QUANTITY IMPLICATURES IN EXTENDED LOGIC PROGRAMMING

GERBEN DE VRIES

Human-Computer Studies Lab, Informatics Institute
University of Amsterdam
gkdvries@science.uva.nl

This paper describes a formalization of quantity implicatures using extended logic programming. In this approach an implicature example is translated into a logic program and from the WFSX semantics of this program quantity implicatures are derived. This formalization provides the insight that scalar implicatures are computationally more complex than exhaustivity implicatures. Furthermore, the approach has a close connection to one using circumscription as in van Rooij and Schulz 2004.

1. Introduction

Traditionally the formal study of Gricean quantity implicatures such as:

(1) Q: "Who is coming to the party?"
A: "John or Mary is coming."
Imp: John or Mary aren't coming both (a). No one (relevant) besides John or Mary is coming (b).

is done in a model-theoretic fashion. In van Lambalgen and Hamm 2004 the authors argue that such methods are inadequate if one wants to bestow any cognitive relevance to formal accounts. In cognitive science, humans are regarded as information processing, ie. computational, machines. Thus, in order for formal approaches to make cognitive sense, they must be computational in nature. In their study of tense and aspect of verbs Van Lambalgen and Hamm create such a computational approach by making use of the formalism of *Logic Programming*.

The above example shows two types of implicatures. The first one, from 'or' to 'not and', is one of the standard incarnations of a *scalar* implicature (Horn 1972). The second type does not have a standard label, we shall dub it an *exhaustivity* implicature, after the exhaustivity function in Groenendijk and Stokhof 1984.

In the following we will apply the formalism of extended logic programming to quantity implicatures and show a difference in computational complexity between scalar and exhaustivity implicatures. We will also see how it is related to a circumscription based approach. Section 2 begins with introducing extended logic programming, which is applied to implicatures in section 3. We will discuss the results of this approach in section 4 and end with a brief conclusion.

2. Extended Logic Programming

We will use a special form of logic programming called: *Extended Logic Programming*. An extended logic program Π is a finite set of *rules* that have the following form:

$$H \quad \leftarrow \quad B_1, \ldots, B_n, not\ B_{n+1}, \ldots, not\ B_m \quad (0 \leq n \leq m)$$

$H, B_1, \ldots, B_m$ are objective literals, meaning that they are either an atom A or its explicit negation $\neg A$[1]. The set of all objective literals of a program Π is called its Herbrand base, denoted by $\mathcal{H}(\Pi)$. The symbol not stands for default negation, hence $not\ L$ is called a default literal. An *interpretation* of a program Π is a set $T \cup not\ F$[2] such that T and F are disjoint subsets of $\mathcal{H}(\Pi)$.

Ordinarily, one works with logic programs using their top-down procedural semantics. However, we will only consider declarative bottom-up semantics. Such semantics define what the valid models of a program Π are. For this paper we use the *Well-Founded Semantics with eXplicit Negation* (WFSX) (Alferes and Pereira 1996). A WFSX model M of a logic program Π is an interpretation of Π that is a fixed-point of the Φ operator, which we will not define further. Such a model M is called a *Partial Stable Model* (PSM). These PSMs can be organized into a downward complete semi-lattice with a unique minimal element called the *Well-Founded Model* (WFM).

3. Application to Implicatures

Before giving the approach, let us get some preliminaries out of the way. First of all, this is a global approach to implicatures, we assume some form of semantical representation, ie. a formula, and start from there. Furthermore, though we look at scalar implicatures, there will be no use of Horn-scales as such. Finally, implicatures are always considered in the context of an (overt) question to avoid at least some contextual issues.

3.1. Translation to a Logic Program

In a typical implicature example there are three formal elements: a question predicate, an answer formula and a domain of individuals. Take the example from the introduction, there we have $come(x)$ as the question predicate, $come(j) \vee come(m)$ as the answer formula and we take $\{j, m, b\}$ as a domain of individuals (adding b as other "relevant" people). A full definition of the function to translate this into a logic program is given in de Vries 2007, we will skip that here.

