Towards overcoming the knowledge acquisition bottleneck in Answer Set Prolog applications: embracing natural language inputs

Chitta Baral, Juraj Dzifcak and Luis Tari
School of Computing and Informatics
Arizona State University
Tempe, AZ 85287-8809

Abstract. Answer set Prolog, or AnsProlog in short, is one of the leading knowledge representation (KR) languages with a large body of theoretical and building block results, several implementations and reasoning and declarative problem solving applications. But it shares the problem associated with knowledge acquisition with all other KR languages; most knowledge is entered manually by people and that is a bottleneck. Recent advances in natural language processing have led to some systems that convert natural language sentences to a logical form. Although these systems are in their infancy, they suggest a direction to overcome the above mentioned knowledge acquisition bottleneck. In this paper we discuss some recent work by us on developing applications that process logical forms of natural language text and use the processed result together with AnsProlog rules to do reasoning and problem solving.

1 Introduction

Answer Set Prolog, or AnsProlog in short, is the logic programming language whose semantics is defined using the answer set semantics [GL88,GL91]. A knowledge base in this language consists of rules of the following form:

$$l_0 \leftarrow l_1, \ldots, l_m,\ not\ l_{m+1}, \ldots,\ not\ l_n.$$

where l_i's are literals. Given an AnsProlog knowledge base KB and a query Q, we say KB entails Q, denoted by $KB \models Q$, if Q is true in all answer sets of KB. A related term, Answer Set Programming (or ASP) often focuses only on encoding a given problem as an AnsProlog program so that each of its answer sets encodes a solution of that problem. For knowledge representation and reasoning both query entailment and problem solving by finding answer sets are important.

Over the last 20 years a large body of work has been done on this language. These include theoretical building block results, several implementations and reasoning and declarative problem solving applications. While the book [Bar03] is a compilation of many of these results, there have been a lot of further developments since this book was written. Some highlights include the use of SAT in finding answer sets [LZ02,Lie05], the extension of AnsProlog to allow probabilities [BGR04], the recent system CLASP [GKNS07] and some recent work on applications involving natural language processing and question answering. In this paper we focus on the last aspect.

While we seem to be getting closer to an adequate language for creating knowledge bases, the obstacle of manual knowledge acquisition still haunts the development of a large body of sizeable knowledge bases. (There are other issues that have also hampered such development. They include lack of appropriate tools for facilitating building of knowledge bases such as GUIs, debuggers and the ability to easily reuse previously coded knowledge modules. Some research addressing these issues have been taken up recently [BDT06,dVS07].) Project Halo [FAW$^+$04], an ambitious attempt to encode knowledge of a high school chemistry book chapter realized the bottleneck of knowledge acquisition in its phase one and has focused on it in its second phase.

Automatic knowledge acquisition through learning has focused mostly on learning from observations. However, some recent developments in natural language processing and natural language resource compilations has brought us closer to the possibility of acquiring knowledge from natural language text. Amongst them is the resource Wordnet [MBF$^+$90], which is much more than an on-line dictionary; and systems such as LCC [MHG$^+$02,MPHS03] and C&C [CCt] that can translate natural language text to a logical form. Although these systems are in their infancy, they suggest a direction to overcome the above mentioned knowledge acquisition bottleneck. In this paper we discuss some recent work by us on developing applications that process logical forms of natural language text and use the processed result together with AnsProlog rules to do reasoning and problem solving. We will particularly focus on our attempt to build a system that can solve puzzles described in a natural language. We will also briefly mention our other attempts[1], some reported or to be reported in detail elsewhere, in reasoning about cardinality of dynamic sets, reasoning about travel, question answering and textual entailment; all dealing with natural language input.

2 Processing natural language using C&C tools and Boxer

In this section we discuss in detail how we use the C&C tools and some post-processing to process natural language and obtain (some) knowledge in AnsProlog syntax which together with additional rules allows us to do reasoning and problem solving.

We start with the C&C [CCt,BCS$^+$04] tools that are used to translate natural language text into a logic form.

[1] Many of these works were done in collaboration with Michael Gelfond and Marcello Balduccini of Texas Tech University and Richard Scherl of Monmouth University.

2.1 C&C tools

The C&C tools consists of a set of tools including a robust wide-coverage CCG parser [CC07], several maximum entropy taggers(for example the POS tagger and Named Entity Recognizer) and a semantic analysis tool Boxer [BCS$^+$04]. Most of them can be used individually, but are designed to be used together. The system is very robust and the tools have been used in various scenarios, such as the RTE challenge [RTE], where the problem is to recognize textual entailment; i.e., to decide if a given hypothesis is entailed by a given text or not. Many of the tools use statistics and a set of trained models on which the statistical methods are applied.

When using this system, the extraction of data from the English text is performed as follows. First, the CCG parser [CCt], together with taggers and other tools, is run on the data to obtain the first basic logic form of the data. Then the Boxer [CCt] system is used on the parser output to obtain a first order logic representation of the English text. This is performed sentence by sentence for the whole text. The Boxer output is, as said, in first order logic and thus contains variables as well as predicates and quantifiers. We then translate the Boxer first order logic output to a set of AnsProlog facts.

We now look more closely at each of the steps presented above.

2.2 CCG parser

The first important step is to obtain the CCG parser output. Note that when doing this, many other tools are used in the process such as the tagger. Nevertheless, the parser is the most important part of this step. The CCG parser is a statistical parser with the focus on wide-coverage and high efficiency, producing a single parse of the text. It is based on the combinatorial categorial grammar(CCG), which is based on the the classical categorial grammar [Woo93]. The CCG parser works on a sentence by sentence basis. That means it parses and analyzes a single sentence and then moves to another and starts anew without using any information obtained from the previous sentence. Thus by itself, it is unable to perform and detect things like anaphoras. However, Boxer does employ a form of anaphora resolution.

