Lexical Reasoning

Claire Gardent
CNRS
LORIA, Campus Scientifique BP 239
Vandoeuvre-les-Nancy, France
gardent@loria.fr

Evelyne Jacquey
CNRS
ATILF,
Nancy, France
ejacquey@atilf.fr

July 14, 2003

Abstract

We first argue that lexical reasoning could help improve precision in question answering. We then indicate how to develop the various NLP tools required to perform the inference base, semantic comparison of question and answer implied by such a lexical reasoning.

1 Introduction

Reasoning is an essential part of natural language processing (NLP): it is used to disambiguate anaphora, scope and ellipses; to recover the implicit meaning conveyed by presuppositions, implicatures and entailments; and to check the felicity of producing an utterance in a given context.

In the 1970s, work on reasoning in NLP was an important trend of research with in particular Roger Schank's PhD thesis on story comprehension [17]. This work however resulted in a negative result: the amount of knowledge and reasoning necessary to fully "understand" a text was both too vast and too complex to be tractable by current reasoners. Since then much of the work in NLP has concentrated on more tractable areas of language such as morphological analysis, grammar writing and syntactic processing.

Over the last decade however, the situation has changed dramatically suggesting that the time is ripe to again investigate and attempt to model the role of inference in NLP. Three main changes have taken place. First, automated reasoners are now available which are highly efficient even when coupled with ontologies populated with several thousands of concepts ([11]). Second, large scale, online, lexical knowledge resources (e.g., [9, 5, 14]) now exist which can be used as a basis to model lexical i.e., domain independent knowledge. Third, new applications (e.g., question answering, information extraction, text summarisation) arose from the web which suggests more tractable reasoning problems than the "full text comprehension task" Roger Schank had set himself.

In this paper, we argue that lexical reasoning could help improve precision in question answering (QA). We then go on to describe an architecture which, based on existing resources, supports lexical reasoning and should permit testing and optimising existing reasoners on linguistic data.

2 Question answering and Lexical reasoning

In open domain question answering, the dominant approach consists in first, finding a set of potentially relevant passages using information retrieval techniques and second, identifying the answer by searching for an entity whose semantic type matches the expected answer type.

Light et al. [15] established an upper bound on the performance of such systems at around 70% (assuming perfect passage retrieval, named entity recognition and question classification) thus clearly showing the limitation of linguistically uninformed approaches. To improve precision, it is increasingly argued, the standard two step approach (passage retrieval followed by type matching) need to be complemented with linguistically informed techniques. In particular, [18] argues that inference could be used to compare questions and answers between and among themselves both with respect to entailment and to equivalence.

No matter how sophisticated existing theorem provers are however, the amount of knowledge needed to perform equivalence and entailment tests between two arbitrary formulas remains a bottleneck because there is neither a model nor a general ontology encoding this knowledge and because it is unclear that existing theorem provers could handle the resulting combinatorics.

Rather than scaling up directly to the full complexity of the task by assuming unrestricted corpora as well as complete world and lexical knowledge, we will here pursue a more restricted approach whose principal aim is to *develop a basic architecture permitting the integration of lexical reasoning within NLP*. In specific, the approach presented in this paper restricts the complexity of the reasoning task in the following several ways.

- It considers (in a first stage) a *single inference task* namely, the task of testing whether given the available lexical knowledge, a potential answer is a real answer to a given question.
- It concentrates on *lexical*, rather than fully blown general knowledge reasoning.
- It uses *Description Logic* (henceforth, DL), rather than the more expressive but less tractable first order logic, as a logical framework.
- It operates on a set of constructed examples rather than on open domain corpora. The underlying aim is to establish a systematic test suite on which existing theorem provers can be compared and optimised with respect to linguistic (rather than mathematical) problems. Once such an architecture is in place, it becomes possible to test existing reasoners on linguistic data, to optimise them and eventually to upgrade the approach with more extensive knowledge and reasoning capacity.

3 The general approach

In this section, we outline the general approach we are pursuing. First, we introduce the logical framework used namely Description Logic. Second, we define the inference task we are interested in investigating namely, the semantic comparison, under lexical knowledge of question and potential answers. Third, we delimit a number of lexical semantics phenomena the approach is intended to cover. Fourth, we identify the NL processing tools needed to carry out our basic inference task.

