Solving Puzzles Described in English by Automated Translation to Answer Set Programming and Learning How to Do that Translation

Chitta Baral
School of Computing, Informatics and DSE
Arizona State University
chitta@asu.edu

Juraj Dzifcak
School of Computing, Informatics and DSE
Arizona State University
juraj.dzifcak@asu.edu

Abstract

We present a system capable of automatically solving combinatorial logic puzzles given in (simplified) English. It uses an ontology to represent the puzzles in ASP which is applicable to a large set of logic puzzles. To translate the English descriptions of the puzzles into this ontology, we use a λ-calculus based approach using Probabilistic Combinatorial Categorial Grammars (PCCG) where the meanings of words are associated with parameters to be able to distinguish between multiple meanings of the same word.

Introduction and Motivation

One of the long term goal of our research is to develop systems that can “comprehend” natural language text and be able to answer questions based on that understanding and associated reasoning. The success of the recent Watson (Ferrucci et al. 2010) and Siri (Roush June 2010) systems has brought a lot of attention towards such a research direction. While Watson is very good at understanding the Jeopardy clues and Siri is able to answer many factoid questions well, our focus is on being able to do “deep reasoning” with natural language text. To elucidate our notion of “deeper reasoning with natural language text” we propose and consider the following challenge problem: *To develop a system that can take as input natural language descriptions of combinatorial puzzles*[1] *(Press 2007) and solve them.*

To solve such a puzzle one needs to be able to reason with the multitude of clues in that puzzle; hence, we use the term “deep reasoning”.

It is well known how to *manually* represent the clues of a combinatorial puzzle in a knowledge representation and reasoning language such as Answer Set Programming (ASP) (Baral 2003) so that one can use a corresponding reasoning engine of that language, such as (Gebser et al. 2007) for ASP, to solve the puzzle. Hence, to develop an automated system to solve such puzzles, we need a way to *automatically translate the natural language text of the puzzle to a representation in ASP or a similar language.*

Towards that goal in this paper we pick ASP as our knowledge representation and reasoning language. To automatically translate English description of puzzles to ASP rules, *we build up on* the basic approach of translating English to ASP (Baral, Dzifcak, and Son 2008) that is inspired by Montague’s work (Montague 1974) on using lambda calculus to give semantics to natural language text.

The reason we do not directly use the approach in (Baral, Dzifcak, and Son 2008) is that it requires that the ASP-λ representations of various words be given a-priori. We believe that construction of such expressions by humans is not scalable and is perhaps one of the important reasons Montague’s work (Montague 1974) has not yet led to large scale natural language understanding systems with the exception of work by Bos, such as in (Bos and Markert 2005).

We propose an approach[2] of learning the ASP-λ representations of various words from training examples of puzzles and their ASP translations and then using them in translating sentences to ASP rules. This approach, which we first presented in (Baral et al. 2011), can be described using the overall architecture given in Figure 1.

There are two parts of the system in Figure 1 which depend on each other: (a) the part in the right of Figure 1 builds a lexicon by learning the of ASP-λ representations of words from training examples of sentences and their ASP representations and assigns them a weight so as to deal with multiple ASP-λ representations of the same word and (b) the part in the left uses the lexicon and translates natural language sentences to ASP rules.

The translation (from English to ASP) system, given in the left hand side of Figure 1, uses a Probabilistic Combinatorial Categorial Grammars (PCCG) (Ge and Mooney 2005) and a lexicon consisting of words, their corresponding ASP-λ representations and associated (quantitative) parameters to do the translation. Since a word may have multiple meanings implying that it may have multiple associated ASP-λ-Calculus formulas, the associated parameters help us in using the “right” meaning in that the translation that has the highest associated probability is the one that is picked.

[1]An example is the well-known Zebra puzzle. http://en.wikipedia.org/wiki/Zebra_Puzzle

[2]The systems and approaches in (Zettlemoyer and Collins 2009; Ge and Mooney 2009; Chen and Mooney 2011; Kwiatkowski et al. 2010; 2011) , which have some similarities with our approach, have not been used in the context of ASP and it is not clear if they could be used in the context of ASP.

The learning system takes a given training set of sentences and their corresponding ASP-λ-Calculus rules, and an initial vocabulary (consisting of some words and their meaning) and uses Inverse λ algorithms[3] and Generalization[4] techniques to obtain the meaning of words which are encountered but are not in the initial lexicon. Because of the generalization step and because of the inherent ambiguity of words having multiple meanings, one may end up with a lexicon where words are associated with multiple ASP-λ-Calculus rules. A parameter learning method is used to assign weights to each meaning of a word in such a way that the probability that each sentence in the training set would be translated to the given corresponding ASP-λ-Calculus rule (using the approach in the left hand side of Figure 1) is maximized. The block diagram of this learning system is given in the right hand side of Figure 1.

