Formalizing Implicatures Using Extended Logic Programming

MSc Thesis (*Afstudeerscriptie*)

written by

Gerben de Vries
0033383
gkdvries@science.uva.nl
(born January 22nd, 1982 in Amsterdam)

under the supervision of **Robert van Rooij**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
May 15th, 2007	Dr. Robert van Rooij
	Prof. Dr. Michiel van Lambalgen
	Prof. Dr. Jeroen Groenendijk
	Prof. Dr. Peter van Emde Boas

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

In this thesis a successful formalization of implicatures using extended logic programming is given. This provides a cognitively more interesting account than traditional formalizations. Focus lies on the class of Q-implicatures, but I-implicatures are dealt with as well. The thesis also includes an equivalence proof between this approach and one using circumscription, such as sketched in Van Rooij and Schulz [2004]. The main conclusion is the difference in complexity between 'exhaustivity' and scalar implicatures, the latter type is harder to compute. This is evidence in favor of the psycholinguistic theory that (scalar) implicatures are only computed in the right context, since they require extra processing.

Acknowledgment

I would like to thank my supervisor Robert van Rooij for developing the subject of this thesis with me while I only had a vague thought about what I wanted. More importantly, I thank him for helping me with this thesis even while it took so long.

The original idea for this thesis arose in Jeroen Groenendijk's class Semantics and Pragmatics. Basically, the exposition below is a very long answer to one of the exam questions. I thank him, in his position as my mentor, for thinking with me about how to make a thesis subject out of the little paper I wrote for his class.

I owe Michiel van Lambalgen thanks for reading my thesis and giving very useful comments. The same goes for Brammert Ottens. Furthermore, I thank my thesis committee.

Finally, and probably most importantly, I thank all of my friends who studied with me in all the different UvA buildings: Gebouw P., PCHH, Daar-boven-het-Atrium, Bushuis, Filobieb, UB, the list goes on.

Contents

Chapter 1

Introduction

Imagine that you're at a friends house, sitting at the dinner table. You're drinking a nice cup of coffee after having finished a 'deep-freeze-pizza-from-the-oven' meal. Later that evening the two of you are planning to go to a party given by a classmate of both of you. Your friend knows that classmate very well, so you ask him: "Will any of our classmates be there?" He responds: "Some of our classmates will be there." You discuss the party some more and after a while you feel a bit of a party vibe coming up, so you ask: "Shall we go?" To which your friend responds: "My cup isn't empty yet." After five more minutes of coffee drinking the two of you are off to the party.

The short story above provides at least two examples of what has become known as *implicatures*: extra pragmatic meaning that is 'implicated' by the semantic content of an utterance. The first example is the answer to your question: "Will any of our classmates be there?". In general you shall infer from the response: "Some of our classmates will be there" that *not all* of them are going to attend the party. This inference is not strictly logical in a traditional sense, since "some" does not imply "not all". You only infer that not all of your classmates are coming because you consider your friend informed about who is coming to the party (after all he knows your classmate who throws the party very well) and he has no reason to hide things from you. If all of them would be coming, then he should have made a more informative statement, given these assumptions. By the fact that he doesn't do this, you infer that not all of your classmates will be coming to the party. In this case the statement: "Not all of our classmates will be there" is an implicature. It is extra pragmatic meaning inferred from the context and the standard semantic content of the utterance. The second response of your friend also carries an implicature. You ask whether your friend is ready to go to the party, but he answers with a statement that his coffeecup still has coffee in it. Because of your knowledge about your friend you know that he likes to completely finish his coffee. Thus you infer that he is not ready to go. In this particular example, the utterance: "My cup isn't empty yet." carries the implicature: "I'm not ready to go."

In the 1960's the philosopher H.P. Grice introduced his theory of pragmatics, which has the concept of an implicature as a main element. Being a philosopher his concept remained somewhat informal, but since then a number of attempts have been made to give a good formalization. Especially the class of so-called quantity-implicatures has proven a good candidate for a formal treatment. Since quantity-implicatures are concerned with stronger statements that could have been made but were not, the first implicature in the above example belongs in this category. Furthermore, there is some generality to the phenomenon, an implicature from "some" to "not all" tends to occur in a lot of different contexts and sentences. The main part of this thesis will be concerned with the formalization of quantity-implicatures. The second example is of a different nature. It depends very much on the specific context of the example, ie. you know your friend always finishes his coffee, and only in this specific context does the implicature occur. This type is often dubbed particularized implicatures and is of no concern in this thesis.

Logic Programming

In Van Lambalgen and Hamm [2004] the authors provide a novel way of formally looking at the semantics of verbs. It is customary in the field of natural language semantics to have a model-theoretic point of view. Van Lambalgen and Hamm argue that this is wrong from the perspective of cognitive science. A widely held view in cognitive science is that humans are information processing machines and therefore the modelling of human intelligence should be in terms of algorithms. After all, algorithms are what makes information processing possible. The authors posit that if we want formal approaches to natural language to be cognitively more relevant, then these approaches had better become computational in nature.

The most elegant formalism in the eyes of Van Lambalgen and Hamm is *logic programming*. One reason for this is that there is evidence that it has cognitive relevance. For instance, it corresponds elegantly to certain, explanatory relevant, neural networks (Stenning and Van Lambalgen [2007]). Another main advantage is the ability of logic programming to deal elegantly with nonmonotonic reasoning. This type of reasoning is essential for day-to-day human life and using it in formal modelling of verbs leads to insightful results, as the authors show. We shall see below that, from the perspective of inferences, implicatures are nonmonotonic in nature.

The above ideas are the starting point for this thesis. The traditional literature on the modelling of implicatures is model-theoretic in nature, hence a computational approach might be very insightful. Since logic programming has some cognitive relevance, results from this thesis can be compared to psychological literature on implicatures.

Thesis Outline

The objective of this thesis is to provide an adequate account of the formalization of implicatures in logic programming that can deal with at least the same data as other current formalization attempts. Furthermore, as mentioned above, this account provides computational insights into implicatures which makes the formalization psycholinguistically more relevant. Finally, the aim is to give a comparison between the approach of this thesis and other formal treatments.

The main type of implicatures that we will treat is the quantity- or Q-implicature, which we saw in the example above. But another type, the I-implicature: inferences based on world knowledge and stereotypes, will be dealt with as well. This class of implicatures looks very much like the knowledge that AI researchers are interested in formalizing, which is precisely the type of knowledge logic programming was designed to handle. Thus we will see that dealing with I-implicatures using logic programming goes very well.

Chapter 2 will introduce the theory of implicatures, its history and current status and the standpoint for this thesis. There will also be a short overview of the recent psycholinguistic work on implicatures. A simplified account of quantity-implicatures, using the formalism of our choice: extended logic programming, will be presented in chapter 3. First, this special version of logic programming is introduced. Then we encounter two ways to look at its semantics: SLDNF-resolution and answer-sets. In between examples are treated. The third chapter also contains a comparison between the approach sketched and an account using circumscription. Some shortcomings that we encounter in chapter 3 will be dealt with in the next chapter. Chapter 4 introduces the WFSX-semantics. In this chapter the final account is given and more complex data is dealt with. We will look at the class of I-implicatures in chapter 5 and apply the theory from the previous two chapters to them. The conclusions are left to chapter 6. To get a bit ahead of things, we will see that the main result is the computational difference between what will be dubbed 'exhaustivity' implicatures and scalar implicatures.

Chapter 2

Implicatures

2.1 History

As mentioned above, back in the 1960's the philosopher H.P. Grice first coined the term *implicature* as a part of his widely influential theory of pragmatics. His theory is aimed at saving the traditional truth-functional account of natural language semantics by providing a set of pragmatic principles which account for the extra meaning that utterances have in specific contexts. The total meaning of an utterance has at least two parts: *what is said* and *what is implicated*. 'What is said' roughly corresponds to the traditional truth-functional part. 'What is implicated' is part of pragmatics and further divided into *conventional* and *conversational* implicatures. Conversational implicatures are again split up into *particularized* and *generalized* conversational implicatures. It is the class of conversational implicatures that we will be concerned with in this thesis and to which we will often just refer as *implicatures*.

Conversational implicatures are, according to Grice, inferred on the basis of assumptions about rational cooperative conversation. These assumptions are summarized rather abstractly in the *cooperative principle* and are made further explicit in his *maxims of conversation*.

The cooperative principle "Make your contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged." (Grice [1989]:26)

The maxims of Quality "Try to make your contribution one that is true," specifically:
"Do not say what you believe to be false"
"Do not say that for which you lack adequate evidence" (Grice [1989]:27)

The maxim of Relation "Be relevant." (Grice [1989]:27)

The maxims of Quantity
Q1: "Make your contribution as informative as is required (for the purposes of the exchange)."

Q2: "Do not make your contribution more informative than is required." (Grice [1989]:26)

The maxims of Manner "Be perspicuous," specifically:
M1: "Avoid obscurity of expression"
M2: "Avoid ambiguity"
M3: "Be brief (avoid unnecessary prolixity)"
M4: "Be orderly" (Grice [1989]:27)

Conversational implicatures arise because the above maxims are being adhered to in rational cooperative conversation. This means that, when we assume that a speaker is following the maxims, a speaker conversationally implicates a proposition ϕ, if ϕ is required to maintain the assumption that the speaker follows the maxims and that the speaker thinks the hearer will realize this requirement. To illustrate this, let us consider the following example taken from Levinson [2000]:16.

(2.1) Q: "What time is it?"
A: "Some of the guests are already leaving."
Imp: (a) "It must be late."
(b) "Not all of the guests are already leaving."

Both (a) and (b) are implicatures of the answer. (a) can be derived by the assumption that the speaker sticks to the maxim of relevance. The fact that guests are leaving is relevant to the time, because guests leave late in the evening, since it is a nice party. By using Q1 we can derive (b). The utterance "all of the guests are already leaving" would have been more informative than the actual answer. Thus by Q1 (and the assumption that the speaker is well informed as to who is leaving) we can assume that this is not the case. Thus we derive (b).

As we see, we can derive implicatures more or less transparently from the utterance and the assumption of rational cooperative conversational activity. That they are derivable is a property of implicatures that is very important to us and which is called *calculability.* Also highly relevant is *cancellability*: an implicature can be defeated by the addition of extra premises. Based on this property we can say the following: if we consider an implicature to be an inference made from the actual utterance, then implicatures are nonmonotonic inferences. Suppose the answer in example 2.1 is: "Some of the guests are already leaving, *in fact all of them are.*" In this case the addition of the premise: "in fact all of them are" cancels the inference to: "not all of the guests are leaving".

In the theorizing after Grice, the traditional set of maxims has been reduced by many authors, collapsing different maxims together into one principle. Most noticeable are the Q- and R-principles of Horn (Horn [1995, 2004] and the comparable Q-,I- and M-principles of Levinson (Levinson [2000]). The status of the different maxims in these principles varies considerably: the maxims of quantity are the most important, followed by those of manner. Early formalization of the derivation of implicatures (and of pragmatics in general) was done by Gazdar (Gazdar [1979]). The focus of his work was on implicatures based on the quantity maxim, primarily scalar (Horn [1972]), but also clausal implicatures (Gazdar [1979]). To date, these types remain the most important implicatures to be studied formally.

2.2 The Common View

The most well-known current pragmatic theory is that of Levinson (Levinson [2000]). This theory is basically a big update of his earlier work: Levinson [1983]. It is based on the notion of a generalized conversational implicature and three guiding heuristic principles. Generalized conversational implicatures are called general because they arise unless there are unusual specific contextual assumptions that defeat it, thus Levinson considers them context independent. Opposite this idea we have the particularized conversational implicature. Such an implicature arises precisely because of specific contextual assumptions and is very much context dependent. In example 2.1 (a) is a particularized implicature. For instance, if the question in the example had been: “Where's Bill?”, then (a) would not have been an implicature of the answer. According to Levinson (b) would have been. He considers (b) to be a typical example of a generalized implicature.

There are three principles behind the derivation of generalized conversational implicatures in Levinson [2000]: Q,I and M. The Q-heuristic provides the background for most of the work on the formalization of implicatures, the I- and M-principles are less relevant, but we will also consider those later on in this thesis.

According to Levinson, the Q-principle roughly corresponds to Grice's first maxim of quantity: Q1[1]. Therefore, implicatures based on this principle are sometimes also called quantity-implicatures. For the speaker the Q-principle entails the following maxim (from Levinson [2000]:76): “Do not provide a statement that is informationally weaker than your knowledge of the world allows, unless providing an informationally stronger statement would contravene the I-principle.” What this boils down to is that the speaker should provide the strongest (relevant) proposition that he knows to be true (ignoring the I-principle for now). The hearer knows that the speaker adheres to this maxim and can therefore infer something about certain statements stronger than the one uttered by the speaker. To determine these statements we need Horn scales.

A Horn scale (Horn [1972]) is a linear ordered set of linguistic alternatives. The use of each alternative in a proposition gives an informationally stronger proposition than the use of those below that alternative. The most famous scale is the one used in example 2.1: ⟨*some*, *all*⟩. Implicature (b) in 2.1 is inferred, using the fact that *all* is higher on the scale than *some*, in the following way: the speaker makes use of *some*, this is the strongest information the speaker can provide, so this must mean that a stronger expression with *all* replacing *some* must not be the case, thus “not all the guests are already leaving” is true. Because of the use of scales, this type of implicature is called *scalar*. The epistemic status of these implicatures is still a matter of debate. Some (Soames [1982]) have argued that the inference is only a weak one and that (b) should

[1]However, whether this is really the case is a matter of debate in the community.

actually be: "The speaker doesn't know/believe that all the guests are already leaving." However, the principle of scalar implicatures remains clear: infer some form of negation of the propositions containing linguistic alternatives stronger than the alternative used, provided that there is a Horn scale containing these alternatives.

The other type of implicature based on the Q-principle is the clausal implicature, which was first introduced by Gazdar (Gazdar [1979]) and which Levinson copies intact. The idea is that if an utterance fails to entail a certain embedded sentence ϕ which a stronger statement (in a scalar sense) would entail, then we infer that the speaker considers ϕ and $\neg\phi$ possible.

2.3 Thesis Starting Points

For this thesis' preliminaries regarding implicatures we shall look at the most recent formal work on implicatures and use it to determine our own viewpoints. Recent work on the (traditional) formalization of scalar implicatures is found in for instance Sauerland [2004]. In Van Rooij and Schulz [2004], Schulz and Van Rooij [2006] and also Spector [2003] a somewhat different approach is taken; their formalization is based on the concept of exhaustive interpretation. Groenendijk and Stokhof [1984] introduced the concept of exhaustive interpretation as a way to account for the particular way we often interpret answers. If one asks: "Who came to the party?" and the answer is: "John and Mary", then we take this to mean that only John and Mary came, the answer exhaustifies all the possible people that can come. Van Rooij and Schulz show that this concept, if modified a bit, can deal with traditional scalar and clausal implicatures elegantly without the need for the postulation of Horn scales. This is a real advantage over the traditional account. In this thesis we shall have no need of Horn scales either.

The popular scalar approach as described by Levinson is considered to be a global one, because implicatures are calculated on the semantic output of a grammar. Recently, *localists* such as Chierchia [2001] have attacked the idea that the global scalar account can handle complex sentences and argue that pragmatic rules should be incorporated into the level of the grammar. They want to proof that global accounts in general cannot be successful, but as mentioned above, there are other successful possibilities for a global approach. In this thesis we will also consider implicatures from a global perspective, we will assume an available semantic representation of a proposition and start from there.

The idea that generalized conversational implicatures are completely context independent is not entirely unproblematic, as for instance indicated in Van Rooij and Schulz [2004]. Their following example illustrates this:

(2.2) Q: "Do you have some apples?"

A: "Yes, I have some apples."

Although the scalar term "some" ordinarily gives rise to the scalar implicature "not all", it does not happen in this example because the implicature is out of place in the context created by this particular question. Thus generalized conversational implicatures are not totally context independent. This doesn't mean that scalar implicatures are not in some sense very general, the phenomenon occurs with a great number of linguistic terms in a lot of contexts. It does mean that, to avoid problems of this kind, we will always consider implicatures in this thesis in the context of an (overt) question.

To sum up the vantage point for implicatures for this thesis: the aim is to provide a global formal account of Q-implicatures without the postulation of Horn scales or comparable concepts and we will always be explicit about the context in which an implicature occurs.

2.4 Psycholinguistic Studies

Since the argument for a computational account is its greater cognitive relevance it is also interesting for us to know what psycholinguistics has to say about implicatures. And, although the literature on psycholinguistic investigations into implicatures is not very large, recently some interesting studies have been conducted. The focus of most of the research is to settle the debate between the 'defaultists' and the 'context dependents'. Defaultists think that scalar implicatures are generated by default and processing is required when they are cancelled. This is for instance the neo-Gricean account favored by Levinson. On the other side we have the believers that scalar implicatures are only generated if the context is right and hence cancellation never occurs (Carston [1998]).

In Breheny *et al.* [2006] this is done by doing a reading time experiment that looks at the difference between the time needed to read and comprehend a sentence with an implicature and one without an implicature. The work in Noveck and Posada [2003] and Bott and Noveck [2004] looks at the reaction of subjects to questions containing scalar terms, in the first research brain data called ERP (event related potential) is measured, in the other a response-time experiment is conducted. The scalar implicatures in all the experiments are from "some" to "not all" and from "or" to "not and". All the authors conclude from their research that a context-driven account of implicatures is to be favored.

This is also the case in Pouscoulous *et al.* [in press], which is a research into the development of implicatures in children. Children tend to have more trouble computing scalar implicatures. The study suggests that this is because children do not yet have enough cognitive capacity to compute scalar implicatures, they require all their processing power to comprehend the basic meaning.

The authors of Storto and Tanenhaus [2004] are interested in a slightly different subject. They want to say something about the localist vs. globalist debate that was mentioned in the previous section (which is completely orthogonal to the default vs. context dependent debate, as Geurts [2007] also notes). The results of their experiments tend to favor a localist approach. Furthermore, it somewhat seems to be evidence more in favor of the neo-Gricean account of default generation. However, the authors themselves have nothing to say about this.

All things considered the psycholinguistic evidence is more in favor of a context-driven than a generate-by-default account. We will see that the approach of this thesis is also on the context-driven track.

Chapter 3

Basic Approach

We will always look at the implicature of an answer to a certain (overt) question, since we saw in the previous chapter that the context in which an implicature occurs is important. The idea of our approach is simple: we define a function that generates an extended logic program based on a question and an answer. This program represents the knowledge of the answerer in the context of the question. Using this program we derive implicatures concerning the knowledge state of the answerer. Thus, one could say that the program represents the mental state of the answerer from a perspective of the questioner.

To be able to define our function for generating a 'Gricean' program, the mechanics of extended logic programming are introduced first. Avoiding unnecessary complication in this chapter, we will consider the simplest semantics for extended logic programs: SLDNF and answer-sets. Combined with a 'Gricean' program function we see what these semantics can do for us in different 'benchmark' examples.

3.1 Extended Logic Programming

Logic Programming is what it says: programming with logic. It arose as a programming method separating the *declarative* part from the *procedural* part of a program. A logic programmer's only job is to specify *what* the program should compute, not *how* it should do this. She does this by formalizing the declarative part of the program using so-called *rules*. These rules describe the concepts and relations relevant for the programming problem at hand. Computation with these rules is left to the programming language. In order to be able to do efficient computations we cannot use arbitrary first (or higher) order logic formulae. Thus the rules that describe our problem are of a special type. In their most basic form they are called Horn clauses. A typical example of a Horn clause program might be:

$$\begin{array}{rcl} come(john) & \leftarrow & come(mary) \\ come(mary) & \leftarrow & come(bill) \\ come(sue) & \leftarrow & come(andy) \\ come(bill). & & \end{array}$$

This program formalizes the knowledge that John comes to the party if

Mary comes, Mary comes if Bill comes and Sue comes if Andy comes, and that we know that Bill comes. Via *SLD*, the standard derivation procedure for Horn clause logic programming, we derive that John comes in the following (informal) way. We ask $come(john)$, this matches with the head (left of the arrow) of the first rule, which means we have to make $come(mary)$ true, $come(mary)$ matches with the second rule, thus $come(bill)$ must be true, since $come(bill)$ holds, we have successfully derived $come(john)$. Note that, for instance, we cannot derive $come(sue)$, because $come(andy)$ doesn't hold.

Logic programming with Horn-clauses is already really powerful. However, it still is monotonic and we saw that implicatures involve some form of nonmonotonic reasoning. Logic programming becomes nonmonotonic when we introduce negation in the body (the right side) of a rule. Let's consider a modified version of the first example:

$$\begin{array}{rcl} come(john) & \leftarrow & come(mary) \\ come(mary) & \leftarrow & come(bill) \\ come(sue) & \leftarrow & not\ come(andy) \\ come(bill). & & \end{array}$$

In the third rule, $come(andy)$ is replaced by $not\ come(andy)$. Intuitively the rule now means: Sue comes to the party if Andy doesn't come, or to be more precise, if we cannot derive that Andy comes. We query the program with $come(sue)$, which matches the head of the third rule, thus the program must make $not\ come(andy)$ true. The standard approach for this is negation as failure (*NAF*): if it is impossible to construct a derivation for $come(andy)$, then $not\ come(andy)$ is true. In the example there is no derivation possible for $come(andy)$ (there isn't even a rule with $come(andy)$ as its head), thus $not\ come(andy)$ holds, and therefore Sue comes to the party.

As the example illustrates this kind of negation is in some sense implicit, everything that we cannot derive is "false", but we cannot explicitly define things false, eg. $not\ come(andy)$ cannot be the head of a clause. To overcome this problem people have introduced a second, more explicit, form of negation into logic programming (Gelfond and Lifschitz [1990, 1991]). This form of logic programming is called *extended logic programming.*

As Gelfond and Lifschitz already note (Gelfond and Lifschitz [1991]), one of the main advantages of this approach is that we can define closed world reasoning for only those predicates that we want it for, simply by introducing something like:

$$\neg P(x) \leftarrow not\ P(x).$$

Thus $\neg P$ holds for those individuals for which we cannot derive that P holds. As we will see, such a rule is very useful in our approach to implicatures. Because of this, and the ability to deal with negation explicitly in general, we prefer extended over normal logic programming.

