Using Description Logics for Recognising Textual Entailment

Paul Bedaride

Using Description Logics for Recognising Textual Entailment

PAUL BEDARIDE

Loria – UHP Nancy 2 – Talaris

paul.bedaride@loria.fr

ABSTRACT. The aim of this paper is to show how we can handle the Recognising Textual Entailment (RTE) task by using Description Logics (DLs). To do this, we propose a representation of natural language semantics in DLs inspired by existing representations in first-order logic. But our most significant contribution is the definition of two novel inference tasks: *A-Box saturation* and *subgraph detection* which are crucial for our approach to RTE.

1   Introduction

Recognising textual entailment (RTE) is performing the following task: given two texts T_1 and T_2 in natural language, determine if we can infer T_2 from T_1. As an example consider the three following sentences:

A: *"Adam has a son who has a son"*,

B: *"Adam has an offspring who has an offspring"*,

C: *"Adam is a grandfather"*

We can infer B from A because a son is an offspring. We can also say that B and C are equivalent because a grandfather is a male who has an offspring who has an offspring and because *Adam* is a male name. But we cannot infer A from B or C because an offspring can be a son or a daughter.

As we can see, recognising textual entailments is far from trivial, involving many issues that are difficult to solve. The main issue is that natural language is highly expressive. Due to this expressivity, it is possible to express the same meaning in several ways, as in B and C. Furthermore modifying, adding or deleting a word in a sentence can completely change its meaning (e.g., *Adam (dis)likes Eve*). Another important issue, related to the first, is that there exists a huge number of synsets (i.e., sets of words with the same meaning). It is difficult to exactly map the relation among them (e.g., an offspring is a child) and to represent all background knowledge

needed for detecting textual entailment. However, as the RTE task is widely considered to be relevant for such tasks as Question-Answering, information retrieval, multi-document summarization and information extraction, the task has received a great deal of attention in recent years.

Several different approaches to this task have been proposed and some of them have been compared in the RTE Pascal challenge [1]. This challenge compares the different approaches using a corpus of annotated pairs of texts, usually referred to as T for Text and H for Hypothesis. For each pair, it is specified whether T entails H or not. One outcome of this comparison is that symbolic methods perform better than statistical methods. Symbolic methods — using techniques and intuitions rooted in semantics, syntax, logic, etc — typically have about 75% accuracy. Statistical methods — based on techniques like n-grams, lexicon, etc — have about 60% accuracy. The method that we describe in this paper is symbolic. It differs from other symbolic methods because it uses Description Logics. The first important reason to choose these logics for the RTE task is that they are decidable and there exists highly optimized reasoners (e.g., RACER [2]) for different inference tasks. Moreover, we can (at least partially) represent background knowledge and the semantic representation of sentences in these logics. Other symbolic techniques which have already been investigated for the RTE task (e.g., lexical alignment to detect synonyms), could perhaps be integrated into our approach, improving the performance, but we are not going to discuss these possibilities here.

As Description Logics (DLs) are the core of our approach to textual entailment, we will start with a very brief introduction to these formalisms. Description Logics are formal languages for knowledge representation. They were inspired by Quillian's semantic network [3] and Minsky's frame semantics [4]. DLs classify knowledge in two parts: the T-Box and the A-Box. The T-Box contains terminological information which is general (good for representing background knowledge). The A-Box contains assertions which are specific (good for representing sentences). Another way to see the division between these two kinds of information is to regard the T-Box as rules which govern our world (e.g., laws from physics, chemistry, biology, etc), and the A-Box as depicting the world's individuals (e.g., a table, a chair, a man, etc).

Description Logics employ the notions of concept, role and individual. Concepts are classes of elements and are interpreted as a subset of a given universe. Roles are links between elements and are interpreted as binary relations of a given universe. Individuals are the elements of a given universe.