A key aspect in being able to use the above approach is to have an ontology of the puzzle representation that will allow one to learn more general representation from given puzzle encoding examples, and use it in new puzzle. In this regards, we noticed that although there are many examples of puzzle encodings in ASP, they were not amenable to be useful in a learning based system where the system learns representations of words from given examples of puzzles and their ASP code and then uses that to translate other puzzles. Considering that in our case the translation of English description of the puzzles into ASP is to be done by an automated system and this system learns aspects of the translation by going over a training set, we need to develop an ontology of representing puzzles in ASP that is applicable to most (if not all) combinatorial logic puzzles. This is one of the main contributions of the paper. Some of the other contributions of this paper are: We developed an overall system that *learns* to solve combinatorial puzzles in English under certain assumptions. We developed a system that learns to translate simplified[5] English to ASP. We developed a small initial lexicon of words and their ASP-λ representations that allows us to learn the representations of other words in puzzles and solve new puzzles using them. We identified some of the background knowledge that is needed to be added to the system.

Assumptions and Background Knowledge

With our longer term goal to be able to solve combinatorial logic puzzles specified in English, as mentioned earlier, we made some simplifying assumptions for this current work. Here we assumed that the domains of puzzles are given (and one does not have to extract them from the puzzle description) and focused on accurately translating the clues. We did a human preprocessing[6] of puzzles to eliminate anaphora and features that may lead to a sentence being translated into multiple ASP rules. Besides translating the given English sentences, we added some domain knowledge related to combinatorial logic puzzles. This is in line with the fact that often natural language understanding involves going beyond literal understanding of a given text and taking into context some background knowledge. We adopted a standard "Generate and test" approach in solving the puzzles. The domain of the puzzle is used to generate all the possible solutions of the puzzle, which are then tested against the constraints obtained from the clues by translating them into ASP representation using our system that learns how to do such translations.

[3]They compute an expression F given ASP-λ-calculus expressions G and H such that F@G or = H or G@F = H.

[4]This includes modifying a known ASP-λ-calculus expression for other words of the same category(Baral et al. 2011).

[5]Our simplified English is different from "controlled" English in that it does not have a pre-specified grammar. We only do some preprocessing to eliminate anaphora and some other aspects.

Puzzle representation and Ontology

For our experiments, we focus on logic puzzles from (Press 2007; 2004; Dell 2005). These logic puzzles have a set of basic domain data and a set of clues.

For example, a puzzle may have the domain data: 1,2,3,4 and 5 are ranks; earl, ivana, lucy, philip and tony are names; earth, fire, metal, water and wood are elements; and cow, dragon, horse, ox and rooster are animals.

Such domain data is encoded using the following format, where $etype(A, t)$ stores the element type t under the index A, while $element(A, X)$ is the predicate storing all the elements X of the type $etype(A, type)$ under the index A. An example of an instance of this encoding is given below.

```
index(1..n).      % size of a tuple
eindex(1..m). % number of tuples

% type and lists of elements of that type,
% one element from each index forms a tuple
etype(1, type1).
element(1, el11). element(1, el12). ...
element(1, el1n).     ...
```

For the given puzzle, the encoding would look like:

```
index(1..4).
eindex(1..5).

etype(1, name).
element(1,earl). element(1,ivana).
element(1,lucy). element(1,philip).
element(1,tony).
etype(2, element).
element(2,earth). element(2,fire).
element(2,metal). element(2,water).
element(2,wood). ...
```

In addition, to avoid permutations among the tuples, the following facts are generated, where $tuple(I, X)$ is the predicate storing the elements X within a tuple I:

```
tuple(1,e11). ... tuple(1,e1n).
```

Using the above ontology clues are expressed as follows:

[6]The people doing the pre-processing were not told of any specific subset of English or any "Controlled" English to use. They were only asked to simplify the sentences so that each sentence would translate to a single clue.

