The NL2KR Platform for building Natural Language Translation Systems

Nguyen H. Vo, Arindam Mitra and Chitta Baral
School of Computing, Informatics and Decision Systems Engineering
Arizona State University
{nguyen.h.vo, amitra7, chitta }@asu.edu

Abstract

This paper presents the NL2KR platform to build systems that can translate text to different formal languages. It is freely-available[1], customizable, and comes with an Interactive GUI support that is useful in the development of a translation system. Our key contribution is a user-friendly system based on an interactive multistage learning algorithm. This effective algorithm employs Inverse-λ, Generalization and user provided dictionary to learn new meanings of words from sentences and their representations. Using the learned meanings, and the Generalization approach, it is able to translate new sentences. NL2KR is evaluated on two standard corpora, Jobs and GeoQuery and it exhibits state-of-the-art performance on both of them.

1 Introduction and Related Work

For natural language interaction with systems one needs to translate natural language text to the input language of that system. Since different systems (such as a robot or database system) may have different input language, we need a way to translate natural language to different formal languages as needed by the application. We have developed a user friendly platform, NL2KR, that takes examples of sentences and their translations (in a desired target language that varies with the application), and some bootstrap information (an initial lexicon), and constructs a translation system from text to that desired target language.

[1] http://nl2kr.engineering.asu.edu/

Our approach to translate natural language text to formal representation is inspired by Montague's work (Montague, 1974) where the meanings of words and phrases are expressed as λ-calculus expressions and the meaning of a sentence is built from semantics of constituent words through appropriate λ-calculus (Church, 1936) applications. A major challenge in using this approach has been the difficulty of coming up with the λ-calculus representation of words.

Montague's approach has been widely used in (Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2010) to translate natural language to formal languages. In ZC05 (Zettlemoyer and Collins, 2005) the learning algorithm requires the user to provide the semantic templates for all words. A semantic template is a λ-expression (e.g. $\lambda x.p(x)$ for an arity one predicate), which describes a particular pattern of representation in that formal language. With all these possible templates, the learning algorithm extracts the semantic representation of the words from the formal representation of a sentence. It then associates the extracted meanings to the words of the sentence in all possible ways and ranks the associations according to some goodness measure. While manually coming up with semantic templates for one target language is perhaps reasonable, manually doing it for different target languages corresponding to different applications may not be a good idea as manual creation of semantic templates requires deep understanding of translation to the target language. This calls for automating this process. In UBL (Kwiatkowski et al., 2010) this process is automated by restricting the choices of formal representation and learning the meanings in a brute force manner. Given, a sentence S and its representation M in the restricted formal language,

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 899–908, Beijing, China, July 26-31, 2015.

it breaks the sentence into two smaller substrings $S1, S2$ and uses higher-order unification to compute two λ-terms $M1, M2$ which combines to produce M. It then recursively learns the meanings of the words, from the sub-instance $< S1, M1 >$ and $< S2, M2 >$. Since, there are many ways to split the input sentence S and the choice of $M1, M2$ can be numerous, it needs to consider all possible splittings and their combinations; which produces many spurious meanings. Most importantly, their higher-order unification algorithm imposes various restrictions (such as limited number of conjunctions in a sentence, limited forms of functional application) on the meaning representation language which severely limits its applicability to new applications. Another common drawback of these two algorithms is that they both suffer when the test sentence contains words that are not part of the training corpus.

Our platform NL2KR uses a different automated approach based on Inverse-λ (section 2.1) and Generalization (section 2.2) which does not impose such restrictions enforced by their higher-order unification algorithm. Also, Generalization algorithm along with Combinatory Categorical Grammar (Steedman, 2000) parser, allows NL2KR to go beyond the training dictionary and translate sentences which contain previously unseen words. The main aspect of our approach is as follows: given a sentence, its semantic representation and an initial dictionary containing the meaning of some words, NL2KR first obtains several derivation of the input sentence in Combinatory Categorical Grammar (CCG). Each CCG derivation tree describes the rules of functional application through which constituents combine with each other. With the user provided initial dictionary, NL2KR then traverses the tree in a bottom-up fashion to compute the semantic expressions of intermediate nodes. It then traverses the augmented tree in a top-down manner to learn the meaning of missing words using Inverse-λ (section 2.1). If Inverse-λ is not sufficient to learn the meaning of all unknown words, it employs Generalization (section 2.2) to guess the meanings of unknown words with the meaning of known similar words. It then restarts the learning process with the updated knowledge. The learning process stops if it learns the meanings of all words or fails to learn any new meaning in an iteration. In the latter case, it shows the augmented tree to the user. The user can then provide meanings of some unknown words and resumes the learning process.

