choice of integrity constraints, is equivalent to a disjunctive logic program interpreted under certain semantics.

- We have showed that other semantics of disjunctive logic programs are not able to represent effectively the different kinds of indefinite information we have identified.

If we are to show conclusively that the abductive approach to representing indefinite information is better than the disjunctive logic programming approach, then further work must take place in two directions:

- We must explore if there are forms of indefinite information which we have not considered which are effectively represented by some semantics of disjunctive logic programming.

- We must explore if more semantics of disjunctive logic programming could be represented abductively.

Our claim can be refuted if some practical example is found which uses a semantics of disjunction which cannot be represented in an abductive framework. Unfortunately, it is possible to prove our claim only by showing that every semantics of disjunctive logic programming can be represented within an abductive framework.

It is highly unlikely that all disjunctive logic programming semantics can be represented in this way, so we will have to settle for evidence which corroborates our claim. If every practically useful example of indefinite information we find can be represented in an abductive framework, then this strongly supports our claim.

In the work we have presented in this thesis, we have focused on examples in which all literals can be assigned true or false under some semantics (usually the stable model or disjunctive stable model semantics). However, no two-valued model can be given to a program containing the clause $p \leftarrow \neg p$. In normal logic programming, it is necessary to use a semantics such as preferred extensions [Dung 91] or three-valued stable models [Prz 91] in which p is undefined. In disjunctive logic programming, how should a clause such as $p \vee q \leftarrow \neg p$ be interpreted? We can use abstract examples such as this to motivate work in both of the activities described above:

- Can concrete, meaningful, examples of a similar form to these abstract examples be found? If so, what interpretation is appropriate? Do existing semantics of disjunctive logic programs, such as the three valued version of the disjunctive stable model semantics [Prz 91], interpret these examples appropriately?

- Can disjunctive semantics which are able to interpret these abstract examples be represented within the abductive framework? How would our abductive framework interpret such examples, if it uses a semantics such as the preferred extension semantics [Dung 91] as the underlying semantics of negation?

By carrying out such work, we would hope to corroborate our claim further, or to identify certain semantics of disjunctive logic programs which are of practical use, and cannot be represented abductively.

In this thesis, we have also argued that certain forms of indefinite information can be represented using normal logic programs. In chapter 6, we showed that exclusive disjunction can be represented in normal logic programs, using the approach of Saccà and Zaniolo [SZ 90].

In chapter 7, we argued that certain examples of apparently disjunctive information do not need any special approach to represent them but rather such statements should be viewed as appearing in the completion of an appropriate program.

For example, the statement "if someone has spots, then they have measles or acne", is implicit in the logic program:

spots ← *acne*
spots ← *measles*

This program has the completion:

spots ↔ *acne* ∨ *measles*

The above statement is the only-if part of this rule. We have argued that causal information which appears to require disjunction is better represented in this way. It may be that other forms of disjunctive information ('inclusive' and exclusive, for example) could also be represented within the completion of a normal logic program. Further exploration will be necessary to determine if this is the case. If most forms of indefinite information can be represented in normal logic programs, either using the completion or using the even loop approach, then this will give further evidence to the redundancy of disjunctive logic programming. It may also suggest that the abductive approach is not necessary either, as normal logic programs augmented with integrity constraints are a sufficient knowledge representation tool.

We believe that this is not the case. We conjecture that normal logic programs are not adequate for representing all forms of indefinite information, and that disjunctive logic programming semantics are overly complex and unnecessary. Abductive logic programs, though, are able to represent the different forms of indefinite information in a way which is straightforward and clear. In this thesis, we have presented the first steps in demonstrating this.

Appendix: Summary of Key Results of the Thesis

In this appendix, we summarise the key transformations and equivalence theorems of the thesis, for use as reference when reading it.

Representing Indefinite Information in an Abductive Framework

Firstly, we summarise the transformations introduced in chapter 3.

