Using Answer Set Programming and Lambda Calculus to Characterize Natural Language Sentences with Normatives and Exceptions*

Chita Baral and **Juraj Dzifcak**
School of Computing and Informatics
Arizona State University
Tempe, AZ 85287-8809

Tran Cao Son
Computer Science Department
New Mexico State University
Las Cruces, NM 88003

Abstract

One way to solve the knowledge acquisition bottleneck is to have ways to translate natural language sentences and discourses to a formal knowledge representation language, especially ones that are appropriate to express domain knowledge in sciences, such as Biology. While there have been several proposals, including by Montague (1970), to give model theoretic semantics for natural language and to translate natural language sentences and discourses to classical logic, none of these approaches use knowledge representation languages that can express domain knowledge involving normative statements and exceptions. In this paper we take a first step to illustrate how one can automatically translate natural language sentences about normative statements and exceptions to representations in the knowledge representation language Answer Set Programming (ASP). To do this, we use λ-calculus representation of words and their composition as dictated by a CCG grammar.

Introduction

Having a knowledge base and being able to reason with it is an essential component in many intelligent systems. However, developing knowledge bases or acquiring the knowledge to put in the knowledge base is time consuming and is a bottleneck. For example, one of the challenges identified by phase 1 of Project Halo[1] was: "Knowledge and question formulation requires highly specialized and expensive personnel (knowledge engineers), which pushes the development cost to about $10,000 per page." As a result one of the main focuses in its phase 2 was developing tools for acquisition of knowledge.

We echo that knowledge acquisition is a bottleneck to developing intelligent knowledge based systems. We think one way to tackle this bottleneck is to develop ways so that knowledge expressed in natural language, say in a book, can be automatically translated to sentences in an appropriate knowledge representation language.

Existing and ongoing research in natural language semantics have explored automatic translation of natural language sentences to sentences in first-order logic. Some of these translations (Moldovan *et al.* 2002; Bos & Markert 2005) have been used in question answering systems that have participated in the TREC QA competitions (Voorhees 2006). The first one did very well in these competitions.

However, as was recognized by AI researchers very early in the AI history, first-order logic is not appropriate for expressing various kinds of knowledge. In particular, knowledge used by intelligent systems and their reasoning involves representation of and reasoning with default statements (e.g., 'most birds fly'), normative statements (e.g., 'normally birds fly'), exceptions (e.g., 'penguins are birds that do not fly'), etc. and these can not be adequately expressed in first-order logic. This has led to the development of several non-monotonic logics. Among them, answer set programming (ASP) or logic programming under the answer set semantics (Marek & Truszczyński 1999; Niemelä 1999) has been known to be a good candidate for knowledge representation and common sense reasoning (Baral 2003). More importantly, there exists a number of good (and free) ASP reasoning systems [2].

Since representation of knowledge in many domains requires the ability to express defaults, normative statements, and exceptions, our goal in this paper is to take a first step towards automatically translating natural language statements to theories in ASP.

To achieve our goal, we take the following approach. We start with a small set of sentences which we would like to be automatically translated. We give a CCG grammar (Steedman 2001; Gamut 1991) for those and other similar sentences and present an equivalent BNF grammar. We introduce the notion of λ-ASP-expressions that combines ASP rules with λ-expressions and then present λ-ASP-expressions for the various categories in our grammar. We then show how sentences built using our grammar can be systematically translated to an ASP theory by applying the λ-ASP-expressions of the various words (and their categories) in the order dictated by the CCG grammar. To make this paper understandable we give a brief background

*Partially supported by ONR grant N00014-07-1-1049 and NSF grants 0420407 and 0220590.
Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] http://www.projecthalo.com/halotempl.asp?cid=20

[2] E.g., `smodels` (http://www.tcs.hut.fi/Software/smodels/) and `dlv` (http://www.dbai.tuwien.ac.at/proj/dlv/).

of ASP, CCG and λ-expressions. Finally we conclude and discuss how to go beyond the first step in this paper.