The lexical entries of CCG include semantic interpretation and syntactic category. The category can not only be a basic category such as noun or verb phrase and gender, but also a complex category that is obtained recursively from the basic categories. For example transitive verbs can be given as $(S\backslash NP)/NP$, where 'S' is a sentence and NP stands for noun phrase. The delimiters '\' and '/' represent the first and the second argument of the verb respectively.

Let us now look at 2 different examples which illustrate the output of the parser.

Example 1. Consider the following sentences:

"The first puzzle category is person."

The output of the CCG parser, together with all the taggers, is as follows:

```
sem(1,
 bapp('S[dcl]',
  fapp('NP[nb]',
   leaf(1,1,'NP[nb]/N'),
   fapp('N',
    leaf(1,2,'N/N'),
    fapp('N',
     leaf(1,3,'N/N'),
     leaf(1,4,'N')))),
  fapp('S[dcl]\NP',
   leaf(1,5,'(S[dcl]\NP)/(S[ng]\NP)'),
   leaf(1,6,'S[ng]\NP')))).

word(1, 1, 'The', 'the', 'DT', 'I-NP', 'O', 'NP[nb]/N').
word(1, 2, 'first', 'first', 'JJ', 'I-NP', 'O', 'N/N').
word(1, 3, 'puzzle', 'puzzle', 'NN', 'I-NP', 'O', 'N/N').
word(1, 4, 'category', 'category', 'NN', 'I-NP', 'O', 'N').
word(1, 5, 'is', 'be', 'VBZ', 'I-VP', 'O', '(S[dcl]\NP)/(S[ng]\NP)').
word(1, 6, 'person.', 'person.', 'VBG', 'I-VP', 'O', 'S[ng]\NP').
```

The output consist of 2 parts, first one is a specific parse tree representing the sentence structure which is used by Boxer for semantic interpretation, with leafs representing the words of the sentence and the non-leaf nodes are the applied combinatorial rules. Next we have one line for each word in the sentence. The structure of the data is as follows. First we have the index of the translated sentence, followed by the index corresponding to the position within the tree, the used word, 'base' form of the word, followed by it's types given by various taggers and finally the category.

Now let us look at a bit more complex sentence.

"Jim Kirby and his partner, asked to name the first man on the moon, nominated Luis Armstrong."

The parser outputs the following:

```
sem(1,
 bapp('S[dcl]',
  bapp('NP[nb]',
   lex('N','NP',
    fapp('N',
     leaf(1,1,'N/N'),
     leaf(1,2,'N'))),
   conj('conj','NP[nb]','NP[nb]\NP[nb]',
    leaf(1,3,'conj'),
    fapp('NP[nb]',
     leaf(1,4,'NP[nb]/N'),
     leaf(1,5,'N')))),
  fapp('S[dcl]\NP',
   leaf(1,6,'(S[dcl]\NP)/(S[to]\NP)'),
   fapp('S[to]\NP',
    leaf(1,7,'(S[to]\NP)/(S[b]\NP)'),
    fapp('S[b]\NP',
     leaf(1,8,'(S[b]\NP)/NP'),
     bapp('NP',
      fapp('NP[nb]',
       leaf(1,9,'NP[nb]/N'),
       fapp('N',
        leaf(1,10,'N/N'),
        leaf(1,11,'N'))),
      fapp('NP\NP',
       leaf(1,12,'(NP\NP)/NP'),
       bapp('NP',
        fapp('NP[nb]',
         leaf(1,13,'NP[nb]/N'),
         leaf(1,14,'N')),
        lex('S[dcl]\NP','NP\NP',
         fapp('S[dcl]\NP',
          leaf(1,15,'(S[dcl]\NP)/NP'),
          lex('N','NP',
           fapp('N',
            leaf(1,16,'N/N'),
            leaf(1,17,'N')))))))))))))).


word(1, 1, 'Jim', 'Jim', 'NNP', 'I-NP', 'I-PERSON', 'N/N').
word(1, 2, 'Kirby', 'Kirby', 'NNP', 'I-NP', 'I-PERSON', 'N').
word(1, 3, 'and', 'and', 'CC', 'O', 'O', 'conj').
word(1, 4, 'his', 'his', 'PRP$', 'I-NP', 'O', 'NP[nb]/N').
word(1, 5, 'partner,', 'partner,', 'NN', 'I-NP', 'O', 'N').
word(1, 6, 'asked', 'ask', 'VBD', 'I-VP', 'O', '(S[dcl]\NP)/(S[to]\NP)').
word(1, 7, 'to', 'to', 'TO', 'B-VP', 'O', '(S[to]\NP)/(S[b]\NP)').
word(1, 8, 'name', 'name', 'VB', 'I-VP', 'O', '(S[b]\NP)/NP').
word(1, 9, 'the', 'the', 'DT', 'I-NP', 'O', 'NP[nb]/N').
word(1, 10, 'first', 'first', 'JJ', 'I-NP', 'O', 'N/N').
word(1, 11, 'man', 'man', 'NN', 'I-NP', 'O', 'N').
word(1, 12, 'on', 'on', 'IN', 'I-PP', 'O', '(NP\NP)/NP').
word(1, 13, 'the', 'the', 'DT', 'I-NP', 'O', 'NP[nb]/N').
word(1, 14, 'moon,', 'moon,', 'NN', 'I-NP', 'O', 'N').
word(1, 15, 'nominated', 'nominate', 'VBD', 'I-VP', 'O', '(S[dcl]\NP)/NP').
word(1, 16, 'Luis', 'Luis', 'NNP', 'I-NP', 'I-PERSON', 'N/N').
word(1, 17, 'Armstrong.', 'Armstrong.', 'NNP', 'I-NP', 'O', 'N').
```

In this case, we obtain a similar output, although more complex. The output is formatted and contains the same information as in the case of the simple sentence. Let us look at the structure of this sentence. Note that the tree is divided into several big parts properly representing the sentence. Note that the first part represents noun phrase 'Jim Kirby and his partner' and the verb phrase 'asked to name the first man on the moon, nominated Luis Armstrong.'. Each of them is then further divided into specific parts. For example the noun phrase is divided again into 'Jim Kirby', 'and' and 'his partner'.Thus one can easily observe the structure of the sentence just by looking at the given tree. Another point to note is that the some categories contain additional information, which is given in the square brackets. These are called features and provide additional information about the category. For example 'S[dcl]' means that the sentence is declarative.