A knowledge base Σ is a pair $\langle T, A \rangle$. T is the T(erminological)-Box, a finite set of expressions called General Concept Inclusions (CGI) with shape $C_1 \sqsubseteq C_2$ where C_1, C_2 are concepts. The intended meaning of $C_1 \sqsubseteq C_2$ is that the set of individuals in C_1 is included in the set of individuals in

C_2. $C_1 \doteq C_2$ is a notation for $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. Formulas of T are also called terminological axioms. A is the A(ssertion)-Box, a finite set of expressions with shape a:C or (a, b):R where C is a concept, R a role and a,b two individuals. The first expression means that the individual a belongs to the set of individuals satisfying C. The second expression means that the relation R holds between a and b. Formulas of A are called assertions.

In the description logic that we used, which is known as $\mathcal{ALCI}$ [5], we can form complex concepts from atomics concepts. They can be made up by negation ($\neg$), conjunction ($\sqcap$) and disjunction ($\sqcup$) of concepts. Roles can either be atomic, or the inverse (R^-) of an atomic role. We can also use the universal quantifier ($\forall$ Role.CONCEPT) to form a complex concept which is true for an individual i if all roles Role which have for first argument i, have for second argument an individual for whom CONCEPT is true. The existential counterpart is defined as in first order logic: ($\exists$Role.CONCEPT) $\equiv$ ($\neg\forall$Role.$\neg$CONCEPT).

Several reasoning tasks can be handled in DLs once we have defined a knowledge base $\langle T, A \rangle$. For example, *instance checking* tests if an individual is an instance of a specified concept. *Relation checking* tests if there exists a relation between two individuals. *Knowledge base consistency* tests if $\langle T, A \rangle$ is consistent. These tasks can be used for defining more complex tasks such as *query individuals* which find all instances of a concept.

We will define two novel reasoning tasks to use DLs for RTE. The most important of these is the *subgraph detection task*, which we will discuss in detail later; here we'll introduce the simpler *A-Box saturation* task. This consists of completing A-Box information according to a given T-Box. Given a knowledge base $\langle T, A \rangle$, we say that A' is a saturation if for each individual a, atomic concept C and role R appearing in $\langle T, A \rangle$ there is an assertion a:C in A' if and only if $\langle T, A \rangle \models$ a:C, and an assertion (a, b):R in A' if and only if $\langle T, A \rangle \models (a, b)$:R.

For example we can have the following T-Box:

$$T = \left\{ \begin{array}{ccc} \text{PARENT} & \doteq & \exists\text{Parent-of.SOMEONE} \\ \text{GRANDFATHER} & \doteq & \exists\text{Father-of.PARENT} \end{array} \right\}$$

expressing respectively that *parent* is equivalent to *someone who is the parent of someone* and *grandfather* is equivalent to *someone who is the father of someone who is a parent*. We can also represent the sentence *"Adam is the father of someone who is the parent of someone"* by the following assertions:

$$A = \left\{ \begin{array}{c} \text{a:ADAM, s1:SOMEONE, s2:SOMEONE} \\ (\text{a}, \text{s1})\text{:Father-of}, (\text{s1}, \text{s2})\text{:Parent-of} \end{array} \right\}$$

By applying the T-Box to the A-Box we can deduce that s1 is a PARENT thanks to the first rule and that a is a GRANDFATHER thanks to the second rule. If we add the two pieces of information to the A-Box, we obtain a saturated A-Box.

There exist automatic theorem provers for different DLs including $\mathcal{ALCI}$ the logic we are going to use. They handle efficiently several reasoning tasks, including instance and relation checking, concept and knowledge base consistency, and getting all instances of a concept. They can also perform A-Box saturation, but the more complex subgraph detection task will require a new algorithm.

2 Representation of sentences in DLs

To start with, we will explain how the meaning of a sentence can be partially represented as a DL formula. We say partially because the expressivity of DLs is limited and the meaning of a text is complex, so our representation is an approximation of the actual meaning of the text. For instance, many syntactic elements such as articles, quantifiers, and modalities will not be considered in our approach. The sentence *"The cat eats an apple"*, for example, will be approximated by *"cat eat apple"*.