Background

Answer Set Programming

We use a broader meaning of the term *answer set programming* (ASP) than originally used in (Marek & Truszczyński 1999; Niemelä 1999). By ASP we mean logic programming under the answer set semantics. We consider ASP as one of the most developed knowledge representation language for the following reasons. ASP is non-monotonic and is expressive enough to represent several classes of problems in the complexity hierarchy. Furthermore, it has solid theoretical foundations with a large body of building block results (e.g., equivalence between programs, systematic program development, relationships to other non-monotonic formalisms) (Baral 2003); it also has a large number of efficient computational tools. Default statements and various forms of exceptions can be naturally represented in ASP (Gelfond & Leone 2002). We now review the basic notions in ASP.

We assume a first order language $\mathcal{L}$. Literals are constructed from atoms in $\mathcal{L}$. A *positive* (or *negative*) literal is of the form A (resp. $\neg A$) where A is an atom in $\mathcal{L}$.

An *ASP program* (or *program*) is a set of rules (ASP rules) of the following form:

$$a \leftarrow a_1, \ldots, a_m,\ not\ b_1, \ldots,\ not\ b_n \qquad (1)$$

where $m, n \geq 0$ and each a, a_j, and b_t is a literal in $\mathcal{L}$; and not represents negation as failure (or default negation). For a rule r of the form (1), $head(r)$ denotes a; $pos(r)$ (*positive* body) denotes the set $\{a_1, \ldots, a_m\}$; and $neg(r)$ (*negative* body) denotes $\{b_1, \ldots, b_n\}$. Intuitively, a rule r of the form (1) states that if all the literals in $pos(r)$ are believed to be true and none of the literals in $neg(r)$ is believed to be true then the literal $head(r)$ must be true.

The notion of answer set semantics for ASP programs is defined in (Gelfond & Lifschitz 1988). Let P be a ground program[3].

Let S be a set of ground literals in the language of P. S satisfies the body of a rule r if $pos(r) \subseteq S$ and $neg(r) \cap S = \emptyset$. S satisfies a rule r if either $head(r) \in S$ or S does not satisfy the body of r. S satisfies a program P if it satisfies every rule in P. S is an *answer set* of P if it is a minimal set of literals satisfying all the rules in P^S where P^S is obtained from P by

(i) Deleting all rules from P that contain some $not\ l$ in their body and $l \in S$.

(ii) All occurrences of $not\ l$ from the remaining rules.

A program P is *consistent* if it has at least one answer set. Given a program P and a literal l, we say that P *entails* l, denoted by $P \models l$, if l belongs to every answer set of P.

ASP and Knowledge Representation

For our purpose of this paper, we will focus on default statements with strong exceptions. We illustrate the use of ASP in knowledge representation on some simple examples.

[3]Rules with variables are replaced by the set of its ground instantiations.

Consider a knowledge base consisting of information about birds, penguins, and some individuals:

- Most birds fly.
- Penguins do not fly.
- Penguins are birds.
- Tim is a bird.
- Tweety is a penguin.

This information can be represented by the following program P_1:

$$fly(X) \leftarrow bird(X),\ not\ \neg fly(X) \qquad (2)$$
$$\neg fly(X) \leftarrow penguin(X) \qquad (3)$$
$$bird(X) \leftarrow penguin(X) \qquad (4)$$
$$bird(tim) \leftarrow \qquad (5)$$
$$penguin(tweety) \leftarrow \qquad (6)$$

It is easy to check that P_1 has a unique answer set: $\{bird(tim),\ fly(tim),\ penguin(tweety),\ bird(tweety),\ \neg fly(tweety)\}$. This implies that $fly(tim)$ and $\neg fly(tweety)$ are entailed by P_1.