The output is then ready to be used by the next step, Boxer.

2.3 Boxer

The next step is to use the Boxer tool on the CCG parser output to obtain a logic form from it. As said before, Boxer performs semantic analysis of the given text. The basic output of Boxer is the Discourse Representation Structures (DRSs) and the box representations of Discourse Representation Theory (DRT). However, in addition, one of the Boxer outputs can be a first order logic form representing the semantics of the sentence. This is very useful as it can be then be transformed into AnsProlog programs. Boxer does take care of some natural language issues such as anaphora resolution, which is not covered by the parser. Let us now take a closer look at the given example.

Example 2. Recall the example sentences presented before:

"The first puzzle category is person."

"Jim Kirby and his partner, asked to name the first man on the moon, nominated Luis Armstrong."

The Boxer first order logic output(after running it on the output given by the CCG parser) of the sentences is:

```
some(_G5981, some(_G6002, and(and(n_puzzle_1(_G5981),
and(a_first_1(_G6002), and(n_category_1(_G6002),
r_nn_1(_G5981, _G6002)))), some(_G6124,
and(v_person_1(_G6124), and(n_event_1(_G6124),
r_agent_1(_G6124, _G6002)))))))
```

and

```
some(_G3117, some(_G3287, some(_G3314, some(_G3252,
some(_G3345, some(_G3469, some(_G3499, some(_G3688,
some(_G3730, some(_G3647, some(_G3803, and(and(p_kirby_1(_G3117),
and(p_jim_1(_G3117), and(n_moon_1(_G3287),
and(p_armstrong_1(_G3314), and(p_luis_1(_G3314),
and(a_first_1(_G3252), and(n_man_1(_G3252),
and(v_nominate_1(_G3345), and(n_event_1(_G3345),
and(r_agent_1(_G3345, _G3287), and(r_patient_1(_G3345, _G3314),
and(r_on_1(_G3252, _G3287), and(a_male_1(_G3469),
and(n_partner_1(_G3499), and(r_of_1(_G3499, _G3469),
and(n_moon_1(_G3688), and(p_armstrong_1(_G3730),
and(p_luis_1(_G3730), and(a_first_1(_G3647), and(n_man_1(_G3647),
and(v_nominate_1(_G3803), and(n_event_1(_G3803),
and(r_agent_1(_G3803, _G3688), and(r_patient_1(_G3803, _G3730),
r_on_1(_G3647, _G3688)))))))))))))))))))))))), some(_G3154,
some(_G3166, some(_G3536, some(_G3548, and(v_ask_1(_G3154),
and(n_proposition_1(_G3166), and(n_event_1(_G3154),
and(r_agent_1(_G3154, _G3117), and(r_theme_1(_G3154, _G3166),
and(some(_G3412, and(v_name_1(_G3412), and(n_event_1(_G3412),
and(r_agent_1(_G3412, _G3117), r_patient_1(_G3412, _G3252))))),
and(v_ask_1(_G3536), and(n_proposition_1(_G3548), and(n_event_1(_G3536),
and(r_agent_1(_G3536, _G3499), and(r_theme_1(_G3536, _G3548),
some(_G3960, and(v_name_1(_G3960), and(n_event_1(_G3960),
and(r_agent_1(_G3960, _G3499), r_patient_1(_G3960, _G3647)
))))))))))))))))))))))))))))))
```

respectively. This can be better visualized as

```
exists X Y(
          n_puzzle_1(X) &
          a_first_1(Y) &
          n_category_1(Y) &
          r_nn_1(X, Y) &
          exists Z(
                 v_person_1(Z) &
                 n_event_1(Z) &
                 r_agent_1(Z, Y)
          )
)
```

and

```
exists X Y Z A B C D E F G H(
       p_kirby_1(X) & p_jim_1(X) &
       n_moon_1(Y) & p_armstrong_1(Z) &
       p_luis_1(Z) & a_first_1(A) &
       n_man_1(A) & v_nominate_1(B) &
       n_event_1(B) &
       r_agent_1(B, Y) &
       r_patient_1(B, Z) &
       r_on_1(A, Y) &
       a_male_1(C) & n_partner_1(D) &
```

```
        r_of_1(D, C) &
        n_moon_1(E) & p_armstrong_1(F) &
        p_luis_1(F) & a_first_1(G) &
        n_man_1(G) & v_nominate_1(H) &
        n_event_1(H)&
        r_agent_1(H, E) &
        r_patient_1(H, F) &
        r_on_1(G, E) &
        exists I J K L(
               v_ask_1(I) & n_proposition_1(J) &
               n_event_1(I) &
               r_agent_1(I, X) &
               r_theme_1(I, J) &
               exists M(
                      v_name_1(M) &
                      n_event_1(M) &
                      r_agent_1(M, X) &
                      r_patient_1(M, A)
               ) &
               v_ask_1(K) & n_proposition_1(_G3548)
               n_event_1(K) &
               r_agent_1(K, D) &
               r_theme_1(K, L) &
               exists N(
                       v_name_1(N) &
                       n_event_1(N) &
                       r_agent_1(N, D) &
                       r_patient_1(N, G)
               )
        )
)
```

respectively.

Let us now look more closely at the given output. Let us first check the format. The predicates are of the form *x_name_number*(*var*), where 'x' is the specifier, 'name' is the actual word, 'number' is the wordnet meaning and 'var' is the variable assigned to it. The specifier gives the basic category of the word. Some of them are as follows:

– 'n' : noun
– 'p' : person
– 'o' : organization
– 'l' : location
– 'a' : adjective
– 'v' : verb
– 'r' : relation

When looking at the output, one can easily observe that the relations specify connections between the nouns. Boxer detects many types of relations including but not limited to: agent, patient, in, at, as, of and with.