The indefinite implication $\{a_1,...,a_n\} <\sim body$ is represented within an abductive framework $< P, A, IC >$ as the program clauses P and integrity constraints IC, where:

P:

$a_1 \leftarrow body, \alpha_1^*$

$\vdots$

$a_n \leftarrow body, \alpha_n^*$

where $\{\alpha_1^*,...,\alpha_n^*\} \subseteq A$

IC:

$\alpha_1^* \vee ... \vee \alpha_n^* \leftarrow body$

In addition, appropriate upper-bound integrity constraints are added depending on how the indefinite implication is to be interpreted:

Inclusive Disjunction	None
Exclusive Disjunction	$\{\neg(a_i \wedge a_j) \leftarrow body \mid \forall i \neq j\}$
Causally Exclusive Disjunction	$\{\neg(a_i \wedge a_j) \mid \forall i \neq j\}$
Disjunction with Minimised Positive Consequences	$\{\alpha_i^* \leftarrow a_i, body \mid \forall i\}$ and minimisation of $\{\alpha_1^*,...,\alpha_n^*\}$

Equivalence of Abductive Representations to Disjunctive Semantics

The following table lists the main theorems of the thesis which show equivalence between the abductive representations of indefinite information, and disjunctive logic programs interpreted under certain semantics. These theorems are proved in chapters 4 and 5.

Theorem 4.1	Abductive Representation of Inclusive Disjunction interpreted using Generalised Stable Models ⇔ Possible Models [SI 93]
Theorem 4.3	Abductive Representation of Causally Exclusive Disjunction interpreted using Generalised Stable Models⇔ Indefinite Closed World Assumption Semantics [KWC 94]
Corollary of Conjecture 4.2.3	Abductive Representation of Disjunction with Minimisation interpreted using Generalised Models ⇔ (?) Perfect Models [Prz 88b]
Theorem 5.1.3	Abductive Representation of Exclusive Disjunction interpreted using Generalised Stable Models ⇔ Exclusive Disjunctive Models [Chapter 5]

Transforming Disjunctive Logic Programs into Exclusive Disjunctive Logic Programs and Normal Logic Programs

We now summarise the transformations of chapters 5 and 6.

Given a disjunctive logic program P which is interpreted using the Indefinite Closed World Assumption semantics, we define a new exclusive disjunctive logic program $<P', IC_x>$:

- Each non-disjunctive clause in P is in P'.
- For each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ in P, P' contains the disjunctive clause $\delta_1 \vee ... \vee \delta_n \leftarrow body$ and the non-disjunctive clauses $a_1 \leftarrow \delta_1, ..., a_n \leftarrow \delta_n$ where δ_i are newly introduced atoms.
- IC_x contains the integrity constraints $\{\neg(\delta_i \wedge \delta_j) \leftarrow body \mid i \neq j\}$ for each newly introduced disjunctive clause.

Theorem 5.2 proves that the exclusive disjunctive models of $<P', IC_x>$ correspond to the I-models of the original program P.

Similarly, we can transform a disjunctive logic program, P, which is to be interpreted under the possible model semantics, into an equivalent exclusive disjunctive program interpreted using the exclusive disjunctive semantics. We form P' from P by substituting each disjunctive clause $a_1 \vee ... \vee a_n \leftarrow body$ with clauses defined as follows:

For each non-empty subset S_i of $\{a_1,...,a_n\}$, we introduce a new atom, s_i. The clause $s_1 \vee ... \vee s_m \leftarrow body$ is in P', where $s_1,...,s_m$ are the $2^n - 1$ newly introduced atoms for this clause. In addition, for each such S_i, the clauses $\{a_j \leftarrow s_i \mid a_j \in S_i\}$ are in P'.