Assuming that the knowledge base is extended with the information about penguins being able to swim and that most birds do not swim. This information can be represented by the two rules:

$$\neg swim(X) \leftarrow bird(X),\ not\ swim(X) \qquad (7)$$
$$swim(X) \leftarrow penguin(X) \qquad (8)$$

Let P_2 be the program P_1 with the above two rules. P_2 entails $swim(tweety)$ and $\neg swim(tim)$.

In general, a default statement of the form "*Most members of a class c have property p*" can be represented by the rule

$$p(X) \leftarrow c(X),\ not\ \neg p(X)$$

which states that for every member m of the class c, unless there is contrary information about m not having property p, then m has the property p.

λ-calculus

λ-calculus was invented by Church to investigate functions, function application and recursion (Church 1936). We assume an infinite but fixed set $\mathcal{V}$ of identifiers. A λ-expression is either a *variable* v in $\mathcal{V}$; or an *abstraction* $(\lambda v.e)$ where v is a variable and e is a λ-expression; or an *application* $e_1 e_2$ where e_1 and e_2 are two λ-expressions. For example,

$$\lambda x.plane(x) \qquad \lambda x.x \qquad \lambda u \qquad y$$

are λ-expressions[4].

Variables in a λ-expression can be bound or free. In the above expressions, only y is free. Others are bound. Various operations can be done on λ-expressions. A α-conversion

[4]It is well known that predicates and functions can be easily expressed by λ-expressions. For brevity, we will often use the conventional representation of predicates and functions instead of their λ-expression.

allows bounded variables to changed their name. A substitution replaces a free variable with λ-expression. A β-reduction could be viewed as a function application, which will be denoted by the symbol @. For example,

$$\lambda x.plane(x)\ @\ boeing767$$

results in

$$plane(boeing767)$$

λ-calculus has been used as a way to formally and systematically translate English sentences to first order logic formulas. This process can be seen in the following example, taken from (Balduccini, Baral, & Lierler 2008), a translation of the sentence "*John takes a plane*" to the logical representation

$$\exists y.[plane(y) \wedge takes(john, y)]$$

The λ-expression for each constituent of the sentence are as follows:

- "John": $\lambda x.(x@john)$.
- "a": $\lambda w.\lambda z.\exists y.(w@y \wedge z@y)$
- "plane": $\lambda x.plane(x)$
- "takes": $\lambda w.\lambda u.(w@\lambda x.takes(u, x))$

We can combine the above λ-expressions to create the formula for the sentence.

- "a plane":
  $\lambda w.\lambda z.\exists y.(w@y \wedge z@y)@\lambda x.plane(x) =$
  $\lambda z.\exists y.(\lambda x.plane(x)@y \wedge z@y) =$
  $\lambda z.\exists y.(plane(y) \wedge z@y)$
- "takes a plane":
  $\lambda w.\lambda u.(w@\lambda x.takes(u, x))@$
  $\lambda z.\exists y.(plane(y) \wedge z@y) =$
  $\lambda u.(\lambda z.\exists y.(plane(y) \wedge z@y)@\lambda x.takes(u, x))$
  $\lambda u.(\exists y.(plane(y) \wedge \lambda x.takes(u, x)@y))$
  $\lambda u.(\exists y.(plane(y) \wedge takes(u, y)))$
- "John takes a plane":
  $\lambda x.(x@john)@\lambda u.[\exists y.(plane(y) \wedge takes(u, y))]$
  $\lambda u.(\exists y.[plane(y) \wedge takes(u, y)])@john$
  $\exists y.[plane(y) \wedge takes(john, y)]$

Combinatorial Categorial Grammar

Although various kinds of grammars have been proposed and used in defining syntax of natural language, combinatorial categorial grammars (CCGs) are considered the most appropriate from the semantic point of view (Steedman 2001). In building the λ-expressions above, CCG parser output would be able to correctly dictate which λ-expression should be applied to which word.