3.1.1 Extended Logic Programs

For the definitions of extended logic programming we will mainly follow the work of Gelfond and Lifschitz [1991], Lifschitz [1996] and Alferes and Pereira [1996]. A *Logic Program* is a finite collection of *rules* of the following form:

$$H \leftarrow L_1, \ldots, L_n.$$

The comma is interpreted as conjunction. We call such a program *normal* if H (the *head*) is an *atom* and the L_i (*body*) are *literals*. Sometimes the body is represented as a set of literals. A literal is either an atom A or its *default* negation *not* A. A program is called *extended* if we allow *objective literals* instead of atoms. An objective literal is either an atom A or its *explicit* negation $\neg A$. The $\neg$ symbol is also used syntactically to denote complementary literals in the sense of explicit negation, ie. $\neg\neg A = A$. Semantically this does not necessarily hold, eg. A and $\neg A$ can both be false. In the extended case, a literal is either objective A or its default negation *not* A. Rules with an empty body are called *facts* and the arrow is usually omitted.

Atoms have the form $p(t_1, \dots, t_n)$, where p is a predicate symbol and the t_i are *terms*. A term over an alphabet $\mathcal{A}$ of a language $\mathcal{L}$ is defined recursively as either a variable, a constant or an expression of the form $f(t_1, \dots, t_n)$ where f is a function symbol of $\mathcal{A}$ and the t_i are terms. A term (resp. atom, literal) is called *ground* if it does not contain variables. The set of all ground terms (resp. atoms) of $\mathcal{A}$ is called the Herbrand universe (resp. base) of $\mathcal{A}$. In the extended case the Herbrand base is the set of all ground objective literals. In this thesis we will not need terms with function symbols in them and therefore we are sloppy with respect to function symbols in our definitions.

It is assumed that the alphabet $\mathcal{A}$ of a program P consists precisely of those constants and predicate and function symbols defined in P. So by the Herbrand universe/base of P we mean the Herbrand universe/base of $\mathcal{A}$. By the grounded version of a program P we mean the possibly infinite set of rules obtained by substituting in all possible ways each of the variables in P by elements of its Herbrand universe. Since work in logic programming is usually restricted to Herbrand interpretations and models, a program and its grounded version can be used interchangeably. In a Herbrand model/interpretation a term in the language is represented in the model by that term, ie. the term a is the object $\mathbf{a}$, the term $f(a, b)$ is the object $\mathbf{f}(\mathbf{a}, \mathbf{b})$, etc. Thus there is no difference between the language of logic programs on the one side and models of the program on the other, the terms are the objects. In the following this difference is therefore ignored, since we are only concerned with Herbrand models.

The semantics of logic programs involve two components: what does a program mean and how do we compute with it. The first is dubbed *declarative* and the second *procedural* semantics. Procedural semantics are intuitively somewhat more understandable than declarative semantics, so we start there first. The aim of procedural semantics is to formalize the informal derivations that we saw in the examples above.

3.1.2 SLDNF - Top down Derivations

As we saw in the informal examples in the previous section we want to derive a goal from the rules of the program. This is done via inference rules that tell us which steps are allowed in a derivation. We already saw that we deal with a normal literal and default literal differently. The most well-known procedural semantics for logic programs with default negation is SLDNF. The following definition is taken from Lifschitz [1996].

SLDNF is about deriving the failure or success of a goal G, which is a finite set of objective literals. The derivable objects in the SLDNF calculus are expressions of the form $\vdash G$ and $\dashv G$. It contains one axiom which says that the empty set is always derivable:

$$\vdash \emptyset,$$

and there are four inference rules associated with a program P:

$$(SP)\frac{\vdash G \cup B}{\vdash G \cup \{L\}} \quad \text{if } L \leftarrow B \in P$$

$$(FP)\frac{\dashv G \cup B \text{ for all } B \text{ such that } L \leftarrow B \in P}{\dashv G \cup \{L\}}$$

$$(SN)\frac{\vdash G \qquad \dashv \{L\}}{\vdash G \cup \{not\, L\}}$$

$$(FN)\frac{\vdash \{L\}}{\dashv G \cup \{not\, L\}}$$

If we can derive $\vdash G$ using these rules and the axiom, then G *succeeds* with respect to the program P. A derivation of $\dashv G$ means that G *fails* in P. A rule should be read in the following way: the expression under the line is derived if we can derive the expression above the line. This illustrated by the following explanation of the four rules. The first one says that a literal from the set of goal literals succeeds if there is a rule in the program with this literal as its head and the derivation of the body of this rule added to the remaining literals in the goal succeeds. The next rule is its dual: a literal fails if for all the rules that have this literal as its head the derivation of the body fails. The other two rules are about the default negation. The derivation of a default negated literal *not* L succeeds if the literal L fails and the other literals in the goal succeed. The last rule says that a default negated literal *not* L fails if the derivation of L succeeds.

Let us look at the following example program from Lifschitz [1996].

$$\begin{array}{lll} p & & \\ q & \leftarrow & p, not\, r \\ q & \leftarrow & r, not\, p \\ r & \leftarrow & p, not\, s \end{array}$$

The failure of $\{q\}$ can be derived as follows. (For readability the curly braces are omitted.)

$$(FP)\dfrac{(FN)\dfrac{(SP)\dfrac{(SP)\dfrac{(SN)\dfrac{\vdash \emptyset \qquad (FP)\dfrac{}{\dashv s}}{\vdash not\, s}}{\vdash p, not\, s}}{\vdash r}}{\dashv p, not\, r} \qquad (FN)\dfrac{(SP)\dfrac{\vdash \emptyset}{\vdash p}}{\dashv r, not\, p}}{\dashv q}$$

When considering the meaning of a program we look at everything that is derivable via SLDNF, thus the following definition is useful. However, this definition is in some sense theoretical, because infinite derivations are possible in SLDNF. It might be impossible to compute the set defined below in practice.

Definition 3.1. *SLDNF(P) is the set of all literals derivable in P via SLDNF.*

$$\mathit{SLDNF}(P) = \{L \mid \vdash \{L\} \textit{ is derivable with SLDNF in } P)\}$$

It is debatable, but as far as procedural semantics go, with its four rules and one axiom SLDNF, is an easily understandable semantics. Because of this we choose this method over more involved procedural semantics. But we saw it has drawbacks: infinite derivations are possible, we must keep that in mind. Thus the introduction of SLDNF here mainly serves the illustrative purpose of getting acquainted with procedural semantics. From a technical, computer science perspective we usually consider SLDNF for so-called schematic programs, ie. programs containing variables. For our purposes grounded, or propositional, programs suffice. Thus we have only seen the SLDNF definitions for grounded programs.

3.2 Translating the Question and Answer

With the basics of extended logic programs in place we need a way to systematically represent the combination of a question and an answer as a program. A typical example that we want to deal with is:

(3.1) Q: "Who came to the party?"

A: "John and Mary came."

Imp: "John, Mary and no one else came."

At the moment we will only consider answers that are propositional formulae, ie. that don't contain quantifiers and variables. Typically these are things like: $\mathit{come}(\mathit{john}) \wedge \mathit{come}(\mathit{mary})$, $\neg \mathit{come}(\mathit{bill})$, etc. Thus we do have predicates and a domain of individuals. Since the result of the translation will be logic program rules, and since the models of a logic program are Herbrand models, the only models we consider possible for our answer formulae must be Herbrand models as well, eg. *mary* will never pick out the same individual as *bill* in the underlying model.

Although the goals of this thesis and Wakaki and Satoh [1997] differ, the similarities between the two are undeniable. The translation procedure proposed in the following resembles very much the translation of a circumscriptive theory into extended logic programming proposed by Wakaki and Satoh[1].

The transformation of a propositional formula into rules will go in a number of steps. The first thing we need is the conjunctive normal form of the formula. The concept of conjunctive normal form is well-known, thus we will not provide a full definition here, one can find it in any introductory logic book. A formula in conjunctive normal form is a conjunction of disjunctions of literals, eg. $(L_1 \vee \neg L_2 \vee L_3) \wedge (\neg L_4 \vee L_5 \vee L_6) \wedge (L_7 \vee \neg L_8 \vee \neg L_9)$. Thus:

Definition 3.2 (CNF function)**.** *Let ϕ be a propositional formula, then CNF(ϕ) is the result of translating ϕ into conjunctive normal form.*

[1]Their hard-to-get paper was discovered by the author after he came up with the translation procedure.

The answer formula for the above example is $come(j) \wedge come(m)$. In this case:

$$CNF(come(j) \wedge come(m)) = come(j) \wedge come(m).$$

The next step is to create clauses out of the CNF. Clauses are sets of literals. In order for a clause to be true, one of its literals needs to be true, thus they have a close connection with a formula in CNF.

Definition 3.3 (*2Clauses* function)**.** *Let ϕ be a propositional formula in conjunctive normal form and let $\phi_1, \ldots, \phi_n$ be its conjuncts. Then*

$$\Sigma = \bigcup_{i \leq n} \{L \mid L \text{ is disjunct of } \phi_i\}.$$

And

$$2Clauses(\phi) = \{\sigma \mid \sigma \in \Sigma \wedge \neg\exists\sigma_2 \in \Sigma : \sigma_2 \subset \sigma\}.$$

Notice the 'clean up' in the second part of the definition in which we remove clauses that are supersets of other clauses, since they are superfluous. Applying this function to the example we get:

$$2Clauses(come(j) \wedge come(m)) = \{come(j)\}, \{come(m)\}.$$

Finally, we transform clauses into extended logic programming rules.

Definition 3.4 (2Rules)**.** *Let Σ be a set of clauses. Then*

$$2Rules(\Sigma) = \{L_j \leftarrow \neg L_1, \ldots, \neg L_{j-1}, \neg L_{j+1}, \ldots, \neg L_n \mid \{L_1, \ldots, L_n\} \in \Sigma\}^2$$

Take as an example the simple disjunction:

$$a \vee b.$$

Which is translated into the clause:

$$\{a, b\}.$$

Resulting in the two rules:

$$\begin{array}{lcl} a & \leftarrow & \neg b \\ b & \leftarrow & \neg a. \end{array}$$

We require two rules for this simple disjunction because the arrow in logic programming is not contrapositive. The two rules represent the disjunction in an intuitive manner: if we know one of the disjuncts to be false, then the other one must be true. The rules for the party example above are easier,

$$2Rules(\{come(j)\}, \{come(m)\}) =$$

$$\begin{array}{c} come(j) \\ come(m). \end{array}$$

We could directly translate a conjunctive normal form into rules without the intermediate step of clauses. However, clauses will prove useful later on and furthermore provide 'cleaner' rules, for instance, we don't have duplicate literals.

To keep things clear, let us put these three auxiliaries together into one translation function.

[2]Recall that $\neg\neg L = L$.

Definition 3.5 (Form2Rules). *The translation function Form2Rules(ϕ) which translates a propositional logic formula ϕ into logic programming rules representing ϕ is defined as follows:*

$$\mathit{Form2Rules}(\phi) = \mathit{2Rules}(\mathit{2Clauses}(\mathit{CNF}(\phi)))$$

The role of the question in the program is simpler, we don't need a full translation of the question into a formula. What the question does is determining which predicate is the one under discussion, ie. the predicate that the questioner wants to know something about. Therefore this predicate is dubbed 'question' predicate. Usually the questioner assumes some form of competence on the part of the answerer regarding this question predicate, else she would not have asked the question. This means that for the question predicate we can do closed world reasoning or make the closed world assumption (CWA), whereas for other predicates we cannot. This results in the following treatment of the question predicate.

For the question predicate we can add the grounded instances, ie. one for every individual, of the following rule to the program.

$$\neg Q(\vec{x}) \leftarrow \mathit{not}\ Q(\vec{x})[3]$$

Intuitively, we can assume that a predicate doesn't hold for individuals ($\neg Q(\vec{x})$) if we have no reason to assume it ($\mathit{not}\ Q(\vec{x})$). In terms of the work in Wakaki and Satoh [1997], which is from a circumscription perspective, this means that we treat the question predicate as the one that needs to be minimized.

In the above example the questioner wants to know who is coming to the party, thus the question predicate is *come*. Which means that we would add grounded instances of:

$$\neg \mathit{come}(x) \leftarrow \mathit{not}\ \mathit{come}(x)$$

to the program.

For now we will assume that the answer contains only the question predicate. The treatment of predicates that are not the question predicate, but that do occur in the answer is postponed and introduced together with the solution for dealing with answers containing quantifiers.

The time has come to put it all together into one program that describes the knowledge-state of the answerer.

Definition 3.6 (*GP* (Gricean Program)). *The function $GP(Q, A, \mathcal{D})$ takes as input a question predicate Q, a propositional formula A and a domain of individuals $\mathcal{D}$[4] It returns an extended logic program.*

$$\begin{array}{rcl} GP(Q, A, \mathcal{D}) & = & \{\neg Q(c_1, \ldots, c_n) \leftarrow \mathit{not}\ Q(c_1, \ldots, c_n) \mid c_1, \ldots, c_n \in \mathcal{D}\} \\ & \cup & \mathit{Form2Rules}(A) \end{array}$$

[3]($\vec{x}$) stands for the vector $x_1, \ldots, x_n$, where n is the arity of the predicate.

[4]If we would have allowed terms with function symbols, then $\mathcal{D}$ should have included these in grounded form as well.

3.3 First Attempt

The application of the above functions to sample data is straightforward, some of it we already saw in the previous section. We determine the propositional formula that represents the answer and we determine what the question predicate is. There must also be a domain of individuals, this is usually trivial and in most examples just a set of people. These three things are the arguments we put in *GP*. *SLDNF* applied to the resulting program gives the implicatures that we are looking for.

3.3.1 "John and Mary came."

Most of the examples we will consider are in a party-going context. We already saw the one below (repeated here for convenience).

(3.2) Q: "Who came to the party?"
A: "John and Mary came."
Imp: "John, Mary and no one else came."

We already noted that the question predicate is *come* and that the answer is straightforwardly translated as $come(j) \wedge come(m)$. Of course there are more individuals who could have come to the party, for brevity we stick to one and call him Bill, abbreviated as b. Accordingly, our domain of individuals $\mathcal{D}$ is $\{j, m, b\}$.

$$P_{3.1} = GP(come(x), come(j) \wedge come(m), \{j, m, b\}) =$$

$$\begin{array}{rcl} come(j) & & \\ come(m) & & \\ \neg come(j) & \leftarrow & not\ come(j) \\ \neg come(m) & \leftarrow & not\ come(m) \\ \neg come(b) & \leftarrow & not\ come(b) \end{array}$$

It is easy to see that the following is true:

$$SLDNF(P_{3.1}) = \{come(j), come(m), \neg come(b)\}.$$

Both $come(j)$ and $come(m)$ are facts of the program and $\neg come(b)$ can be derived as follows:

$$(SP)\dfrac{(SN)\dfrac{\vdash \emptyset \qquad (FP)\dfrac{}{\dashv come(b)}}{\vdash not\ come(b)}}{\vdash \neg come(b)}$$

These results are what we are looking for. We assume that the answerer communicates as much information as possible, ie. he will mention everybody that he knows to have been at the party. So in this example we derive that Bill didn't come to the party. In fact, for every individual not mentioned in the answer we can derive that that individual didn't come.

3.3.2 "John or Mary came."

While there was definitely an implicature at work in example 3.1, it didn't contain a traditional scalar implicature yet. Let's see how we fare with the following.

(3.3) Q: "Who came to the party?"

A: "John or Mary came."

Imp: "John or Mary, but not both came."
"No one else but John or Mary came."

The scalar implicature we expect to arise in this example is that John and Mary didn't come both. The question predicate remains the same as in (3.1), as does the domain of individuals. $come(j) \vee come(m)$ is the easy translation of the answer. Everything together results in the program:

$$P_{3.3} = GP(come(x), come(j) \vee come(m), \{j, m, b\}) =$$

$$\begin{array}{rcl} come(j) & \leftarrow & \neg come(m) \\ come(m) & \leftarrow & \neg come(j) \\ \neg come(m) & \leftarrow & not\ come(m) \\ \neg come(j) & \leftarrow & not\ come(j) \\ \neg come(b) & \leftarrow & not\ come(b). \end{array}$$

Via SLDNF we can, as in the previous example, derive $\neg come(b)$. Other objective literals are not derivable. If we try to derive $come(j)$, $\neg come(j)$, $come(m)$ and $\neg come(m)$ we get infinite derivations. Thus, with respect to the individuals not mentioned we still derive that they weren't at the party, but the scalar implicature: $\neg(come(j) \wedge come(m))$ is not derivable. The answer $come(j) \vee come(m)$ has multiple models. One of those models is cancelled by the scalar implicature mentioned, but the others are not. It seems that, if we want to derive the scalar implicature, then we need to look at the models of the program $P_{3.3}$. Possible models is precisely what declarative semantics of logic programming is about, so we turn to this subject next.

3.4 Second Attempt

3.4.1 Answer-sets

Sticking to the 'simplicity first' approach we look at the simplest declarative semantics for extended logic programs: answer-sets, first introduced by Gelfond and Lifschitz (Gelfond and Lifschitz [1990, 1991]). For the definition we will follow the style of Alferes and Pereira [1996], but give a more recent one by Lifschitz, Tang and Turner (Lifschitz *et al.* [1999]). To give the answer-sets semantics we need four things: interpretations, models, an ordering among interpretations/models and the Γ-operator. Note that the definitions apply to the grounded version of a program.

Interpretations are basically assignments of truth-values, they say which literals are true and which are false. As we will see, some interpretations are also models of a program and some models are answer-sets.

Definition 3.7 (Interpretation)**.** *An interpretation I of a program P is any subset of the Herbrand base $\mathcal{H}$ of P. I is consistent iff it contains no pair of complementary objective literals L, $\neg L$.*

Looking at the example program $P_{3.4}$:

(3.4)

$$\begin{array}{rcl} a & \leftarrow & b, c \\ b & \leftarrow & not\, \neg b \\ \neg c & \leftarrow & \neg a. \end{array}$$

We can easily see that $I_1 = \{a, b\}$ and $I_2 = \{\neg a, b, c\}$ are interpretations of the program $P_{3.4}$, since they are concerned with the same literals as $P_{3.4}$.

An interpretation I can be viewed as a function $I : \mathcal{H} \rightarrow V$ where $V = \{0, 1\}$.

$$I(A) = \begin{cases} 1 & \text{if } A \in I \\ 0 & \text{otherwise} \end{cases}$$

Based on this we can define a truth valuation function for formulae.

Definition 3.8 (Truth valuation)**.** *Let I be an interpretation and C the set of all formulae of the language, then the function $\hat{I} : C \rightarrow V$ is the truth valuation corresponding to I. $\hat{I}$ is recursively defined as follows:*

- *$\hat{I}(L) = I(L)$, if L is an objective literal.*
- *$\hat{I}(not\, L) = 1 - I(L)$, if $not\, L$ is a default literal.*
- *if S and T are formulae then*
    - *$\hat{I}((S, T)) = min(\hat{I}(S), \hat{I}(T))$.*
    - *$\hat{I}(T \leftarrow S) = 1$ if $\hat{I}(S) \leq \hat{I}(T)$, and 0 otherwise.*

For instance, if we look at $\hat{I}_1$ based on the interpretation I_1, then $\hat{I}_1(b, c) = min(I_1(b), I_1(c)) = min(1, 0) = 0$ and $\hat{I}_1(not\, b) = 1 - I_1(b) = 1 - 1 = 0$.

Now we need to separate the interpretations that actually make a program true, these are called models, from those that don't. The above defined truth valuation function is used to define what a model of a program is.

Definition 3.9 (Model)**.** *An interpretation I is called a model of a program P iff for every ground instance of a program rule $H \leftarrow B$ it holds that $\hat{I}(H \leftarrow B) = 1$.*

It is easy to check that I_1 is a model of $P_{3.4}$. However I_2 is not, because $\hat{I}_2(a \leftarrow b, c) = 0$, since $\hat{I}_2(b, c) > \hat{I}_2(a)$.

To decide which models are also answer-sets we need the following ordering on interpretations and models, which is defined only for normal logic programs.

Definition 3.10 (Classical Ordering)**.** *If I and J are two interpretations then we say $I \leq J$ if $I \subseteq J$. Take $\mathcal{I}$ a collection of literals, then an interpretation I is called least in $\mathcal{I}$ if $I \leq J$ for any other interpretation $J \in \mathcal{I}$.*

An inconsistent answer-set is one which contains both a positive literal and its negation. The recent definition of answer-sets in Lifschitz *et al.* [1999], given here, has no room for inconsistent answer-sets and differs in this respect from versions in Gelfond and Lifschitz [1990, 1991]; Lifschitz [1996]; Alferes and Pereira [1996], whose definitions allowed, but didn't agree with respect to inconsistent answer-sets. The Γ-operator, defined below, is necessary to decide which models are answer-sets of a program.

Definition 3.11 (The Γ-operator)**.** *Let P be an extended logic program and I a consistent interpretation. The GL-transformation of P modulo I is the program $\frac{P}{I}$ obtained from P by:*

- *first denoting every objective literal in $\mathcal{H}$ of the form $\neg A$ by a new atom, say $\neg\_A$;*
- *replacing in both P and I these objective literals by their new denotations;*
- *then performing the following operations:*
    - *remove from P all rules which contain a default literal $not\, A$ such that $A \in I$;*
    - *removing from the remaining rules all default literals.*

Since $\frac{P}{I}$ is a definite program (contains only atoms) it has a unique least model J.

Replace in J the atoms $\neg\_A$ by $\neg A$ and call this interpretation J'. Then $\Gamma(I, P) = J'$.

Let us calculate $\Gamma(I_1, P_{3.4})$. First, we replace every $\neg A$ by $\neg\_A$ resulting in:

$$\begin{array}{rcl} a & \leftarrow & b, c \\ b & \leftarrow & not\, \neg\_b \\ \neg\_c & \leftarrow & \neg\_a. \end{array}$$

Then we remove all rules containing a default literal *not* A such that $A \in I_1$. In our case there are none. However there is a default literal in the program which we remove in the next step, giving:

$$\begin{array}{rcl} a & \leftarrow & b, c \\ b & \leftarrow & \\ \neg\_c & \leftarrow & \neg\_a. \end{array}$$

If we take all literals except b to be 0, then every rule in the program is true, thus the least model of this program is: $\{b\}$. This means: $\Gamma(I_1, P_{3.4}) = \{b\}$.