3.1 Description Logic

Description Logic (DL, [1]) provides a natural logical framework in which to reason about concepts. In this framework, concepts can be defined and ordered under subsumption whilst assertions can be made about properties of, and relations between, individuals. Moreover, inference services are available which support the retrieval both of all the concepts true of an individual and of all the individuals satisfying a given concept. In short, DLs support the definition and use of subsumption hierarchies i.e., precisely the type of hierarchies we will use to represent and classify word meanings. The particular language we assume has the following syntax.

$$C, D \rightarrow A|\top|\bot|\neg A \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C$$

The semantics of this language is given below with Δ the domain of interpretation and I the interpretation function which assigns to every atomic concept A, a set $A^I \subseteq \Delta$ and to every atomic role R a binary relation $R^I \subseteq \Delta \times \Delta$.

$$\begin{array}{rcl}
\top^I & = & \Delta \\
\bot^I & = & \emptyset \\
(\neg A)^I & = & \Delta \backslash A^I \\
(C \sqcap D)^I & = & C^I \cap D^I \\
(C \sqcup D)^I & = & C^I \cup D^I \\
(\forall R.C)^I & = & \{a \in \Delta \mid \forall b(a,b) \in R^I \rightarrow b \in C^I\} \\
(\exists R.C)^I & = & \{a \in \Delta \mid \exists b \in C^I \wedge (a,b) \in R^I n\}
\end{array}$$

3.2 The task

The task at hand consists in testing whether, given the available lexical knowledge, a potential answer is a real answer to a given question. In other words, we want to test whether the denotation of a sentence entails the denotation of another sentence under such lexical relations as synonymy, troponymy, opposition.

As a first approximation, we assume here that a potential answer A is an answer to a question Q provided its DL representation C_A is subsumed by the DL representation C_Q of the question[1] given some lexical ontology KB_{Lex} encoding lexical semantic information of the type contained in WordNet and VerbNet. That is, a text with semantic representation C_A provides the answer to a text with semantic representation C_Q provided

$$KB_{Lex} \models C_A \sqsubseteq C_Q$$

For yes/no questions, we furthermore draw on [3]'s proposal and assume that two queries are sent to the prover: one where the body (i.e., that part of the semantic representation associated with the sentence after the wh-phrase has been extracted) of the question is negated and the other where it is not. The answer is "yes" if the first query is successfull and "no" if the second query succeeds.

[1]A more sophisticated account of the relation between question and answer is given in [3] which could be used to refine the approach proposed here.

3.3 Question/Answers "lexical semantics" mismatches

Most current question answering systems select possible answers based on simple bag-of-words approach scoring by word intersection between question and potential answers. As mentioned in section 2, such approaches however have their limits and it is clear that a more semantic approach could improve precision: as [13] points out, most current QA systems cannot even differentiate between "the cat ate the mouse" and "the mouse ate the cat" as they include too little syntactic knowledge for that purpose.

In this section, we illustrate by means of examples the type of question/answer mismatches lexical reasoning could allow to cover. To do this, we assume that NL sentences are translated into DL concepts and that the relevant lexical knowledge is encoded in a DL ontology. Sections 4 and 5 then indicate how these each of these processes can be automatised.

Synonymy. Synonyms are words with equivalent meaning (usually within a given context). Taking synonymy into account will permit proving that (1a) provides a positive answer to (1b) for instance.

(1) a. Text: John builds a house.
 b. Question: Does John construct a house?.
 c. Answer: Yes.

Assuming that synonyms are represented by the same concept, then the queries testing whether (1a) is an answer to (1b) are as given in (2). Since the first query (2a) is verified, the answer is "yes".

(2) a. $\models_? (\exists \mathit{agent.John} \sqcap \mathit{BuildConstruct} \sqcap \exists \mathit{patient.House}) \sqsubseteq (\exists \mathit{agent.John} \sqcap \mathit{BuildConstruct} \sqcap \exists \mathit{patient.House})$ TRUE
 b. $\models_? (\exists \mathit{agent.John} \sqcap \mathit{BuildConstruct} \sqcap \exists \mathit{patient.House}) \sqsubseteq (\exists \mathit{agent.John} \sqcap \neg(\mathit{BuildConstruct} \sqcap \exists \mathit{patient.House}))$ FALSE

A slightly more sophisticated treatment of synonymy where concepts were defined rather than primitive would further allow to cover cases such as (3)

(3) a. Text: The bag contains nothing.
 b. Question: Is the bag emtpy?
 c. Answer: Yes.