Following (Gamut 1991), a *combinatorial categorical grammar* (CCG) can be characterized by

- a set of *basic categories*,
- a set of *derived categories*, each constructed from the basic categories, and
- some syntactical rules describing the concatenation and determining the category of the result of the concatenation.

Moreover, every lexical element is assigned to at least one category. The following is an example of a very simple categorical grammar, called CCG_1:

- CCG_1 has two basic categories: N and S.
- The derived categories of CCG_1 are:
  - A basic category is a derived category;
  - If A and B are categories then the expressions $(A\backslash B)$ and (A/B) are categories;

  Thus, $(N\backslash S)$, $(S\backslash N)$, $(N\backslash(N\backslash S))$, (N/S), $(N/S)\backslash N$ are derived categories of CCG_1.
- The syntactic rule for CCG_1:
  - If α is an expression of category B and β is an expression of category $(A\backslash B)$ then the concatenation $\alpha\beta$ is of category A.
  - If α is an expression of category B and β is an expression of category (A/B) then the concatenation $\beta\alpha$ is of category A.
- CCG_1 contains the following objects: *Tim* whose category is NP and *swims* whose category is $(S\backslash NP)$.

Intuitively, the category of 'swims' is $S\backslash NP$ means that if an NP (a noun phrase) is concatenated to the left of 'swims' then we obtain a string of category S, i.e., a sentence. Indeed, 'Tim' being an NP, when we concatenate it to the left of 'swims' we obtain 'Tim swims', which is a sentence.

A Simple Language and its Automatic λ-Calculus Based Translation to ASP

In this section, we present all the ingredients and illustrate how those ingredients can be used to translate a class of natural languages sentences into ASP rules.

Example Sentences

We start with a set of sentences containing the normative 'most' and other constructs of interest to us. These are the sentences we used in the previous section.

1. Most birds fly.
2. Penguins are birds.
3. Penguins do swim.
4. Penguins do not fly.
5. Tweety is a penguin.
6. Tim is a bird.

In the above sentences, the first sentence is a default (or normative) statement expressing the fact that birds fly by default. The second sentence represents a subclass relationship. The third and fifth sentences are statement about different properties of a class (penguins). The last two sentences are statement about individuals. It should be noted that even though several approaches have been developed to automatically translate natural language sentences to first order logic, none of them translate default statements to an implemented logic (Hella, Väänänen, & Westerståhl 1997). For convenience, we will refer to the above collection of sentences as L_{bird}. We will show how those sentences can be automatically translated to ASP rules. We start with a CCG grammar for L_{bird}.

CCG for L_{bird}

The CCG for the language L_{bird} consists of the three basic categories:

- N – stands for 'noun'
- NP – stands for 'noun phrase' (representing a class)
- S – stands for 'sentence'
- $NP(obj)$ – stands for 'noun phrase' (representing an object)

We will make use of bidirectional CCG, i.e., given two categories A and B, both A/B and $A\backslash B$ are derived categories. The syntactic rules for determining the category of $\alpha\beta$ are as follows:

- (**right concatenation**:) if α is of category B and β is of category A/B then $\beta\alpha$ is of category A, and
- (**left concatenation**:) if α is of category B and β is of category $A\backslash B$ then $\alpha\beta$ is of category A.

Observe also that a word can be assigned to multiple categories.

The categories for each of the word in the language L_{bird} is given in the following table (words of similar categories are grouped together for simplicity of the reading; the reference column is for later use):

Word	Categories	Reference
fly	$S\backslash(S/(S\backslash NP))$ $S\backslash NP$	F1 F2
$flies$	$S\backslash NP(obj)$	F3
$swim$	$S\backslash(S/(S\backslash NP))$ $S\backslash NP$	S1 S2
$swims$	$S\backslash NP(obj)$	S3
$most$	$(S/(S\backslash NP))/NP$	M
do $do\ not$	$(S/(S\backslash NP))\backslash NP$ $(S/(S\backslash NP))\backslash NP$	D1 D2
is	$(S/NP)\backslash NP$ $(S/N)\backslash NP(obj)$	I1 I2
are $are\ not$	$(S/NP)\backslash NP$ $(S/NP)\backslash NP$	A1 A2
$birds$ $penguins$ $a\ penguin$ $bats$ $tweety, tim$ $fictional$ $a\ fictional$	N, NP N, NP $N, NP(obj)$ N, NP $NP(obj)$ NP/N NP/N	B1, B2 P1, P2 AP1, AP2 BA1, BA2 T F FA