Finally, we can transform an exclusive disjunctive logic program $<P', IC>$ into a normal logic program with integrity constraints $<N(P), IC>$ by applying Dung's transformation:

Given a disjunctive logic program, P, we define the corresponding normal logic program, N(P), to be the program formed by substituting each disjunctive clause, $a_1 \vee ... \vee a_n \leftarrow body$, in P with the set of normal clauses:

$$
\begin{aligned}
& a_1 \leftarrow body, \neg a_2, ..., \neg a_n \\
& \vdots \\
& a_i \leftarrow body, \neg a_1, ..., \neg a_{i-1}, \neg a_{i+1}, ..., \neg a_n \\
& \vdots \\
& a_n \leftarrow body, \neg a_1, ..., \neg a_{n-1}.
\end{aligned}
$$

Theorem 6.2 proves that there is a 1-1 correspondence between those stable models of $N(P)$ which satisfy the exclusive integrity constraints, IC, and exclusive disjunctive models of $<P, IC>$.

In chapter 7, we carried out an analysis of how effective various semantics of disjunctive logic programming are at representing the different kinds of indefinite information we identified. The following table summarises these results:

	References	Exclusive Disjunction	'Inclusive' Disjunction	Causally Exclusive Disjunction	Causal Information	Simulate Abduction
Disjunctive Stable Models	[Prz 91]	Yes, with ICs	No	No	Yes	Yes
Possible Models	[SI 92]	Yes, with ICs	Yes	No	No	Yes, with ICs
Indefinite CWA	[KWC 94]	Yes, with ICs	No	Yes	No	Yes
Extensions of GCWA	[Ross 90] [BD 95]	No	No	No	Limited	No
Extensions of WGCWA	[SI 92] [BD 95]	No	No	No	No	No

References

[ABW 88] K. Apt, H. Blair and A. Walker. *Towards a Theory of Declarative Knowledge*. Foundations of Deductive Databases and Logic Programming, ed J. Minker. Morgan-Kaufman, Washington (1988), 89-148.

[AD 95] C. Aravindan, P.M. Dung. *On the Correctness of Unfold/Fold Transformation of Normal and Extended Logic Programs*. Journal of Logic Programming 24(3) (1995), 201-217.

[BLM 89] C. Baral, J. Lobo and J. Minker. *Generalised Well-Founded Semantics for Logic Programs*. In 10th International conference on Automated Deduction (July 1989), 102-116.

[BLM 90a] C. Baral, J. Lobo and J. Minker. *Generalised Disjunctive Well-Founded Semantics for Logic Programs*: Declarative Semantics. Proceedings of the 5th International symposium on Methodologies for Expert Systems (1990), 465-473.

[BLM 90b] C. Baral, J. Lobo and J. Minker. *Generalised Disjunctive Well-Founded Semantics for Logic Programs*: Procedural Semantics in proceedings of the 5th International symposium on Methodologies for Expert Systems (1990), 456-464.

[BLM 91] C. Baral, J. Lobo and J. Minker. WF3: *A Semantics for Negation in Normal Disjunctive Logic Programs*, Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems (1991), 459-468.

[BJ 74] G.S. Boulos and R.C. Jeffrey. *Computability and Logic*. Cambridge University Press (1974).

[BDKT 97] A. Bondarenko, P.M. Dung, R.A. Kowalski and F.Toni. *An Abstract Argumentation-theoretic Approach to Default Reasoning*. Artificial Intelligence Journal 93 (1-2) (1997), 63-102.

[BD 95] S. Brass and J. Dix. *Disjunctive Semantics based on Partial and Bottom-Up Evaluation*. Proceedings of the 12th International conference on Logic Programming, Tokyo. MIT (1995), 199-213.

[Bry 89] F. Bry. *Negation in Logic Programming: A Formalisation in Constructive Logic*. Information Systems and Artificial Intelligence: Integration Aspects, ed D.Karaginnis, (1989), 30-46.

[Cha 89] E. Chan. *A Possible World Semantics for Non-Horn Databases.* Research report CS-89-47, Dept of Computer Science, Univ. of Waterloo (1989).

[Clark 78] K. Clark. *Negation as Failure.* Logic and Databases, ed H. Gallaire & J. Minker (1978), 293-322.

[CTT 90] L. Console, D. Thesedier Dupre and P. Torasso. *A Completion Semantics for Object-level Abduction.* Proceedings of AAAI symposium in Automated Abduction (1990).