Assuming "empty" is defined as "containing nothing" and the lexical knowledge base KB_{Lex} is updated accordingly with

$$\mathit{Empty} \doteq \mathit{Contain} \sqcap \exists \mathit{patient}.\bot$$

then the query in (4) is valid thus confirming that (3a) provides a positive answer to (3b).

(4) $KB_{Lex} \models_? (\mathit{Contain} \sqcap \exists \mathit{theme.Bag} \sqcap \exists \mathit{patient}.\bot) \sqsubseteq (\mathit{Empty} \sqcap \exists \mathit{theme.Bag})$ TRUE

Antonymy. As illustrated in (5a-b), a word mismatch between question and answer can result from the use of antonyms i.e., words with opposite meaning.

(5) a. Text: The bag is empty.
b. Question: Is the bag full?.
c. Answer: No.

Assuming that the antonymy relation between "empty" and "full" is captured by the presence in KB_{Lex} of the following statements

$$(\mathit{Full} \sqsubseteq \neg \mathit{Empty}) \qquad\qquad (\mathit{Empty} \sqsubseteq \neg \mathit{Full})$$

then (5a) can be proved to provide a negative answer to (5b) as the query in (6) will then returns true.

(6) $KB_{Lex} \models_? (\exists \mathit{theme}.\mathit{Bag} \sqcap \mathit{Empty}) \sqsubseteq (\exists \mathit{theme}.\mathit{Bag} \sqcap \neg \mathit{Full})$ TRUE

Hyponymy. Example (7) shows that the mismatch between question and answer can also come from hyponymy as "liquid, solid" and "steam" are hyponyms of "state".

(7) a. Text: The water is liquid, solid or steam.
b. Question: What are the three states of the water?

Assuming that hyponymy translates into subsumption and that therefore KB_{Lex} contains the following statements:

$$\begin{array}{rcl} \mathit{Liquiq} & \sqsubseteq & \mathit{State} \\ \mathit{Solid} & \sqsubseteq & \mathit{State} \\ \mathit{Steam} & \sqsubseteq & \mathit{State} \end{array}$$

Then if (7a-b) are assigned the following DL concept terms

$$\begin{array}{rl} A: & (\mathit{Liquid} \sqcup \mathit{Solid} \sqcup \mathit{Steam}) \sqcap \exists \mathit{theme}.\mathit{Water} \\ Q: & \mathit{State} \sqcap \exists \mathit{theme}.\mathit{Water} \end{array}$$

it follows that

$$KB_{Lex} \models (\mathit{Liquid} \sqcup \mathit{Solid} \sqcup \mathit{Steam}) \sqcap \exists \mathit{theme}.\mathit{Water}) \sqsubseteq (\mathit{State} \sqcap \exists \mathit{theme}.\mathit{Water})$$

and thus that (7a) is an answer to (7b).

Troponomy. Troponymy is another lexical relation encoded in Wordnet which can help overcome question/answer mismatches of the type illustrated in example (8).

(8) a. Text: John is running.
b. Question: Is John moving?
c. Answer: Yes.

Specifically, a verb V_2 is a troponym of some other verb V_1 if $V_1 - ing$ is $V_2 - ing$ in a certain manner. Hence "to run" is a troponym of "to move" and furthermore "to run" implies "to move". Assuming that the lexical concepts representing "runs" and "move" are:

$$\begin{array}{l} \mathit{Move} \doteq \mathit{Movement} \sqcap \exists \mathit{agent}.\mathit{Animate} \\ \mathit{Run} \doteq \mathit{Move} \sqcap \exists \mathit{manner}.\mathit{Fast} \sqcap \exists \mathit{means}.\mathit{Foot} \end{array}$$

Then provided these definitions are included in KB_{Lex},

$$KB_{Lex} \models (\mathit{Run} \sqcap \exists \mathit{agent}.\mathit{John}) \sqsubseteq (\mathit{Move} \sqcap \exists \mathit{agent}.\mathit{John})$$

is true and it can therefore be proved that (8b) is a positive answer to (8a).