During the definition of our representation we should remember that our final goal is recognising textual entailment, hence we should struggle to have the same representation for the same meaning whenever possible. The main idea of our approach is to represent each sentence by an A-Box, the background knowledge by a T-Box and then to check if the model of the entailed sentence is a subgraph of the graph of the entailing sentence.

We now describe our approach step by step. We first discuss how to represent sentences in DLs. We start by introducing predicate-arguments dependencies, then we discuss modifiers, and we finish by explaining adjectives and negation.

Predicates-arguments dependencies. Our representation of sentences is based on Davidson's semantics [6] which represents events as individuals. For example, the sentence *"John loves Mary"* is represented by the first order formula $love(e) \wedge john(j) \wedge mary(m) \wedge agent(e,j) \wedge patient(e,m)$. Here e stands for the event of *loving*, j stands for the individual named *John*, and m stands for the individuals named *Mary*. j is the agent of the event e, and m is its patient.

By using Davidson's semantics we only need to use unary or binary predicates. This fits well with our DLs approach by making a correspondence between unary predicates and concepts, and binary predicates and roles. The sentence *"John loves Mary"* is represented in DL as `e:LOVE`, `j:JOHN`, `m:MARY`, `(e,j):Agent`, `(e,m):Patient`.

We have agreed then on a semantic form, but we don't know which set of basic concepts and roles we will use for representing the meaning of words and the relations between words. To define our signature (i.e., the set of basic concepts and roles), we use the linguistic database FrameNet [7]

4

based on frame semantics. FrameNet is composed of semantic frames which involve frame elements and which are evoked by certain lexical units. For instance, the *Commercial_transaction* frame describes a common situation involving a *buyer*, a *seller*, some *goods* and some *money* and it is evoked by such words as *buy, sell, pay, cost, spend*, etc.

To specify the signature which allows us to represent verb semantics, we link the frame semantics to our representation, and we link frames to concepts which represent the sense of verbs, and frame elements to relations which connect verbs to their arguments.

For example, when we want to represent a verb like *sell*, we start by looking up in FrameNet the corresponding frame. FrameNet tells us that the concept *sell* is represented by the frame `COMMERCIAL_TRANSACTION` and by the thematic relations `Buyer`, `Seller`, `Goods` and `Money`. Then for the sentence *"Adam buys chocolate in the supermarket for 2 euros"*, we have the following representation as a DL A-Box:

$$A = \left\{ \begin{array}{c} \texttt{ct:COMMERCIAL\_TRANSACTION} \\ \texttt{a:ADAM, s:SUPERMARKET, c:CHOCOLATE, p:2\_EUROS} \\ \texttt{(ct, a):Buyer, (ct, s):Seller, (ct, c):Goods, (ct, p):Money} \end{array} \right\}$$

Verb modifiers. The meaning of a verb can be affected by modifiers (e.g., place, time, manner, etc). For example in the sentence *"The dog barks loudly"*, *loudly* affects the meaning of *bark* by adding to it the fact that the sound produced by the bark is noisy. We must be able to say that *"The dog barks loudly"* entails that *"The dog barks"* but not the converse. Verbs may have many modifiers of the same type, but this is not a problem with Davidson-style representations. For each modifier we simply conjoin a concept which represents the modifier sense to the A-Box individual corresponding to the verb. For example, the sentence *"John bought a car on Monday 8 may at 5pm"* has the following representation:

$$A = \left\{ \begin{array}{c} \texttt{ct:COMMERCIAL\_TRANSACTION} \sqcap \texttt{MONDAY} \sqcap \texttt{8\_MAY} \sqcap \texttt{5PM} \\ \texttt{j:JOHN, c:CAR} \\ \texttt{(ct, j):Buyer, (ct, c):Goods} \end{array} \right\}$$

Adjectives. In Davidson's approach, adjectives are represented as unary predicates applied to the variable which represents the word to which the adjective is applied. This representation can easily be used in DLs for adjectives that modify nouns; such adjectives are essentially treated in the same way that verb modifiers are. But adjectives can also occur following the copula as in *"The cat is big"*. How do we treat them? As our final goal is to recognize textual entailment, we have to be able to check that *"The big cat"* is equivalent to *"The cat is big"*.