The definition of answer-sets semantics is given using the Γ-operator.

Definition 3.12 (Answer-sets semantics)**.** *A consistent interpretation I of an extended logic program P is an answer-set of P iff $\Gamma(I, P) = I$, ie. I is a fixedpoint of Γ. The answer-sets semantics of P is the set $AS(P)$ defined as:*

$$AS(P) = \{I \subseteq \mathcal{H} \mid \Gamma(I, P) = I\}.$$

Thus I_1 is not an answer-set of $P_{3.4}$, however the interpretation: $\{b\}$ is. Not all programs have answer-sets, take for example the trivial program

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

Neither $\emptyset$ nor $\{p\}$ are fixedpoints of the Γ-operator.

In Lifschitz [1996] it is shown that SLDNF is sound, but unfortunately not complete, in relation to answer-sets.

Theorem 3.1 (Soundness of SLDNF)**.** *For any extended logic program P and objective literal L,*

- *If L succeeds with respect to P, then L belongs to all answer-sets for P.*
- *If L fails with respect to P, then L does not belong to any answer-set of P.*

If we examine the Γ-operator closely, we see that what it does is determine the result of a program given certain assumptions about default literals. It removes all rules with *not* A for which $A \in I$ holds. Since the head of such a rule can never be true if A holds this seems very intuitive. Furthermore all default literals *not* A such that $A \notin I$ holds are removed. Again very intuitive, A is not in I, thus basically *not* A succeeds and can be removed from the rule. When considered this way, every answer-set is a way to interpret the program, ie. an answer-set describes a possibility under which the program is true.

In this thesis we take the approach that the program resulting from GP represents the knowledge of the answerer. Thus the answer-sets of this program are the epistemic possibilities or possible worlds of the answerer.

3.4.2 "John or Mary came." - revisited

We look again at example 3.3. Of interest to us now are the answer-sets of the resulting program $P_{3.3}$, which are the following:

$$AS(P_{3.3}) =$$

$$\begin{array}{c}\{\neg come(j), come(m), \neg come(b)\}\\ \{come(j), \neg come(m), \neg come(b)\}.\end{array}$$

Let us see how we conclude that the first answer-set, call it $I_{P_{3.3-1}}$, is an answer-set of $P_{3.3}$. We must calculate $\Gamma(I_{P_{3.3-1}}, P_{3.3})$, beginning with translating all $\neg A$, we get:

$$\begin{array}{rcl} come(j) & \leftarrow & \neg\_come(m)\\ come(m) & \leftarrow & \neg\_come(j)\\ \neg\_come(m) & \leftarrow & not\ come(m)\\ \neg\_come(j) & \leftarrow & not\ come(j)\\ \neg\_come(b) & \leftarrow & not\ come(b). \end{array}$$

We then remove all rules with *not* A such that A is in $I_{P_{3.3-1}}$. This holds for the third rule. Then we remove the remaining default literals, resulting in:

$$\begin{array}{rcl} come(j) & \leftarrow & \neg\_come(m)\\ come(m) & \leftarrow & \neg\_come(j)\\ \neg\_come(j) & \leftarrow & \\ \neg\_come(b). & \leftarrow & \end{array}$$

Since $\neg\_come(j)$ and $\neg\_come(b)$ are facts of this program, they must be true in its minimal model. And since we have the rule: $come(m) \leftarrow \neg\_come(j)$, $come(m)$ must be in the minimal model as well. Thus, after translating back, we conclude that $\Gamma(I_{P_{3.3-1}}, P_{3.3}) = I_{P_{3.3-1}}$ holds.

We consider answer-sets to be worlds that the speaker holds possible. From this perspective we conclude from the answer-sets of $P_{3.3}$ that the speaker knows: $\neg come(b)$, he knows that Bill didn't come. This we could already derive via SLDNF. In neither of the two answer-sets is it the case that both John and Mary came to the party. Thus, additionally the answer-set semantics allow us to derive the scalar inference that we were looking for, the speaker knows $\neg(come(j) \wedge come(m))$. On top of that, we derive the following, what is usually considered a clausal implicature, the speaker holds possible: $come(m)$, $come(b)$, $\neg come(m)$ and $\neg come(b)$.

3.5 Answers with Quantifiers

Up until now we have made an oversimplification of answers by considering only propositional formulae. What about arbitrary first-order formulae[5]? In general, the translation of arbitrary first-order logic formulae into rules is impossible, since logic programming is based on a subset of first-order logic. Nevertheless, if we restrict ourselves to finite domains, then we can define an extension of *Form2Rules* which translates first order formulae into rules. For this, we need a new function.

It is well known that all first order formulae are equivalent to a first order formula in prenex normal form, ie. all quantifiers are in front of the formula. For the following we will assume that every formula is in prenex normal form. The universal quantifier is easily translated by creating a large conjunction. For every individual in the domain the formula that is quantified over should hold. Dually, the existential quantifier can be translated by creating a large disjunction. This method will create quite a number of rules. Although an unfortunate problem, it is not insuperable. We define the following function for this translation:

Definition 3.13 (2Prop)**.** *Let ϕ be a first order formula in prenex normal form and $\mathcal{D}$ a domain of individuals. Then 2Prop is recursively defined as follows:*

- *$2Prop(\forall x \phi(x), \mathcal{D}) = 2Prop(\phi(d_1), \mathcal{D}) \wedge \ldots \wedge 2Prop(\phi(d_n), \mathcal{D})$, where n is the number of individuals in $\mathcal{D}$ and for each d_i, if $i \neq j$, then d_j is an individual different from d_i.*
- *$2Prop(\exists x \phi(x), \mathcal{D}) = 2Prop(\phi(d_1), \mathcal{D}) \vee \ldots \vee 2Prop(\phi(d_n), \mathcal{D})$, where n is the number of individuals in $\mathcal{D}$ and for each d_i, if $i \neq j$, then d_j is an individual different from d_i.*
- *$2Prop(\phi, \mathcal{D}) = \phi$, if ϕ contains no quantifiers.*

The use of quantifiers and thus variables in formulae will inevitably introduce the use of the equality predicate. We recall that the only models under consideration are Herbrand models, this means that the equality relation is completely fixed. The only equality is between syntactically identical constants. We can use this fact to clean up the clauses that we generate. If a clause contains a literal with equality between two syntactically identical constants or vice versa, then this literal is always true, thus the clause is always satisfied and we can remove this clause from our set. On the other hand if we have a literal with equality between two syntactically nonidentical constants or vice versa, then we can remove this literal from the clause, since it will never be satisfied. To keep the number of generated rules down, we will formalize these ideas in the following clean-up function.

[5]Technically we are not dealing with first-order logic, since we are only concerned with Herbrand models. However the formulae that we study follow the first-order logic syntax.

Definition 3.14 (*EqClean* function)**.** *Let Σ be a set of clauses. Then,*

$$\begin{aligned}
\Sigma' = \{ & \{L_1, \ldots, L_n\} \mid \\
& \{L_1, \ldots, L_n, d_{1a} \neq d_{1b}, \ldots, d_{ma} \neq d_{mb}, d_{(m+1)a} = d_{(m+1)b}, \ldots, d_{ka} = d_{kb}\} \in \Sigma, \\
& L_1, \ldots, L_n \textit{ are not equality or inequality literals,} \\
& \textit{for all } i \leq m\textit{: } d_{ia} \textit{ is syntactically identical to } d_{ib}, \\
& \textit{for all } m < j \leq k\textit{: } d_{ja} \textit{ is syntactically nonidentical to } d_{jb}, \\
& d_{1a}, \ldots, d_{ma}, d_{1b}, \ldots, d_{mb}, d_{(m+1)a}, \ldots, d_{ka}, d_{(m+1)b}, \ldots, d_{kb} \in \mathcal{D}\}
\end{aligned}$$

and,

$$\textit{EqClean}(\Sigma) = \{\sigma \mid \sigma \in \Sigma' \wedge \neg\exists \sigma_2 \in \Sigma' : \sigma_2 \subset \sigma\}.$$

Suppose we have the following clauses:

$$\{P(a), P(b), a = a, b = c\}, \{P(b), P(c), a \neq a, b = c\}.$$

The first clause is always true, since $a = a$ is always true, thus we remove it from our set. In the second clause $a \neq a$ and $b = c$ can never be true, therefore we can remove those literals. The application of the cleanup function to these clauses would result in only one clause:

$$\{P(b), P(c)\}.$$

The redefinition of *Form2Rules* incorporates our newly defined functions.

Definition 3.15 (Form2Rules)**.** *The translation function Form2Rules$(\phi, \mathcal{D})$ which translates a first-order logic formula in prenex form ϕ into logic programming rules representing this formula given a finite domain $\mathcal{D}$ is defined as follows:*

$$\textit{Form2Rules}(\phi, \mathcal{D}) = \textit{2Rules}(\textit{EqClean}(\textit{2Clauses}(\textit{CNF}(\textit{2Prop}(\phi, \mathcal{D}))))).$$

With the introduction of quantifiers into answers we will inevitably get answers that contain other predicates than the question predicate. For instance, we already saw an equivalence relation and we will see more in the next example. There are two good alternatives for treating these other predicates. We can explicitly define the extension of such a predicate in the program, ie. for every individual we include a fact as background knowledge stating whether this individual has this property or not. This means that we add rules to the program generated by GP. The other option is to treat these predicates as fixed predicates in a circumscription sense. Just as in Wakaki and Satoh [1997] we introduce the following rules for a predicate R:

$$\begin{array}{rcl}
R(\vec{x}) & \leftarrow & \textit{not } \neg R(\vec{x}) \\
\neg R(\vec{x}) & \leftarrow & \textit{not } R(\vec{x})
\end{array}$$

The result of these rules is that each consistent answer-set of a program contains either $R(d)$ or $\neg R(d)$ for each d in $\mathcal{D}$. The first solution overrides this second solution, which mean that we can implement the second one by default and introduce the extension of a predicate as background knowledge if we need that.

Since *Form2Rules* now has two arguments and we also want to deal with other predicates than the question, we need a redefinition of GP.

Definition 3.16 (*GP* (Gricean Program))**.** *The function $GP(Q, A, \mathcal{D})$ takes as input a question predicate Q, a first-order formula A in prenex normal form and a domain of individuals $\mathcal{D}$. It returns an extended logic program. Q' is the set of all the predicates in A that are not Q.*

$$\begin{array}{rcl} GP(Q, A, \mathcal{D}) & = & \{\neg Q(c_1, \ldots, c_n) \leftarrow \mathit{not}\, Q(c_1, \ldots, c_n) \mid c_1, \ldots, c_n \in \mathcal{D}\} \\ & \cup & \{R(c_1, \ldots, c_n) \leftarrow \mathit{not}\, \neg R(c_1, \ldots, c_n) \mid \\ & & c_1, \ldots, c_n \in \mathcal{D}, R \in Q'\} \\ & \cup & \{\neg R(c_1, \ldots, c_n) \leftarrow \mathit{not}\, R(c_1, \ldots, c_n) \mid \\ & & c_1, \ldots, c_n \in \mathcal{D}, R \in Q'\} \\ & \cup & \mathit{Form2Rules}(A, \mathcal{D}) \end{array}$$

This new definition of *GP* clearly extends the original one for the propositional case. The result is the same if we put in a propositional answer. First of all, the result of *2Rules* is still the same, since the function *2Prop* leaves a formula without quantifiers unchanged. Secondly, in the propositional case we didn't consider predicates other than the question, thus there will be no extra rules introduced for dealing with other predicates and also the function *EqClean* changes nothing.

For a proof later on in this thesis we will need to get more formal about these claims: we require a theorem that states that under the Herbrand semantics, ie. only allowing Herbrand models, a formula A is equivalent to $\mathit{EqClean}(\mathit{2Clauses}(\mathit{CNF}(\mathit{2Prop}(A, \mathcal{D}))))$ given a domain $\mathcal{D}$. This requires two lemmas.

Lemma 3.1. *$M = \langle \mathcal{D}, I \rangle$ is a Herbrand model of the first order formula ϕ in prenex normal form iff M is a Herbrand model of $\mathit{2Prop}(\phi, \mathcal{D})$.*

Proof. We prove this by induction on the complexity of prenex normal form formulae.

(base) The formula ϕ contains no quantifiers, thus $\mathit{2Prop}(\phi, \mathcal{D}) = \phi$. Which means that $M \models \mathit{2Prop}(\phi, \mathcal{D}) \Leftrightarrow M \models \phi$.

(induction) $\forall x$ The induction hypothesis is the following: for ϕ with free variable x it holds that for all $d \in \mathcal{D}$, M is a Herbrand model of $\phi(d)$ iff M is a Herbrand model of $\mathit{2Prop}(\phi, \mathcal{D})$. Now, the claim M is a Herbrand model of $\mathit{2Prop}(\forall x \phi(x), \mathcal{D})$ is by definition of *2Prop* equal to: M is a Herbrand model of $\mathit{2Prop}(\phi(d_1), \mathcal{D}) \wedge \ldots \wedge \mathit{2Prop}(\phi(d_n), \mathcal{D})$ (where $d_1, \ldots, d_n$ are all the different individuals in $\mathcal{D}$)). This is the same as saying that M is a Herbrand model of $\mathit{2Prop}(\phi(d_1), \mathcal{D})$ *and* M is a Herbrand model of $\mathit{2Prop}(\phi(d_2), \mathcal{D})$, etc. Now by the induction hypothesis we know that this is the case iff M is a Herbrand model of $\phi(d_1)$ *and* M is a Herbrand model of $\phi(d_2)$, etc. Which is equivalent to claiming that M is a Herbrand model of $\forall x \phi(x)$.

$\exists x$ The induction hypothesis is the following: for ϕ with free variable x it holds that for all $d \in \mathcal{D}$, M is a Herbrand model of $\phi(d)$ iff M is a Herbrand model of $\mathit{2Prop}(\phi, \mathcal{D})$. Now, the claim M is a Herbrand model of $\mathit{2Prop}(\exists x \phi(x), \mathcal{D})$ is by definition of *2Prop* equal to: M is a Herbrand model of $\mathit{2Prop}(\phi(d_1), \mathcal{D}) \vee \ldots \vee \mathit{2Prop}(\phi(d_n), \mathcal{D})$ (where $d_1, \ldots, d_n$ are all the different individuals in $\mathcal{D}$)). This is the same as saying that M is a Herbrand model of *2Prop*$(\phi(d_1), \mathcal{D})$ *or* M is a Herbrand model of *2Prop*$(\phi(d_2), \mathcal{D})$, etc. Now by the induction hypothesis we know that this is the case iff M is a Herbrand model of $\phi(d_1)$ *or* M is a Herbrand model of $\phi(d_2)$, etc. Which is equivalent to claiming that M is a Herbrand model of $\exists x \phi(x)$.

□

Lemma 3.2. *$M = \langle \mathcal{D}, I \rangle$ is a Herbrand model of the set of clauses Σ iff M is a Herbrand model of the set of clauses EqClean(Σ).*

Proof. ($\Rightarrow$) Assume M is a Herbrand model of Σ and suppose towards contradiction that M is not a Herbrand model of *EqClean*(Σ). Thus there is a clause σ in *EqClean*(Σ) that is not satisfied by M. By definition of *EqClean*, σ must be a subset of, or equal to a clause in Σ. If it is equal, then we have a contradiction, since it would imply that M is also not a model of Σ. Thus, suppose it is a proper subset of a clause in Σ, call this clause σ'. By definition of *EqClean*, $\sigma - \sigma'$ contains only equality literals. Furthermore, we know by that definition and the fact that equality is fixed in Herbrand semantics that these literals are all false. Therefore M cannot be a model for σ' and thus M cannot be a model for Σ, which contradicts our assumption.

($\Leftarrow$) Suppose M is a Herbrand model of *EqClean*(Σ) and towards contradiction that M is not a Herbrand model of Σ. By definition of *EqClean* all clauses in *EqClean*(Σ) must be subsets of clauses in Σ. Hence M must be a model for Σ and therefore we have a contradiction.

□

These two lemmas allow us to prove the following theorem.

Theorem 3.2. *$M = \langle \mathcal{D}, I \rangle$ is a Herbrand model of the formula ϕ in prenex normal form iff M is a Herbrand model of EqClean(2Clauses(CNF(2Prop$(A, \mathcal{D})$))).*

Proof. This proof is trivial using the two lemmas 3.1 and 3.2 combined with the fact that it is well-known that formulae translated to conjunctive normal form and then further translated to clauses are semantically equal to their non-translated counterparts. □

3.5.1 “Some boys came to the party.”

Now that we have a translation for quantifiers we can look at the most famous of scalar implicatures: “some” implicates “not all”. Again, the example is about the now familiar party.

(3.5) Q: “Who came to the party?”

A: “Some boys came.”

Imp: “Not all the boys came to the party.”
“No girls came to the party.”

The most elegant translation of the answer is: $\exists x(\mathit{boy}(x) \wedge \mathit{come}(x))$. Perhaps, one can convincingly argue that "some" means "at least two", but for the example this does not really matter, thus we stick to the simple translation above. We have seen two options for dealing with other predicates than the question predicate: adding information about them as background knowledge or treating them as fixed predicates. We begin with the later option and hence do not state explicitly as background knowledge who is a boy and who is a girl.

The program that we are looking for is the result of $\mathit{GP}(\mathit{come}(x), \exists x(\mathit{boy}(x) \wedge \mathit{come}(x)), \{j, m, b\})$. For illustrative purposes, let us look at the last step in the *GP* function to see how it works: $\mathit{Form2Rules}(\exists x(\mathit{boy}(x) \wedge \mathit{come}(x)), \{j, m, b\})$. We will look at all the interesting steps. The first one being *2Prop*. C and B abbreviate *come* and *boy* respectively.

$$\begin{gathered}
\mathit{2Prop}(\exists x(B(x) \wedge C(x)), \{j, m, b\}) = \\
\mathit{2Prop}(B(j) \wedge C(j), \{j, m, b\}) \vee \\
\mathit{2Prop}(B(m) \wedge C(m), \{j, m, b\}) \vee \mathit{2Prop}(B(b) \wedge C(b), \{j, m, b\}) = \\
(B(j) \wedge C(j)) \vee (B(m) \wedge C(m)) \vee (B(b) \wedge C(b))
\end{gathered}$$

We take the *CNF* and *2Clauses* steps together, giving the clauses:

$$\begin{gathered}
\{B(j), \neg B(m), \neg B(b)\}, \{B(j), \neg B(m), \neg C(b)\}, \{B(j), \neg C(m), \neg B(b)\}, \\
\{B(j), \neg C(m), \neg C(b)\}, \{C(j), \neg B(m), \neg B(b)\}, \{C(j), \neg C(m), \neg B(b)\}, \\
\{C(j), \neg B(m), \neg C(b)\}, \{C(j), \neg C(m), \neg C(b)\}.
\end{gathered}$$

The function *EqClean* changes nothing to these clause. The result of *2Rules* applied to them is the first chunk of rules in the program below, which is the entire program for this example.

$$P_{4.6} = \mathit{GP}(C(x), \exists x(B(x) \wedge C(x)), \{j, m, b\}) =$$

$$\begin{array}{rcl@{\qquad}rcl}
B(j) & \leftarrow & \neg B(m), \neg B(b) & C(j) & \leftarrow & \neg B(m), \neg B(b) \\
B(m) & \leftarrow & \neg B(j), \neg B(b) & B(m) & \leftarrow & \neg C(j), \neg B(b) \\
B(b) & \leftarrow & \neg B(j), \neg B(m) & B(b) & \leftarrow & \neg C(j), \neg B(m) \\
B(j) & \leftarrow & \neg B(m), \neg C(b) & C(j) & \leftarrow & \neg C(m), \neg B(b) \\
B(m) & \leftarrow & \neg B(j), \neg C(b) & C(m) & \leftarrow & \neg C(j), \neg B(b) \\
C(b) & \leftarrow & \neg B(j), \neg B(m) & B(b) & \leftarrow & \neg C(j), \neg C(m) \\
B(j) & \leftarrow & \neg C(m), \neg B(b) & C(j) & \leftarrow & \neg B(m), \neg C(b) \\
C(m) & \leftarrow & \neg B(j), \neg B(b) & B(m) & \leftarrow & \neg C(j), \neg C(b) \\
B(b) & \leftarrow & \neg B(j), \neg C(m) & C(b) & \leftarrow & \neg C(j), \neg B(m) \\
B(j) & \leftarrow & \neg C(m), \neg C(b) & C(j) & \leftarrow & \neg C(m), \neg C(b) \\
C(m) & \leftarrow & \neg B(j), \neg C(b) & C(m) & \leftarrow & \neg C(j), \neg C(b) \\
C(b) & \leftarrow & \neg B(j), \neg C(m) & C(b) & \leftarrow & \neg C(j), \neg C(m)
\end{array}$$

$$\begin{array}{rcl}
\neg C(j) & \leftarrow & \mathit{not}\ C(j) \\
\neg C(m) & \leftarrow & \mathit{not}\ C(m) \\
\neg C(b) & \leftarrow & \mathit{not}\ C(b) \\
\neg B(j) & \leftarrow & \mathit{not}\ B(j) \\
\neg B(m) & \leftarrow & \mathit{not}\ B(m) \\
\neg B(b) & \leftarrow & \mathit{not}\ B(b) \\
B(j) & \leftarrow & \mathit{not}\ \neg B(j) \\
B(m) & \leftarrow & \mathit{not}\ \neg B(m) \\
B(b) & \leftarrow & \mathit{not}\ \neg B(b)
\end{array}$$

This program has twelve answer-sets.

$$AS(P_{4.6}) =$$

$$\begin{gathered}
\{B(j), B(m), B(b), C(j), \neg C(b), \neg C(m)\}\\
\{B(j), B(m), B(b), C(m), \neg C(j), \neg C(b)\}\\
\{B(j), B(m), B(b), C(b), \neg C(j), \neg C(m)\}\\
\{B(j), B(m), \neg B(b), C(j), \neg C(m), \neg C(b)\}\\
\{B(j), B(m), \neg B(b), C(m), \neg C(j), \neg C(b)\}\\
\{B(j), \neg B(m), B(b), C(j), \neg C(m), \neg C(b)\}\\
\{B(j), \neg B(m), B(b), C(b), \neg C(j), \neg C(m)\}\\
\{B(j), \neg B(m), \neg B(b), C(j), \neg C(m), \neg C(b)\}\\
\{\neg B(j), B(m), B(b), C(m), \neg C(j), \neg C(b)\}\\
\{\neg B(j), B(m), B(b), C(b), \neg C(j), \neg C(m)\}\\
\{\neg B(j), B(m), \neg B(b), C(m), \neg C(j), \neg C(b)\}\\
\{\neg B(j), \neg B(m), B(b), C(b), \neg C(j), \neg C(m)\}
\end{gathered}$$

The first thing to gather from all these sets is that it is never the case that a girl came to the party, thus we can derive the second implicature: "No girls came to the party." from these answer-sets. Furthermore there is no answer-set with more than one boy, in which all the boys came. This means that the first implicature, the scalar one, is also derivable from these answer-sets, because in no possibility with more than one boy did all the boys come to the party.