3.4 The tools

Simplified as it is, the reasoning task just described (testing whether given the available lexical knowledge, a potential answer is a real answer to a given question) cannot currently be carried out using existing NLP and reasoning resources. Two things are missing: (i) a means to associate a DL based semantic representation with NL expressions and (ii) a DL ontology encoding lexical knowledge. The aim of the rest of this paper is to show how such tools can be developed on the basis of existing resources and how they can be integrated in an overall architecture which will support lexical reasoning. Specifically, the general architecture we propose consists of the following components:

1. A test suite: a set of question/answer pairs extracted from the CBC4Kids corpus or constructed on the basis of these. Question and answers are in natural language (e.g., french or english).

2. A lexical ontology: this ontology written in Description Logic and implemented in RACER encodes lexical knowledge (e.g., the rich verb semantics encoded in VerbNet [14], synonymy and antonymy between concepts as capture in WordNet etc.)

3. A TAG meta-grammar and a meta-grammar compiler which permits the semi-automatic creation of a TAG grammar on the basis of factorised syntactic and semantic information.

4. The TAG grammar produced by the meta-grammar compiler. Importantly, this grammar encodes both syntactic and semantic information so that any sentence generated by that grammar is associated not only with a syntactic structure but also with a DL based semantic representation of the type illustrated in section 3.3.

5. A parser which given a sentence returns the semantic representation(s) associated by the grammar with this sentence.

6. A description logic theorem prover (RACER) which is used to check whether the semantic representations of the potential answer entails that of the question.

In this paper, we concentrate on components 2, 3 and 4. Component 5 is already available (for example the XTAG parser is online which could be adapted to fit the grammar described in section 4) as well as component 6 (`http://kogs-www.informatik.uni-hamburg.de/~race/`). Component 1 (a systematic test suite for lexical reasoning) has yet to be specified and developed. We leave this question open for the moment.

4 Constructing semantic representations

The first obstacle against systematically testing the inference task sketched in section 3.2 is that there is no existing tool which given some NL sentence S would return a DL based semantic representation for S.

A simple way to carry this out is to define a grammar which assigns to each sentence it generates both a syntactic structure and a DL semantic representation. Given this grammar and a sentence, a parser for this grammar will then not only recognize the input sentence (provided it is grammatically correct) but also build the syntactic structure and semantic representation associated by the grammar to this sentence. We now describe such a grammar and indicate how its production can be semi-automatised.

Specifying a grammar describing both syntax and semantics. The grammar we are using is a Tree Adjoining Grammar (TAG, [12]) which is based on a flat semantic representation similar to that described in [6, 2] and uses the unification based syntax/semantic interface proposed in [10]. It can be sketched as follows (see [10] for further details).

A TAG consists of a set of elementary trees and of two tree combining operations called substitution and adjunction. In the approach presented in [10], each elementary tree is furthermore associated with a flat semantic formula representing its meaning. Further, some of the tree nodes are decorated with unification variables and constants occuring in this formula. The idea behind this is that the association between tree nodes and unification variables encodes the syntax/semantics interface – it specifies which node in the tree provides the value for which variable in the final semantic representation.

As trees combine during derivation, two things happen: (i) variables are unified – both in the tree and in the associated semantic representation – and (ii) the semantics of the derived tree is constructed from the conjunction of the semantics of the combined trees. A simple example will illustrate this.

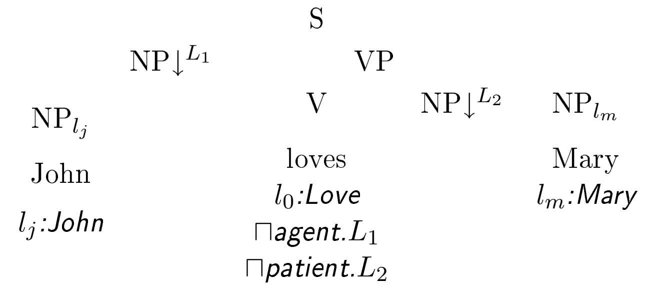

Figure 1: "John loves Mary"

Suppose the elementary trees for "John", "loves" and "Mary" are as given in Fig. 1 where a downarrow (↓) indicates a substitution node and C^x/C_x abbreviate a node with category C and a top/bottom feature structure including the feature-value pair { **index** : x}. On substitution, the root node of the tree being substituted in is unified with the node at which substitution takes place. Further, when derivation ends, the top and bottom feature structures of each node in the derived tree are unified. Thus in this case, L_1 is unified with l_j and L_2 with l_m. Hence, the resulting semantics is:

$$l_0{:}\mathit{Love} \sqcap \mathit{agent}.l_j \sqcap \mathit{patient}.l_m, \quad l_j{:}\mathit{John} \;\; l_m{:}\mathit{Mary}$$

The intuition between the flat representation used is that labels (e.g.,l_0, l_j, l_m) stand in for the formula they label[2]. Hence the above formula should be viewed as equivalent to the DL formula:

$$\mathit{Love} \sqcap \mathit{agent.John} \sqcap \mathit{patient.Mary}$$

That is, the semantic representations constructed by this grammar are compatible with the kind of semantic representations assumed in section 3.3.