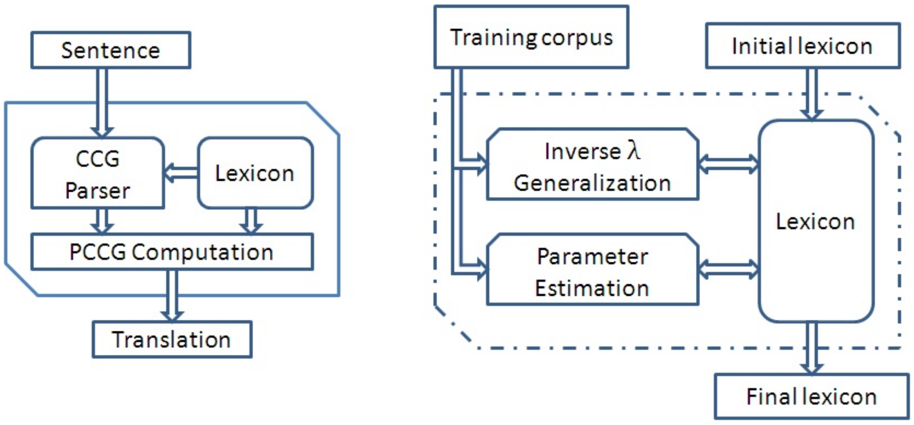

Figure 1: Overall system architecture

```
%Tony was the third person to have
%his fortune told.
:- tuple(I, tony), not tuple(I, 3).

%The person with the Lucky Element
%Wood had their fortune told fifth.
:- tuple(I, wood), not tuple(I, 5).

%Earl's lucky element is Fire.
:- tuple(I, earl), not  tuple(I, fire).

%Earl arrived immediately before
%the person with the Rooster.
:- tuple(I, earl), tuple(J, rooster),
   tuple(I, X), tuple(J, Y),
   etype(A, rank), element(A, X),
   element(A, Y), X != Y-1.
```

Generic modules and background knowledge

Given the puzzle domain data, we combine their encodings with additional modules responsible for generation and background knowledge. The following rules are responsible for generation of all the possible tuples, where we assume that all the elements are exclusive.

```
1{tuple(I,X):element(A,X)}1.
:- tuple(I,X), tuple(J,X),
   element(K,X), I != J.
```

In addition, a module with rules defining generic/background knowledge is used so as to provide higher level knowledge which the clues define. For example, a clue might discuss maximum, minimum, or genders such as woman. Thus rules defining concepts and knowledge such as maximum (max), minimum, within range, sister is a woman and others are added. For example, the concept "max" is encoded as:

```
notmax(A, X) :- element(A, X),
       element(A, Y), etype(A,numerical),
       X != Y, Y > X.
max(A, X) :- not notmax(A,X),
        element(A,X),
        etype(A,numerical).
```

Extracting relevant facts from the puzzle clues

A sample of clues with their corresponding representations was given earlier. Let us take a closer look at the clue "Tony was the third person to have his fortune told.", encoded as :- $tuple(I, tony), not\ tuple(I, 3)$. This encoding specifies that if "Tony" is assigned to tuple I, while the rank "3" is not assigned to the tuple I, we obtain False. Thus this ASP rule limits all the models of it's program to have "Tony" assigned to the same tuple as "3". One of the questions one might ask is where are the semantic data for "person" or "fortune told". They are missing from the translation since with respect to the actual goal of solving the puzzle, they do not contribute anything new or relevant. The fact that "Tony" is a "person" is not relevant. With this encoding, we attempt to encode only new and relevant information with regards to the solutions of the puzzle. This is to keep the structure of the encodings as simple and as general as possible. In addition, if the rule would be encoded as :- $person(tony), tuple(I, tony), not\ tuple(I, 3)$., the fact $person(tony)$ would have to be added to the program in order for the constraint to give it's desired meaning. However, this does not seem reasonable as there is no apparent reason to add it (outside for the clue to actually work), since "person" is not part of the domain data of the puzzle.

Translating Natural language to ASP

To translate the English descriptions into ASP, we adopt the approach in (Baral et al. 2011). This approach uses Inverse-λ computations and generalization techniques together with learning. However for this paper, we had to adapt the approach to the ASP language and develop an ASP-λ-Calculus. An example of a clue translation using combinatorial categorial grammar (Steedman 2000) and ASP-λ-calculus is given in table 1.

The overall learning algorithm

The input to the overall learning algorithm is a set of pairs $(S_i, L_i), i = 1, ..., n$, where S_i is a sentence and L_i its corresponding logical form. The output of the algorithm is a PCCG defined by the lexicon L_T and a parameter vector

Θ_T. As given by (Baral et al. 2011), the parameter vector Θ_i is updated at each iteration of the algorithm. It stores a real number for each item in the dictionary. The overall learning algorithm is given as follows:

- **Input:** A set of training sentences with their corresponding desired representations $S = \{(S_i, L_i) : i = 1...n\}$ where S_i are sentences and L_i are desired expressions. Weights are given an initial value of 0.1.