In the case of the sex of individuals it is very reasonable to include information about this as background knowledge. In this example $P_{4.6}$ would be extended with the following facts: $\{B(j), B(b), \neg B(m)\}$.

Since Mary is now defined as a girl and the answer is only about boys, one would expect to derive the fact $\neg C(m)$ in this extended program. Unfortunately, infinite loops occur when trying to derive $\neg C(m)$. Later on we will see a top-down method that can derive the fact that Mary didn't come to the party. For now, we need to look at the answer-sets. The extended version of $P_{4.6}$ has two answer-sets.

$$AS(P_{4.6}) =$$

$$\begin{gathered}
\{C(j), \neg C(b), \neg C(m), B(j), B(b), \neg B(m)\}\\
\{C(b), \neg C(j), \neg C(m), B(j), B(b), \neg B(m)\}
\end{gathered}$$

Clearly, Mary didn't come to the party. Furthermore, we see that there is no answer-set in which all the boys, ie. John and Bill, came to the party. Thus, as before, the answer-sets provide us with the scalar implicature that we were looking for: "Not all the boys came to the party."

3.5.2 "Two people came to the party."

Using quantifiers we can also represent cardinal numbers, such as in this example:

(3.6) Q: "How many people came to the party?"

A: "Two people came."

Imp: "Exactly two people came to the party."

We interpret "two" in a standard neo-Gricean way and take it to mean "at least two". We have the same question predicate as before: $come(x)$. We must notice however, that the question is about how many people came to the party, which is the cardinality of the extension of the predicate, not the extension itself. We cannot represent this cardinality explicitly since we refrain from introducing numbers as individuals. Because of this, the translation of the answer is a bit more complex, we have to count in first-order logic: $\exists x \exists y (x \neq y \wedge come(x) \wedge come(y))$. We don't need any background theory and we can also keep $\{j, m, b\}$ as our standard domain of individuals.

$$P_{3.6} = GP(come(x), \exists x \exists y(x \neq y \wedge come(x) \wedge come(y)), \{j, m, b\}, \emptyset) =$$

$$\begin{array}{rcl}
come(j) & \leftarrow & \neg come(b) \\
come(j) & \leftarrow & \neg come(m) \\
come(b) & \leftarrow & \neg come(j) \\
come(b) & \leftarrow & \neg come(m) \\
come(m) & \leftarrow & \neg come(j) \\
come(m) & \leftarrow & \neg come(b) \\
\neg come(j) & \leftarrow & not\ come(j) \\
\neg come(m) & \leftarrow & not\ come(m) \\
\neg come(b) & \leftarrow & not\ come(b)
\end{array}$$

Via SLDNF there isn't much useful derivable, except for the trivial facts. The answer-sets are the most interesting in this case. We have three of them.

$$AS(P_{3.6}) =$$

$$\begin{array}{c}
\{come(j), come(m), \neg come(b)\} \\
\{come(b), come(m), \neg come(j)\} \\
\{come(j), come(b), \neg come(m)\}
\end{array}$$

Thus we have three possibilities, ie. John and Mary came, John and Bill came and Mary and Bill came. In every possibility there are only two people that came. Thus, we get an "exactly two" interpretation, precisely the implicature we were looking for.

When we add natural numbers as objects to our domain we can give a somewhat simpler translation of the example. We need a new predicate:

$$NumAtParty(x)$$

which holds if at least x people where at the party. Furthermore we require background theory in the form of the (grounded instances of the) rule:

$$NumAtParty(x) \leftarrow NumAtParty(x + 1).$$

Let us see what program we get in this case (keeping rules ungrounded for brevity).

$$\begin{array}{l}
P_{3.6_B} = GP(NumAtParty(x), NumAtParty(2), \\
\quad \{0, 1, 2, 3, 4\}, \{NumAtParty(x) \leftarrow NumAtParty(x + 1)\}) =
\end{array}$$

$$\begin{array}{rcl} \mathit{NumAtParty}(2) & & \\ \neg \mathit{NumAtParty}(x) & \leftarrow & \mathit{not}\ \mathit{NumAtParty}(x) \\ \mathit{NumAtParty}(x) & \leftarrow & \mathit{NumAtParty}(x+1) \end{array}$$

As we see the program is rather simple and we can easily conclude that

$$\begin{array}{l} \mathit{SLDNF}(P_{3.6_B}) = \\ \{\mathit{NumAtParty}(0), \mathit{NumAtParty}(1), \\ \quad \mathit{NumAtParty}(2), \neg \mathit{NumAtParty}(3), \neg \mathit{NumAtParty}(4)\}. \end{array}$$

This is also the only answer-set of the program. We see that we get an “exactly two”, or more precise, “no more than two” reading. Because we represent numbers differently, this second solution is somewhat less complex, in the sense that the implicature already follows from SLDNF, than the previous one.

3.6 Remarks

Roughly we have seen two types of implicatures in the examples: implicatures based on the not mentioning of individuals in the answer, such as “Bill didn’t come” in example 3.1, and scalar implicatures. The first type we can perhaps aptly dub ‘exhaustivity’ implicatures[6]. The difference between exhaustivity and scalar implicatures is arguably the most noticeable thing so far. Apart from the second treatment of the last example, every scalar implicature is obtained by considering the answer-sets. In contrast, exhaustivity implicatures are most of the time already achieved by using SLDNF (although not in the complex example 4.6 with existentials). This hints at a difference in complexity between exhaustive implicatures and scalar implicatures. So, if nothing else, this distinction is interesting. It might be a thing for psycholinguistic research to look into. We will come back to this in the next chapter.

Scalar implicatures are only generated when one computes the answer-sets of a program, this is a computationally expensive procedure, as we will see when considering computational complexity. In practice SLDNF is used to work with logic programs, answer-sets are only considered in special situations. To map this distinction to a psycholinguistic standpoint: ‘just’ considering SLDNF is what natural language users ordinarily do. Only when the context is right do they consider all the answer-sets, which lets them derive scalar implicatures. Thus, in terms of the psycholinguistic debate on whether scalar implicatures are generated by default or are only generated in the right context, this approach is in the latter category.

In terms of epistemic strength, the implicatures derived in the current approach are strong. In example 3.3 we derive $\neg(\mathit{come}(j) \wedge \mathit{come}(m))$ and in 4.6 we derive “Not all boys came to the party”. Epistemically it means that we consider those implicatures knowledge of the speaker. This is correct for the examples treated thus far, however, in the context of negation we might want to consider epistemically weaker implicatures. We want ‘not knowledge that’ instead of ‘knowledge that not’.

[6]In Geurts [2007] the author uses the term multiplicatures for more or less the same type of implicatures.

3.7 Comparison with Circumscription

In this section we will see a proof of equivalence between the approach of this chapter using extended logic programming and answer-sets and the solution in Van Rooij and Schulz [2004] using circumscription. This proof is somewhat more technical than the rest of this thesis and can be skipped without missing anything vital for the later chapters.

Although Van Rooij and Schulz [2004]; Schulz and Van Rooij [2006] is definitely not the only recent work on the formalization of implicatures (we already mentioned Sauerland [2004] and Spector [2003]) it is the most advanced. Therefore it is the most obvious to compare the results in this chapter with their work. Moreover, upon studying their results the similarities immediately become apparent.

Circumscription is a form of nonmonotonic reasoning introduced by John McCarthy (McCarthy [1987]). Many authors have expanded on his ideas, a good overview article is Lifschitz [1994].

In Wakaki and Satoh [1997] a translation of circumscription into extended logic programs is provided, however, the resulting equivalence theorem is not good enough for our aims. It only states equivalence between the intersection of the answer-sets of the translated circumscriptive theory and the consequences of the theory itself[7]. We are interested in equivalence between the models of a circumscriptive theory and answer-sets, since our claims about implicatures are based on the contents of the actual answer-sets, not on their intersection. Nonetheless, other results from their paper will be useful for us.

Ordinarily circumscription is a syntactic operation applied to a first-order theory. However, we consider circumscription from a model-theoretic perspective: interpretation in minimal models. There are three types of predicates in circumscription, the predicate that needs to be minimized (P), those predicates that remain fixed (Q) and those that are variable (Z). The first step in comparing circumscription with our answer-sets approach is to define minimal models. It is useful to note again that everywhere in this thesis we are only considering Herbrand models. Furthermore, in contrast to the rest of this thesis, the symbol Π is used as a variable for programs and P refers to the minimized predicate.

Definition 3.17 (Minimal Models). *Let $M_1 = \langle \mathcal{D}, I \rangle$ and $M_2 = \langle \mathcal{D}, J \rangle$ be first order models. $\mathcal{D}$ is a domain of individuals and I and J are interpretation functions for the predicates and relations. We say that with respect to the minimized predicate P and the fixed predicates Q:*

$$\begin{aligned}&M_1 <_{P,Q} M_2, \textit{ iff } I(P) \subset J(P)\\&\quad \textit{and for all other predicates } P' \in Q\textit{: } I(P') = J(P').\end{aligned}$$

M is a minimal model for ϕ with respect to a minimized predicate P and fixed predicates Q iff:

$$M \models \phi, \textit{ and } \neg\exists M' : M' \models \phi \wedge M' <_{P,Q} M.$$

The question predicate in our approach is the minimized predicate P. The other predicates in our program are fixed predicates and thus in Q. In this

[7]It is also unfortunate that the proof of the theorem was omitted from their paper due to space constraints, since it is now only available in an extended version in Japanese, a language in which the author of this thesis, unfortunately, is not proficient.

way, our translation of a question and an answer roughly corresponds to the translation of a circumscriptive theory in Wakaki and Satoh [1997].

We need a simple lemma about the totality of answer-sets for our comparison with circumscription. Since minimal models are total models, the corresponding answer-sets must be total as well.

Lemma 3.3. *If we have an extended logic program Π and we treat every predicate in Π as either a minimized or a fixed predicate (thus for every predicate P in Π we have either $\neg P \leftarrow$ not P and/or $P \leftarrow$ not $\neg P$). Then every answer-set for this program is total, ie. for every literal L it either contains L or $\neg L$.*

Proof. Assume the answer-set I for the program Π for which holds: $I = \Gamma(I, \Pi)$. Now assume towards contradiction that we have a literal L, such that $L \notin I$ and $\neg L \notin I$.

If L is a literal of the minimized predicate, then we know by the assumption $L \notin I$ and the rule $\neg L \leftarrow$ *not* L that $\neg L \in \Gamma(I, \Pi)$, since $\neg L \notin I$, this contradicts the assumption that I is an answer-set.

If L is a fixed predicate, then we know by the assumption $L \notin I$ and the rule $\neg L \leftarrow$ *not* L and the assumption $\neg L \notin I$ and the rule $L \leftarrow$ *not* $\neg L$ that $L, \neg L \in \Gamma(I, \Pi)$, again contradicting the assumption that I is an answer-set.

□

In order to prove equivalence between the minimal models of a circumscriptive theory and the answer-sets of a program resulting from this theory we need to assume that there are no variable predicates. As we see in lemma 3.3, answer-sets of programs in which only minimized and fixed predicates are translated are total, but for programs without this constraint, ie. with variable predicates, this is not the case. To a non-total answer-set there correspond a number of total minimal models. This isn't completely problematic, we could try to prove correspondence modulo variable predicates. However, for the sake of simplicity we omit variable predicates. Also, as far as we have seen in this thesis, we can deal with implicatures without them.

With the above lemma we can prove that every answer-set of a program based on a question and answer is a minimal model of the corresponding answer.

Theorem 3.3. *Let A be a consistent answer to a question about predicate P and $\mathcal{D}$ a domain of individuals. $\Pi = GP(P, A, \mathcal{D})$. Q is the set of predicates in Π without P. Thus $Q \cup \{P\}$ is the set of all predicates in Π.*

If $J \in AS(\Pi)$, then $M = \langle \mathcal{D}, I \rangle$ is a minimal Herbrand model of A with respect to the minimized predicate P and the fixed predicates Q. Where I is defined as:

$$\begin{array}{l} \textit{for every } P' \in \{P\} \cup Q: \\ \quad I(P') = \{\langle d_1, \ldots, d_n \rangle \mid P'(d_1, \ldots, d_n) \in J, d_1, \ldots, d_n \in \mathcal{D}\} \end{array}$$

Proof. By theorem 3.2 we know that the Herbrand models of A are equal to the Herbrand models of $\mathit{EqClean}(\mathit{2Clauses}(\mathit{CNF}(\mathit{2Prop}(A, \mathcal{D}))))$. Thus, when we refer to A this can also be a reference to its translated variant. Suppose that $J \in AS(\Pi)$ and towards contradiction that the model $M = \langle \mathcal{D}, I \rangle$ of A is not a minimal model. We have two options, M is not a model of A or M is a model of A but not minimal. By lemma 3.3 we know that J is a total answer-set, so M is a total model, which is what we need.

(case 1) M is not a model of A. Thus there must be a clause in A which is not true under M. Let this clause be: $\{L_1, \ldots, L_n\}$. Since it is false, it holds that for every $i \leq n$: $M \models \neg L_i$. And also for every $i \leq n$: $\neg L_i \in J$. Since J is an answer-set, $\Gamma(J, \Pi) = J$ and thus for every $i \leq n$: $\neg L_i \in \Gamma(J, \Pi)$. The translation of the clause $\{L_1, \ldots, L_n\}$ results in the set of rules:

$$\{L_i \leftarrow \neg L_1, \ldots, \neg L_{i-1}, \neg L_{i+1}, \ldots, \neg L_n \mid i \leq n\}$$

Now, since every $\neg L_i \in \Gamma(J, \Pi)$, we have by the rules above that every L_i is in $\Gamma(J, \Pi)$. Since $\Gamma(J, \Pi) = J$, every L_i is in J. Thus J contains a contradiction and cannot be an answer-set, which contradicts the assumption that J is an answer-set.

(case 2) M is not minimal. This means that there is a model of A, $M' = \langle \mathcal{D}, I' \rangle$, such that $I'(P) \subset I(P)$ and thus $M' <_{P,Q} M$. The answer-set based on M' is J'. Thus A can be made true with at least one less positive P-literal than that is true in M. Call these literals $L'_1, \ldots, L'_n$. For each of these literals the following holds: $M \models L'_i$ and $M' \models \neg L'_i$. Thus for every clause in A with an L'_i in it there must also be a literal L''_i, $L'_i \neq L''_i$, such that $M' \models L''_i$ holds. We can write this clause as: $\{L'_i, L''_i, L_1, \ldots, L_n\}$, the translation of which, results in, among others, the following two rules:

$$\begin{array}{lcl} L'_i & \leftarrow & \neg L''_i, \neg L_1, \ldots, \neg L_n \\ L''_i & \leftarrow & \neg L'_i, \neg L_1, \ldots, \neg L_n \end{array}$$

We know: $I'(P) \subset I(P)$ and $I'(P') = I(P')$ for all $P' \in Q$. Thus, if L''_i is a positive P-literal, or a Q-literal, then $L''_i \in J$ holds. Hence by $\Gamma(J, \Pi) = J$, $L''_i \in \Gamma(J, \Pi)$ is true. Therefore L'_i cannot be in $\Gamma(J, \Pi)$ by the first rule above, since that would require $\neg L''_i \in \Gamma(J, \Pi)$ to be true. L'_i is a positive P-literal, thus there is no default rule with L'_i in its head. But since $M \models L'_i$, we know that $L'_i \in J$, this means that $\Gamma(J, \Pi) \neq J$, which is a contradiction with the fact that J is an answer-set.

Suppose that L''_i is a negative P-literal. We know $L''_i \in J$ leads to a contradiction by the argument above. Thus we assume that $L''_i \notin J$ is the case. Since $\neg L''_i$ is positive and it is in J and also $L''_i \in J'$ holds, we know that L''_i must be one of the $L'_1, \ldots, L'_n$, say L'_j. Thus $L'_i \in \Gamma(J, \Pi)$ is true if $L'_j \in \Gamma(J, \Pi)$ holds. So, since positive P-literals cannot be in $\Gamma(J, \Pi)$ via a default rule, for an arbitrary L'_i to be in $\Gamma(J, \Pi)$ there needs to be at least another L'_j in $\Gamma(J, \Pi)$ already. For this L'_j the same holds again, or we run into a contradiction as we saw above, etc. Thus, they are all interdependent, and all of them cannot be in $\Gamma(J, \Pi)$, which contradicts the assumption that J was an answer-set.

$\square$

With theorem 3.3 we have one direction of the equivalence that we are looking for. The other direction is a bit more problematic, which the following example will illustrate.

(3.7) Q: $come(x)$?

A: $(come(j) \vee come(m) \vee come(b)) \wedge (come(j) \leftrightarrow come(m))$

Clauses of A: $\{come(j), come(m), come(b)\}, \{\neg come(j), come(m)\},$
$\{come(j), \neg come(m)\}$

It is easy to see that this answer has two minimal models: one in which only Bill comes and one in which John and Mary come. However, the resulting program has only one answer-set.

$$\begin{array}{rcl} come(j) & \leftarrow & \neg come(m), \neg come(b) \\ come(m) & \leftarrow & \neg come(j), \neg come(b) \\ come(b) & \leftarrow & \neg come(m), \neg come(j) \\ \neg come(j) & \leftarrow & \neg come(m) \\ come(m) & \leftarrow & come(j) \\ \neg come(m) & \leftarrow & \neg come(j) \\ come(j) & \leftarrow & come(m) \\ \neg come(x) & \leftarrow & not\ come(x) \end{array}$$

This program does have the answer-set: $\{come(b), \neg come(j), \neg come(m)\}$, but has no answer-set in which both Mary and John come.

A solution to this problem is given in Wakaki and Satoh [1997]. The authors add an extra step to their translation to extended logic programming rules, which we will see below. Let Σ be a set of clauses, then $Th(\Sigma)$ is the set of clauses which are theorems of Σ. Looking at example 3.7, the theorems would be:

$$\{\{come(j), come(m), come(b)\}, \{\neg come(j), come(m)\},$$
$$\{come(j), \neg come(m)\}, \{come(m), come(b)\}, \{come(j), come(b)\}\}.$$

A clause in $Th(\Sigma)$ that is not properly subsumed by any other clause in $Th(\Sigma)$ is called a *characteristic* clause. A clause ϕ is properly subsumed by a clause ψ iff $\psi \subseteq \phi$ and not $\phi \subseteq \psi$. They define $\mu Th(\Sigma)$ as the set of all characteristic clauses in $Th(\Sigma)$. In example 3.7 we would have the following set of characteristic clauses:

$$\{\{come(m), come(b)\}, \{come(j), come(b)\},$$
$$\{\neg come(j), come(m)\}, \{\neg come(m), come(j)\}\}.$$

The translation of these clauses results in the program:

$$\begin{array}{rcl} come(j) & \leftarrow & \neg come(b) \\ come(b) & \leftarrow & \neg come(j) \\ come(m) & \leftarrow & \neg come(b) \\ come(b) & \leftarrow & \neg come(m) \\ \neg come(j) & \leftarrow & \neg come(m) \\ come(m) & \leftarrow & come(j) \\ \neg come(m) & \leftarrow & \neg come(j) \\ come(j) & \leftarrow & come(m) \\ \neg come(x) & \leftarrow & not\ come(x). \end{array}$$

This program does have the two answer-sets that we want:

$$\{come(b), \neg come(j), \neg come(m)\}$$
$$\{\neg come(b), come(j), come(m)\}.$$

The proof of equivalence that we will give, requires the use of characteristic clauses. However, we have refrained from introducing characteristic clauses into our own answer-sets based approach described in the previous sections, because the problem of example 3.7 is a somewhat contrived one and does not occur often in natural language. Furthermore, the procedure to compute characteristic clauses is not at all trivial and computationally quite complex. We do keep in mind that in some cases problems might occur because of this decision. For the proof we need special versions of *Form2Rules* and *GP* that make use of charateristic clauses. We will need a function that translates clauses into characteristic clauses first. Such a function is definitely not trivial, but we only need the fact that it exists for this proof, so we will not spell out the entire definition.

Definition 3.18 (*2CC*)**.** *2CC(σ) gives the characteristic clauses of a set of clauses σ.*

With this auxiliary we can define new versions of *Form2Rules* and *GP*.

Definition 3.19 ($\mathit{Form2Rules}_{CC}$)**.** *The function $\mathit{Form2Rules}_{CC}(\phi, \mathcal{D})$ which translates a first-order logic formula in prenex form ϕ into logic programming rules representing this formula given a finite domain $\mathcal{D}$ is defined as follows:*

$$\mathit{Form2Rules}(\phi, \mathcal{D}) = \mathit{2Rules}(\mathit{2CC}(\mathit{EqClean}(\mathit{2Clauses}(\mathit{CNF}(\mathit{2Prop}(\phi, \mathcal{D}))))))$$

This function may start to look very complicated, but compared to previous definitions only one extra step is added: the conversion to characteristic clauses. Thus, this function still does to same, it applies a number of steps in a row to go from a formula to a set of extended logic programming rules.

The special version of *GP* is trivial.