Implementing this grammar. Writing an extensive grammar with DL semantics (as required by the inference task we want to investigate) is not a trivial task however. Indeed grammar writing in general (i.e., with or without a semantic dimenstion) is a reputedly time- and expertise-costly enterprise. [7] for instance estimates to 11 man/year the time needed to write ERG, one of the biggest available grammar for English. It is therefore crucial that grammar writing be made as easy as possible.

For TAG grammars, various tools have been developed which support a partial automation of grammar generation [16, 4, 19, 8]. The tool we are using is inspired from these and uses three-dimensional classes to factorise syntactic, semantic and interfacing information: one dimension is for the specification of syntactic structure using tree descriptions (i.e., trees which are underspecified with respect to dominance and/or precedence thus representing a set of possible trees rather than a single tree); a second dimension is used to specify semantic information using sets of literals; the third dimension has an interface function and supports variable sharing between classes.

Since space is too limited to define this tool in any details, we will instead try to illustrate its workings by means of the example metagrammar given in Figure 2 and from which the lexical entries for intransitive verbs (e.g., "run") can be compiled out.

The first class definition says that trees for the lexeme "run" can be obtained by conjoining the classes `Intransitive`, `LemmeRun` and `UnaryRel(run)`. The interface constraints further ensure that the values referred to by the interface features `subj` and `agent` are the same. Intuitively this last constraint ensures that the unification variable appearing on the subject node of the verb be the same as that appearing in the semantic representation associated with the verb and abbreviated by the class `UnaryRel(run)`.

The class `Intransitive` is then defined as the conjunction of the classes `SubjCan` and `ActiveMorph` where each of these two classes abbreviates a tree description capturing the syntactic configuration in an intransitive TAG tree, of a canonical subject and of a finite verb respectively.

The class `LemmeRun` abbreviates three single node tree descriptions each corresponding to one of the three possible variants of "run".

Finally the class `UnaryRel` with parameter `Pred` abbreviates a semantic representation of the form `!L:Pred ⊓ agent.?A }` with `!L` a skolemised constant, `$Pred` the value of the parameter and `?A` the value designed by the feature `agent` through the interface `[agent=?A]`.

In sum, the language used to describe a grammar supports the factorisation of structural information through tree descriptions; of semantic information through conjunction of literals; and of value sharing though interface declarations and constraints.

A compiler for this metagrammar language is currently being developed by Denys Duchier at INRIA Lorraine and tested by various linguists there for different applications. Several

[2]For more details on flat semantics and their interpretation see [6, 2]

```
class LexemeRun =
  Intransitive::[suj=?A]
& LemmeRun
& UnaryRel(run)::[agent=?A]

class Intransitive = SubjCan
                 & ActiveMorph

class SubjCan = <syn>{
        node[cat = s]{
            node(subst)[cat=n,idx=?W]
            node[cat=v]
        }
} := [suj=?W]

class ActiveMorph=<syn>{
        node[cat=s]{
            node[cat=v]{
                node}}}

class LemmeRun =
  <syn> { node [cat=v,phon=run ] }
| <syn> { node [cat=v,phon=runs] }
| <syn> { node [cat=v,phon=ran] }

class UnaryRel(Pred) =
                 <sem> {!L:$Pred ⊓ agent.?A }  := [agent=?A]
```

Figure 2: Toy metagrammar

medium size TAG grammars have already been rapidly implemented using that tool and we are currently using it to develop a TAG grammar with DL semantics which will deliver the sort of semantic representation assumed in section 3.3.

5 Lexical knowledge and Description Logic Ontology

As illustrated by the examples in (3.3), the lexical knowledge required to bridge words mismatches between questions and answers, includes *inter alia*: synonymy (words with similar meaning e.g., build/construct), antonymy (words with opposite meanings e.g., dry/wet but also full/empty), hyperonymy (more or less general means of referring to an object e.g., liquid,solid,steam/state), troponymy (further specification of a verb semantics e.g., move/run) and fine grained verb semantics (e.g., the fact that giving Y to Z implies that Z has Y).