Figure 1: Words in L_{bird} and their categories

The language generated by the CCG is the set of sentences whose category is S. Using the above mentioned combinatorial rules, sentences in L_{bird} can be shown to be of category S (see, e.g., (Steedman 2001)). For example, the derivation of the sentence "*Most birds fly*" is done as follows.

- *Most birds* is of category $S/(S\backslash NP)$ (applying the right concatenation rule of $B2$ to M);
- *Most birds fly* is of category S (applying the left concatenation rule of $S/(S\backslash NP)$ to F1).

Graphically, this derivation is represented as follows:

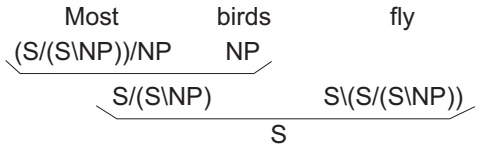

Figure 2: Derivation of the category of *Most birds fly*

A Context Free Grammar for the CCG of L_{bird}

Each CCG, whose only syntactical rules are the left- and right-concatenation rules, is equivalent to a context-free grammar (CFG) (see, e.g., (Gamut 1991)) which can be used in syntactical checking. As parsing a sentence using a CFG is often more efficient, and for many, more intuitive than using a CCG, we include a CFG equivalent to the CCG of L_{bird} in Figure 3. It should be noted, however, that the CFG lacks the directionality information that CCG has. For example, the CFG in Figure 3 will not tell us how to obtain the semantics (i.e., λ-expression) of 'most birds' from the semantics of 'most' and 'birds'. I.e., whether to apply the λ-expression of 'most' to the λ-expression of 'birds' or vice-versa. The CCG grammar gives us that information.

S	$\rightarrow$	$X_5\ X_3 \mid X_6\ NP \mid X_{11}\ N \mid X_5\ X_7 \mid NP(obj)\ X_8 \mid NP\ X_3$
X_6	$\rightarrow$	$NP\ X_1$
X_{11}	$\rightarrow$	$NP(obj)\ X_{10}$
X_5	$\rightarrow$	$NP\ X_2 \mid X_4\ NP$
NP	$\rightarrow$	$X_9\ N$
X_4	$\rightarrow$	$most$
NP	$\rightarrow$	$birds \mid bats \mid penguins$
$NP(obj)$	$\rightarrow$	$tweety \mid a\ penguin$
N	$\rightarrow$	$birds \mid bats \mid penguins \mid a\ penguin$
X_3	$\rightarrow$	$fly \mid swim$
X_2	$\rightarrow$	$do \mid do\ not$
X_1	$\rightarrow$	$are \mid are\ not \mid is$
X_7	$\rightarrow$	$fly \mid swim$
X_8	$\rightarrow$	$flies \mid swims$
X_9	$\rightarrow$	$fictional \mid a\ fictional$
X_{10}	$\rightarrow$	is

Figure 3: CFG for the CCG of L_{bird}

λ-ASP-Expression for Categories in L_{bird}

As discussed earlier, λ-expressions can be used to translate natural language sentences into first order logic representation. As our goal is to obtain an ASP representation, we expand the notion of λ-expressions to λ-ASP-Expression which allows constructs of ASP rules. We then carry over all operations on λ-expressions to λ-ASP-expressions.