Definition 3.20 (GP_{CC})**.** *The function $GP_{CC}(Q, A, \mathcal{D})$ takes as input a question predicate Q, a first-order formula A in prenex normal form and a domain of individuals $\mathcal{D}$. It returns an extended logic program. Q' is the set of all the predicates in A that are not Q.*

$$\begin{array}{rcl} GP_{CC}(Q, A, \mathcal{D}) & = & \{\neg Q(c_1, \ldots, c_n) \leftarrow \mathit{not}\ Q(c_1, \ldots, c_n) \mid c_1, \ldots, c_n \in \mathcal{D}\} \\ & \cup & \{R(c_1, \ldots, c_n) \leftarrow \mathit{not}\ \neg R(c_1, \ldots, c_n) \mid \\ & & c_1, \ldots, c_n \in \mathcal{D}, R \in Q'\} \\ & \cup & \{\neg R(c_1, \ldots, c_n) \leftarrow \mathit{not}\ R(c_1, \ldots, c_n) \mid \\ & & c_1, \ldots, c_n \in \mathcal{D}, R \in Q'\} \\ & \cup & \mathit{Form2Rules}_{CC}(A, \mathcal{D}) \end{array}$$

We also need some definitions from Lifschitz [1996] that are used for basic programs. Basic programs are extended logic programs without default literals.

Definition 3.21 (Consequence-operator)**.** *Let Π be a basic program, ie. it contains no default literals, and let X be a set of literals. Then we define the function T from sets of literals to sets of literals as follows:*

$$T_\Pi(X) = \{H \mid H \leftarrow B_1, \ldots, B_n \in \Pi, B_1, \ldots, B_n \in X\}.$$

The set of consequences of a basic program can be defined as follows:

Definition 3.22 (Consequences)**.** *The set $Cn(\Pi)$ of consequences of a basic program Π is:*

$$Cn(\Pi) = \bigcup_{n \geq 0} T_\Pi^n(\emptyset).$$

Where $T_\Pi^0(\emptyset) = \emptyset$, $T_\Pi^1(\emptyset) = T_\Pi(T_\Pi^0(\emptyset))$, $T_\Pi^2(\emptyset) = T_\Pi(T_\Pi^1(\emptyset))$, etc...

As Lifschitz shows in Lifschitz [1996] this set of consequences is the least model of a basic program. Thus in definition 3.11 of the Gamma-operator we could have equivalently used $Cn(\frac{P}{I})$, resulting in the following definition:

Definition 3.23 (The Γ-operator (Alternative definition))**.** *Let P be an extended logic program and I a consistent interpretation. The GL-transformation of P modulo I is the program $\frac{P}{I}$ obtained from P by:*

- *first denoting every objective literal in $\mathcal{H}$ of the form $\neg A$ by a new atom, say $\neg\_A$;*
- *replacing in both P and I these objective literals by their new denotations;*
- *then performing the following operations:*
    - *remove from P all rules which contain a default literal not A such that $A \in I$;*
    - *removing from the remaining rules all default literals.*

$J = Cn(\frac{P}{I})$.
Replace in J the atoms $\neg\_A$ by $\neg A$ and call this interpretation J'. Then $\Gamma(I,P) = J'$.

Using these definitions we can prove an equivalence.

Theorem 3.4. *Let A be a consistent answer to a question about predicate P and $\mathcal{D}$ a domain of individuals. $\Pi = GP_{CC}(P, A, \mathcal{D})$. Q is the set of predicates in Π without P. Thus $Q \cup \{P\}$ is the set of all predicates in Π.*

If $M = \langle \mathcal{D}, I \rangle$ is a minimal Herbrand model of A with respect to the minimized predicate P and the fixed predicates Q, then $J \in AS(\Pi)$, where $J =$

$$\begin{aligned}&\{P'(d_1,\ldots,d_n) \mid \\ &\quad \textit{for every } P' \in Q \cup \{P\}{:}\langle d_1,\ldots,d_n\rangle \in I(P') \wedge d_1,\ldots,d_n \in \mathcal{D}\} \cup \\ &\{\neg P'(d_1,\ldots,d_n) \mid \\ &\quad \textit{for every } P' \in Q \cup \{P\}{:}\langle d_1,\ldots,d_n\rangle \notin I(P') \wedge d_1,\ldots,d_n \in \mathcal{D}\}.\end{aligned}$$

Proof. We know from theorem 3.2 that $\mathit{EqClean}(\mathit{2Clauses}(\mathit{CNF}(\mathit{2Prop}(\phi,\mathcal{D}))))$ and A are equivalent. The further translation to characteristic clauses changes nothing in this respect. So again when refering to A we can also mean the characteristic clauses of A.

Assume that M is a minimal Herbrand model of A and towards contradiction that J is not an answer-set, thus $J \neq \Gamma(J,\Pi)$. We have two possibilities: there is a literal L such that $L \notin J$ and $L \in \Gamma(J,\Pi)$ or $L \in J$ and $L \notin \Gamma(J,\Pi)$.

(case 1) Assume there is a literal L such that $L \notin J$ and $L \in \Gamma(J,\Pi)$.

By induction on the number of applications of the consequence operator we can proof that, if for a literal L' it holds that: $L' \in Cn(\frac{\Pi}{J})^T$, then $L' \in J$ must hold (where the superscript T means that we have translated each literal $\neg\_X$ in $Cn(\frac{\Pi}{J})$ back to $\neg X$). $L' \in Cn(\frac{\Pi}{J})^T$ means by definition that $L' \in (\bigcup_{n\geq 0} T^n_{\frac{\Pi}{J}}(\emptyset))^T$. Thus, $L' \in (T^n_{\frac{\Pi}{J}}(\emptyset))^T$ for some n.

(base) $T^0_\Pi(\emptyset) = \emptyset$, thus it holds trivially. The real first step is $T^1_\Pi(\emptyset) = T_\Pi(\emptyset)$. Now $T_\Pi(\emptyset) = \{H \mid H \leftarrow B_1,\ldots,B_n \in \Pi, B_1,\ldots,B_n \in \emptyset\}$. There is trivially no $B_i \in \emptyset$, thus the only heads H in $T_{\Pi}(\emptyset)$ are facts in the program. There are two ways that we can have facts H in $\frac{\Pi}{J}$. First H can be a singleton clause of A. In this case H must be in J, or M is not a model of A. The second way is if H is a fact resulting from removing the default literal from a rule: $H \leftarrow not\ \neg H$. If the default literal is removed, then this means that $\neg H \notin J$, which implies, since J is total and non-contradictory (because it is based on a total model M), that $H \in J$ holds.

(induction) The induction hypothesis is the following: if $L' \in (T^n_{\frac{\Pi}{J}}(\emptyset))^T$, then $L' \in J$. Now we need to show for $n+1$, that if $L' \in (T^{n+1}_{\frac{\Pi}{J}}(\emptyset))^T$ then $L' \in J$ holds. We recall that $T^{n+1}_{\frac{\Pi}{J}}(\emptyset) = T_{\frac{\Pi}{J}}(T^n_{\frac{\Pi}{J}}(\emptyset))$. Furthermore, $T_{\frac{\Pi}{J}}(T^n_{\frac{\Pi}{J}}(\emptyset)) = \{H \,|\, H \leftarrow B_1, \ldots, B_n \in \Pi, B_1, \ldots, B_n \in T^n_{\frac{\Pi}{J}}(\emptyset)\}$. Now, for every B_i that is in $T^n_{\frac{\Pi}{J}}(\emptyset)$ we know by the induction hypothesis that $B_i \in J$. Thus, since J is total and non-contradictory, $\neg B_i \notin J$. The rule: $H \leftarrow B_1, \ldots, B_n$ was a translation of the clause: $\{H, \neg B_1, \ldots, \neg B_n\}$. Since M is a model for this clause, and we know that $\neg B_1, \ldots, \neg B_n$ are not in J and thus they are also not true in M, it must be that H is true in M and therefore $H \in J$.

By the alternative definition of Γ with Cn, we know that $L \in \Gamma(J, \Pi)$ is equal to claiming that $L \in Cn(\frac{\Pi}{J})^T$. Now we have shown that if $L \in Cn(\frac{\Pi}{J})^T$ then $L \in J$ must be true, which contradicts our assumption that L is not in J.

(case 2) Assume there is a literal L such that $L \in J$ and $L \notin \Gamma(J, \Pi)$.

Suppose that L is a literal of a fixed predicate. Since J is total and non-contradictory, we know that if $L \in J$, then $\neg L \notin J$, thus by the default rule: $L \leftarrow not\ \neg L$, which exists for a fixed predicate, L must be in $\Gamma(J, \Pi)$. This contradicts our assumption. If L is a negative P-literal, then we know that $L \in \Gamma(J, \Pi)$ holds by the rule $L \leftarrow not\ \neg L$, which exists for negative P-literals. Again, this is contradicting our assumption.

The last option is that L is a positive P-literal. From all the minimal models of A we can construct a special formula. Suppose we have n minimal models. Then the formula ϕ_i is a conjunction of all the positive P-literals true in the i-th minimal model. The disjunction of all these ϕ_i we call ψ. Thus ψ looks like: $\phi_1 \vee \ldots \vee \phi_n$. This ψ is a theorem of A. If it isn't then there should be a model of A in which ψ doesn't hold. Take this model to be M'. The set of every positive P-literal true in M' is a superset of the set of positive P-literals true in some minimal model M_i of A. Therefore ϕ_i must still be true in M' and thus ψ is true in M', which gives a contradiction.

We can take the CNF of ψ and translate the result into clauses. Lets call the set of these clauses Δ. It is easy to see that each clause in Δ contains at most a number of literals equal to the number of minimal models of A. Now since these clauses are theorems of A, there must be clauses that are at least as strong as these in Σ. There cannot be a clause that is stronger in Σ. Suppose we have such a clause σ. Then there is a $\delta \in \Delta$ such that $\sigma \subset \delta$ holds. Now for each M_i (minimal model of A) it holds that there is an $L_i \in \delta$ such that $M_i \models L_i$, since every minimal model must satisfy δ. Suppose that there is only one such L_i for each M_i, thus one literal can be true in multiple models, but there is not more than one literal true in a model. Since $\sigma \subset \delta$ holds, there must be an L_j such that $L_j \in \delta$ and $L_j \notin \sigma$. However for this L_j it holds that there is a model M_j, such that $M_j \models L_j$ is true. Now σ is assumed to be a theorem of A, but then M_j cannot be a model of A, since $M_j \not\models \sigma$, which is a contradiction. It might be however, that there is a model M_m such that there are multiple $L_m \in \delta$ for which holds: $M_m \models L_m$. For such an L_m it can hold that $L_m \notin \sigma$ and $L_m \in \delta$, while σ remains true in all minimal models. Because the number of literals in σ is less than the number of minimal models of A, there must be an $L_{kl} \in \sigma$ such that there are two minimal models M_k and M_l which make L_{kl} true. This means that L_{kl} is in the conjunctions ϕ_k and ϕ_l. Which implies by the construction of the clauses in Δ via a conjunctive normal form of ψ that σ is also a clause in Δ.

Thus, every clause in Δ that is not properly subsumed by another clause in Δ is also in Σ. Suppose without loss of generality that our literal L, $L \in J$ and $L \notin \Gamma(J, \Pi)$, is in ϕ_1, thus $M_1 \models L$ is true. And we assume also without loss of generality that $M = M_1$. For every $i \neq 1$ we can find a positive P-literal L', such that $M_i \models L'$ (and thus L' is in ϕ_i) and $M_1 \models \neg L'$ hold. Suppose that we cannot do this for some i, then M_i would be more minimal than M_1, since M_1 makes the same literals true as M_i, and we can't have $M_1 = M_i$ it must be that $M_i \subset M_1$ holds, which contradicts the fact that M_1 is a minimal model. With these L' we can construct the disjunction: $L \vee L'_1 \vee \ldots \vee L'_n$ (note that some L'_i can be equal). The clause ω of this disjunction is in Δ. None of the L'_i is true in M_1. Though it might be that a clause properly subsuming ω is in Σ and not ω itself, it is always the case that such a clause will contain L which is true in M_1 and thus in M and literals $L''_1, \ldots L''_n$, that are false in M_1 and thus false in M. The fact that there is such a clause in Σ means that there will be a rule in our program looking like:

$$L \quad \leftarrow \quad \neg L''_1, \ldots, \neg L''_n$$

The literals $\neg L''_1, \ldots, \neg L''_n$ are all negative P-literals. Since $L''_1, \ldots L''_n$ are false in M, it holds that $\neg L''_1, \ldots, \neg L''_n \in J$. And thus by: $\neg P \leftarrow \textit{not}\ P$, they must be in $\Gamma(J, \Pi)$. From which follows that $L \in \Gamma(J, \Pi)$ must be true, which contradicts our assumption.

□

The following theorem stating equivalence between answer-sets and the minimal models of circumscription in the context of a question and an answer is now easy to prove.

Theorem 3.5. *Let A be a consistent answer to a question about predicate P and $\mathcal{D}$ a domain of individuals. $\Pi = GP_{CC}(P, A, \mathcal{D})$. Q is the set of predicates in Π without P. Thus $Q \cup \{P\}$ is the set of all predicates in Π.*

$M = \langle \mathcal{D}, I \rangle$ is a minimal Herbrand model of A with respect to the minimized predicate P and the fixed predicates Q, iff $J \in AS(\Pi)$, where $J =$

$$\begin{aligned}
&\{P'(d_1, \ldots, d_n) \mid \\
&\quad \textit{for every } P' \in Q \cup \{P\} \colon \langle d_1, \ldots, d_n \rangle \in I(P') \wedge d_1, \ldots, d_n \in \mathcal{D}\} \cup \\
&\{\neg P'(d_1, \ldots, d_n) \mid \\
&\quad \textit{for every } P' \in Q \cup \{P\} \colon \langle d_1, \ldots, d_n \rangle \notin I(P') \wedge d_1, \ldots, d_n \in \mathcal{D}\}.
\end{aligned}$$

Proof. The proof is trivial from theorems 3.3 and 3.4, which each provide a direction of the equivalence. Theorem 3.3 is even a bit stronger than necessary since it doesn't depend on characteristic clauses. □

We conclude that the approach described in this chapter has a very close connection to a circumscription based solution. The main difference is the fact that our approach distinguishes between facts already derivable using topdown derivations and facts that are only derivable when considering all the possibilities, ie. all answer-sets or minimal models. In the next chapter we will see a refinement of the solutions in this chapter that allows for more implicatures, but we will also loose the strong connection with circumscription.

Chapter 4

Advanced Approach

Already in chapter two we mentioned that not all the literature agrees with the view that scalar implicatures are epistemically as strong as we derive them in chapter three. In example 3.3 there can be contexts in which the implicature: "The answerer does not know that John and Mary come" might be more appropriate than the stronger: "(The answerer knows that) John and Mary don't come both". This will become even more clear in the context of negation, which we will see in the example below. The goal of this chapter is to provide a solution that can deal with both weak and strong epistemic interpretations via the introduction of a different declarative semantics, called: WFSX.

4.1 "John didn't come."

Let us consider the following example with negation.

(4.1) Q: "Who came to the party?"
A: "John didn't come to the party."
Imp (a): "All other people came to the party."
Imp (b): "It is possible that other people came to the party."

As mentioned in Van Rooij and Schulz [2004] the literature is divided as to what is the correct implicature in this example. Is it the strong variant (a), or the epistemically weaker (b)? (a) can be derived in our approach by considering $\neg come$ as our question predicate, a suggestion also found in Van Rooij and Schulz [2004].

$$P_{4.1} = GP(\neg come(x), \neg come(j), \{j, m, b\}, \{\}) =$$

$$\begin{array}{rcl} \neg come(j) & & \\ come(m) & \leftarrow & not\ \neg come(m) \\ come(j) & \leftarrow & not\ \neg come(j) \\ come(b) & \leftarrow & not\ \neg come(b) \end{array}$$

The answer-set of $P_{4.1}$ is:

$$\{\neg come(j), come(b), come(m)\}.$$

And clearly this is also the outcome of $\mathit{SLDNF}(P_{4.1})$. Thus we derive the exhaustivity implicature with respect to $\neg \mathit{come}$, which is variant (a).

As mentioned before, answer-sets intuitively represent different possibilities of interpreting the program. Under different assumptions of the truth of the default literals the program behaves differently. With this in mind, we modify our closed world assumption (CWA) into something weaker, which would allow us to derive variant (b). Suppose that we treat the question predicate P the same as the other predicates:

$$\begin{array}{rcl} P(\vec{x}) & \leftarrow & \mathit{not}\ \neg P(\vec{x})) \\ \neg P(\vec{x}) & \leftarrow & \mathit{not}\ P(\vec{x})) \end{array}$$

In this example this new approach would give the program $P'_{4.1}$:

$$\begin{array}{rcl} \neg \mathit{come}(j) & & \\ \mathit{come}(m) & \leftarrow & \mathit{not}\ \neg \mathit{come}(m) \\ \mathit{come}(j) & \leftarrow & \mathit{not}\ \neg \mathit{come}(j) \\ \mathit{come}(b) & \leftarrow & \mathit{not}\ \neg \mathit{come}(b) \\ \neg \mathit{come}(m) & \leftarrow & \mathit{not}\ \mathit{come}(m) \\ \neg \mathit{come}(j) & \leftarrow & \mathit{not}\ \mathit{come}(j) \\ \neg \mathit{come}(b) & \leftarrow & \mathit{not}\ \mathit{come}(b) \end{array}$$

Via SLDNF we can only derive the obvious $\neg \mathit{come}(j)$. The answer-sets of this program are more interesting.

$$AS(P'_{4.1}) =$$

$$\begin{array}{c} \{\neg \mathit{come}(j), \mathit{come}(m), \mathit{come}(b)\} \\ \{\neg \mathit{come}(j), \neg \mathit{come}(m), \mathit{come}(b)\} \\ \{\neg \mathit{come}(j), \mathit{come}(m), \neg \mathit{come}(b)\} \\ \{\neg \mathit{come}(j), \neg \mathit{come}(m), \neg \mathit{come}(b)\} \end{array}$$

The answer-sets tell us that as far as Bill and Mary are concerned, the speaker keeps all options open: Mary might come, or Mary might not come and the same is true for Bill. This solution is perhaps a bit too strong. Because answer-sets are total, Bill and Mary either come or they don't, but it might be nice to have an option of undefinedness for not coming. It would be nice if we could strengthen the interpretation to a strong one, ie. the first answer-set. With the current solution it is not clear how, since by definition answer-sets cannot be subsets of each other, it is difficult to define a relation between them to model this. If we would have some form of partial models, then a nicer solution might be achieved. The more complex WFSX-semantics provide this for us.

4.2 WFSX - Well-founded Semantics with Explicit Negation

We will see that the declarative and procedural semantics described below are going to incorporate both SLDNF and answer-sets. They can provide the results gained with SLDNF and answer-sets, but can do more, since they allow for partial models. These semantics are the final semantics of our choice.

Alferes and Pereira (Alferes and Pereira [1996]) introduce a declarative semantics for extended logic programs based on the well-founded semantics for normal logic programs (van Gelder *et al.* [1991]). They dub this semantics: WFSX, Well-Founded Semantics with eXplicit negation.

4.2.1 WFSX

To introduce the WFSX semantics we need some new and some slightly different definitions compared to those for answer-sets. Thus some concepts are redefined, but it will always be clear whether the concepts from answer-sets or from WFSX are used.

Definition 4.1 (Interpretation)**.** *An interpretation I of a language $\mathcal{L}$ is any set*

$$T \cup not\ F^1$$

where T and F are disjoint subsets of objective literals over the Herbrand base of $\mathcal{L}$, and:

$$\textit{if } \neg L \in T \textit{ then } L \in F \textit{ (Coherence Principle)}$$

The set T contains all ground objective literals true *in I and the set F contains all ground objective literals* false *in I. The truth value for the remaining objective literals is* undefined.

We look again at the simple example program $P_{3.4}$:

(4.2)

$$\begin{array}{rcl} a & \leftarrow & b, c \\ b & \leftarrow & not\ \neg b \\ \neg c & \leftarrow & \neg a \end{array}$$

We can easily see that $I_1 = \{a, b, not\ c\}$ and $I_2 = \{\neg a, b, c, not\ a, not\ \neg c\}$ are interpretations of the program $P_{3.4}$.

Again, as with answer-sets, it is easy to see that an interpretation I can be viewed as a function $I : \mathcal{H} \rightarrow V$ where $V = \{0, u, 1\}$.

$$I(A) = \begin{cases} 1 & \text{if } A \in I \\ 0 & \text{if } not\ A \in I \\ u & \text{otherwise} \end{cases}$$

Based on this we can define a truth valuation function for formulae. Although Alferes and Pereira never mention this, the third truth value behaves in the sense of Kleene's strong three-valued logic. However, the definition of $\leftarrow$ differs from the usual interpretation of implication in this logic.

Definition 4.2 (Truth valuation)**.** *Let I be an interpretation and C the set of all formulae of the language, then the function $\hat{I} : C \rightarrow V$ is the truth valuation corresponding to I. $\hat{I}$ is recursively defined as follows:*

- *$\hat{I}(L) = I(L)$, if L is an objective literal.*

[1]where $not\ \{a_1, ..., a_n\}$ stands for $\{not\ a_1, .., not\ a_n\}$

- $\hat{I}(not\ L) = 1 - I(L)$, *if not L is a default literal.*
- $\hat{I}((S,T)) = min(\hat{I}(S), \hat{I}(T))$, *if S and T are formulae.*
- $\hat{I}(L \leftarrow S) = \begin{cases} 1 & \text{if } \hat{I}(S) \leq \hat{I}(L) \text{ or } \hat{I}(\neg L) = 1 \text{ and } \hat{I}(S) \neq 1 \\ 0 & \text{otherwise} \end{cases}$

  if L is an objective literal and S a formula.

Take $\hat{I}_1$ to be the truth valuation function based on I_1, then, for instance: $\hat{I}_1(\neg c \leftarrow \neg a) = 1$ and $\hat{I}_1(not\ \neg b) = u$. In the case of $\hat{I}_2$, we see: $\hat{I}_2(\neg c \leftarrow \neg a) = 0$.

The definition of a model using the function $\hat{I}$ remains the same as with answer-sets. But not every such model is a model according to the WFSX semantics. To define WFSX a couple of extra notions are required: an extended definition of the modulo transformation and a definition of the least and coherence operators. To be able to give these, we need to expand the language with a proposition $\mathbf{u}$ such that for every interpretation I, $I(\mathbf{u}) = u$.

Definition 4.3 ($\frac{P}{I}$ transformation)**.** *Let P be an extend logic program, I an interpretation, then $\frac{P}{I}$ is the program obtained from P by performing the following four operations:*

1. *Remove from P all rules containing a default literal not A such that $A \in I$.*
2. *Remove from P all rules which in the body contain an objective literal L such that $\neg L \in I$.*
3. *Remove from all remaining rules of P their default literals not A such that not $A \in I$.*
4. *Replace all the remaining default literals in P by* $\mathbf{u}$.