The four lexical relations of synonymy, antonymy, hyperonymy, troponymy have been extensively studied and encoded within WordNet. Thus for these relations, a data base exists which provides the knowledge required to establish that the assertions in 3.3 entails their corresponding questions and are thus positive answers to these same questions.

This knowledge base however does not support logical inference: it is not equiped with a reasoner which allows one to check whether one proposition entails another. To put this knowledge base to work, we now sketch how this knowledge could be encoded into a DL knowledge base.

Synonyms. In WordNet, quasi-synonyms (i.e., words that have the same meaning within a certain context) are grouped within so-called "synsets" the intuition being that synsets stand for lexicalised concepts. Furthermore, each synset is associated with a numerical identifier and with a gloss giving the intuitive meaning of the lexicalised concept. So for instance, the synset { robin, American robin, Turdus migratorius } is associated with the gloss *Large American thrush having a rust-red breast and abdomen .*

A simple encoding of synonymy within DL would consist in associating WN synset elements (i.e., quasi-synonyms) with the same DL concept. In a first approximation, such DL concepts might be undefined (for instance, "robin", "American robin" and "Turdus migratorius" would be associated with the undefined DL concept *Robin*). In a further refinement step, one could try to define this concept e.g., on the basis of the gloss so that for instance, *Robin* might be defined as

$$\mathit{Robin} \doteq \mathit{large} \sqcap \mathit{American} \sqcap \mathit{Thrush} \sqcap \exists \mathit{has}.(\mathit{RustRedBreast} \sqcap \mathit{Abdomen})$$

Hyponymy. WordNet systematically encode hyponymy relations between synsets thus defining a taxonomy. For instance the hyperonyms for the synset {robin, American robin, Turdus migratorius} are:

```
robin, American robin, Turdus migratorius
       => thrush
           => oscine, oscine bird
               => passerine, passeriform bird
                   => bird
                       => vertebrate, craniate
                           => chordate
                               => animal, animate beast, creature, fauna
                                   => organism, being
                                       => living thing, animate thing
                                           => object, physical object
                                               => entity, physical thing
```

Assuming as above that each synset Syn_k translates into a DL concept C_k, then for each Synset pair Syn_i , Syn_j in WordNet s.t. Syn_i is a hyperonym of Syn_j, a DL statement of the form $C_i \sqsubseteq C_j$ is added to KB_{Lex}.

Opposition. WordNet encodes both antonymy (a relation between paired words e.g., heavy/light) and semantic opposition (a relation between words related by antonyms e.g., heavy/weightless). Both relations can be used to capture the type of mismatches illustrated in section 3.3. Specifically, given two words w_1, w_2 and their associated DL concepts C_1, C_2 , if w_1, w_2 are either

WN opposites or WN antonyms, then either

$$C_1 \sqsubseteq \neg C_2 \quad and \quad C_2 \sqsubseteq \neg C_1 \tag{1}$$

or

$$C_1 \doteq \neg C_2 \tag{2}$$

The first case is illustrated by the pair full/empty, the second by the pair dry/wet.

Troponymy. As we saw in section 3.3, a verb V_2 is a troponym of some other verb V_1 if $V_1 - ing$ is $V_2 - ing$ in a certain manner M. Generally, this implication relation between concepts can be captured by encoding the lexical concepts associated with troponym pairs according to the following schema:

> If V_2 is a troponym of V_1, C_2 is the DL concept representing the meaning of V_2 and $V_1 - ing$ means $V_2 - ing$ in a certain manner M, then the DL concept representing the meaning of V_1 should be of the form $\mathsf{C_2} \sqcap \exists \mathsf{manner}.\mathsf{M}$ (with M the DL concept representing M).