The simplest way to recognise textual entailment is to have the same representation for the same meaning. We will thus represent adjectives and

the verb *to be* in the same way. That is, for adjectives we add a concept representing the adjective to the event individual which represents the word to which the adjective is applied. And we consider the verb *to be* as an isolated verb; we don't create any individual for it, but we add to the individual representing the subject of the verb the concept representing the verb's copula. That is, *"the big cat"* and *"the cat is big"* will be represented in the same way, by: `c:CAT ⊓ BIG`.

Negation. Even though, we have negation in our representation language, modeling natural language negation is difficult. The problem is scope. For example, for the sentence *"The dog doesn't bark loudly"* there are two possible interpretations. In the first interpretation, negation takes narrow scope and applies to *loudly*. In this reading we mean that the dog barks but that it doesn't bark loudly. In the second interpretation, negation takes wide scope and applies to *bark loudly*. It means that we don't know if the dog barks, but if it barks it doesn't do it loudly.

Scope is an ubiquitous phenomenon in natural language. Besides negation, it also plays a role for quantifiers and verb arguments (e.g., *"John sees the girl with the telescope"*). We can try to analyse all possibilities, but this soon leads to an exponential blowup (e.g., two negations in a sentence can give rise to four different interpretations, three negation to eight different meanings, and so on). Moreover, we must have a YES or NO answer for the RTE task, hence what should we do if the possibilities don't all agree?

Our choice of representation for negation is motivated by our mechanism for recognising textual entailment. This mechanism is a mix between logical implication and syntactic similarity. Let's analyse a concrete example. We take the following sentences: (A) "John didn't buy a fruit", (B) "John didn't buy a fruit at midnight", (C) "John didn't buy an apple", and (D) "John didn't buy a big fruit" because they represent the most common kinds of scope negation. With a standard reading of the sentences we are able to detect the following entailments (and only those): (A)⇒(B), (A)⇒(C) and (A)⇒(D).

To detect (A)⇒(B), we must have a logical implication between the negation of the verb (i.e., *"buy"*) and the negation of the verb and its modifiers (i.e., *"buy at midnight"*). So we must have a scope for the negation on verb and its modifier, because otherwise we won't detect $\neg$`BUY` ⊑ $\neg$(`BUY` ∧ `MIDNIGHT`). To detect (A)⇒(C), we must have a logical implication between the concept `FRUIT` and the concept `APPLE`. Lexical knowledge will give us the implication `APPLE` ⊑ `FRUIT`, but we need the contraposed form $\neg$`FRUIT` ⊑ $\neg$`APPLE`. So we must have the negation of concepts associated with verb objects to detect this textual entailment. Finally, to detect (A)⇒(D), we must have a logical implication between the negation of the verb arguments (i.e., *"fruit"*) and the negation of the verb arguments and

6

their adjectives. This is similar to the first case, so we have a scope for the negation on the verb arguments and their adjectives $\neg(\mathtt{FRUIT} \wedge \mathtt{BIG})$.

3 Representing knowledge

Now that we have seen how to represent sentences in DLs by encoding them into the A-Box, we will see how we use background knowledge to detect textual entailments such as *"a cat eats"* $\Rightarrow$ *"an animal eats"*. The knowledge required to detect this entailment is lexical knowledge which explains that a cat is an animal, thus that the $\mathtt{CAT}$ concept is subsumed by the $\mathtt{ANIMAL}$ concept.