Also note that the links between the words are given using both relations and variables, as the same variable can be used in place of multiple words (for example, ‘X’ is used for both ‘Jim’ and ‘Kirby’ thus connecting them), indicating they are the same with respect to semantics. This information can then be used to effectively extract information from the sentence.

Another important thing to notice when looking at this example is that it seems like some parts of it are repeating itself, only with different variables. An example of it is the second existential quantifier, ‘exists I J K L’, where ‘I’ and ‘J’ are used in the exact same way and connect the exact same words as ‘K’ and ‘L’. Also note that this ‘doubling’ was not in the parser output. The problem here seems to be the current version of Boxer. As we will discuss later, the translation is not perfect and particular words such as ‘his’ (which is present in the sentence) cause trouble. However, one must add that the overall quality of the obtained logic form is still good.

2.4 Obtaining the answer set programs

Once we obtain the output form from Boxer, the next step is to translate the output given as a first order logic form into a set of AnsProlog facts. This is done in multiple steps, the main parts of the transformation are as follows.

Translating Boxer first order logic output to AnsProlog facts:

- Remove all quantifiers from the output
- Modify each predicate of the form $x\_name\_number(var)$, where ‘x’ is the specifier(‘n’ for noun, ‘v’ for verb, etc.), ‘name’ is the actual word, ‘number’ is the wordnet meaning and ‘var’ is the variable assigned to the word. Change any such predicate into ‘x(name, number, var).’. Note that this change is required to be able to reason about specific words and meanings in AnsProlog.
- In case of negation, replace the statement of the form $not(p_1, p_2, ..., p_n)$, where p_i is a predicate for $i = 1, 2, ..., n$, with $-p_1., -p_2., ..., -p_n.$. This is done to preserve negation and to be able to use it in reasoning.

Example 3. Let us recall the 2 examples.

- The first puzzle category is person
- Jim Kirby and his partner, asked to name the first man on the moon, nominated Luis Armstrong.

In this case, the given Boxer output will be translated to:

```
n(puzzle,1,g5981). a(first,1,g6002).
n(category,1,g6002).
r(nn,1,g5981, g6002).
```

```
v(person,1,g6124). n(event,1,g6124).
r(agent,1,g6124, g6002).

object(g5981).
object(g6002).
object(g6124).
```

and respectively into:

```
p(kirby,1,g3117). p(jim,1,g3117).
n(moon,1,g3287).
p(armstrong,1,g3314). p(luis,1,g3314).
a(first,1,g3252). n(man,1,g3252).
v(nominate,1,g3345). n(event,1,g3345).
r(agent,1,g3345, g3287).
r(patient,1,g3345, g3314).
r(on,1,g3252, g3287).
a(male,1,g3469). n(partner,1,g3499).
r(of,1,g3499, g3469).
n(moon,1,g3688). p(armstrong,1,g3730).
p(luis,1,g3730). a(first,1,g3647).
n(man,1,g3647).

object(g3117).
object(g3287).
object(g3314).
object(g3252).
object(g3345).
object(g3287).
object(g3469).
object(g3499).
object(g3688).
object(g3730).
object(g3647).
```

Notice that by performing this transformation, we keep all the information contained in the predicates and variables, but lose other information contained in the first order logic form given by the CCG parser as well as Boxer. Although this does not cause a problem with the applications that we are working on, in future one may also need to use the other knowledge contained in the logic form.

The translation also includes some more specific parts with regards to answer set solvers, such as Smodels [NS97] reserved words translation (for example Boxer output 'eq' is translated into 'equal', since 'eq' is a reserved word in Smodels), changing the case of some letters and adding domain predicates for each variable.

Once we obtain the AnsProlog facts from the text, we can combine them with the rest of the knowledge base.

3 An application: Solving puzzles

We now present an application of the given method. We will use the specified method to solve a set of puzzles obtained from [blp07]. These puzzles each happen to have 4 domains, each containing 5 elements. To solve any of these puzzles, one needs to form tuples of 4 elements, where for each tuple it contains 1 element from each domain, while all the tuples are mutually exclusive(e.g. any element can only be in a single tuple). In addition, the conditions regarding the tuples are given in natural language and a solution must satisfy all of these conditions.

Let us now take a look at a particular puzzle, puzzle number 37 from [blp07].

Example 4 (Quiz game). The domain of the puzzle contains information about a person, his partner, the place they are from and question they have been asked.

- The persons are: Dick Eames, Helen Ingle, Jim Kirby, Joanna Long and Paul Riggs.
- The partners are: Ben Cope, Laura Mayes, Nick O'Hara, Ray Shaw and Tina Urquart
- The towns are: Blunderbury, Errordon, Messford, Slipwood and Wrongham.
- The questions are to name: Babylon wonder, First man on the moon, Mona Lisa artist, Six-day war length and Trafalgar commander.

There is a total of 7 clues given as follows:

1. The two ladies from Blunderbury were asked which of the Seven Wonders of the world was in Babylon and got it half right when they replied the hanging baskets.
2. Helen Ingle and her team-mate were from Errordon.
3. Jim Kirby and his partner, asked to name the first man on the Moon, nominated Louis Armstrong.
4. Laura Mayes, who was half of the team from Slipwood, and Tina Urquart were partnered with persons sharing a first name initial.
5. Ray Shaw and his partner both knew really that the British commander at Trafalgar wasn't Nelson Eddy but their minds went blank just at the wrong moment.
6. Ben Cope and Dick Eames, who isn't from Wrongham, were not partners in the game.
7. Paul Riggs and his partner Nick O'Hara were not the pair who, asked who painted Mona Lisa, replied Leonardo Da... um...Caprio.