[CTT 91] L. Console, D. Thesedier Dupre and P. Torasso. *On the Relationship between Abduction and Deduction.* Journal of Logic and Computation 1(5) (1991).

[CT 90] L. Console and P. Torasso. *Hypothetical Reasoning in Causal Models.* Int J. of Intelligent Systems, 5(1) (1990), 83-124.

[Deck 96] H. Decker. *Is Disjunction in Logic Programming Inclusive or Exclusive or What?* In Disjuctive Logic Programming and Databases: Non-Monotonic Aspects. Dagstuhl seminar report 150, (1996).

[DD 92] M. Denecker and D. De Schreye. SLDNFA: *An Abductive Procedure for Normal Abductive Programs.* Proceedings International conference and symposium in Logic Programming, (1992), 686-700.

[Dix 91] J. Dix. *Classifying Semantics of Logic Programs.* Logic programming and non-monotonic reasoning: Proceedings of the 1st International Workshop (1991), 166-180.

[Dix 92] J. Dix. *Classifying Semantics of Disjunctive Logic Programs.* Proceedings Joint International conference and symposium on Logic Programming (1992), 798-812.

[DGM 94] J. Dix, G. Gottlob and V. Marel. *Causal Models of Disjunctive Logic Programs.* Proceedings of the 11th International conference on Logic programming. (1994), 798-812.

[DM 94] J. Dix and M. Muller. *An Axiomatic Approach to Semantics of Disjunctive Programs.* Proceedings 11th International conference on Logic Programming (1994), 303-320.

[DLN 90] F. Donini, M. Lenzerini, D. Nardi, F. Pirri and M. Schaerf. *Non-monotonic Reasoning.* Artificial Intelligence Review (1990) 4, 163-210.

[Dung 91] P. Dung, *Negation as Hypothesis: An Abductive Foundation for Logic Programming*. Proceedings 8th International conference on Logic Programming (1991).

[Dung 92] P. Dung. *Acyclic Disjunctive Programs with Abductive Procedure as Proof Procedure*. Proceedings International conference on Fifth Generation Computer Systems (1992), 555-561.

[Dung 93] P. Dung. *The Acceptability of Arguments and its Fundamental Role in Non-monotonic Reasoning and Logic Programming*. Proceedings from the International Joint Conference on Artificial Intelligence, (1993), 852-857.

[DK 89] P.M. Dung and K. Kanchansut. *A Fixpoint Approach to Declarative Semantics of Logic Programs*. ed E.L.Lusk and R.A.Overbeek. Proceedings North American conference on Logic Programming (1989), 604-625.

[EGG 93] T. Eiter, G. Gottlob and Y. Gurevich. *Curb your Theory! A Circumscriptive Approach for Inclusive Interpretation of Disjunctive Information*. Proceedings 13th International joint conference on Artificial Intelligence (1993), 3-8.

[Eshghi 90] K. Eshghi. *Diagnosis as Stable Models*. Proceedings from the 1st International Workshop on Principles of Diagnosis, Menlo Park, (1990), 39-48.

[EK 88] K. Eshghi and R. Kowalski. *Abduction through Deduction*. Technical report, Department of Computing, Imperial College, London (1988).

[EK 89] K. Eshghi and R. Kowalski. *Abduction Compared with Negation by Failure*. Proceedings 6th International conference on Logic Programming, Lisbon (1989), 234-255.

[FL 93] J. Fernandez and J. Lobo. *A Proof Procedure for Stable Theories*. Submitted to Journal of Logic Programming (1993).

[Fitting 85] M. Fitting. *A Kripke-Kleene Semantics of Logic Programs*. Journal of Logic Programming 4 (1985), 295-312.

[Fung 95] T. Fung. *Abduction by Deduction*. Ph.D. Thesis, Imperial College, University of London (1995).

[FK 97] T. Fung and R. Kowalski. *The IFF Proof Procedure for Abductive Logic Programming*. Journal of Logic Programming 33:2 (1997), 151-165.

[Gelfond 94] M. Gelfond. *Logic Programming and Reasoning with Incomplete Information.* Annals of Mathematics and Artificial Intelligence. 12(1994), 89-116.