Another distinguishing feature of NL2KR is its user-friendly interface that helps users in creating their own translation system. The closest system to NL2KR is the UW Semantic Parsing Framework (UW SPF) (Artzi and Zettlemoyer, 2013) which incorporates the algorithms in (Zettlemoyer and Collins, 2005; Kwiatkowski et al., 2010) . However, to use UW SPF for the development of a new system, the user needs to learn their coding guidelines and needs to write new code in their system. NL2KR does not require the users to write new code and guides the development process with its rich user interface.

We have evaluated NL2KR on two standard datasets: GeoQuery (Tang and Mooney, 2001) and Jobs (Tang and Mooney, 2001). GeoQuery is a database of geographical questions and Jobs contains sentences with job related query. Experiments demonstrate that NL2KR can exhibit state-of-the-art performance with fairly small initial dictionary. The rest of the paper is organized as follows: we first present the algorithms and architecture of the NL2KR platform in section 2; we discuss about the experiments in section 3; and finally, we conclude in section 4.

2 Algorithms and Architecture

The NL2KR architecture (Figure 1) has two subparts which depend on each other (1) NL2KR-L for learning and (2) NL2KR-T for translation. The NL2KR-L sub-part takes the following as input: (1) a set of training sentences and their target formal representations, and (2) an initial lexicon or dictionary consisting of some words, their CCG categories, and their meanings in terms of λ-calculus expressions. It then constructs the CCG parse trees and uses them for learning of word meanings.

Learning of word meanings is done by using Inverse-λ and Generalization (Baral et al., 2012; Baral et al., 2011) and ambiguity is addressed by a Parameter Learning module that learns the weights of the meanings. The learned meanings update the lexicon. The translation sub-part uses this updated lexicon to get the meaning of all the words in a new sentence, and combines them to get the meaning of the new sentence. Details of each module will be presented in the following subsections.

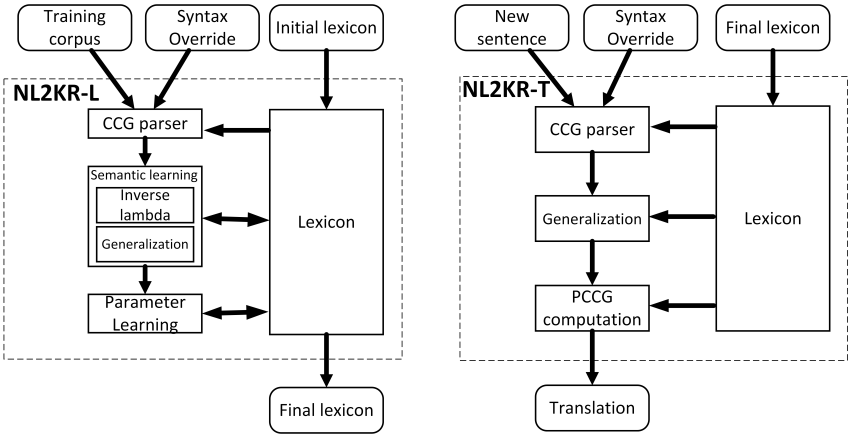

Figure 1: Architecture of NL2KR

The NL2KR platform provides a GUI (Figure 2) with six features: λ-application, Inverse-λ, Generalization, CCG-Parser, NL2KR-L and NL2KR-T. The fourth feature is a stand-alone CCG parser and the first four features can help on user with constructing the initial lexicon. The user can then use NL2KR-L to update the lexicon using training data and the NL2KR-T button then works as a translation system.

2.1 Inverse-λ

Inverse-λ plays a key role in the learning process. Formally, given two λ-expressions H and G with $H = F@G$ or $H = G@F$, the Inverse-λ operation computes the λ expression F. For example, given the meaning of "is texas" as $\lambda x2.x2@stateid(texas)$ and the meaning of "texas" as $stateid(texas)$, with the additional information that "is" acts as the function while "texas" is the argument, the Inverse-λ algorithm computes the meaning of "is" as $\lambda x3.\lambda x2.x2@x3$ (Figure 4). NL2KR implements the Inverse-λ algorithm specified in (Baral et al., 2012). The Inverse-λ module is separately accessible through the main GUI (Figure 2).

2.2 Generalization

Generalization (Baral et al., 2012; Baral et al., 2011) is used when Inverse-λ is not sufficient to learn new semantic representation of words. In contrast to Inverse-λ which learns the exact meaning of a word in a particular context, Generalization learns the meanings of a word from similar words with existing representations. Thus, Generalization helps NL2KR to learn meanings of words that are not even present in the training data set. In the current implementation, two words are considered as similar if they have the exact same CCG category. As an example, if we want to generalize the meaning of the word "plays" with CCG category $(S\backslash NP)/NP)$ and the lexicon already contains an entry for "eats" with the same CCG category, and the meaning $\lambda y.\lambda x.eats(x, y)$, the algorithm will extract the template $\lambda y.\lambda x.WORD(x, y)$ and apply the template to $plays$ to get the meaning $\lambda y.\lambda x.plays(x, y)$.