We use two repositories of lexical knowledge to detect textual entailment. The first is FrameNet, which we already used to represent text with the same meaning in the same way. The second is WordNet [8], which records different lexical relations between synsets, like synonymy, antonymy or hyponymy. To check that T entails H, we retrieve SLT and SLH, the synsets list of the words of T and H using WordNet. For each synset ST and SH of SLT and SLH we check if there exists a lexical relation between them. If there exists a synonymy relation between ST and SH we add the following CGI to the background knowledge: $\mathtt{ST} \doteq \mathtt{SH}$. If there exists an antonymy relation between ST and SH we add the following CGI to the background knowledge: $\mathtt{ST} \sqsubseteq \neg\mathtt{SH}$ and $\mathtt{SH} \sqsubseteq \neg\mathtt{ST}$. And finally for the hyponymy relation we have three different cases. If ST is an hyponym of SH then we get the following CGI: $\mathtt{SH} \sqsubseteq \mathtt{ST}$. If SH is an hyponym of ST then we get $\mathtt{ST} \sqsubseteq \mathtt{SH}$. And if SH and ST share an hyponym we get $\mathtt{ST} \sqsubseteq \neg\mathtt{SH}$ and $\mathtt{SH} \sqsubseteq \neg\mathtt{ST}$.

For example, to detect the textual entailment *"a cat eats"* $\Rightarrow$ *"an animal eats"* we check lexical relations between senses of *cat* and *animal* using WordNet. We get that *cat* is an hyponym of *animal* and we obtain the CGI: $\mathtt{CAT} \sqsubseteq \mathtt{ANIMAL}$. By applying this CGI to the representation of *"a cat eats"* we obtain the following saturated A-Box for the sentence *"a cat eats"*:

$$A = \{\mathtt{i}{:}\mathtt{INGESTION}, \mathtt{c}{:}\mathtt{CAT} \sqcap \mathtt{ANIMAL}, (\mathtt{i}, \mathtt{c}){:}\mathtt{Ingestor}\}$$

4 Inference detection - Subgraph detection

We have now a way to represent sentences and use background knowledge to detect textual entailment, and this brings us to the second, and more complex, of our novel inference tasks: subgraph detection. It remains to specify how we check if a sentence entails another sentence. To understand what this involves, we must first note that a saturated A-Box can be represented as one or more oriented and labeled graphs (see, for example Figure 1.1). What we call subgraph detection is divided into three steps. First, we create A-Boxes for the pair (T,H). Then we saturate them with the T-Box created
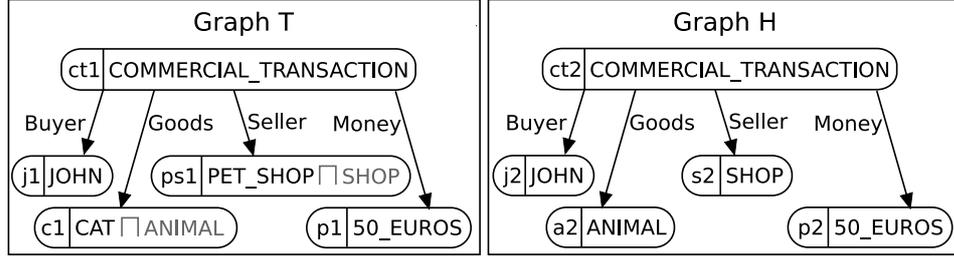
Figure 1.1: The graph H is a subgraph of the graph T

by using WordNet. Finally we traverse the graphs corresponding to these saturated A-Boxes to check if the second is a subgraph of the first. By doing this we verify if all the information in H is also in T. The algorithm is shown on Figure 1.2.

We need to do this because existing theorem provers for DLs focus on tasks which involve one A-Box and one T-Box. There is no existing tool which handles relations between two DL knowledge bases, and this is what we required for RTE.

To illustrate our algorithm, we use the example in Figure 1.1 which aims to show the entailment between the sentence T: *"John buys a cat at the pet shop for 50 euros"* and the sentence H: *"A shop sells an animal for 50 euros to John"*. These sentences are represented by the following A-Boxes:

$$T = \left\{ \begin{array}{c} \texttt{ct1:COMMERCIAL\_TRANSACTION} \\ \texttt{j1:JOHN, ps1:PET\_SHOP, c1:CAT, p1:50\_EUROS} \\ \texttt{(ct1, j1):Buyer, (ct1, ps1):Seller, (ct1, c1):Goods, (ct1, p1):Money} \end{array} \right\}$$

$$H = \left\{ \begin{array}{c} \texttt{ct2:COMMERCIAL\_TRANSACTION} \\ \texttt{j2:JOHN, s2:SHOP, a2:ANIMAL, p2:50\_EUROS} \\ \texttt{(ct2, j2):Buyer, (ct2, s2):Seller, (ct2, a2):Goods, (ct2, p2):Money} \end{array} \right\}$$