[GL 88] M. Gelfond and V. Lifschitz. *The Stable Model Semantics for Logic Programs.* Proceedings 5th Logic Programming symposium, MIT Press, Cambridge MA (1988), 1070-1080.

[GL 91] M. Gelfond and V. Lifschitz. *Classical Negation in Logic Programs and Disjunctive Databases.* New Generation Computing, 9 (1991), 365-385.

[Hogger 90] C.J. Hogger. *Essentials of Logic Programming.* Oxford University Press (1990).

[Inoue 91] K. Inoue. *Extended Logic Programs with Default Assumptions.* Proceedings 8th International conference on Logic Programming, Paris (1991), 490-504.

[IS 91] N. Iwayama and K. Satoh. *Computing Abduction using the TMS.* Proceedings 8th International conference on Logic Programming, Paris (1991), 505-518.

[KKT 93] A. Kakas, R. Kowalski and F. Toni. *Abductive Logic Programming.* Journal of Logic and Computation 2(6), (1993).

[KM 90] A. Kakas and P. Mancarella. *Generalised Stable Models: A Semantics for Abduction.* Proceedings 9th European conference on Artificial Intelligence (1990), 385-391.

[KM 92] A. Kakas and P. Mancarella. *Preferred Extensions are Partial Stable Models.* Journal of Logic Programming 14 (3,4), (1992), 341-349.

[KWC 94] Q. Kong, M. Williams and G. Chen. *The Indefinite Closed World Assumption.* Data and Knowledge Engineering 12 (1994), 297-311.

[KS 87] R. Kowalski and F. Sadri. *A Theorem Proving Approach to Database Integrity.* Foundations of Deductive Databases and Logic Programming. ed. J. Minker. Morgan Kaufmann (1987), 313-362.

[KT 94] R. Kowalski and F. Toni. *Argument and Reconciliation.* International symposium of Fifth Generation Computer Systems '94. Workshop on Legal Reasoning, Tokyo (1994), 9-16.

[Kunen 87] K. Kunen. *Negation in Logic Programming.* Journal of Logic Programming 4 (1987), 289-308.

[LT 95] V. Lifschitz and H. Turner. *From Disjunctive Programs to Abduction*. Technical report, University of Texas at Austin (1995).

[LT 85] J. Lloyd and R. Topor. *A Basis for Deductive Database System*. Journal of Logic Programming 2 (1985), 93-109.

[LMR 92] J. Lobo, J. Minker and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press (1992).

[Min 82] J. Minker. *On Indefinite Databases and the Closed World Assumption*. In Lecture notes in Computer Science 138, Springer Verlag (1982), 292-308.

[Min 94] J. Minker. *Overview of Disjunctive Logic Programming*. Annals of Mathematics and Artificial Intelligence 12(1994)1-2, 1-24.

[MR 90] J. Minker and A. Rajasekar. *A Fixpoint Semantics for Disjunctive Logic Programs*. J. Logic Progr. 9(1990), 45-74.

[Peirce 58] C.S. Peirce. *Collected Papers of Charles Sanders Peirce. Vol 2*, (1931-1958). ed Hartshorn et. al. Harvard University Press.

[PP 94] A. Pettorossi, M. Proietti. *Transformation of Logic Programs*. Journal of Logic Programming 19/20 (1994), 261-320.

[Poole 88] D. Poole. *Representing Knowledge for Logic Based Diagnosis*. Proceedings from the International Conference on Fifth Generation Computer Systems (1988), 1282-1290.

[Poole 89a] D. Poole. *Explanation and Prediction: An Architecture for Default and Abductive Reasoning*. Computational Intelligence 5(1) (1989), 97-110.

[Poole 89b] D. Poole. *Normality and Faults in Logic-based Diagnosis*. Proceedings from the 11th International Joint Conference on Artificial Intelligence (1989), 1304-1310.

[PE 92] C. Preist and K. Eshghi. *Consistency-Based and Abductive Diagnoses as Generalised Stable Models*. Proceedings International conference on Fifth Generation Computer Systems. Tokyo (1992), 514-521.