  An initial feature vector Θ_0. An initial lexicon L_0.
- **Output**: An updated lexicon L_{T+1}. An updated feature vector Θ_{T+1}.
- **Algorithm:**

```
– Set $L_0$
– For t = 1 . . . T
– Step 1: (Lexical generation)
– For i = 1...n.
  ∗ For j = 1...n.
  ∗ Parse sentence $S_j$ to obtain $T_j$
  ∗ Traverse $T_j$
    · apply $INVERSE\_L$, $INVERSE\_R$ and $GENERALIZE_D$ to find new ASP-$\lambda$-calculus expressions of words and phrases $\alpha$.
  ∗ Set $L_{t+1} = L_t \cup \alpha$
– Step 2: (Parameter Estimation)
– Set $\Theta_{t+1} = UPDATE(\Theta_t, L_{t+1})$
```

- return $GENERALIZE(L_T, L_T), \Theta(T)$

To translate the clues, a trained model was used to translate these from natural language into ASP. This model includes a dictionary with ASP-λ-calculus formulas corresponding to the semantic representations of words. These have their corresponding weights.

Evaluation

To evaluate the clue translation, we manually selected 800 clues. Standard 10 fold cross validation was used. $Precision$ measures the number of correctly translated clues, save for permutations in the body of the rules, or head of disjunctive rules. $Recall$ measures the number of correct exact translations, $F\text{-}measure$ is the harmonic mean of Recall and Precision.

To evaluate the puzzles, we used the following approach. A number of puzzles were selected and all their clues formed the training data for the natural language module. The training data was used to learn the meaning of words and the associated parameters and these were then used to translate the English clues to ASP. These were then combined with the corresponding puzzle domain data, and the generic/background ASP module. The resulting program was solved using $clingo$, an extension of $clasp$ (Gebser et al. 2007). $Accuracy$ measured the number of correctly solved puzzles. A puzzle was considered correctly solved if it provided a single correct solution. If a rule provided by the clue translation from English into ASP was not syntactically correct, it was discarded. We did several experiments. Using the 50 puzzles, we did a 10-fold cross validation to measure the accuracy. In addition, we did additional experiments with 10, 15 and 20 puzzle *manually chosen* as training data. The $C\&C$ parser (Clark and Curran 2007) was used to obtain the majority of the syntactic categories. The background knowledge for the 50 puzzles contained 62 different ASP facts and rules, with the puzzles having 17.6 clues on average (880 total).

Results

The results are given in tables 2 and 3. The "10-fold" corresponds to experiments with 10-fold cross validation, "10-s", "15-s" and "20-s" corresponds to experiments where 10, 15 and 20 puzzles were manually chosen as training data respectively.

Precision	Recall	F-measure
87.13	85.86	86.49

Table 2: Clue translation performance.

	Accuracy
10-Fold	27/50 (54%)
10-s	21/40 (52.5%)
15-s	23/35 (65.71%)
20-s	25/30 (83.33%)

Table 3: Performance on puzzle solving.

The results for clue translation to ASP are comparable to translating natural language sentences to Geoquery and Robocup domains in (Baral et al. 2011), and used in similar works such as (Zettlemoyer and Collins 2007) and (Ge and Mooney 2009). Our results are close to the values reported there, which range from 88 to 92 percent for the database domain and 75 to 82 percent for the Robocup domain.

Conclusion and Future work

In this paper we presented a learning based approach to solve combinatorial logic puzzles in English. Our system uses an initial dictionary and sample puzzles in simplified English and their ASP representations to learn how to translate new puzzles (given in simplified English) to ASP and solve them, where the unique answer set of the translated ASP program corresponds to the solution of the puzzle. In the process it may use some general knowledge modules. Our system used results and components from various AI sub-disciplines including natural language processing, knowledge representation and reasoning, machine learning and ontologies as well as the functional programming concept of λ-calculus. There are many ways to extend our work. The simplified English limitation might be lifted by better natural language processing tools and additional sentence analysis. We could also apply our approach to different types of puzzles. A modified encodings might yield a smaller variance in the results. Finally we would like to submit that solving puzzles given in a natural language could be considered as a challenge problem for human level intelligence as it encompasses various facets of intelligence that we listed earlier. In particular, one has to use a reasoning system and can not substitute it with surface level analysis that is often used in information retrieval based methods.

For future work, we would like to add additional background knowledge which can be found in different puzzles.