As an example, let us see what the result of $\frac{P_{3.4}}{I_1}$ is:

$$\begin{array}{rcl} a & \leftarrow & b, c \\ b & \leftarrow & \mathbf{u} \end{array}$$

Notice that by the definition the resulting program is always non-negative, its literals are either objective or $\mathbf{u}$. This is necessary to be able to apply the *least* operator.

Definition 4.4 (Least operator)**.** *least(P), where P is a non-negative program, is defined as the set of literals $T \cup not\ F$ obtained in the following manner:*

- *Replace in P every negative objective literal $\neg L$ by a new atomic symbol, say L' and call this program P'.*
- *Let $I = T' \cup not\ F'$ be the model of P' such that there is no model J for which holds that $J \neq I$ and there is a ground atom A such that $I(A) > J(A)$, this means that I is the least 3-valued model of P'*[2].
- *We get $T \cup not\ F$ from I by reversing the first replacement.*

[2]This definition of least 3-valued model is non-constructive, but short. Alferes and Pereira also provide a constructive definition using a fixed-point operator, similar to the consequence operator for answer-sets that we saw in section 3.7.

least$(\frac{P_{3.4}}{I_1})$ is:

$$\{not\ a, not\ \neg a, not\ \neg b, not\ c, not\ \neg c\}$$

Definition 4.5 (Coherence operator)**.** *$Coh(QT \cup not\ QF) = T \cup not\ F$, if QT does not contain a pair of contradictory objective literals A and $\neg A$. The interpretation $T \cup not\ F$ is obtained as follows:*

$$T = QT \text{ and } F = QF \cup \{\neg L | L \in T\}.$$

The coherence operator applied to *least*$(\frac{P_{3.4}}{I_1})$ results in the same set.

All the operators from above are put together into one final operator.

Definition 4.6 (Φ-operator)**.** *Let P be a logic program, I an interpretation and $J = least(\frac{P}{I})$. If $Coh(J)$ exists, then $\Phi(I, P) = Coh(J)$. Otherwise $\Phi(I, P)$ is undefined.*

Using the Φ-operator we define the WFSX semantics.

Definition 4.7 (*WFSX*; PSM and WFM)**.** *We call an interpretation I of a program P a Partial Stable Model (PSM) of P iff*

$$\Phi(I, P) = I$$

If there is no J among the PSM's of P such that $J \subset I$, then I is the F-least[3] *PSM and this is called the Well Founded Model (WFM) of P. The WFM is the intended meaning of P.*

$WFSX(P)$ is the set of all the PSMs of P.

We already saw that I_1 is not a PSM of $P_{3.4}$, since $I_1 \neq Coh(least(\frac{P_{3.4}}{I_1}))$. Let us see if $I_3 = \{not\ a, not\ \neg a, b, not\ \neg b, not\ c, not\ \neg c\}$ is. The first step is to calculate $\frac{P_{3.4}}{I_3}$:

$$\begin{array}{rcl} a & \leftarrow & b, c \\ b & \leftarrow & \\ \neg c & \leftarrow & \neg a \end{array}$$

When we apply the *least*-operator the result is:

$$\{not\ a, not\ \neg a, b, not\ \neg b, not\ c, not\ \neg c\}$$

Applying the *Coh*-operator does not change this result. Thus we conclude that $I_3 = Coh(least(\frac{P_{3.4}}{I_3}))$ and that therefore I_3 is a PSM of $P_{3.4}$.

Alferes and Pereira show that every noncontradictory program always has a unique WFM and that there exists a bottom-up and top-down procedure to compute it. The WFM is equal to the intersection of all PSMs. The relation between answer-sets and WFSX is given by the following theorem.

Theorem 4.1. *Let P be and extended logic program, then:*

$$\text{if } J \in AS(P), \text{ then } J \cup \{not\ L \mid L \notin J\} \in WFSX(P).$$

Furthermore, suppose P has an answer-set. Then WFSX is sound with respect to answer-set semantics. Thus, for every literal L:

$$L \in WFSX(P) \Rightarrow L \in AS(P)$$

[3]Least in the so-called Fitting-order.

We see that WFSX generalizes answer-sets semantics: it assigns meaning to the same and more programs than answer-sets. WFSX also has the very nice property that the PSMs under set inclusion are organized into a downward complete semilattice, its least element being the WFM. Alferes and Pereira also show that if a program has answer-sets, then the maximal elements in this semi-lattice are the answer-sets of the program. This will prove a very useful property.

4.2.2 SLX

Alferes and Pereira also define a top down derivation procedure, in the vein of SLDNF, called SLX. SLX is somewhat more complex than SLDNF and we will refrain from giving its precise definitions. The most important advantage of SLX over SLDNF is the ability to deal with loops. This allows Alferes and Pereira to prove that SLX is sound and theoretically complete with respect to the WFM of a program. Thus everything that is derivable using SLX is also in the WFM of the program. This differs greatly with respect to SLDNF, we cannot derive everything that is in the WFM (or intersection of the answer-sets) with SLDNF. We are not really interested in what the top down derivations look like, except for illustrative purposes, we just want to know which literals are derivable top down. So, whenever we want to known those literals we just look at the WFM. We will see that the distinction in the previous chapter of looking at SLDNF or answer-sets is replaced by the distinction of looking at just the WFM or looking at all the PSMs of a program.

4.2.3 Computational Complexity

Without going into the full details of computational complexity theory, it is relevant to look into the matter with regard to the WFSX semantics. The field of computational complexity is concerned with the computational difficulty of computable functions. This complexity is studied using so-called decision problems, ie. problems that have a yes/no answer as output. For example: for a certain natural number x, is x a prime number? Different classes of complexity are distinguished on the basis of the relation between the size of the input of a decision problem and the time and space required to solve it. When this relation is polynomial, ie. describable by a function in which the dominant term is of the form n^c, where n represents the size of the input and c a constant, then the problem is in the complexity class P. Usually problems in P are considered to be tractable, meaning that they can be solved well in practice. When the computation costs grow faster in the size of the input, for instance exponential, then solving such a problem becomes intractable. This means that, although we might be able to define algorithms to solve them, they require a very long running time.

For logic programming the decision problem is the following: does a literal L belong to the chosen semantics of the program P? For full extended logic programming with functions the decision problem is very complex, far beyond the class P, both for the WFSX and the answer-set semantics (Dantsin *et al.* [1997]). However, when considering only grounded functionless programs, as in this thesis, things are brighter. For the WFSX semantics the decision problem is polynomial (Alferes and Pereira [1996]). The answer-set semantics are more complex, since the decision problem is Co-NP-complete. This seems a difficult label, but for us the relevant thing is that at the moment there exist only super polynomial (greater than polynomial) algorithms to solve the decision problem and it is highly debatable that we will do better in the future[4].

The fact that the decision problem for the WFSX semantics has such nice computational properties is due to the fact that the well founded model (WFM) of a program is bottom-up computable in polynomial time (Alferes and Pereira [1996]) and we only need the WFM to solve the decision problem. However, to solve the decision problem for the answer-set semantics we require the computation of all the answer-sets, since we need to take their intersection to determine whether a literal is true or not. To get all the answer-sets we cannot be a lot more efficient than just trying out all possible interpretations as answer-sets.

Unfortunately nothing is said in the literature about the complexity of computing the entire semilattice of partial stable models (PSMs) under the WFSX semantics, mostly because of the fact that it isn't very interesting to many people. However, it is interesting for this thesis since the derivation of a lot of implicatures depends on them. Based on the decision problems of answer-sets it seems reasonable to conjecture that the complexity of computing all the PSMs is the same as the complexity of computing all the answer-sets. Both the number of answer-sets and the number of PSMs is exponential in the amount of literals and furthermore the procedures to compute them are similar (not much better than just testing all possibilities).[5].

The complexity of SLX is never clearly stated by the authors of Alferes and Pereira [1996]. Though it is mentioned that it is computationally more complex than computing the WFM bottom-up (Alferes *et al.* [1995]). The lack of information about the complexity of SLX is another reason for not giving the definitions in the previous section. The important point from this section is that there is a clear difference in computational complexity between the WFM, which is computable top-down, but more efficiently bottom-up, and the complete semilattice of PSMs, for which there are only super polynomial algorithms.

4.3 Strong and Weak Epistemics

When it comes to interpreting WFSX semantics, we keep the view introduced with answer-sets: partial stable models represent different possible worlds of the speaker. Only now, literals can have the truth-value undefined, which allows for more possible worlds. Furthermore, a PSM can be a subset of another PSM, removing another constraint of answer-sets.

4.3.1 Weak Epistemic Interpretation

In our quest to define a weaker interpretation of an answer by using WFSX, we need to remain able to provide for the strong interpretation as well. Since there are more, or just as much, partial stable models as answer-sets for a program, the thought arises that considering all PSMs as possible worlds is epistemically weaker than considering just the answer-sets. Thus, this might just be what we need to provide for a weaker epistemic account. Nonetheless, we still want to

[4]It all depends on whether P equals NP.

[5]If the complexity would clearly be better, one would expect to find so in the literature.

be able to derive the strong results that answer-sets give us. Fortunately, we already saw that if a program has answer-sets, then the maximal elements in the semilattice of PSMs correspond to these answer-sets. Thus if we only consider those PSMs, then we have our original strong results. To make this notion a bit more formal let us define a simple operator that selects the maximal elements in the semilattice of PSMs. Note that in general there might be no maximal elements, however, the examples that we consider in this thesis all have maximal elements.

Definition 4.8 (MaxEl operator)**.** *Suppose that $WFSX(P)$ is the set of partial stable models of a program P under the WFSX-semantics. Then:*

$$MaxEl(WFSX(P)) = \{M \mid \neg\exists M' \in WFSX(P) : M \subset M' \wedge M \neq M'\}.$$

Considering all the PSMs is not yet enough to get a weaker interpretation. Recall that we have the proposition **u** in our language, which always has the truth-value u. We are going to use this proposition to get an epistemically weaker program. In our weaker gricean logic program we are going to introduce the following rules for the question predicate instead of the original CWA assumption:

$$\begin{array}{rcl} \neg Q(\vec{x}) & \leftarrow & not\, Q(\vec{x})) \\ Q(\vec{x}) & \leftarrow & \mathbf{u} \end{array}$$

Intuitively, what these rules do is state that by default either an individual doesn't have property Q, or we are undecided about it. Based on this idea we also define a function GP_{weak}.

Definition 4.9 (GP_{weak})**.** *The function $GP_{weak}(Q, A, \mathcal{D})$ takes as input a question predicate Q, a first-order formula A in prenex normal form and a domain of individuals $\mathcal{D}$. It returns an extended logic program. Q' is the set of all the predicates in A that are not Q.*

$$\begin{array}{rcl} GP_{weak}(Q, A, \mathcal{D}) & = & \{\neg Q(c_1, \ldots, c_n) \leftarrow not\, Q(c_1, \ldots, c_n) \mid c_1, \ldots, c_n \in \mathcal{D}\} \\ & \cup & \{Q(c_1, \ldots, c_n) \leftarrow \mathbf{u} \mid c_1, \ldots, c_n \in \mathcal{D}\} \\ & \cup & \{R(c_1, \ldots, c_n) \leftarrow not\, \neg R(c_1, \ldots, c_n) \mid \\ & & c_1, \ldots, c_n \in \mathcal{D}, R \in Q'\} \\ & \cup & \{\neg R(c_1, \ldots, c_n) \leftarrow not\, R(c_1, \ldots, c_n) \mid \\ & & c_1, \ldots, c_n \in \mathcal{D}, R \in Q'\} \\ & \cup & Form2Rules(A, \mathcal{D}) \end{array}$$

The weaker closed world assumption (CWA) introduced above is only useful when we consider all the PSMs. When we only look at the maximal elements, which in our examples correspond to answer-sets, then we notice that these do not contain literals that are undefined. So we might as well have had the original CWA, without the $Q(x) \leftarrow \mathbf{u}$ rule. This rule is basically ineffective because in an answer-set $Q(x)$ is either true or false, not undefined.

There might not be a difference between the original CWA and the weaker one when considering the maximal elements, there is, however, a difference when we consider the minimal element, ie. the well founded model. Furthermore, under the original CWA more literals can be derived with SLX than we could with SLDNF.

This new approach gives us three levels of epistemic strength. We derive the strongest interpretation by applying *MaxEl* to either GP or GP_{weak}. Considering all the PSMs of GP_{weak} gives the weakest epistemic interpretation, and looking at all the PSMs of GP is somewhere in between.

To illustrate these three levels we shall consider both GP and GP_{weak} of example 3.3 (repeated here for convenience).

(4.3) Q: "Who came to the party?"
A: "John or Mary came."
Imp: "John or Mary, but not both came."
"No one else but John or Mary came."

Let us call $P_{3.3}$ the program obtained using GP and $P_{3.3-weak}$ the program obtained using GP_{weak}. The PSMs in $WFSX(P_{3.3})$ form the following semilattice, where C stands for *come*:

$$\begin{tikzcd}
\begin{array}{c}\{\neg C(j), \mathit{not}\ C(j),\\ C(m), \mathit{not}\ \neg C(m),\\ \neg C(b), \mathit{not}\ C(b)\}\end{array} & & \begin{array}{c}\{C(j), \mathit{not}\ \neg C(j),\\ \neg C(m), \mathit{not}\ C(m),\\ \neg C(b), \mathit{not}\ C(b)\}\end{array} \\
& \{\neg C(b), \mathit{not}\ C(b)\} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

The semilattice of the PSMs in $WFSX(P_{3.3-weak})$ is bigger:

$$\begin{tikzcd}
\begin{array}{c}\{\neg C(j), \mathit{not}\ C(j),\\ C(m), \mathit{not}\ \neg C(m),\\ \neg C(b), \mathit{not}\ C(b)\}\end{array} & & \begin{array}{c}\{C(j), \mathit{not}\ \neg C(j),\\ \neg C(m), \mathit{not}\ C(m),\\ \neg C(b), \mathit{not}\ C(b)\}\end{array} \\
\begin{array}{c}\{\neg C(j), \mathit{not}\ C(j),\\ C(m), \mathit{not}\ \neg C(m)\}\end{array} \arrow[u] & \{\neg C(b), \mathit{not}\ C(b)\} \arrow[ul] \arrow[ur] & \begin{array}{c}\{C(j), \mathit{not}\ \neg C(j),\\ \neg C(m), \mathit{not}\ C(m)\}\end{array} \arrow[u] \\
& \{\} \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

In both semilattices the maximal elements are the same, they correspond to the answer-sets we found earlier and they give the answer-set account that we know: either John or Mary came to the party and we are sure that Bill didn't come. We will dub this the strong epistemic interpretation. When we look at the semilattice for $P_{3.3}$ we see that every PSM contains the fact that Bill didn't come. Thus this is an implicature in this interpretation. Basically, this implicature is derivable by only considering the well founded model, since everything that holds in the WFM holds in all other PSMs too. When it comes to John and Mary, the implicature that we can derive is weaker: "The speaker doesn't know that John and Mary came", since the WFM isn't decided about whether John or Mary comes. Finally, what we will call the epistemically weak interpretation is derived by considering all the PSMs of $P_{3.3-weak}$. Next to the weaker implicature mentioned above, we derive a weaker implicature regarding Bill: "The speaker doesn't know that Bill didn't come".

4.3.2 "John didn't come." - revisited

Let us look at example 4.1 under the WFSX-semantics using GP_{weak}.

$$P''_{4.1} = GP_{weak}(\neg come(x), \neg come(j), \{j, m, b\}, \{\}) =$$

$$\begin{array}{rcl}
\neg come(j) & & \\
come(m) & \leftarrow & not\ \neg come(m) \\
come(j) & \leftarrow & not\ \neg come(j) \\
come(b) & \leftarrow & not\ \neg come(b) \\
\neg come(m) & \leftarrow & \mathbf{u} \\
\neg come(j) & \leftarrow & \mathbf{u} \\
\neg come(b) & \leftarrow & \mathbf{u}
\end{array}$$

The WFSX of $P''_{4.1}$ can be represented by the following semilattice:

$$\begin{tikzcd}
 & \begin{array}{c}\{\neg C(j), not\ C(j), \\ C(m), not\ \neg C(m), \\ C(b), not\ \neg C(b)\}\end{array} & \\
\begin{array}{c}\{\neg C(j), not\ C(j), \\ C(m), not\ \neg C(m)\}\end{array} \arrow[ur] & & \begin{array}{c}\{\neg C(j), not\ C(j), \\ C(b), not\ \neg C(b)\}\end{array} \arrow[ul] \\
 & \{\neg C(j), not\ C(j)\} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

This is exactly the weaker interpretation that we were looking for. The speaker considers it possible that Bill comes and that Mary comes. Furthermore, we can consider the maximal elements of this semilattice with *MaxEl*. In this case this is the top element:

$$\{\neg C(j), not\ C(j), C(m), not\ \neg C(m), C(b), not\ \neg C(b)\}$$

Which is the strong interpretation that we derived earlier: both Mary and Bill come.

4.3.3 "John *and* Mary didn't come."

Somewhat more complicated is the following example with a conjunction under a negation. That we intend the conjunction to fall under the negation is accentuated by the emphasis on "and".

(4.4) Q: "Who came to the party?"

A: "John *and* Mary didn't come to the party." (It is not the case that John *and* Mary came to the party.)

Imp (a): "John or Mary came to the party and all other people came too."

Imp (b): "It is possible that John or Mary came to the party and other people too."

The translation of the answer suggested by the emphasis on "and" is the following: $\neg(come(j) \wedge come(m))$[6]. Thus $P_{4.4} =$

$$\begin{array}{rcl}
\neg come(j) & \leftarrow & come(m) \\
\neg come(m) & \leftarrow & come(j) \\
come(m) & \leftarrow & not\ \neg come(m) \\
come(j) & \leftarrow & not\ \neg come(j) \\
come(b) & \leftarrow & not\ \neg come(b) \\
\neg come(m) & \leftarrow & \mathbf{u} \\
\neg come(j) & \leftarrow & \mathbf{u} \\
\neg come(b) & \leftarrow & \mathbf{u}.
\end{array}$$

It has the following semilattice of PSMs.

$$\begin{tikzcd}
\{\neg C(j), not\ C(j), C(m), not\ \neg C(m), C(b), not\ \neg C(b)\} & & \{\neg C(m), not\ C(m), C(j), not\ \neg C(j), C(b), not\ \neg C(b)\} \\
\{\neg C(j), not\ C(j), C(m), not\ \neg C(m)\} \arrow[u] & \{C(b), not\ \neg C(b)\} \arrow[ul] \arrow[ur] & \{\neg C(m), not\ C(m), C(j), not\ \neg C(j)\} \arrow[u] \\
& \{\} \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

This set of PSMs clearly represents the expected weak interpretation. It is possible that John or Mary came to the party and it is also possible that Bill came. The two maximal elements allow for the derivation of the strong scalar implicature: "John or Mary came to the party", and furthermore the exhaustive implicature: "All other people came to the party".

4.4 Dealing with Modalities

We can apply the approach for dealing with negations to examples containing modalities as well. But we must keep it simple. Therefore we only look at the epistemic modality: "possibly", such as in the following example[7].

(4.5) Q: "Who is coming to the party?"

A: "Possibly John or Mary is coming."

Imp: "It is impossible that John and Mary come."

The use of the epistemic modality "possibly" refers to the knowledge state of the answerer. Thus it says something about the worlds that the answerer holds

[6]We stress this interpretation so much because $\neg come(j) \wedge \neg come(m)$ has no interesting implicature.

[7]There are approaches to logic programming that incorporate modal operators into the language, see, for instance, Baldoni *et al.* [1998]; Orgun and Ma [1994]. However, these modal logic programming languages do not contain an explicit negation and often even lack default negation, which is why we shall not look into them here.

possible, ie. PSMs. This is the reason that we will only consider epistemic modalities, they connect directly to our approach. Other modalities, such as deontic or temporal ones, would require the introduction of extra structure in our models before we can treat them.