Naturally, the goal of the puzzle is to figure out each pair of people, where they are from and what question they had been asked. In this case, the solution is:

(Joanna Long, Tina Urquart, Blunderbury, Babylon wonder)
(Paul Riggs, Nick O'Hara, Wrongham, Six-day war length)
(Jim Kirby, Laura Mayes, Slipwood, First man on the moon)
(Helen Ingle, Ben Cope, Errordon, Mona Lisa artist)
(Dick Eames, Ray Shaw, Messford, Trafalgar commander)

From this one can easily observe that to solve the puzzles in general, one can do the following. Extract the facts and clues from the text, enumerate all the possibilities for the extracted data and use clues to derive rules to impose restrictions on the possible assignments. Thus in this case, one needs to extract both the puzzles facts and their respective clues into answer set programming facts and rules. Let us first look at the fact extraction.

However, looking back at the example one also notices that he needs to know which names corresponds to men and woman and which have the same initial. This additional knowledge is assumed to be included in the knowledge base.

3.1 Facts extraction

The facts are extracted directly from specific input sentences. As said before, all the puzzles happen to have 4 domains, each of which contains 5 elements. The problem is that in many cases, some domain elements are not mentioned in the clues at all. Furthermore, many domains include elements similar to the other domain, for example first names, making it impossible to have a completely automatic extraction. Thus, we use the following type of English sentences to define the domains and their elements. (This is similar to the specification of domains in the Constraint Lingo system [FMT04].) For the given puzzle we have (assuming we take into account only first names and represent the questions by single word, we can as they are all different) the following:

```
The first puzzle category is person.
The second puzzle category is partner.
The third puzzle category is town.
The fourth puzzle category is question.
The persons are Dick, Helen,  Jim, Paul and Joanna.
The partners are Ben, Laura, Nick, Ray and Tina.
The towns are Blunderbury, Errordon, Messford, Slipwood and Wrongham.
The questions are wonder, moon, mona, war and Trafalgar.
```

The sentence always start with 'The', followed by one of the following: 'first', 'second', 'third', 'fourth'. This specifies which of the 4 domains the element it is in, for example 'first' means 1st domain, 'second' means 2nd etc. We use this to properly connect each puzzle category to one and only one domain. Next we have the specifier 'puzzle category is' followed by the respective category. In the first case, it is 'person' but it can be any word. Then we have a next sentence of the format 'The *category* are', where *category* can be any of the above categories followed by the elements of that particular category. Note that in the last case, we simplified the questions to one word. This was done in order to improve the performance of the system, in particular the clue extraction part. However, one might use the whole questions if necessary. These facts are automatically extracted and there is a set of specific rules that perform this extraction. Next a set of rules is used to enumerate all possible tuples of 4 elements. These rules are all contained in the knowledge base about puzzles. In this particular case, we use 2 predicates h and at, where the first specifies the first triple of a tuple, while the second predicate includes the first and last element of the tuple. For example $h(joanna, tina, blunderbury), at(joanna, wonder)$ represents a tuple $(joanna, tina, blunderbury, wonder)$. This tuple represents the knowledge that Joanna and Tina were the pair from Blunderbury and were asked the question to name the Babylon wonder. The reason for splitting the tuples of 4 elements into smaller ones is to improve performance and also to allow us to have a simpler set of clues extraction rules.

3.2 Clues Extraction

The more interesting part of the natural language processing of the puzzle sentences is the clues extraction. Our initial approach is as follows. We have a set of templates of rules which specify the conditions of the puzzle; i.e., they specify which tuple assignments are correct. We then have an extraction system that 'supplies' these rules. We will illustrate the way it works on an example. Consider the following rule template [2]:

```
sat(O1) :- h(X,Y,Z), at(X,T), c1x(X,O1), c1y(Y,O1), c1z(Z,O1),
           c1t(T), cond(O1).
```

The head of the rule is the satisfaction predicate; a rule later on will then require this to be true thus making sure the condition holds for all answer sets. The 'h' and 'at' predicates are used to connect the domains of the puzzles. The template then allows us to specify 4 variables, e.g. 4 domains, first one, second one, third one and fourth one using predicates 'c1x', 'c1y', 'c1z' and 'c1t' respectively. Then the extraction system 'feeds' these predicates based on clues, making this a real rule. The last predicate says it's a positive rule and also specifies the rule identifier 'O1' used later to ensure all the rules created from templates are connected to the sentences they were extracted from, all of the extracted rules are satisfied and that we get only correct answer sets.

For example if we have the clue of the form 'Jim Kirby and his partner, asked to name the first man on the moon, nominated Luis Armstrong.', we can obtain(assuming we have the proper domains) that the first domain element is Jim and that the fourth is moon(again assuming simplified representations for questions). Then the extraction system would extract c1x(jim), c1y(Y), assuming Y is a variable representing the second domain, c1z(Z), again assuming Z represents the third domains, and c1t(moon). Then the above template rule would be equivalent to:

```
sat(O1) :- h(jim,Y,Z), at(jim,moon), cond(O1).
```

[2] An alternative template where h is a binary predicate would work fine too. Also, our use of four domains each with five elements can be easily generalized where the numbers are not predetermined but are part of the puzzle.

Note that this is done for all the clues. Clearly, the effectiveness of this system depends on the accurate extraction from the logic forms. Also, note that during extraction, we always pick a unique identifier, given by the $O1$ variable. This then ensures that the clue is only used for the cases we extracted.

The main idea when extracting clues is as follows. We try to connect each clue to a sentence of a particular type. In general, one might assume a sentence has a subject, an object and a verb. Now depending on the verb, and the connection between subject and object, we will select which clue will be used for the sentence. To do this, we look for relation(s) which connects the subject and the object, and if they are present in the facts representing the sentence. For example for the sentence 'Jim Kirby and his partner, asked to name the first man on the moon, nominated Luis Armstrong.' and the given clue template, we use the following set of extraction rules:

```
extract_template_1(X,Y,Z,T,O1) :- n(man,1,O6), r(on,1,O6,O7),
  l(T,1,O7), p(X,1,O4), r(agent,1,O5,O4), v(nominate,1,O5),
  v(ask,1,O1),v(name,1,O2), r(agent,1,O1,O3), r(agent,1,O2,O3).
c1x(X, O1) :- extract_template_1(X,Y,Z,T,O1).
c1y(Y, O1) :- extract_template_1(X,Y,Z,T,O1).
c1z(Z, O1) :- extract_template_1(X,Y,Z,T,O1).
c1t(T, O1) :- extract_template_1(X,Y,Z,T,O1).
cond(O1) :- extract_template_1(X,Y,Z,T,O1).
```

Here assume X, Y, Z, T are variables representing the four puzzle domains respectively and Oi, where i is a positive integer, represents any variable given by boxer. In this case, we use 'O1' as the identifier of the sentence. Note that it does not matter which one of the sentence variables we use, since boxer uses exclusive variables for each sentence. This kind of extraction is performed for all the clues.