Earl's	lucky	element	is	fire.
NP/NP	NP/NP	NP	$(S\backslash NP)/NP$	NP
NP/NP	NP		$(S\backslash NP)/NP$	NP
NP			$(S\backslash NP)$	
NP			$(S\backslash NP)$	
			S	

Earl's	lucky	element	is	fire.
$\lambda x.(x@earl)$	$\lambda x.x$	$\lambda x.x$	$\lambda x.\lambda y. : -tuple(I,x), not\ tuple(I,y).$	$fire$
$\lambda x.(x@earl)$	$\lambda x.x$		$\lambda x.\lambda y. : -tuple(I,x), not\ tuple(I,y).$	$fire$
$earl$			$\lambda x. : -tuple(I,x), not\ tuple(I,fire).$	
$earl$			$\lambda x. : -tuple(I,x), not\ tuple(I,fire).$	
			$: -tuple(I,earl), not\ tuple(I,fire).$	

Table 1: CCG and λ-calculus derivation for "Earl's lucky element is fire."

Our current modules only covered the puzzles we used for evaluation, however many new puzzles may require different background knowledge. For example, geographical information or locations of world cities are required in order to solve some of the other puzzles. In this work, we assumed that the domain of the puzzle is given for each puzzle. However, it might be possible to extract this information from the actual clues and the puzzle's accompanying story. Such system would then be able to automatically generate the domain of a puzzle, while the presented system can be used to translate the clues. Another limitation of the presented work is that the clues had to be manually pre-processed by humans to get rid of anaphoras. This can be avoided by programs that can preprocess anaphora and can split the clues in a such a way that the presented system can learn from them. With such improvements in place, the system could be made fully automatic. It could then just be given a set of puzzles with ASP solutions, and it would learn from it and then be able to solve new puzzles without any human intervention. *That is our goal and we propose it as a challenge to the community.*

Acknowledgement

We acknowledge support by NSF (grant number 0950440), IARPA and ONR-MURI for this work.

References

Baral, C.; Gonzalez, M.; Dzifcak, J.; and Zhou, J. 2011. Using inverse λ and generalization to translate english to formal languages. In *Proceedings of the International Conference on Computational Semantics, Oxford, England, January 2011. http://arxiv.org/abs/1108.3843*.

Baral, C.; Dzifcak, J.; and Son, T. C. 2008. Using ASP and lambda calculus to characterize NL sentences with normatives and exceptions. In *AAAI*, 818–823.

Baral, C. 2003. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press.

Bos, J., and Markert, K. 2005. Recognising textual entailment with logical inference. In *HLT/EMNLP*.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. In *AAAI*.

Clark, S., and Curran, J. R. 2007. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics* 33.

Dell. 2005. Logic puzzles.

Ferrucci, D. A.; Brown, E. W.; Chu-Carroll, J.; Fan, J.; Gondek, D.; Kalyanpur, A.; Lally, A.; Murdock, J. W.; Nyberg, E.; Prager, J. M.; Schlaefer, N.; and Welty, C. A. 2010. Building watson: An overview of the deepqa project. *AI Magazine* 31(3):59–79.

Ge, R., and Mooney, R. J. 2005. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of CoNLL.*, 9–16.

Ge, R., and Mooney, R. J. 2009. Learning a compositional semantic parser using an existing syntactic parser. In *Proceedings of ACL-IJCNLP.*, 611–619.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. *Clasp* : A conflict-driven answer set solver. In *LPNMR*, 260–265.

Kwiatkowski, T.; Zettlemoyer, L.; Goldwater, S.; and Steedman, M. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *the 2010 Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, 1223–1233.

Kwiatkowski, T.; Zettlemoyer, L. S.; Goldwater, S.; and Steedman, M. 2011. Lexical generalization in ccg grammar induction for semantic parsing. In *EMNLP*, 1512–1523.

Montague, R. 1974. *Formal Philosophy. Selected Papers of Richard Montague*. Yale University Press.

Press, P. 2004. Logic problems.

Press, P. 2007. England's best logic problems.

Roush, W. June 2010. The story of siri, from birth at sri to acquisition by apple: Virtual personal assistants go mobile. *Xconomy; http://www.xconomy. com/san-francisco/2010/06/14/the-story-of-sirifrom-birth-at-sri-to-acquisition-by-apple-virtualpersonal-assistants-go-mobile/.*

Steedman, M. 2000. *The syntactic process*. MIT Press.

Zettlemoyer, L., and Collins, M. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of EMNLP-CoNLL*, 678–687.

Zettlemoyer, L., and Collins, M. 2009. Learning context-dependent mappings from sentences to logical form. In *ACL*.