Suppose that we have a formula with the "possible" modality: $\Diamond\phi$. We are going to apply the same solution as we chose for dealing with the question predicate in the context of negation. This means the following:

$$\begin{aligned} \phi &\leftarrow \mathit{not}\ \neg\phi \\ \neg\phi &\leftarrow \mathbf{u}. \end{aligned}$$

However, the formula ϕ can be complex, so we need further translation after we have applied the above solution. Lets see how example 4.5 would work out. The formula for the answer is: $\Diamond(\mathit{come}(j) \vee \mathit{come}(m))$. Which is translated to:

$$\begin{aligned} (\mathit{come}(j) \vee \mathit{come}(m)) &\leftarrow \mathit{not}\ \neg(\mathit{come}(j) \vee \mathit{come}(m)) \\ \neg(\mathit{come}(j) \vee \mathit{come}(m)) &\leftarrow \mathbf{u}. \end{aligned}$$

On the first line we can distribute both the explicit and default negation. That we can distribute the explicit negation over the disjunction is clear enough. The distribution of the default negation over the resulting conjunction requires some explanation. Suppose we have $\mathit{not}\ (\phi \wedge \psi)$. Informally this means that $\phi \wedge \psi$ is not derivable, ie. has the truthvalue 0. Thus ϕ or ψ must be 0, which implies that ϕ or ψ is not derivable, hence $\mathit{not}\ \phi \vee \mathit{not}\ \psi$.

$$\begin{aligned} (\mathit{come}(j) \vee \mathit{come}(m)) &\leftarrow (\mathit{not}\ \neg\mathit{come}(j) \vee \mathit{not}\ \neg\mathit{come}(m)) \\ (\neg\mathit{come}(j) \wedge \neg\mathit{come}(m)) &\leftarrow \mathbf{u}. \end{aligned}$$

A conjunction on the left can be split into separate copies of the same rule but with a different conjunct as its head:

$$\begin{aligned} (\mathit{come}(j) \vee \mathit{come}(m)) &\leftarrow (\mathit{not}\ \neg\mathit{come}(j) \vee \mathit{not}\ \neg\mathit{come}(m)) \\ \neg\mathit{come}(j) &\leftarrow \mathbf{u} \\ \neg\mathit{come}(m) &\leftarrow \mathbf{u}. \end{aligned}$$

For a disjunction on the right we can do something similar:

$$\begin{aligned} (\mathit{come}(j) \vee \mathit{come}(m)) &\leftarrow \mathit{not}\ \neg\mathit{come}(j) \\ (\mathit{come}(j) \vee \mathit{come}(m)) &\leftarrow \mathit{not}\ \neg\mathit{come}(m) \\ \neg\mathit{come}(j) &\leftarrow \mathbf{u} \\ \neg\mathit{come}(m) &\leftarrow \mathbf{u}. \end{aligned}$$

Finally, a left-hand side disjunction can be translated as we normally translate a disjunction, but we extend the new rules with the body on the right-hand side:

$$\begin{aligned} \mathit{come}(m) &\leftarrow \neg\mathit{come}(j), \mathit{not}\ \neg\mathit{come}(j) \\ \mathit{come}(j) &\leftarrow \neg\mathit{come}(m), \mathit{not}\ \neg\mathit{come}(j) \\ \mathit{come}(m) &\leftarrow \neg\mathit{come}(j), \mathit{not}\ \neg\mathit{come}(m) \\ \mathit{come}(j) &\leftarrow \neg\mathit{come}(m), \mathit{not}\ \neg\mathit{come}(m) \\ \neg\mathit{come}(j) &\leftarrow \mathbf{u} \\ \neg\mathit{come}(m) &\leftarrow \mathbf{u}. \end{aligned}$$

If we combine this with our standard default negation rule for the question predicate and take the small domain: $\{j, m\}$, then we get the following program for example 4.5:

$P_{4.5} =$

$$\begin{array}{rcl}
come(m) & \leftarrow & \neg come(j), not\ \neg come(j) \\
come(j) & \leftarrow & \neg come(m), not\ \neg come(j) \\
come(m) & \leftarrow & \neg come(j), not\ \neg come(m) \\
come(j) & \leftarrow & \neg come(m), not\ \neg come(m) \\
\neg come(j) & \leftarrow & \mathbf{u} \\
\neg come(m) & \leftarrow & \mathbf{u} \\
\neg come(j) & \leftarrow & not\ come(j) \\
\neg come(m) & \leftarrow & not\ come(m).
\end{array}$$

The semilattice of PSMs for this program is the following (C for *come*):

$$\begin{tikzcd}
\begin{array}{c}\{\neg C(j), not\ C(j),\\ C(m), not\ \neg C(m)\}\end{array} & \begin{array}{c}\{\neg C(j), not\ C(j),\\ \neg C(m), not\ C(m)\}\end{array} & \begin{array}{c}\{C(j), not\ \neg C(j),\\ \neg C(m), not\ C(m)\}\end{array} \\
\{\neg C(j), not\ C(j)\} \arrow[u] \arrow[ur] & & \{\neg C(m), not\ C(m)\} \arrow[u] \arrow[ul] \\
& \{\} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

When we consider the entire semilattice of PSMs we get a weak epistemic interpretation. Basically all options are open. To arrive at the suggested implicature, we need a strong interpretation. To get one, we know that we have to apply *MaxEl*, which gives us:

$$\begin{array}{c}\{\neg C(j), not\ C(j),\\ C(m), not\ \neg C(m)\}\end{array} \qquad \begin{array}{c}\{\neg C(j), not\ C(j),\\ \neg C(m), not\ C(m)\}\end{array} \qquad \begin{array}{c}\{C(j), not\ \neg C(j),\\ \neg C(m), not\ C(m)\}\end{array}.$$

From these three PSMs we can conclude the implicature: "It is not possible that John and Mary come."

Let us look at another simpler example: $come(j) \wedge \Diamond come(m)$, "John and possibly Mary comes". This formula results in the following program:

$P_{come(j) \wedge \Diamond come(m)} =$

$$\begin{array}{rcl}
come(m) & \leftarrow & not\ \neg come(m) \\
\neg come(m) & \leftarrow & \mathbf{u} \\
come(j) & & \\
\neg come(j) & \leftarrow & not\ come(j) \\
\neg come(m) & \leftarrow & not\ come(m),
\end{array}$$

which has the semilattice:

$$\begin{tikzcd}
\begin{array}{c}\{C(j), not\ \neg C(j),\\ C(m), not\ \neg C(m)\}\end{array} & & \begin{array}{c}\{C(j), not\ \neg C(j),\\ \neg C(m), not\ C(m)\}\end{array} \\
& \{C(j), not\ \neg C(j)\} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

There is no predicted implicature here, however, the PSMs represent the knowledge that we would expect: John comes to the party and it is possible that Mary does too, ie. there is a PSM in which she comes.

The result of another possible answer: $\Diamond(come(j) \wedge come(m))$, "Possibly John and Mary come", is a bit more problematic. The corresponding program: $P_{\Diamond(come(j)\wedge come(m))} =$

$$\begin{array}{rcl}
come(m) & \leftarrow & not\ \neg come(j), not\ \neg come(m) \\
come(j) & \leftarrow & not\ \neg come(m), not\ \neg come(j) \\
\neg come(j) & \leftarrow & come(m), \mathbf{u} \\
\neg come(m) & \leftarrow & come(j), \mathbf{u} \\
\neg come(j) & \leftarrow & not\ come(j) \\
\neg come(m) & \leftarrow & not\ come(m),
\end{array}$$

has the following semilattice:

$$\begin{tikzcd}
\{C(j), not\ \neg C(j), C(m), not\ \neg C(m)\} & & \{\neg C(j), not\ C(j), \neg C(m), not\ C(m)\} \\
& \{\} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Also in this case there is no predicted implicature. The problem is that one would expect PSMs in which Mary comes and John doesn't and vice versa, since these seem to be possibilities we would expect given the answer "possibly John and Mary come". Nevertheless, "possibly John and Mary come" is still clearly true in this semilattice, there is a possible world in which they come, but they don't come in all of them.

The most troublesome is the final example of this section: $come(j) \vee \Diamond come(m)$, "John comes or possibly Mary comes". The first thing to note is that this utterance seems to be a bit contrived to begin with. What would one mean when uttering this sentence? Perhaps one intends to convey that either John or Mary comes, but that the chance of John coming is significantly higher than the chance of Mary coming. In this case "possibly" would not have the epistemic meaning we intend to have it. Nonetheless, the logical formula makes perfect sense and we would like for it to receive the right interpretation. The program for this formula is:

$P_{come(j)\vee\Diamond come(m)} =$

$$\begin{array}{rcl}
come(m) & \leftarrow & not\ \neg come(m), \neg come(j) \\
come(j) & \leftarrow & \neg come(m), not\ \neg come(m) \\
come(j) & \leftarrow & come(m), \mathbf{u} \\
\neg come(m) & \leftarrow & \neg come(j), \mathbf{u} \\
\neg come(j) & \leftarrow & not\ come(j) \\
\neg come(m) & \leftarrow & not\ come(m).
\end{array}$$

This program has the following semilattice of PSMs:

$$\begin{tikzcd}
\begin{array}{c}\{\neg C(j), \mathit{not}\, C(j),\\ C(m), \mathit{not}\, \neg C(m)\}\end{array} & & \begin{array}{c}\{\neg C(j), \mathit{not}\, C(j),\\ \neg C(m), \mathit{not}\, C(m)\}\end{array} \\
& \{\neg C(j), \mathit{not}\, C(j)\} \arrow[ul] \arrow[ur] & \\
& \{\} \arrow[u] &
\end{tikzcd}$$

What is clearly wrong is that there is no PSM in which John comes. Thus the semilattice seems more to represent the utterance: "Possibly Mary comes" (if the empty well founded model wasn't there this would actually be the case), than "John comes or possibly Mary comes". Both the disjunction and the modal "possibility" are existential operators[8] and in this example they somehow interfere, resulting in wrong predictions.

The above sketched approach goes some way to provide a direction in which to find a solution for dealing with modalities in a WFSX framework. However problems immediately pop-up, since the link between modalities and the semilattice of PSMs run via rules. We have no way of directly defining the restrictions that a modal operator imposes on the semilattice of PSMs.

4.5 Conclusions and Remarks

Before we draw conclusions for this chapter we shall look at example 4.6 again (repeated here for readability).

(4.6) Q: "Who came to the party?"

A: "Some boys came."

Imp: "Not all the boys came to the party."
"No girls came to the party."

We remarked in chapter 3 that even with adding the sex of the individuals as background knowledge we still could not derive anything interesting using SLDNF. Fortunately under the WFSX semantics we can derive something interesting in a top-down fashion. We recall that the program extended with the background knowledge has the following two answer-sets:

$$\{C(j), \neg C(b), \neg C(m), B(j), B(b), \neg B(m)\}$$
$$\{C(b), \neg C(j), \neg C(m), B(j), B(b), \neg B(m)\}$$

The semilattice under the WFSX semantics is as follows:

[8] The disjunction says that there is *a* disjunct that is true, "possibility" says there is *a* world in which the formula holds.

$$\begin{tikzcd}
\begin{array}{c}\{C(j), \mathit{not}\ \neg C(j),\\ \neg C(b), \mathit{not}\ C(b),\\ \neg C(m), \mathit{not}\ C(m),\\ B(j), \mathit{not}\ \neg B(j),\\ B(b), \mathit{not}\ \neg B(b),\\ \neg B(m), \mathit{not}\ B(m)\}\end{array} & & \begin{array}{c}\{\neg C(j), \mathit{not}\ C(j),\\ C(b), \mathit{not}\ \neg C(b),\\ \neg C(m), \mathit{not}\ C(m),\\ B(j), \mathit{not}\ \neg B(j),\\ B(b), \mathit{not}\ \neg B(b),\\ \neg B(m), \mathit{not}\ B(m)\}\end{array} \\
& \begin{array}{c}\{\neg C(m), \mathit{not}\ C(m),\\ B(j), \mathit{not}\ \neg B(j),\\ B(b), \mathit{not}\ \neg B(b),\\ \neg B(m), \mathit{not}\ B(m)\}\end{array} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

We see that the well founded model contains the fact that Mary doesn't come. So in contrast to SLDNF we can already derive the implicature: "No girl came to the party", without looking at all the partial stable models. We can derive it efficiently bottom-up and also top-down, since the WFM is computable using SLX.

As mentioned earlier, there is a clear difference between the complexity of the computation of the well founded model on the one hand and the semilattice of partial stable models on the other. Furthermore, after reworking out example 4.6 under the WFSX semantics, we saw that the difference between what we dubbed 'exhaustivity' and scalar implicatures has become even clearer. Under the WFSX semantics exhaustivity implicatures are now all based on the well founded model and derivable in a bottom-up or top-down fashion, without looking at all the possible partial stable models. On the other hand all scalar implicatures still require that we compute all PSMs[9]. Since the computation of the WFM is computationally far less complex than the calculation of all the PSMs, we conclude that there is a complexity difference between exhaustivity implicatures and scalar implicatures. It would be very nice to set up experiments to look into this from a psycholinguistic point of view and see whether this theoretical difference has any real world significance.

To work with a program under WFSX semantics one doesn't need to compute the PSMs, ie. one can use SLX or the bottom-up procedure to compute the WFM. We saw the same kind of distinction in chapter three. As in that chapter, we have to look at all the models (PSMs) to derive the scalar implicatures. The calculation of these PSMs is an expensive extra computational step. From a pyscholinguistic perspective we can say again that humans ordinarily derive the WFM and only when the context is right for scalar implicatures one computes all the PSMs. Thus, even clearer than in the previous chapter, our approach is in the line of the context-dependent explanation.

The WFSX semantics also provides a way to deal with implicatures of different epistemic strengths. However, the presented solution is not without its

[9]Actually, we saw one example in which this wasn't the case: the second way of treating cardinal numbers. However, this treatment was somewhat different than the rest of this thesis and more importantly it is the case that the implicature community is very much divided as to whether examples with cardinal number are real scalar implicatures, or just something else. So this is not a problem.

flaws. Despite this fact, these semantics deserves more attention to find a better and more widely applicable solution.

Chapter 5

Other Types of Implicatures

Next to the Q-implicatures, which we encountered in the previous two chapters, we already saw in chapter two that there are other types of implicatures. Sticking to Levinson [2000] there are at least the classes of I- and M-implicatures that we can look at as well. We expect to deal especially well with the I-implicatures, since they have close resemblance to the type of default knowledge that AI researchers want to formalize using logic programming, for an introduction, see Baral and Gelfond [1994].

5.1 I-implicatures

While Q-implicatures are derived based on the idea that the speaker provides as much information as possible, I-implicatures are derived using an opposite intuition. Levinson (Levinson [2000]) calls the responsible maxim the *Principle of Informativeness* or just I-principle. To get an idea, let us consider the following example:

(5.1) Q: "Who is John meeting tonight?"
A: "John is meeting his secretary tonight."
Imp: "John is meeting a woman, who is not his wife, but is his secretary, tonight."

In the implicature the use of the word "secretary" is strengthened, via stereotypical world knowledge, to describe a woman, not the wife of the person in question[1]. The hearer interprets the utterance more specific, she amplifies the informational content of the utterance. The idea of the I-principle becomes clear in the example: knowledge of the syntactic and semantic use of expressions allows the speaker to say less and the hearer to interpret more.

On the speakers side we have the maxim of Minimization: "Say as little as necessary." The speaker produces as minimal linguistic information as is

[1]One could perhaps convincingly argue here that the inference that 'his secretary' is not his wife can also be due to the Q-principle.

sufficient to communicate what she wants, while keeping the Q-principle in mind. As a result of this, the hearer enriches the informational content of the utterance as much as possible, unless it is somehow clear, for example from the context, that the speaker has broken the maxim of Minimization. More specifically, this means that the hearer assumes as much temporal, causal and referential connections and stereotypical relations as are consistent with the utterance and context, she doesn't multiply entities referred to beyond what is strictly necessary and she assumes the existence or actuality of what the sentence is about if that is consistent.

5.1.1 Applying ELP

As we mentioned, the power of logic programming in AI has always been the formalization of default reasoning such as in example 5.1. Typical examples such as Tweety the bird who flies, and the penguin that by default doesn't fly can be considered examples of I-implicatures from our perspective. Thus we expect that logic programming is particularly suited to deal with at least some I-implicatures. Unfortunately, a big portion of the I-implicatures are based on the syntax of the utterance in question and not strictly on the semantic representation. Since the syntax of the utterance completely disappears in our rule-based representation, we cannot deal with those kinds of I-implicatures. The implicature of the following example cannot be dealt with.

(5.2) - "John closed the door and left the house."

Imp: "John first closed the door and then left the house."

The temporal reading of the implicature is based on the idea that the left conjunct of "and" ordinarily occurs temporally before the right conjunct. Such information is lost in our system. If we want to deal with this, we have to add temporal structure.

However, example 5.1 can be dealt with perfectly. The answer can be translated as:

$$\exists x(\mathit{Meet}(j,x) \land \mathit{Secretary}(j,x)).$$

Clearly the question predicate is $\mathit{Meet}(j,x)$. The domain of individuals is again: $\{j,m,b\}$. Since I-implicatures are based on world knowledge we need to formalize this knowledge in a background theory in order to derive them. As we saw in chapter 3, we add this theory to the program generated with GP. An old AI trick is needed for this theory: abnormality predicates. The idea behind abnormality predicates is that they are false unless there is an abnormal situation going on. This notion will become clearer when considering the background knowledge that we are going to need in this example, which is the (grounded version of the) following theory:

$$\begin{array}{rcl}
\neg \mathit{Male}(x) & \leftarrow & \mathit{Secretary}(j,x), \mathit{not}\ \mathit{ab}_1(x) \\
\neg \mathit{Married}(j,y) & \leftarrow & \mathit{Secretary}(j,x), \mathit{not}\ \mathit{ab}_2(j,x) \\
\neg \mathit{Secretary}(j,x) & \leftarrow & \mathit{Male}(x), \mathit{not}\ \mathit{ab}_1(x) \\
\neg \mathit{Secretary}(j,x) & \leftarrow & \mathit{Married}(j,x), \mathit{not}\ \mathit{ab}_2(j,x).
\end{array}$$

For simplicity, the background theory is centered around John, but in general it means the following. A person is not male, ie. female, if that person is a secretary and there is nothing abnormal going on for this person with regards to the relation with his secretary, ie. $ab_1(x)$ doesn't hold. This also holds vice versa, someone is not a secretary of someone if he is male, unless there is something abnormal going on. For being married more or less the same holds: two people are not married if the one is the secretary of the other, and there is nothing abnormal going on, which is defined by another abnormality predicate than the one used for the secretary relation. For this example we don't know of any abnormalities so we don't add facts/rules with ab_1 or ab_2.

If we put all the arguments into *GP* we get the following program $P_{5.1}$ (M for *Meet* and S for *Secretary*, also the default negation rules are ungrounded to save space):

$$\begin{array}{rclrcl}
M(j,j) & \leftarrow & \neg M(j,m), \neg M(j,b) & S(j,j) & \leftarrow & \neg M(j,m), \neg M(j,b) \\
M(j,m) & \leftarrow & \neg M(j,j), \neg M(j,b) & M(j,m) & \leftarrow & \neg S(j,j), \neg M(j,b) \\
M(j,b) & \leftarrow & \neg M(j,j), \neg M(j,m) & M(j,b) & \leftarrow & \neg S(j,j), \neg M(j,m) \\
M(j,j) & \leftarrow & \neg M(j,m), \neg S(j,b) & S(j,j) & \leftarrow & \neg S(j,m), \neg M(j,b) \\
M(j,m) & \leftarrow & \neg M(j,j), \neg S(j,b) & S(j,m) & \leftarrow & \neg S(j,j), \neg M(j,b) \\
S(j,b) & \leftarrow & \neg M(j,j), \neg M(j,m) & M(j,b) & \leftarrow & \neg S(j,j), \neg S(j,m) \\
M(j,j) & \leftarrow & \neg S(j,m), \neg M(j,b) & S(j,j) & \leftarrow & \neg M(j,m), \neg S(j,b) \\
S(j,m) & \leftarrow & \neg M(j,j), \neg M(j,b) & M(j,m) & \leftarrow & \neg S(j,j), \neg S(j,b) \\
M(j,b) & \leftarrow & \neg M(j,j), \neg S(j,m) & S(j,b) & \leftarrow & \neg S(j,j), \neg M(j,m) \\
M(j,j) & \leftarrow & \neg S(j,m), \neg S(j,b) & S(j,j) & \leftarrow & \neg S(j,m), \neg S(j,b) \\
S(j,m) & \leftarrow & \neg M(j,j), \neg S(j,b) & S(j,m) & \leftarrow & \neg S(j,j), \neg S(j,b) \\
S(j,b) & \leftarrow & \neg M(j,j), \neg S(j,m) & S(j,b) & \leftarrow & \neg S(j,j), \neg S(j,m)
\end{array}$$

$$\begin{array}{rcl}
\neg M(j,x) & \leftarrow & \textit{not } M(j,x) \\
\neg S(j,x) & \leftarrow & \textit{not } S(j,x) \\
S(j,x) & \leftarrow & \textit{not } \neg S(j,x)
\end{array}$$

$$\begin{array}{rcl}
\neg \textit{Male}(x) & \leftarrow & \textit{Secretary}(j,x), \textit{not } \textit{ab}_1(x) \\
\neg \textit{Married}(j,y) & \leftarrow & \textit{Secretary}(j,x), \textit{not } \textit{ab}_2(j,x) \\
\neg \textit{Secretary}(j,x) & \leftarrow & \textit{Male}(x), \textit{not } \textit{ab}_1(x) \\
\neg \textit{Secretary}(j,x) & \leftarrow & \textit{Married}(j,x), \textit{not } \textit{ab}_2(j,x).
\end{array}$$

This program has a lot of partial stable models. Since John can meet every person[2], that gives us three options already. Furthermore the person that John is meeting must be his secretary. But the others might be his secretary too, since we don't apply closed world reasoning to the secretary predicate. This leaves us with nine PSMs for every person that John is meeting. Thus including the well founded model, which is just the empty-set, we have 28 PSMs. To sum these all up would be a bit too much, so we only focus on who John is meeting and that way reduce nine PSMs to one. We also leave out the default (*not* $\neg L$ is obviously in the PSM if L is) and the abnormality literals.

[2]Even himself because we didn't add to the program that such a thing is usually impossible, which would not be difficult, but only unnecessarily complicates matters.

$$\begin{tikzcd}
\begin{array}{c}\{M(j,j), \neg M(j,b),\\ \neg M(j,m), S(j,j),\\ \neg \mathit{Male}(j),\\ \neg \mathit{Married}(j,j)\}\end{array} & \begin{array}{c}\{M(j,b), \neg M(j,j),\\ \neg M(j,m), S(j,b),\\ \neg \mathit{Male}(b),\\ \neg \mathit{Married}(j,b)\}\end{array} & \begin{array}{c}\{M(j,m), \neg M(j,b),\\ \neg M(j,j), S(j,m),\\ \neg \mathit{Male}(m),\\ \neg \mathit{Married}(j,m)\}\end{array} \\
& \{\} \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

We see in all the partial stable models that the person that John is meeting is his secretary and furthermore is female and not his wife, exactly the I-implicatures that we were looking for.

Now let us see what happens when we explicitly define in the background knowledge what the sex of the individuals in our domain is. We add the obvious: $\mathit{Male}(j)$, $\mathit{Male}(b)$ and $\neg\mathit{Male}(m)$ as background theory. What we get is one possible partial stable model, which is of course also the well founded model (all abnormality predicate literals are left out, they are all default negated).

$$\begin{array}{c}
\{M(j,m), \mathit{not}\ \neg M(j,m), \neg M(j,b), \mathit{not}\ M(j,b), \neg M(j,j), \mathit{not}\ M(j,j),\\
S(j,m), \mathit{not}\ \neg S(j,m), \neg S(j,b), \mathit{not}\ S(j,b), \neg S(j,j), \mathit{not}\ S(j,j),\\
\mathit{Male}(j), \mathit{not}\ \neg\mathit{Male}(j), \mathit{Male}(b), \mathit{not}\ \neg\mathit{Male}(b), \neg\mathit{Male}(m), \mathit{not}\ \mathit{Male}(m),\\
\mathit{not}\ \neg\mathit{Married}(j,j), \mathit{not}\ \mathit{Married}(j,j), \mathit{not}\ \neg\mathit{Married}(j,b),\\
\mathit{not}\ \mathit{Married}(j,b), \neg\mathit{Married}(j,m), \mathit{not}\ \mathit{Married}(j,m)\}
\end{array}$$

Clearly this is the result we desired. The only option left is that John is meeting Mary, since she is female and furthermore he is not married to her. Note that if we would add more females to the domain we will incidentally derive a scalar implicature: John will only be meeting one woman tonight. Also, the well founded model will still contain the fact that John is not meeting a man, ie. for all men we have the fact that John is not meeting them is in the WFM.