In light of the above discussion, the translation of natural language sentence begins with the development of λ-ASP-expressions for words and categories in the language

of interest. For the language L_{bird}, we present the λ-ASP-expressions of various categories (a dash '–' represents all categories associated to the word) in Figure 4.

Word	Cat.	λ-ASP-expression
fly	F1	$\lambda x.fly(x)$
	F2	$\lambda x.fly(X) \leftarrow x@X$
$flies$	F3	$\lambda x.fly(x)$
$swim$	S1	$\lambda x.swim(x)$
	S2	$\lambda x.swim(X) \leftarrow x@X$
$swims$	S3	$\lambda x.swim(x)$
$most$	M	$\lambda u\lambda v.(v@X \leftarrow u@X, \; not \; \neg v@X)$
do	D1	$\lambda u\lambda v.v@X \leftarrow u@X$
$do\ not$	D2	$\lambda u\lambda v.\neg v@X \leftarrow u@X$
is	–	$\lambda v.\lambda u.u@v$
are	A1	$\lambda u\lambda v.v@X \leftarrow u@X$
$are\ not$	A2	$\lambda u\lambda v.\neg v@X \leftarrow u@X$
$birds$	–	$\lambda x.bird(x)$
$penguins$	–	$\lambda x.penguin(x)$
$a\ penguin$	–	$\lambda x.penguin(x)$
$bats$	–	$\lambda x.bat(x)$
$tweety/tim$	T	$\lambda x.x@tweety/tim$
$fictional$	F	$\lambda v\lambda u.fictional(u) \wedge v@u$
$a\ fictional$	FA	$\lambda v\lambda u.fictional(u) \wedge v@u$

Figure 4: λ-ASP-expressions for L_{bird}

λ-Calculus + CCG $\Rightarrow$ ASP Rules — An Illustration

We will now illustrate the techniques for automatic translation of several sentences in L_{bird} to ASP rules.

- Let us start with the sentence "*Most birds fly*". The CCG derivation (Fig. 2) tells us how this sentence is constructed. Namely, 'birds' is concatenated to the right of 'most' to create 'most birds'; this will be concatenated to the left of 'fly' to create a sentence whose category is S (i.e., a syntactically correct sentence). During this derivation, the category M, B2, and F1 are used for 'most', 'birds', and 'fly', respectively. From Fig. 4, we know that the λ-ASP-expression for the category M, B2, and F1 are:

  $$\begin{array}{lcl} M & : & \lambda u\lambda v.(v@X \leftarrow u@X, \; not \; \neg v@X) \\ B2 & : & \lambda x.bird(x) \\ F1 & : & \lambda y.fly(y) \end{array}$$

  Concatenating 'birds' to 'most' implies that the λ-ASP-expression for 'most birds' is obtained by applying M to B2, i.e., it is the result of

  $$(\lambda u\lambda v.(v@X \leftarrow u@X, \; not \; \neg v@X))@(\lambda x.bird(x))$$

  or

  $$\lambda v.(v@X \leftarrow \lambda x.bird(x)@X, \; not \; \neg v@X)$$

  which reduces to

  $$\lambda v.(v@X \leftarrow bird(X), \; not \; \neg v@X).$$

  The λ-ASP-expression for 'most birds fly', obtained by applying the above expression to F1, is:

  $$(\lambda v.(v@X \leftarrow bird(X), \; not \; \neg v@X))@(\lambda y.fly(y))$$

  It simplifies to:

  $$\lambda y.fly(y)@X \leftarrow bird(X), \; not \; \neg \lambda y.fly(y)@X$$

  which yields

  $$fly(X) \leftarrow bird(X), \; not \; \neg fly(X).$$

- *Penguins are birds.* This sentence is obtained by concatenating 'penguins' (P2) to the left of 'are' (A1) and 'bird' (B2) to the right of 'penguins are'. The ASP rule for this sentence is obtained by applying the λ-ASP-expression for A1 on P2 and then on B2. The first step gives us:

  $$\begin{array}{c} \lambda u\lambda v.(v@X \leftarrow u@X)@\lambda x.penguin(x) = \\ \lambda v.(v@X \leftarrow \lambda x.penguin(x)@X) = \\ \lambda v.(v@X \leftarrow penguin(X)) \end{array}$$