Implicatures are non-monotonic inferences. A motivation to use logic programming is its ability to deal with non-monotonicity elegantly, via the default negation

[1] The use of this explicit negation is why this version is called *extended* logic programming.
[2] $not\ \{L_1, \ldots, L_n\}$ denotes the set $\{not\ L_1, \ldots, not\ L_n\}$.

not. In this approach, we apply non-monotonic reasoning to the question predicate, which we do by introducing a *default rule* for every individual:

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

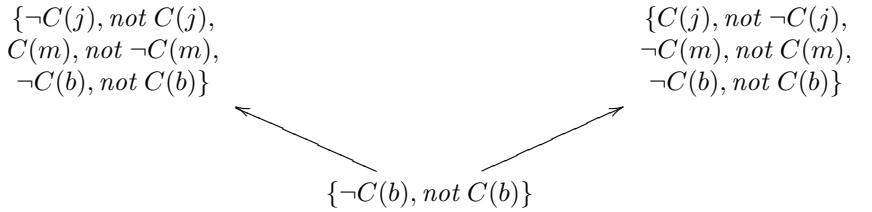
Intuitively these rules say that we can conclude the fact that a person will not come, if we cannot derive (default negation) the fact that that person will come. We assume to have all knowledge, ie. a closed world, about whether people are coming or not.

The general approach to translate the answer formula is to transform it into conjunctive normal form (CNF) and then generate rules from this form. For every disjunction in this CNF we do the following: for every literal in the disjunction we introduce a rule with that literal as its head and the rest of the literals negated in the body. In this example, $come(j) \vee come(m)$ is already in conjunctive normal form. Thus, we get the following rules, which we combine with the above rules to complete the program Π_1: [3]

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

3.2. Application of WFSX

To derive implicatures we look at the well-founded model (WFM) and the partial stable models (PSMs) of the above program Π_1. Both are given in the following semi-lattice (C stands for $come$):



The bottom element is the WFM. The WFM is enough to derive the second implicature (1b): "No one besides John or Mary is coming." After all, Bill is not coming and this would be the case for every other extra individual in the domain. For the first implicature (1a) we must look at the maximal elements in the semi-lattice, making it somewhat more complicated. In this case, these are the two PSMs at the top. In both models Mary and John don't come both, thus we conclude this as an implicature.

3.3. Quantifiers

To show the generality of the method in the previous section we will work out a more complex example. Suppose the answer in example 1 was different:

[3] Notice that a simple binary disjunction generates two rules, because the arrow in logic programming is not contrapositive.

(2) A: "Some boys are coming to the party."
Imp: Not all boys (a) and no girls (b) are coming to the party.

The answer formula is: $\exists x(boy(x) \wedge come(x))$[4], which contains an existential quantifier. If we have a finite domain of individuals, as is often the case, then such a quantifier can be translated into a large disjunction (or conjunction in the case of the universal quantifier) in which every disjunct is an instance of the original formula with the quantifier variable replaced by an individual from the domain. For this example, using the domain $\{j, m, b\}$, we get: $(boy(j) \wedge come(j)) \vee (boy(m) \wedge come(m)) \vee (boy(b) \wedge come(b))$. The conjunctive normal form of this formula can be translated to a set of rules (omitted here due to spatial reasons). Let us call this set of rules Π_2.

The use of quantifiers inevitably introduces the use of other predicates than the question. In this example we have the boy-predicate. This predicate does not require a non-monotonic interpretation (like the question predicate). It seems most intuitive to explicitly state the extension of boy, since the sex of an individual is usually something static. Thus, we add the background knowledge: $\{boy(j), \neg boy(m), boy(b)\}$ to Π_2. See de Vries 2007 on how to deal with predicates that don't have a fixed extension. For the question we add the same default rules as in example 1 to Π_2, since it has not changed.

The WFSX semantics of Π_2 is given by the following semi-lattice (C for $come$ and B for boy):

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

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

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

Again, the exhaustivity implicature (2b): "No girls are coming to the party" is derivable from the WFM, since all the girls in our domain, ie. Mary, are not coming. For the scalar implicature (2b) we look at the maximal elements. In both, there is only one boy coming, which means that we can derive: "Not all boys are coming to the party".