But why do we need these abnormality predicates? A first intuition, upon looking at this example, can be that background knowledge of the following form should suffice:

$$\begin{array}{rcl}
\neg\mathit{Male}(x) & \leftarrow & \mathit{Secretary}(j,x), \mathit{not}\ \mathit{Male}(x)\\
\neg\mathit{Married}(j,x) & \leftarrow & \mathit{Secretary}(j,x), \mathit{not}\ \mathit{Married}(j,x)\\
\neg\mathit{Secretary}(j,x) & \leftarrow & \mathit{Male}(x), \mathit{not}\ \mathit{Secretary}(j,x)\\
\neg\mathit{Secretary}(j,x) & \leftarrow & \mathit{Married}(j,x), \mathit{not}\ \mathit{Secretary}(j,x).
\end{array}$$

The meaning of these rules seems to be almost identical to the ones given above, without making use of 'ugly' abnormality predicates. The first thing to note is that the last two rules are actually weaker than the default rule: $\neg S(j,x) \leftarrow \mathit{not}\ S(j,x)$, which is already included in the program. The last two rules are basically the same, except for the added male/married predicate, so whenever these rules can be used the default rule applies too.

If we don't have information about the sex of the individuals in our background knowledge, then in this example the results are the same when using these new rules as when using the abnormality predicate rules. Things differ however when we add information about the sex of the individuals to the background knowledge. What we get is the same amount of PSMs as before (28). Furthermore, they are almost completely the same, except for the fact that it is now the case that John's secretary might be male. There are PSMs where John meets John and where John meets Bill, who are both male. However, in no PSM is John married to his secretary. What we see is that the explicit definition of the sex of the individuals overwrites the default rule. This is not the right result. One would expect that for every person that is defined male it is not possible that John is meeting this person, unless some explicit evidence to the contrary is given. Which is exactly what the solution using abnormality predicates does.

5.1.2 Combination with a Q-implicature

The above example contained a simple I-implicature and no more. However, it might be the case that an utterance is expected to give rise to both Q- and I-implicatures. These implicatures together might be inconsistent. Levinson argues that in such a case the Q-implicature takes precedence over the I-implicature. In Levinson [2000] he gives examples of conjunctions which have a possible I-implicature in one conjunct but have this implicature cancelled by the fact that the other conjunct entails its contradiction. For instance, this is the case in:

(5.3) Q: "Who is John meeting tonight?"

A: "John is meeting his wife or his secretary."

Imp: The I-implicature: "John is not meeting his wife" arising from the second conjunct, is cancelled by the Q-implicature: "Possibly John is meeting his wife" from the first conjunct.

The implicature from the first conjunct is what is dubbed a clausal implicature and we briefly saw it before in example 3.3.

It is clear that the question predicate must be $Meet(j, x)$. The translation of the answer is also not too difficult: $\exists x(Meet(j, x) \wedge ((Married(j, x) \wedge \neg Male(x)) \vee Secretary(j, x)))$. The domain of individuals and the background theory remain the same as in the last version of example $P_{5.1}$. The program resulting from GP is given below. We abbreviate it a lot. First of all, we use M for $Meet$, Mr for $Married$, Ml for $Male$ and S for $Secretary$. Also, we give the rules in a more compact notion. Every literal in the body is now a set of literals. If we take a literal from each of those sets, we get a rule of the program. Hereby we compress around 70 rules into one. Thus, $P_{5.3} =$

$$\begin{array}{rcl}
M(j,j) & \leftarrow & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \{\neg M(j,m), \neg S(j,m)\} \\
M(j,b) & \leftarrow & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \{\neg M(j,j), \neg S(j,j)\}, \\
& & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \{\neg M(j,m), \neg S(j,m)\} \\
M(j,m) & \leftarrow & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \{\neg M(j,j), \neg S(j,j)\} \\
Mr(j,j) & \leftarrow & \{\neg M(j,j), \neg S(j,j)\}, \\
& & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \{\neg M(j,m), \neg S(j,m)\} \\
Mr(j,b) & \leftarrow & \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \{\neg M(j,j), \neg S(j,j)\}, \\
& & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \{\neg M(j,m), \neg S(j,m)\} \\
Mr(j,m) & \leftarrow & \{\neg M(j,m), \neg S(j,m)\}, \\
& & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \{\neg M(j,j), \neg S(j,j)\} \\
\neg Ml(j,j) & \leftarrow & \{\neg M(j,j), \neg S(j,j)\}, \\
& & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \{\neg M(j,m), \neg S(j,m)\} \\
\neg Ml(j,b) & \leftarrow & \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \{\neg M(j,j), \neg S(j,j)\}, \\
& & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \{\neg M(j,m), \neg S(j,m)\} \\
\neg Ml(j,m) & \leftarrow & \{\neg M(j,m), \neg S(j,m)\}, \\
& & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \{\neg M(j,j), \neg S(j,j)\} \\
S(j,j) & \leftarrow & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \\
& & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \{\neg M(j,m), \neg S(j,m)\} \\
S(j,b) & \leftarrow & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \\
& & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \{\neg M(j,j), \neg S(j,j)\}, \\
& & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \{\neg M(j,m), \neg S(j,m)\} \\
S(j,m) & \leftarrow & \{\neg M(j,m), \neg Mr(j,m), Ml(m)\}, \\
& & \{\neg M(j,b), \neg Mr(j,b), Ml(b)\}, \{\neg M(j,b), \neg S(j,b)\}, \\
& & \{\neg M(j,j), \neg Mr(j,j), Ml(j)\}, \{\neg M(j,j), \neg S(j,j)\}
\end{array}$$

$$\begin{array}{rcl}
\neg M(j,x) & \leftarrow & not\ M(j,x) \\
\neg S(j,x) & \leftarrow & not\ S(j,x) \\
S(j,x) & \leftarrow & not\ \neg S(j,x) \\
\neg Ml(x) & \leftarrow & S(j,x), not\ Ml(x) \\
\neg Mr(j,x) & \leftarrow & S(j,x), not\ Mr(j,x) \\
\neg S(j,x) & \leftarrow & Ml(x), not\ S(j,x) \\
\neg S(j,x) & \leftarrow & Mr(j,x), not\ S(j,x) \\
Ml(j) & & \\
Ml(b) & & \\
\neg Ml(m) & &
\end{array}$$

This program has the following semilattice of partial stable models (again the complementary default negated literals are left out, as are the literals about the sex of the individuals):

$$\begin{tikzcd}
\begin{array}{c}\{M(j,m), \neg M(j,b),\\ \neg M(j,j), \neg S(j,j),\\ \neg S(j,b), \neg S(j,m),\\ Mr(j,m)\}\end{array} & & \begin{array}{c}\{M(j,m), \neg M(j,j),\\ \neg M(j,b), \neg S(j,b),\\ \neg S(j,j), S(j,m)\\ \neg Mr(j,m)\}\end{array} \\
& \begin{array}{c}\{M(j,m), \neg M(j,j),\\ \neg M(j,b), \neg S(j,b),\\ \neg S(j,j)\}\end{array} \arrow[ul] \arrow[ur] & \\
& \begin{array}{c}\{M(j,m), \neg S(j,b),\\ \neg S(j,j)\}\end{array} \arrow[u] &
\end{tikzcd}$$

When we apply a strong epistemic interpretation by using *MaxEl* we see that the result is exactly what we want. There are two possibilities, John might be meeting his wife and John might be meeting his secretary, which is female and not his wife. Furthermore John is only meeting one person, which is not an implicature we mentioned earlier in this example, but it is just the scalar implicature from "or" to "not and" that we saw before.

The well founded model does not completely live up to our expectations. It contains the fact that John is going to meet Mary, which makes sense, since Mary is the only woman and either John is meeting his wife, or his secretary which is a woman by default. Hence, from the perspective of the minimal interpretation of the question predicate we would expect that we also derive that John is not meeting himself or Bill, however, these facts are not in the WFM. This way we miss the derivation of the exhaustivity implicature on the basis of the WFM (of course we do get it from our maximal elements).

The problem is due to the interaction between the disjunction in our answer: "John is meeting his wife *or* his secretary" and the disjunction created when translating the existential quantifier. This is best illustrated with an example taken from the program. To derive $\neg M(j,b)$ we need to derive *not* $M(j,b)$, which implies that every derivation of $M(j,b)$ should fail. Looking at the program we see that an example of a salient rule for $M(j,b)$ is:

$$M(j,b) \quad \leftarrow \quad Ml(j), \neg S(j,j), \neg Mr(j,m), \neg S(j,m)$$

Both $Ml(j)$ and $\neg S(j,j)$ are derivable, thus if we want this rule to fail then the derivation of $\neg Mr(j,m)$ or $\neg S(j,m)$ should fail. However they both do not fail. Nevertheless, we as outsiders can see that one of the two fact should clearly fail, since either John is going to meet his wife or his secretary and we already know (since this is derivable) that this is Mary. So, while we can see that $M(j,b)$ should fail for this rule, and hence other similar ones, because of the fact that the disjunction "John is meeting his wife or his secretary" is always true, the derivation procedure doesn't 'know' that $M(j,b)$ should fail because it cannot use the fact that the aforementioned disjunction should always be true.

Two solutions to this problem easily come to mind. First one could use functions to model the secretary and wife relations. In which case the answer would be the much less complex: $M(j, \mathit{Wife}(j)) \vee M(j, \mathit{Sec}(j))$. However, using functions destroys some of the computational complexity properties mentioned in the fourth chapter.

A second option is to introduce a discourse referent for an existential quantifier, instead of the translation to the large disjunction, in the vein of Discourse Representation Theory (DRT) (Kamp and Reyle [1993]). However, this would require extra rules to model the equivalence relation to ensure that this referent is equal to an individual in our domain and that it has the same properties as this individual and vice versa. This solution would destroy the equivalence result as it is proven in chapter three.

It is somewhat unfortunate that, in this complex example, we lose the dichotomy that we noticed in chapters three and four, between exhaustivity implicatures and the WFM on the one hand and scalar implicatures and the semilattice of PSMs on the other. The two solutions briefly presented above will most likely return this dichotomy, but the extra complexities that they introduce might not be worth the effort. Fleshing out these solutions is beyond the scope of this thesis.

5.1.3 Another Example: Conditional Perfection

Levinson [2000] provides an example of another phenomenon attributed to the I-principle. Since it is not based on the syntactic structure of the utterance, we can account for it in our approach as well, quite easily in fact. We work out the following example of conditional perfection adapted from Levinson [2000]:117:

(5.4) Q: "When will I get five dollars?"
A: "If you mow the lawn, I'll give you five dollars."
Imp: "If and only if you mow the lawn, will I give you five dollars."

We keep the formalization as simple as possible, we only introduce what is necessary to derive the conditional perfection. Thus, the question predicate in this example is: 'the getting of the five dollars'. So we introduce

$$\neg \mathit{FiveDollars} \quad \leftarrow \quad \mathit{not}\ \mathit{FiveDollars}$$

for this literal. Furthermore, 'mowing the lawn', should remain a fixed predicate, thus we have the rules:

$$\begin{array}{rcl} \neg \mathit{MowLawn} & \leftarrow & \mathit{not}\ \mathit{MowLawn} \\ \mathit{MowLawn} & \leftarrow & \mathit{not}\ \neg \mathit{MowLawn}. \end{array}$$

What remains is a simple translation of the utterance into logic: $\mathit{MowLawn} \rightarrow \mathit{FiveDollars}$. Putting it all together gives us the program $P_{5.4}$.

$$\begin{array}{rcl} \neg \mathit{MowLawn} & \leftarrow & \neg \mathit{FiveDollars} \\ \mathit{FiveDollars} & \leftarrow & \mathit{MowLawn} \\ \neg \mathit{FiveDollars} & \leftarrow & \mathit{not}\ \mathit{FiveDollars} \\ \neg \mathit{MowLawn} & \leftarrow & \mathit{not}\ \mathit{MowLawn} \\ \mathit{MowLawn} & \leftarrow & \mathit{not}\ \neg \mathit{MowLawn} \end{array}$$

As far as the well founded model is concerned, nothing is derivable in this program, it is just the empty set. Nevertheless, the maximal elements of the semilattice give us the conditional perfection that we want.

$$\begin{tikzcd} \{\mathit{MowLawn}, \mathit{not}\ \neg\mathit{MowLawn}, \mathit{FiveDollars}, \mathit{not}\ \neg\mathit{FiveDollars}\} & & \{\neg\mathit{MowLawn}, \mathit{not}\ \mathit{MowLawn}, \neg\mathit{FiveDollars}, \mathit{not}\ \mathit{FiveDollars}\} \\ & \{\} \arrow[ul] \arrow[ur] & \end{tikzcd}$$

As we see, there are two ways the utterance is true, either the lawn is mown and the five dollars is received, or the lawn remains untouched in which case getting the five dollars will never happen. Thus the conditional is perfected into a biconditional.

5.1.4 Remarks

Judging from the worked out examples, we can say that extended logic programming handles the type of default reasoning that is required for I-implicatures rather well. Unfortunately, the results of the most complex example were not entirely satisfiable. If we want to deal with I-implicatures that arise from syntactic features of the utterance, then we have to expand our approach to deal with syntax and see if we are able to derive the desired results. Although difficult, there is no reason to consider this impossible.

5.2 M-implicatures

Levinson defines a third principle responsible for implicatures, the M-principle. The idea behind M-implicatures is that we can use marked expressions to indicate nonstereotypical situations. The speaker is required to use a marked expression when such a situation occurs, so that the hearer is capable of recognizing this intention and make an appropriate nonstereotypical interpretation, usually one that contrasts with the stereotypical interpretation due to the I-principle. The following example from Levinson [2000]:138 makes the idea clear.

(5.5) a. "Sue smiled."

b. - "The corners of Sue's lips turned slightly upwards."
   - Imp: Sue produced a smirk or grimace.

Both the utterance *a* and *b* describe the fact the Sue smiled. But since *b* is a nonstereotypical way of saying this, we interpret the utterance as meaning that Sue's smile was not a real smile out of happiness, but a smirk or grimace.

From this example and also from the idea behind the M-principle in general, we see that M-implicatures depend very much on the form and not so much on the semantic content of the utterance. Thus, as with I-implicatures based more on syntax than semantics, our approach is not suited to deal with them. A solution for M-implicatures might be to explicitly state in the program, via some special predicate, that a marked expression was used and that we therefore cannot apply the default reasoning of I-implicatures. However, such a solution is more stating that an M-implicature holds than actually deriving an M-implicature and thus not to be preferred.

Chapter 6

Conclusion

In the previous chapters we have expounded a formalization using extended logic programming that can deal with a lot of different examples of implicatures. At the heart lies the function *GP* that generates an extended logic program on the basis of formulae representing a question and an answer and a domain of individuals. Then we apply either procedural or declarative semantics to derive implicatures. We have seen two versions of both types of semantics, SLDNF and answer-sets in the basic approach of chapter three and WFSX and SLX in the advanced approach of chapter four.

The formalization provides a very satisfying account of traditional scalar implicatures with "or" and "some". We have also seen how we can deal with implicatures with cardinal numbers. An epistemically weaker solution is also given to deal with those implicatures that require an epistemically weaker reading, such as scalar terms in the context of negation. Furthermore we can deal with I-implicatures that are not based on the syntax but on the semantic content of an utterance. Nevertheless, some of the more difficult data, such as modalities and the mixing of Q- and I-implicatures, requires further investigation to provide a more satisfying account. In terms of dealing with the same data that other approaches can, we are almost as far as the work of Van Rooij and Schulz [2004]; Schulz and Van Rooij [2006].

An important point of this thesis is the computational difference found between exhaustivity implicatures and scalar implicatures. The result of this thesis is that it is easier to compute exhaustivity, eg. the fact that people not mentioned in the answer will not be becoming, than that it is to compute a full scalar implicature, eg. inferring from the fact that Mary or Bill comes that they won't come both. Scalar implicatures almost always involve multiple models, where one partial model is enough to determine the exhaustivity implicature. This result is not something found elsewhere in the literature and worth taking into account in the psycholinguistic studies of implicatures.

In the light of the debate whether scalar implicatures are computed by default or only when the context is right, this thesis favors the latter option, since we have seen that scalar implicatures are computationally expensive. The derivation of the scalar implicatures is a clear extra step on top of the computation of the well founded model. To know what is true in the program one only needs this well founded model. From a psycholinguistic perspective the well founded model can be regarded as the default behaviour and when the context is right the computationally expensive partial stable models are computed to derive scalar implicatures.

Chapter three also included an equivalence theorem between this thesis' approach using extended logic programming and answer-sets semantics and the approach by Van Rooij and Schulz [2004] using circumscription of the answer formula. With adding the extra step of characteristic clauses this theorem is proven.

Despite the fact that the approach in this thesis has problems of its own, a lot of implicatures can be dealt with. Furthermore it provides a fresh computationally motivated view on formalizing implicatures that can be expected to be psychologically relevant. As we set out in the introduction, this was an important goal of this thesis.

Bibliography

Jose Julio Alferes and Luis Moniz Pereira. *Reasoning with logic programming*, volume 1111. Springer-Verlag Inc., New York, NY, USA, 1996.

Jose Julio Alferes, Carlos Viegas Damasio, and Luis Moniz Pereira. A logic programming system for nonmonotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.

Matteo Baldoni, Laura Giordano, and Alberto Martelli. A modal extension of logic programming: Modularity, beliefs and hypothetical reasoning. *Journal of Logic and Computation*, 8(5):597–635, 1998.

Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

Lewis Bott and Ira A. Noveck. Some utterances are underinformative: The onset and time course of scalar inferences. *Journal of Memory and Language*, 51(3):437–457, 2004.

Richard Breheny, Napoleon Katsos, and John Williams. Are generalised scalar implicatures generated by default? an on-line investigation into the role of context in generating pragmatic inferences. *Cognition*, 100(3):434–463, 2006.

Robyn Carston. Informativeness, relevance and scalar implicature. In R. Carston and S. Uchida, editors, *Relevance Theory: Applications and Implications*, pages 179–236. John Benjamins, Amsterdam, 1998.

Gennaro Chierchia. Scalar implicatures, polarity phenomena, and the syntax/pragmatics interface. Ms., University of Milan, 2001.

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. In *IEEE Conference on Computational Complexity*, pages 82–101, 1997.

Gerald Gazdar. *Pragmatics: Implicature, Presupposition and Logical Form.* Academic Press, New York, 1979.

Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In *Logic programming*, pages 579–597. MIT Press, Cambridge, MA, USA, 1990.

Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

Bart Geurts. Quasi-local implicatures or local quasi-implicatures? ms., 2007.

Herbert Paul Grice. *Studies in the Way of Words*. Harvard University Press, Cambridge, Massachusetts, 1989.

Jeroen Groenendijk and Martin Stokhof. *Studies in the Semantics of Questions and the Pragmatics of Answers*. PhD thesis, University of Amsterdam, 1984.

Larry Horn. *On the Semantic Properties of Logical Operators in English*. PhD thesis, University of California, 1972.

Larry Horn. Presupposition and implicature. In Shalom Lappin, editor, *The Handbook of Contemporary Semantic Theory*, pages 299–319. Blackwell's, Oxford, 1995.

Larry Horn. Implicature. In *The Handbook of Pragmatics*, pages 3–28. Blackwell, Oxford, 2004.

Hans Kamp and Uwe Reyle. *From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer, Dordrecht, 1993.

Stephen Levinson. *Pragmatics*. Cambridge University Press, Cambridge, 1983.

Stephen Levinson. *Presumptive Meanings*. MIT Press, Cambridge, 2000.

Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.

Vladimir Lifschitz. Circumscription. In Dov Gabbay, Christopher J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, pages 298–352. Oxford University Press, 1994.

Vladimir Lifschitz. Foundations of logic programming. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, pages 69–127. CSLI Publications, Stanford, California, 1996.

John McCarthy. Circumscription: A form of non-monotonic reasoning. In M. L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 145–151. Kaufmann, Los Altos, CA, 1987.

Ira A. Noveck and Andres Posada. Characterizing the time course of an implicature: An evoked potential study. *Brain and Language*, 85:203–210, 2003.

Mehmet Ali Orgun and Wanli Ma. An overview of temporal and modal logic programming. In D M Gabbay and H J Ohlbach, editors, *Proceedings of ICTL'94: The 1st International Conference on Temporal Logic*, pages 445–479, Berlin Heidelberg, 1994. Springer-Verlag.

Nausicaa Pouscoulous, Ira A. Noveck, Guy Politzer, and Anne Bastide. Evidence for the production of scalar implicature in young children. *Language Acquisition*, in press.

Uli Sauerland. Scalar implicatures in complex sentences. *Linguistics and Philosophy*, 27(3):367–391, 2004.

Katrin Schulz and Robert Van Rooij. Pragmatic meaning and non-monotonic reasoning: The case of exhaustive interpretation. *Linguistics and Philosophy*, 29(2):205–250, 2006.

Scott Soames. How presuppositions are inherited: a solution to the projection problem. *Linguistic Inquiry*, 13:483–545, 1982.

Benjamin Spector. Scalar implicatures: Exhaustivity and gricean reasoning. In Balder ten Cate, editor, *Proceedings of the ESSLLI 2003 Student Session*, 2003.

Keith Stenning and Michiel Van Lambalgen. *Human Reasoning and Cognitive Science*. to appear with MIT Press, 2007.

Gianluca Storto and Michael K. Tanenhaus. Are scalar implicatures computed online? In *Proceedings of WECOL 2004*, 2004.

Allen van Gelder, Kenneth Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

Michiel Van Lambalgen and Fritz Hamm. *The Proper Treatment of Events*. Blackwell, 2004.

Robert Van Rooij and Katrin Schulz. Exhaustive interpretation of complex sentences. *Journal of Logic, Language and Information*, 13(4):491–519, 2004.

Toshiko Wakaki and Ken Satoh. Compiling prioritized circumscription into extended logic programs. In *IJCAI (1)*, pages 182–189, 1997.