[Prz 88a] T.C. Przymusinski. *On the Declarative Semantics of Deductive Databases and Logic Programming*. Foundations of Deductive Databases and Logic Programming, ed J.Minker, (1988), 193-216.

[Prz 88b] T.C. Pryzmusinski. *Perfect Model Semantics*. Proceedings 5th International conference and symposium on Logic Programming (1988), 1081-1096.

[Prz 90] T.C. Przymusinski. *Stationary Semantics for Disjunctive Logic Programs and Deductive Databases*. Proceedings North American conference on Logic Programming (1990), 42-59.

[Prz 91] T.C. Przymusinski *Stable Semantics for Disjunctive Programs*. New Generation Comp. J. 9(1991), 401-424.

[Prz 95] T.C. Przymusinski. *Static Semantics for Normal and Disjunctive Logic Programs*. Technical report, University of California, Riverside (1995).

[RLM 89] A. Rajasekar, J. Lobo and J. Minker. *The Weak Generalised Closed World Assumption*. J. Automated Reasoning (1989), 293-307.

[RNW 83] J. Reggia, D. Nau and P. Wang. *Diagnosis Expert Systems based on a Set Covering Model*. International Journal of Man Machine Studies 19 (1983), 437-460.

[Rei 78] R. Reiter. *On Closed World Databases*. In Logic and Data Bases, ed H. Gallaire and J. Minker, Plenum Press, NY (1978), 55-76.

[Rei 87] R. Reiter. *A Theory of Diagnosis from First Principles*. Artificial Intelligence 32 (1987), 57-95.

[Ross 89] K. Ross. *Well-founded Semantics for Disjunctive Logic Programs*. Proceedings 1st International conference. on Deductive and Object Oriented Databases, Kyoto (1989), 352-369.

[RT 88] K. Ross and R. Topor. *Inferring Negative Information from Disjunctive Databases*. J. Automated Reasoning 4(1988), 397-424.

[SZ 90] D. Saccà and C. Zaniolo. *Stable Models and Non-determinism in Logic Programs with Negation*. Proceedings of the ACM symposium on Principles of Database Systems (1990), 205-218.

[SZ 97] D. Saccà and C. Zaniolo. *Deterministic and Non-deterministic Stable Models*. To be published in the Journal of Logic and Computation (1997).

[Sak 89] C. Sakama. *Possible Model Semantics for Disjunctive Databases*. Proceedings 1st International conference on Deductive and Object Oriented Databases (1989), 337-351.

[SI 93] C. Sakama and K. Inoue. *Negation in Disjunctive Logic Programs.* Proceedings ICLP-93, Budapest (1993), 703-719.

[SI 94] C. Sakama and K. Inoue. *On the Equivalence between Disjunctive and Abductive Logic Programs.* Proceedings 11th International conference on Logic Programming (1994), 489-503.

[TK 95a] F. Toni and R.A. Kowalski. *Reduction of Abductive Logic Programs to Normal Logic Programs.* Proceedings from the International Conference on Logic Programming, L.Sterling, ed., Tokyo (1995), 367-381.

[TK 95b] F. Toni and R.A. Kowalski. *An Argumentation-theoretic Approach to Logic Program Transformation.* Proceedings from LOPSTR'95 International Workshop on Logic Program Synthesis and Transformation. M. Proietti ed., LNCS 1048, Springer Verlag (1995), 61-75.

[VEK 76] M. van Emden and R. Kowalski. *The Semantics of Predicate Logic as a Programming Language.* Journal of the ACM 23(1976), 733-742.

[VGRS 88] A.van Gelder, K.A. Ross and J. Schlipf. *Unfounded Sets and Well-founded Semantics for General Logic Programs.* Proceedings of the 7th symposium on Principles of Database Systems, (1988), 221-230.

[YH 85] A. Yahya and L. Henschen. *Deduction in Non-Horn Databases.* J. Automated Reasoning 1(1985), 141-160.

BIBL.
LONDIN.
UNIV.