2.3 Combinatory Categorial Grammar

Derivation of a sentence in Combinatory Categorial Grammar (CCG) determines the way the constituents combine together to establish the meaning of the whole. CCG is a type of phrase structure grammar and clearly describes the predicate-argument structure of constituents.

Figure 3 shows an example output of NL2KR's CCG parser. In the figure, "John" and "home" have the category [N] (means noun) and can change to [NP] (means noun phrase). The phrase"walk home" has the category [S\NP], which means that it can combine with a constituent with category [NP] ("John" in this case) from left with the *backward application* to form category [S] (sentence). The word "walk" has the category [(S\NP)/NP], which means it can combine with a constituent with category [NP] ("home") from right through the *forward application* combinator to form category [S\NP] (of "walk home").

A detailed description on CCG goes beyond the scope of this paper (see (Steedman, 2000) for more details). Since, natural language sentences can have various CCG parse trees, each expressing a different meaning of the sentence, a key challenge

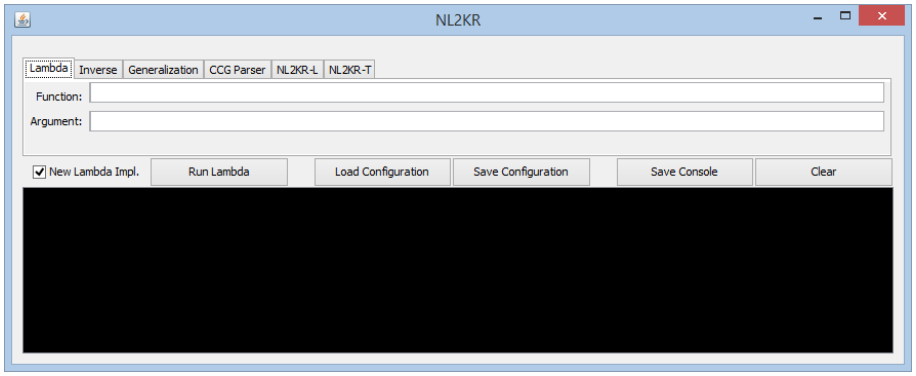

Figure 2: NL2KR's main GUI, Version 1.7.0001

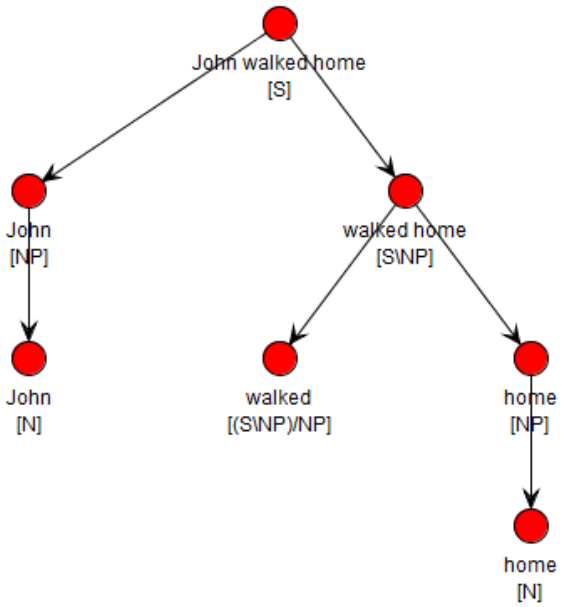

Figure 3: CCG parse tree of "John walked home".

in the learning and the translation process is to find a suitable CCG parse tree for a sentence in natural language. We overcome this impediment by allowing our learning and translation subsystem to work with multiple weighted parse trees for a given sentence and determining on the fly, the one that is most suitable. We discuss more on this in sections 2.4-2.6.

Existing CCG parsers (Curran et al., 2007; Lierler and Schüller, 2012) either return a single best parse tree for a given sentence or parse it in all possible ways with no preferential ordering among them. In order to overcome this shortcoming and generate more than one weighted candidate parse trees, we have developed a new parser using beam search with Cocke-Younger-Kasami(CYK) algorithm. NL2KRs CCG parser uses the C&C model (Curran et al., 2007) and constraints from the Stanford parser (Socher et al., 2013; Toutanova et al., 2003) to guide the derivation of a sentence. The output of the CCG parser is a set of k weighted parse trees, where the parameter k is provided by the user.

NL2KR system allows one to use the CCG parser independently through the interactive GUI. The output graphs look like the one in Figure 3. It can be zoomed in/out and its nodes can be moved around, making it easier to work with complex sentences.