The table in Figure 3 summarises the relation between WN relations and DL encoding (with τ a function mapping words onto DL concepts).

w_1, w_2 are in the same synset	$\tau(w_1) = \tau(w_2)$
w_1 is a hyponym of w_2	$\tau(w_1) \sqsubseteq \tau(w_2)$
w_2 is a troponym of w_1 w.r.t manner M	$\tau(w_1) \doteq \tau(w_2) \sqcap \exists \mathsf{manner}.\tau(M)$
w_1 is an opposite of w_2	$(\tau(w_1) \sqsubseteq \neg\tau(w_2)$ and $\tau(w_2) \sqsubseteq \neg\tau(w_1))$ or $(\tau(w_1) \doteq \neg\tau(w_2))$

Figure 3: Relation between WN relations and DL encoding

In sum, there are some obvious ways in which existing lexical bases could be used to construct DL ontologies supporting lexical reasoning. Clearly, the translation techniques just presented are only a first approximation of what could and should be done. In particular, other lexical bases should be considered (e.g., VerbNet and Framenet) and finer grained distinctions need to be made (e.g., to differentiate between the various types of troponymy encoded in WordNet). There is also the issue of ensuring that the constructed ontology remains consistent (it is unlikely that the hand-constructed WordNet does not contain inconsistencies). Nonetheless these translation techniques give us some preliminary guidelines on how to proceed and we are currently developing the software necessary to translate WordNet into a DL ontology that can be used to support lexical reasoning.

6 Conclusion

We have given some evidence that lexical reasoning could help improve precision in question answering. Furthermore, we have indicated how the various NLP tools required to perform an inference based, semantic comparison of question and answer, could be developed.

Current work concentrates on developing these tools, specifying a test suite and validating the approach on the resulting test suite.

References

[1] F. Baader, D. Calvanese, D. Hardi D. McGuiness, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge, 2003.

[2] J. Bos. Predicate logic unplugged. In Paul Dekker and Martin Stokhof, editors, *Proceedings of the 10th Amsterdam Colloquium*, pages 133–142, 1995.

[3] Johan Bos and Malte Gabsdil. First-order inference and the interpretation of questions and answers. In *Proceedings of Goetalog 2000. Fourth Workshop on the Semantics and Pragmatics of Dialogue*, 2000.

[4] Marie-Helene Candito. *Representation modulaire et parametrable de grammaires electroniques lexicalisees.* PhD thesis, University of Paris 7, 1999.

[5] C.Johnson, C. Fillmore, M. Petruckand C. Baker, M. Ellsworth, and J. Ruppenhofer. Framenet: Theory and practice. Technical report, Berkeley, 2002.

[6] A. Copestake, A. Lascarides, and D. Flickinger. An algebra for semantic construction in constraint-based grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, Toulouse, France, 2001.

[7] Ann Copestake and Dan Flickinger. An open-source grammar development environment and broad-coverage english grammar using hpsg. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000)*, Athens, Greece, 2000.

[8] Benoit Crabbe and Bertrand Gaiffe. A new metagrammar compiler. In *Proc. of the sixth TAG+ Workshop*, Venice, Italy, 2002.

[9] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[10] Claire Gardent and Laura Kallmeyer. Semantic construction in ftag. In *Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary, 2003.

[11] Volker Haarslev and Ralf Moller. High performance reasoning with very large knowledge bases: A practical case study. In *IJCAI*, pages 161–168, 2001.

[12] A. K. Joshi and Y. Schabes. Tree-Adjoning Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69–123. Springer, 1997.

[13] Boris Katz and Jimmy Lin. Selectively using relations to improve precision in question answering. In *EACL03 workshop on question answering*, Budapest, 2003.

[14] Karin Kipper, Hoa Trang Dang, and Martha Palmer. Class based construction of a verb lexicon. In *Proceedings of AAAI-2000 Seventeenth National Conference on Artificial Intelligence*, Austin TX, July 30 - August 3 2000.

[15] M. Light, G. Mann, E. Riloff, and E. Breck. Analyses for elucidating current question answering technology. *Journal of Natural Language Engineering*, 2001.

[16] Gerald Gazdar Roger Evans and David Weir. Encoding lexicalised tree adjoining grammars with nonmonotonic inheritance hierarchy. In *Proceedings of ACL95*, 1995.

[17] R.C. Schank. Conceptual dependency: a theory of natural language understanding. *Cognitive Psychology*, 1972.

[18] Bonnie Webber, Claire Gardent, and Johan Bos. Position statement: Inference in question answering. In *Proceedings of the third international conference on language resources and evaluation (LREC)*, Las Palmas, Canary Islands, Spain, 2002.

[19] Fei Xia, Martha Palmer, K. Vijay-Shanker, and Joseph Rosenzweig. Consistent grammar development using partial-tree descriptions for ltags. In *Proc. of the fourth TAG+ Workshop*, Philadelphia, 1998.