  The second step starts with

  $$(\lambda v.(v@X \leftarrow penguin(X)))@(\lambda y.bird(y))$$

  and results in:

  $$bird(X) \leftarrow penguin(X)$$

- *Penguins do not fly.* The CCG derivation for this sentence is similar to the previous sentence and the categories involved in the derivation are also similar.

  $$\begin{array}{c} ((\lambda u\lambda v.\neg v@X \leftarrow u@X)@(\lambda x.penguin(x))) \\ @(\lambda x.fly(x)) = \\ \neg fly(X) \leftarrow penguin(X) \end{array}$$

We can easily check that rules (2)-(8) are obtained from the automatic translation of the corresponding sentences using the techniques presented in this section.

Discussion

The grammar presented in this work is sufficient to represent an interesting set of sentences, including default statements and strong exceptions. It is, however, very simple and lacks several constructs so as to be able to capture more complex sentences. For example, it lacks conjunctions (e.g. and, or, etc.), adverbs (e.g. quickly, slowly, etc.), and other auxiliary verbs (e.g. can, might, etc.). To be able to handle most of these constructs is one of our immediate goals.

The hardest problem in dealing with these constructs lies in the requirement to properly specify the actual category representing the word. I.e., making sure that our category can only be used in grammatically correct sentences and in all such sentences, as well as ensuring that the semantical representation is what we want. We intend to address these issues in our future work. Additionally, the construction of λ-expressions presented in this work requires human engineering. We are exploring ways to make the process of acquiring these automatic.

We now give a glimpse of the representation of the conjunction 'and'. Assume that L_{bird} is extended with the sentence '*Parrots and Penguins are birds*,' with 'Parrots' of category NP. Intuitively, this suggests that the category NP/NP can be assigned to 'and'. The λ-ASP-expression for 'and' could be

$$\lambda u\lambda v.u \wedge \lambda u\lambda v.v \quad \text{shortened as} \quad \lambda u\lambda v.u \mid v$$

The λ-ASP-expressions for 'Parrots and Penguins' is

$$((\lambda u \lambda v.u \mid v)@(\lambda x.parrot(x)))@\lambda x.penguin(x) = (\lambda v.\lambda x.parrot(x) \mid v)@\lambda x.penguin(x) = \lambda x.parrot(x) \mid \lambda x.penguin(x)$$

For 'Parrots and penguins are':

$$\lambda v.v@X \leftarrow parrot(X) \mid penguin(X)$$

Adding 'birds': $bird(X) \leftarrow parrot(X) \mid penguin(X)$ which stands for two rules:

$bird(X) \leftarrow penguin(X)$ and
$bird(X) \leftarrow parrot(X)$

The above example shows that the translation of a more complex sentence may result in a formula which does not conform to the syntax of ASP rules in (1). In these cases, an intermediate representation language or/and a post processing step may be necessary.

Conclusion, Related Work and Future work

In this paper we have taken a first step to automatically translate natural language sentences that include phrases such as "normally" and "most." Such sentences are common place in many domains, such as in Biology. For example, the following is part of a sentence from (Turner *et al.* 2006): "Although the levels of the P2X(7) receptor protein in mouse kidney are normally very low ...".

Earlier we mentioned recent works where automatic translation of natural language sentences to first order logic formulas have been used in building question answering systems. However, such question answering systems are unable to answer questions that involve reasoning where non-monotonic reasoning is needed. This is discussed in (Balduccini, Baral, & Lierler 2008; Baral, Dzifcak, & Tari 2007) and an alternative is presented where facts are extracted from the natural language text and non-monotonic rules in ASP are added to do the reasoning. In this paper we go further than that and discuss how to translate natural language text to ASP theories. The work by (Kuhn 2007) is concerned with a controlled natural language and this approach cannot work on arbitrary natural language text, unlike ours. The methodologies used in our work are also completely different.