We compute the background knowledge for detecting the entailment between these two sentences by using WordNet and we obtain the following T-Box:

$$BK = \left\{ \begin{array}{ccc} \texttt{CAT} & \sqsubseteq & \texttt{ANIMAL} \\ \texttt{PET\_SHOP} & \sqsubseteq & \texttt{SHOP)} \end{array} \right\}$$

By applying this background knowledge to the DLs representation of the sentences T and H we obtain the graphs of the Figure 1.1.

Now we need to check whether a graph GH is a subgraph of another graph GT. Our approach is divided in two parts: node checking and arc checking. The first step consists in checking that there exists a function f which links to each node NH of GH a node NT of GT such that the concept associated to NH is subsumed by the concept associated to NT. The next

```
deftype BIJ = Dict: {IND -> IND}
deftype UNBIJ = Dict: {IND -> (List: IND)}

/* the main function */
def main(t:ABOX, h:ABOX) : BOOL is
  bij:BIJ // nodes that have just one correspondance
  unbij:UNBIJ // nodes that have more than one correspondance
  foreach ind in h.getIndividuals do //get bij and unbij
    inds = t.indsSatisfying(ind.concepts)
    if len(inds)>1 then
      unbij[ind] = inds
    elif len(inds)==1 then
      bij[ind] = inds
    else
      print ''no correspondence for individual '' + ind.name
      stop
  return testAllBijection(bij,unbij)

/* find all bijections and test them */
def testAllBijections (bij:BIJ,unbij:UNBIJ) : BOOL is
  if len(unbij)<=0 then
    testBijection(bij) // we have a bijection and we test it
  else
    ind, List[Ind] = unbij.pop
    foreach i in List[Ind] do
      if testAllBijections(bij+(ind,[i]),unbij) then
        return True
    unbij.append(ind,Lisd[Ind])
    return False

/* test if with this bijection the entailment is correct */
def testBijection(bij:BIJ) : BOOL is
  foreach (src,trg,name) in h.getRelations do: // test all relations
    if not t.hasRelation(bij[src],bij[trg],name) then
      return False
  return True.
```

Figure 1.2: Algorithm for subgraph detection

step is to check for each arc A of GH between nodes N1 and N2 if there is an arc between $f(\text{N1})$ and $f(\text{N2})$ in GT which have the same label as A.

Now that we know how to check if a graph GH is a subgraph of a graph GT, we will check if the graph of the sentence H is a subgraph of the graph of the sentence T. The first step is to find if there exists a function f. In our example, finding this function is easy, and it is defined like this: $f(\text{ct2}) = \text{ct1}$, $f(\text{j2}) = \text{j1}$, $f(\text{a2}) = \text{c1}$, $f(\text{s2}) = \text{ps1}$, $f(\text{p2}) = \text{p1}$. Now for the second step we must check arcs, and we can see easily that the arcs of the graph of the sentence H exist in the graph of the sentence T via the function f. For instance, the arc Buyer between ct2 and j2 exists between $f(\text{ct2}$ and $f(\text{j2})$, that is to say ct1 and j1.

We have used a simple example, but subgraph checking works with more complex graphs. By more complex graphs we mean graphs containing identically labelled nodes, or more than one relation between two nodes. The limit of our algorithm is when we have existentially quantified information, because in the saturated A-Box we don't expand existentials. So if we compare the saturated A-Boxes of *"Adam is the father of someone who is a parent of someone"* and *"Adam is a grandfather"* we will have many nodes in the first sentence and only one in the second. Thus the first sentence implies the second but not the converse.