To ensure this condition is satisfied later, we have the following rule:

```
:- not sat(O1), cond(O1).
```

This rule guarantees all of the extracted conditions are satisfied.

So far we have discussed the extraction of facts and clues. Now let us look at the enumeration part. This is done using the following simple set of rules:

```
nh(X,Y,Z) :-  h(X,Y1,Z1), Z != Z1.
nh(X,Y,Z) :-  h(X,Y1,Z1), Y != Y1.
nh(X,Y,Z) :-  h(X1,Y1,Z), X != X1.
nh(X,Y,Z) :-  h(X1,Y1,Z), Y != Y1.
nh(X,Y,Z) :-  h(X1,Y,Z1), Z != Z1.
nh(X,Y,Z) :-  h(X1,Y,Z1), X != X1.

h(X,Y,Z) :- d1(X), d2(Y),d3(Z), not nh(X,Y,Z).

n_at(X,T) :- at(X1,T), X != X1.
n_at(X,T) :- at(X,T1), T != T1.
at(X,T) :- d1(X), d4(T), not n_at(X,T).
```

These rules enumerate all the possible combinations. The predicates $d1, d2, d3$ and $d4$ represent the four respective domains of puzzle elements. The 'h' predicate describes the first three domains; the 'at' predicate domains 1 and 4; and domain 1 elements are used as the identifiers.

Let us now provide an idea on how to use the system. With both facts extraction and puzzle specification already present, the system works in a simple way. The system takes English sentences for facts and the English sentences representing the clues of a particular puzzle. It transforms the puzzle domain information and clues into first order logic using CCG parser and Boxer, then uses a script to make them answer set programming facts. Then the general knowledge and extraction rules are added and the answer sets are computed. The resulting answer sets represent the solutions to the puzzles.

Thus, combining the extracted facts, the interface which 'feeds' the rule templates, and the general module about puzzles that includes enumeration, rule templates and some additional information (for example for the given puzzle we need information about names first initial and men/woman names) we obtain an AnsProlog program with the following answer set:

```
h(joanna,tina,blunderbury)
h(paul,nick,wrongham)
h(jim,laura,slipwood)
h(helen,ben,errordon)
h(dick,ray,messford)

at(paul,war)
at(joanna,wonder)
at(jim,moon)
at(helen,mona)
at(dick,trafalgar)
```

One can easily verify that it corresponds to the solution of the puzzle.

Let us now present some of the current limitations of the system. First, since the system uses a set of external tools, it automatically inherits the issues that come with them. At the moment, the most problematic part seems to be current version of the C&C parser and Boxer. In case of the parser, it has problems recognizing most types of quotes. Also, many words not found in its dictionary are not dealt with properly. In case of Boxer, since it uses the output of the parser, the same problems persist. Also, in case of some sentences, particularly longer ones, they seem to be translated incorrectly by Boxer despite the fact that the output of the parser seems correct. We hope at least some of these issues will be fixed by the next release of C&C tools.

4 Other applications

In this section we briefly discuss two other applications involving Answer Set Prolog and natural language.

4.1 Reasoning about travel

In this subsection, we use examples where there are AnsProlog rules encoding background knowledge about travel and certain narrative about a person's travel given in English and a question asked in English. Consider the following story about John's travel.

1. John took a flight from Paris to Baghdad in the morning of June 15.
2. John would take his laptop on the plane with him.
3. A few hours later the plane stopped in Rome, where John was arrested.
4. The police confiscate John's possession.

By using a natural language understanding system for the above story, we extracted the following facts.

– Facts extracted from sentence 1:

```
h(timepoint(morning),1).  object(flight).
h(origin(flight,paris),1).
city(paris). o(take(john,flight),1).  person(john).
h(dest(flight,baghdad),1). city(baghdad).
h(date(june,15),1).
day(15).  month(june).
```

– Facts extracted from sentence 2:

```
o(take(john,laptop),2). person(john). object(laptop).
h(is_in(john,plane),2). vehicle(plane).
h(owned_by(laptop,john),2).
```

– Facts extracted from sentence 3:

```
o(stop_in(plane,rome),3). o(arrest_in(john,rome),4).
h(arrested_in(john,rome),5). h(timepoint(afternoon),3).
vehicle(plane). person(john). city(rome).
```

– Facts extracted from sentence 4:

```
o(confiscate_pos_of(police,john),5). person(john).
person(police).
```

The predicates h and o indicate *holds* and *occurs* for the above facts. For instance, $o(take(john, flight), 1)$ means that the action about John taking a flight occurs at timepoint 1.