In the linguistics community, there have been many papers about representing "generics" and "generalized quantifiers" beyond "forall" and "there exists" (e.g., (Pelletier & Asher 1995)). However, these papers often propose or use relatively undeveloped, unimplemented, and narrowly studied formalisms. We hope our first step in this paper will motivate them to use established and implemented Knowledge Representation languages such as ASP. Among the closest to our work, in (Pelletier & Asher 1995) normative statements are referred to as generics and the paper discusses generics, defaults and several non-monotonic logics. But it does not give a way to automatically translate natural language sentences with generics to theories in a non-monotonic logic.

One of our future goal is also to go beyond the small set of sentences that we covered in this paper and consider additional natural language constructs. In particular, we are exploring natural language constructs involved in directing an agent or a robot and developing ways to automatically translate them to formulas in formal goal description languages such as temporal logic.

References

Balduccini, M.; Baral, C.; and Lierler, Y. 2008. Knowledge representation and Question Answering. In Vladimir Lifschitz and Frank van Harmelen and Bruce Porter., ed., *In Handbook of Knowledge Representation*.

Baral, C.; Dzifcak, J.; and Tari, L. 2007. Towards overcoming the knowledge acquisition bottleneck in answer set prolog applications: Embracing natural language inputs. In *ICLP*, vol 4670 of *LNCS*, 1–21. Springer.

Baral, C. 2003. *Knowledge Representation, reasoning, and declarative problem solving with Answer sets*. Cambridge University Press, Cambridge, MA.

Bos, J., and Markert, K. 2005. Recognizing textual entailment with logical inference. In *Proceedings of EMNLP*.

Church, A. 1936. An unsolvable problem of elementary number theory. *Am. Jou. of Mathematics* 58:345–363.

Gamut, L.T.F. 1991. *Logic, Language, and Meaning*. The University of Chicago Press.

Gelfond, M., and Leone, N. 2002. Logic programming and knowledge representation – the A-Prolog perspective. *Artificial Intelligence* 138(1-2):3–38.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP*, 1070–1080.

Kuhn, T. 2007. AceRules: Executing Rules in Controlled Natural Language. In *Proc. 1st ICWRRS*.

Hella, L., Väänänen, J., and Westerståhl, D. 1997. Definability of polyadic lifts of generalized quantifiers. *Journal of Logic, Language and Information* 6:305–335.

Marek, V., and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective*, 375–398.

Moldovan, D. I.; Harabagiu, S. M.; Girju, R.; Morarescu, P.; Lacatusu, V. F.; Novischi, A.; Badulescu, A.; and Bolohan, O. 2002. Lcc tools for question answering. In *TREC*.

Montague, R. 1970. English as a formal language. B. Visentini, et al., eds*Linguaggi nella societa e nella tecnica*.

Niemelä, I. 1999. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and AI* 25(3,4):241–273.

Pelletier, F., and Asher, N. 1995. Generics and defaults. In van Bentham, J., and ter Meulen, A., eds., *Handbook of Logic and Language*. Elsevier. 1125–1175.

Steedman, M. 2001. *The syntactic process*. MIT press.

Turner, C.; Tam, F. W. K.; Lai, P.-C.; Tarzi, R. M.; Burnstock, G.; Pusey, C. D.; Cook, H. T.; and Unwin, R. J. 2006. Increased expression of the pro-apoptotic atp-sensitive p2x7 receptor in experimental and human glomerulonephritis. *Nep. Dial. Transplant.* 22(2):386–395.

Voorhees, E.M. 2006. Overview of the TREC 2006. In *Proceedings of the Fifteenth Text REtrieval Conference*, 1. `http://trec.nist.gov/`