[4] Arguably, 'some' means 'at least two', which better captures the plurality of 'boys'. However, since we derive the same implicatures, a translation with two existential quantifiers would only unnecessarily complicate matters.

4. Discussion

There are a number of interesting things to say about the approach above: its computational complexity, the connection with circumscription and the possibility to deal with epistemically weaker implicatures.

4.1. Computational Complexity

In both examples the exhaustivity implicature is derivable from the well-founded model (WFM) and the scalar implicature requires the maximal partial stable models (PSMs). This is interesting, because the WFM is efficiently computable in polynomial time. However, to get the maximal PSMs we need to compute all of the PSMs, which at the moment requires super-polynomial time, ie. we cannot do it efficiently.

Because of this difference in computational complexity we conclude that there is a complexity difference between exhaustivity implicatures and scalar implicatures, at least for these examples. In fact, the phenomenon generalizes very well over different types of examples (de Vries 2007). Thus, we have an interesting direction for further research, both in the formal domain as well as in psychology.

In psycholinguistics there is a debate whether scalar implicatures are computed by default (Levinson 2000) or are context dependent (Carston 1998). Defaultists argue that the cancellation of scalars requires extra computational effort, while context dependents argue the opposite: the computation of the implicature itself is costly. The approach in this paper favors the latter position, since we found that scalars are computationally complex and furthermore, to work with a logic program in a traditional, top-down sense one only needs the cheaper WFM.

4.2. Connection with Circumscription

The method of this paper has a close connection to circumscription. The translation from example to logic program is similar to the work of Wakaki and Satoh 1997. Furthermore, under some, not trivial, but, non-critical, assumptions it is proven in de Vries 2007 that there is a one-to-one correspondence between the maximal elements in the semi-lattice of PSMs[5] of a program Π based on answer formula ϕ and question predicate Q and the models of the circumscription of ϕ with respect to Q. This is exactly the circumscription approach described in van Rooij and Schulz 2004. However, their final method is a two step approach which allows for weak and strong epistemic interpretations and differs from purely applying circumscription.

4.3. Epistemic Strength

One could say that the derivation of the scalar implicature, computing all the PSMs and then taking the maximal elements, looks somewhat contrived. Actually, however, this two step process is a good thing. For instance, when it comes to impli-

[5] Technically this is proven for answer-sets, however the answer-sets of a program correspond one-to-one to the maximal elements, see Alferes and Pereira 1996.

catures under negation, one often feels that an epistemically weaker implicature is required. Such an epistemically weaker interpretation can be: looking at all the PSMs, instead of considering just the maximal elements (which would be the strong interpretation). This idea is different from van Rooij and Schulz 2004, see de Vries 2007 for more.

5. Conclusion

The formalism of extended logic programming can deal nicely with quantity implicatures. The approach shows that there is a difference in computational complexity between exhaustivity and scalar implicatures, the latter being more complex.

Acknowledgements

I thank Robert van Rooij for proofreading this paper and supervising my master's thesis on which it is based.

Bibliography

Alferes, J. J. and Pereira, L. M.: 1996, *Reasoning with logic programming*, Vol. 1111, Springer-Verlag Inc., New York, NY, USA

Carston, R.: 1998, Informativeness, relevance and scalar implicature, in R. Carston and S. Uchida (eds.), *Relevance Theory: Applications and Implications*, pp 179–236, John Benjamins, Amsterdam

de Vries, G.: 2007, Formalizing implicatures using extended logic programming, *Master's thesis*, University of Amsterdam

Groenendijk, J. and Stokhof, M.: 1984, *Studies in the Semantics of Questions and the Pragmatics of Answers*, Ph.D. thesis, University of Amsterdam

Horn, L.: 1972, *On the Semantic Properties of Logical Operators in English*, Ph.D. thesis, University of California

Levinson, S.: 2000, *Presumptive Meanings*, MIT Press, Cambridge

van Lambalgen, M. and Hamm, F.: 2004, *The Proper Treatment of Events*, Blackwell

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

Wakaki, T. and Satoh, K.: 1997, Compiling prioritized circumscription into extended logic programs, in *IJCAI (1)*, pp 182–189