With the above story about John, let's suppose we are interested in asking the following questions: (1) Where is John on June 15 evening? (2) Who has the laptop? The questions are automatically translated into the following AnsProlog rules:

– AnsProlog rule for question 1:

```
answer_Q1(C) :- city(C), h(is_in(P,C),T), h(date(june,15),T),
                h(timepoint(evening),T), P=john.
```

– AnsProlog rule for question 2:

```
answer_Q2(P) :- person(P), h(owned_by(laptop,P),n).
```

In the AnsProlog rule for question 2, n is a constant that is assumed to be the last time point of the travel story. To answer the first question, we need to be able to reason that normally a person P is in city $C2$ if the destination of his trip X is $C2$. There can be exceptions to this rule if for some reason, his trip is aborted at a certain timepoint. One of the possible reasons for P's trip to be aborted is that P is arrested in another city $C1$.

```
h(is_in(P,C2),n) :- person(P), city(C1;C2), object(X),
                      h(origin(X,C1),T), h(dest(X,C2),T),
                      not h(aborted(P,X),T1), T < T1.
h(aborted(P,X),T+1) :- person(P), city(C), object(X),
                         o(arrest_in(P,C),T),
                         h(dest(X,C1),T1), C != C1.
```

By using the above rules, the fact $h(is\_in(john, baghdad), n)$ cannot be inferred, as $h(aborted(john, trip), 5)$ becomes true due to the extracted fact $o(arrest\_in(john, rome), 4)$. With the fact $h(arrested\_in(john, rome), 5)$, we can infer $h(is\_in(john, rome, 5)$ using the following rule:

```
h(is_in(P,C),T) :- person(P), city(C), h(arrested_in(P,C),T).
```

Using the inertia axiom, the facts $h(is\_in(john, rome), n)$ and $h(date(june, 15), n)$ are inferred. We assume that the last timepoint n corresponds to evening when the previous to last timepoint $n - 1$ is afternoon and no timepoint is assigned for evening by the following rule:

```
h(timepoint(evening),n) :- h(timepoint(afternoon),n-1),
                             not h(timepoint(evening),T).
```

With the facts $h(is\_in(john, rome), n)$, $h(date(june, 15), n)$ and $h(timepoint(evening), n)$, we are able to infer the final answer $answer\_Q1(rome)$ for the first question.

To answer the second question, the system needs to reason with the fact that an ownership of an object OBJ is transferred from person $P1$ to P if P confiscates $P1$'s possession. This is captured by the following two AnsProlog rules:

```
h(owned_by(OBJ,P),T+1) :- person(P;P1), object(OBJ),
       o(confiscate_pos_of(P,P1),T), h(owned_by(OBJ,P1),T).
-h(owned_by(OBJ,P1),T+1) :- person(P;P1), object(OBJ),
       o(confiscate_pos_of(P,P1),T), h(owned_by(OBJ,P1),T).
```

With the above rules, the facts $h(owned\_by(laptop, police), 6)$ and $-h(owned\_by(laptop, john), 6)$ are inferred so that we can infer the answer $answer\_Q2(police)$ for the second question.

4.2 RTE examples

RTE is an acronym for "Recognizing Textual Entailment." In an RTE task, given a hypothesis-text pair, *h*-*t*, the goal is to be able to answer "yes" if the system verifies that h can be entailed from t. The module returns "no" if it realizes that h cannot be entailed from t, while it returns "unknown" if a conclusion cannot be reached. Natural language processing is used to extract AnsProlog facts $AF(h)$ and $AF(t)$ corresponding both h and t. Query rules $R(h)$ are generated from $AF(h)$ using a rule generator, and rules $Th(h,t)$ relevant to h and t are used for reasoning. To determine if h entails t, we use the answer set solver Smodels [NS97] with respect to $Th(h,t)$, $R(h)$ and $AF(t)$. We now illustrate this with respect to two *h*-*t* pairs from RTE [DGM06,BHDD$^+$06].

Example 5 (Yoko Ono Example).

- T: Yoko Ono unveiled a bronze statue of her late husband, John Lennon, to complete the official renaming of England's Liverpool Airport as Liverpool John Lennon Airport.
- H: Yoko Ono is John Lennon's widow.

After using NLP, the following facts $AF(h)$, $AF(t)$ are extracted from both h, t:
$AF(h) = \{h(widow\_of(x3, x6), 1)$, $name(x6, john)$, $name(x3, yoko)$, $name(x6, lennon)$, $name(x3, ono)\}$
$AF(t) = \{h(husband\_of(x1, x), 0)$, $o(pass\_away(x1), 0))$, $name(x1, john)$, $name(x1, lennon)$, $name(x, yoko)$, $name(x, ono)\}$.
The rule generator of the RTE module takes $AF(h)$ and translates it into the following AnsProlog rules $R(h)$:

```
answer(yes) :- h(widow_of(X3,X6),1),name(X6,john),
               name(X3,yoko),name(X6,lennon),name(X3,ono).
answer(no) :- -h(widow_of(X3,X6),1),name(X6,john),
               name(X3,yoko),name(X6,lennon),name(X3,ono).
answer(unknown) :- not answer(yes), not answer(no).
```

Using the following rule, $Th(h,t)$, that encodes that X is a widow if her husband $X1$ passed away together with the facts $AF(t)$ and query rules $R(h)$, the system returns the fact $answer(yes)$ to indicate that h entails t.

```
h(widow_of(X,X1),T+1) :- o(pass_away(X1),T),
                          h(husband_of(X1,X),T).
```

Now let us consider another RTE example.

Example 6 (Linnaean Society).

– T: Robinson became a fellow of the Linnaean Society at the age of 29 and a successful gardening correspondent of The Times.
– H: Robinson was a member of the Linnaean Society.

After natural language processing, the following facts $AF(h)$, $AF(t)$ are extracted from both h, t:
$AF(h) = \{\ name(x1, robinson),\ name(x5, linnaean),\ name(x5, society),$
$h(is(x1, member(x5)), 1)\ \}$
$AF(t) = \{\ name(x1, fellow),\ name(x1, robinson),\ name(x5, linnaean),$
$name(x5, society),\ o(become(x1, fellow(x5)), 0)\ \}$

The rule generator of the RTE module takes $AF(h)$ and translates it into the following AnsProlog rules $R(h)$:

```
answer(yes) :- h(is(X1,member(X5)),1),name(X1,robinson),
                name(X5,linnaean),name(X5,society).
answer(no) :- -h(is(X1,member(X5)),1),name(X1,robinson),
                 name(X5,linnaean),name(X5,society).
answer(unknown) :- not answer(yes), not answer(no).
```

The following static and dynamic causal rules, $Th(h, t)$, are used to describe person $X1$ is a member/fellow of society $X2$ if $X1$ becomes a member/fellow of $X2$, while $X1$ is a member of $X2$ if $X1$ is a fellow of $X2$.

```
h(is(X1,member(X2)),T+1) :- o(become(X1,member(X2)),T).

h(is(X1,fellow(X2)),T+1) :- o(become(X1,fellow(X2)),T).

h(is(X1,member(X2)),T) :- h(is(X1,fellow(X2)),T).

-h(is(X1,fellow(X2)),T) :- -h(is(X1,member(X2)),T).
```

With the rules $Th(h, t)$, the facts $AF(t)$ and the query rules $R(h)$, the system then returns the fact $answer(yes)$ to indicate that h indeed entails t.