9

5   Tests and Conclusion

To test our algorithm we have made an implementation in Python which uses the DL prover RACER. The application takes a file as input which contains pairs T,H of texts which have been annotated by hand with respect to whether T entails H or not, and it generates the semantics[1] by using the C&C Tools and Boxer [9], and adds the relevant lexical knowledge to detect entailment using WordNet. After computing all this information, it will use the algorithm we describe in the previous section to test if T entail H. We have tested our algorithm on PARC sentence pairs from the University of Illinois at Urbana-Champaign, which contains 76 pairs selected to show relevant issues important to the textual entailment. We can test our implementation on only 75 pairs, as the semantic generation step fails in one of them. We obtain the following results:

		By Hand		Sum
		True	False	
Application	True	23	10	33
	False	18	24	42
Sum		41	34	75

These tests have shown that what we do works for what we want to do. That is to say, it works for detecting entailments between simple sentences (with verbs, noun, verb modifiers, noun modifiers and negation), with simple lexical knowledge. As we said at the start of the paper, the present system is not intended to handle entailments which need complex knowledge, or entailments which hold due to modality, time expressions, quantification or counting. The incorrect cases in the test set were usually of this kind.

Our approach is limited by the expressivity of our representation, which handles only a tiny fragment of the English language. Due to the expressivity of DLs, some fragments of English will be hard to represent. For instance, modality needs ideas form modal logic to be represented correctly (e.g., *"John is an alleged murderer"* is represented by the formula $john(j) \wedge alleged(murderer(j))$ in neo-Davison's semantic). Nevertheless, by using more expressives description logics, we can handle some other fragments, such as *articulate connective* examples (e.g., *"if Mary comes, then John comes too"*).

Currently we are working on the implementation of a syntactic analyser which translates text into our DL representation by using FrameNet, and testing the use of more expressive logics. For instance, we can use the one-of operator $\mathcal{O}$ [5] to have constraints on labelled nodes in terminological axioms. This could be useful for representing sentences with disjunction

---

[1]At present the system doesn't use FrameNet, and instead we take the verb as concept and basic roles as agent and patient. Becuase of this we will miss converse cases (i.e., *to sell* and *to buy*) in our test.

on individuals like *"John loves Mary or Jane"*. We can also use hybrid logics [10] as $\mathcal{H}(@)$ for having more expressive constraints on labelled nodes, and to represent articulate connectives.

Bibliography

[1] I. Dagan, O. Glickman, and B. Magnini. The PASCAL Recognising Textual Entailment Challenge. In *Machine Learning Challenges Workshop*, pages 177–190, 2005.

[2] V. Haarslev and R. Möller. Description of the RACER system and its applications. In *Proc. Int. Workshop on DLs (DL-2001)*, pages 131–141, 2001.

[3] M. Quillian. Semantic memory. In A. Collins and E. Smith, editors, *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, pages 80–101. Kaufmann, 1988.

[4] M. Minsky. A framework for representing knowledge. Report A. I. MEMO 306, MIT, A.I. Lab., 1974. McGraw-Hill, P. Winston (Ed.), "Psychology of Computer Vision", 1975.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[6] D. Davidson. *Essays on Actions and Events*. Clarendon, Oxford, 1980.

[7] C. Baker, C. Fillmore, and J. Lowe. The Berkeley FrameNet project. In C. Boitet and P. Whitelock, editors, *Proc. of the 36th Annual Meeting of the ACL and 17th Int. Conf. on Computational Linguistics*, pages 86–90. Morgan Kaufmann Publishers, 1998.

[8] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, May 1998.

[9] J. Curran, S. Clark, and J. Bos. Linguistically Motivated Large-Scale NLP with C&C and Boxer. In *Proceedings of the Demonstrations Session of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07), Prague, Czech Republic, 2007.* To appear.

[10] C. Areces and B. ten Cate. Hybrid logics. In P. Blackburn, F. Wolter, and J. van Benthem, editors, *Handbook of Modal Logics*. Elsevier, 2006.

[11] J. Bos and K. Markert. When logical inference helps determining textual entailment (and when it doesn't). In *Pascal, Proceedings of the Second Challenge Workshop, Recognizing Textual Entailment*, 2006.