5 Conclusions

In this paper we discussed initial steps taken towards automatically extracting knowledge from natural language text and using them together with other hand coded knowledge to do reasoning and problem solving. Research in this direction has the potential to alleviate the knowledge acquisition bottleneck that one faces when building knowledge bases.

We briefly discussed three examples: solving puzzles, answering questions about travel and recognizing textual entailment. We reiterate that these are only initial steps and a lot remains to be done. For example, although our formulation could solve several puzzles, it needs to be further generalized to further broaden the class of puzzles it can solve. In many puzzles there are relationship between the elements of the domain and the clues use those relations. Some of these issues are also mentioned in [FMT04]. Thus the formulation in this paper needs to be further improved to solve a broader class of puzzles.

In these initial works we have focused on obtaining facts from the natural langauge processing part. However, the NLP systems that we use produce logic forms and a next step would be to extract more general knowledge (such as simple rules), in the AnsProlog syntax, from the logic forms and use them.

An interesting dimension of working on natural language based applications is that we have come across interesting knowledge representation and reasoning issues such as reasoning about intentions [BG05] and reasoning about the cardinality of dynamic sets. An example of the later is to find out the cardinality of a particular class based on English statements about elements joining and leaving that class or a subclass. We were told that among the 28 questions that could not be answered by any of the TREC QA systems in 2006, 6 of them involved such reasoning about cardinality of sets. The lesson we learned from these attempts is that reasoning with respect to natural language text and queries not only involves using natural language processing but also sometimes requires developing interesting knowledge representation rules, such as rules to reason about membership and cardinality of sets in presence of incomplete knowledge.

Acknowledgements

We thank Yulia Lierler, Steve Maiorano, Jean Michel Pomarede, Michael Gelfond, Richard Scherl, Marcello Balduccini and Vladimir Lifschitz for discussions on the topic of this paper. We thank DTO for its support of this research.

References

[Bar03] C. Baral. *Knowledge representation, reasoning and declarative problem solving.* Cambridge University Press, 2003.

[BCS$^+$04] Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. Wide-coverage semantic representations from a ccg parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04), Geneva, Switzerland.*, 2004.

[BDT06] Chitta Baral, Juraj Dzifcak, and Hiro Takahashi. Macros, macro calls and use of ensembles in modular answer set programming. In *In proceedings of 22nd Int'l Conference on Logic Programming*, pages 376–390, 2006.

[BG05] Chitta Baral and Michael Gelfond. Reasoning about intended actions. In *Proceedings of AAAI 05*, pages 689–694, 2005.

[BGR04] C. Baral, M. Gelfond, and N. Rushton. Probabilistic reasoning with answer sets. In *Proceedings of LPNMR7*, pages 21–33, 2004.

[BHDD$^+$06] Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL*

Challenges Workshop on Recognising Textual Entailment. Springer-Verlag, 2006.

[blp07] England's best logic problems. England's best logic problems. Penny Press, Spring 2007.

[CC07] Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics (to appear)*, 2007.

[CCt] http://svn.ask.it.usyd.edu.au/trac/candc/wiki/.

[DGM06] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *MLCW 2005*, pages 177–190. Springer-Verlag, 2006.

[dVS07] Marina de Vos and Torsten Schaub, editors. *Proceedings of LPNMR07 Workshop on Software engineering for answer set programming: SEA'07*. 2007.

[FAW$^+$04] N. Friedland, P. Allen, M. Witbrock, J. Angele, S. Staab, D. Israel, V. Chaudhri, B. Porter, K. Barker, and P. Clark. Towards a quantitative, platformindependent analysis of knowledge systems. In *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 507–515, 2004.

[FMT04] R. Finkel, M. Marek, and M. Truszczyn'ski. Constraint lingo: towards high-level constraint programming. *SoftwarePractice & Experience*, 34(15):1481 – 1504, 2004.

[GKNS07] Martin Gebser, Benjamin Kaufmann, Andr Neumann, and Torsten Schaub. Clasp: A conflict-driven answer set solver. In *LPNMR'07*, pages 260–265, 2007.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080. MIT Press, 1988.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–387, 1991.

[Lie05] Yuliya Lierler. Cmodels - sat-based disjunctive answer set solver. In *LPNMR*, pages 447–451, 2005.

[LZ02] Fangzhen Lin and Yuting Zhao. Assat: Computing answer sets of a logic program by sat solvers. In *Proceedings of AAAI-02*, 2002.

[MBF$^+$90] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography (special issue)*, 3(4):235– 312, 1990.

[MHG$^+$02] D. Moldovan, S. Harabagiu, R. Girju, P. Morarescu, A. Novischi, F. Lacatusu, A. Badulescu, and O. Bolohan. Lcc tools for question answering. In E. Voorhees and L. Buckland, editors, *Proceedings of TREC 2002*, 2002.

[MPHS03] D. Moldovan, M. Pasca, S. Harabagiu, and M. Surdeanu. Performance issues and error analysis in an open-domain question answering system. *ACM Transaction on Information Systems*, 21(2):133–154, 2003.

[NS97] I. Niemalä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *In proceedings of the 4th International Conference of Logic Programming and Nonmonotonic Reasoning*, pages 420–429, 1997.

[RTE] http://www.pascal-network.org/challenges/rte/.

[Woo93] Mary McGee Wood. Categorial grammars. *Routledge.*, 1993.