2.4 Multistage learning approach

Learning meanings of words is the major component of our system. The inputs to the learning module are a list of training sentences, their target formal representations and an initial lexicon consisting of triplets of the form $<$*word, CCG category, meaning*$>$, where meanings are represented in terms of λ-calculus expressions. The output of the algorithm is a final dictionary containing a set of 4-tuples *(word, CCG category, meaning, weight)*.

Interactive Multistage Learning Algorithm (IMLA) NL2KR employs an Interactive Multistage Learning Algorithm (Algorithm 1) that runs many iterations on the input sentences. In each iteration, it goes through one or more of the following stages:

Stage 1 In Stage 1, it gets all the unfinished sentences. It then employs *Bottom Up-Top Down* algorithm (Algorithm 2) to learn new meanings (by Inverse-λ). For a sentence, if it has computed the meanings of all its constituents, which can be combined to produce the given representation, that sentence is considered as learned. Each

Algorithm 1 IMLA algorithm

```
1: function IMLA(initLexicon,sentences, sentsMeanings)
2:  regWords ← ∅
3:  generalize ← false
4:  lexicon ← initLexicon
5:  repeat
6:   repeat
7:    repeat
8:     for all s ∈ sentences do
9:      newMeanings ← BT(s,lexicon,sentsMeanings)
10:     lexicon ← lexicon ∪ newMeanings
11:     for all n ∈ newMeanings do
12:      ms ← GENERALIZE(regWords, n)
13:      lexicon ← lexicon ∪ ms
14:     end for
15:    end for
16:   until newMeanings = ∅
17:   if generalize=false then
18:    generalize ← true
19:    for all t ∈ unfinishedSents do
20:     words ← GETALLWORDS(t)
21:     ms ← GENERALIZE(words)
22:     lexicon ← lexicon ∪ ms
23:     regWords ← regWords ∪ words
24:    end for
25:   end if
26:  until newMeanings = ∅
27:  INTERATIVELEARNING
28: until unfinishedSents = ∅ OR userBreak
29: lexicon ← PARAMETERESTIMATION(lexicon,sentences)
30: return lexicon
31: end function
```

new meaning learned by this process is used to generalize the words in a waiting list. Initially, this waiting list is empty and is updated in stage 2. When no more new meaning can be learned by *Bottom Up-Top Down* algorithm, IMLA (Algorithm 1) enters stage 2.

Stage 2 In this stage, it takes all the sentences for which the learning is not yet finished and applies Generalization process on all the words of those sentences. At the same time, it populates those words into the waiting list, so that from now on, *Bottom Up-Top Down* will try to generalize new meanings for them when it learns some new meanings. It then goes back to stage 1. Next time, after exiting stage 1, it directly goes to stage 3.

Stage 3 When both aforementioned stages can not learn all the sentences, the *Interactive Learning* process is invoked and all the unfinished sentences are shown on the interactive GUI (Figure 4). Users can either skip or provide more information on the GUI and the learning process is continued.

After finishing all stages, IMLA (Algorithm 1) calls Parameter Estimation (section 2.5) algorithm to compute the weight of each lexicon tuple.

Bottom Up-Top Down learning For a given sentence, the CCG parser is used for the CCG parse trees like the one of *how big is texas* in Figure 4. For each parse tree, two main processes are called, namely "bottom up" and "top down". In the first process, all the meanings of the words in the sentences are retrieved from the lexicon. These meanings are populated in the leaf nodes of a parse tree (see Figure 4), which are combined in a bottom-up manner to compute the meanings of phrases and full sentences. We call these meanings, the *current meanings*.

In the "top down" process, using Inverse-λ algorithm, the given meaning of the whole sentence (called the *expected meaning* of the sentence) and the current meanings of the phrases, we calculate the expected meanings of each of the phrases from the root of the tree to the leaves. For example, given the expected meaning of *how big is texas* and the current meaning of *how big*, we use Inverse-λ algorithm to get the meaning (expected) of *is texas*. This expected meaning is used together with current meanings of *is* (*texas*) to calculate the expected meanings of *texas* (*is*). The expected meanings of the leaf nodes we have just learned will be saved to the lexicon and will be used in the other sentences and in subsequent learning iteration. The "top down" process is stopped when the expected meanings are same as the current meanings. And in both "bottom up" and "top-down" processes, the beam search algorithm is used to speed-up the learning process.

Interactive learning In the interactive learning process it opens a GUI which shows the unfinished sentences. Users can see the current and expected meanings for the unfinished sentences. When the user gives additional meanings of word(s), the λ-application or Inverse-λ operation is automatically performed to update the new meaning(s) to related

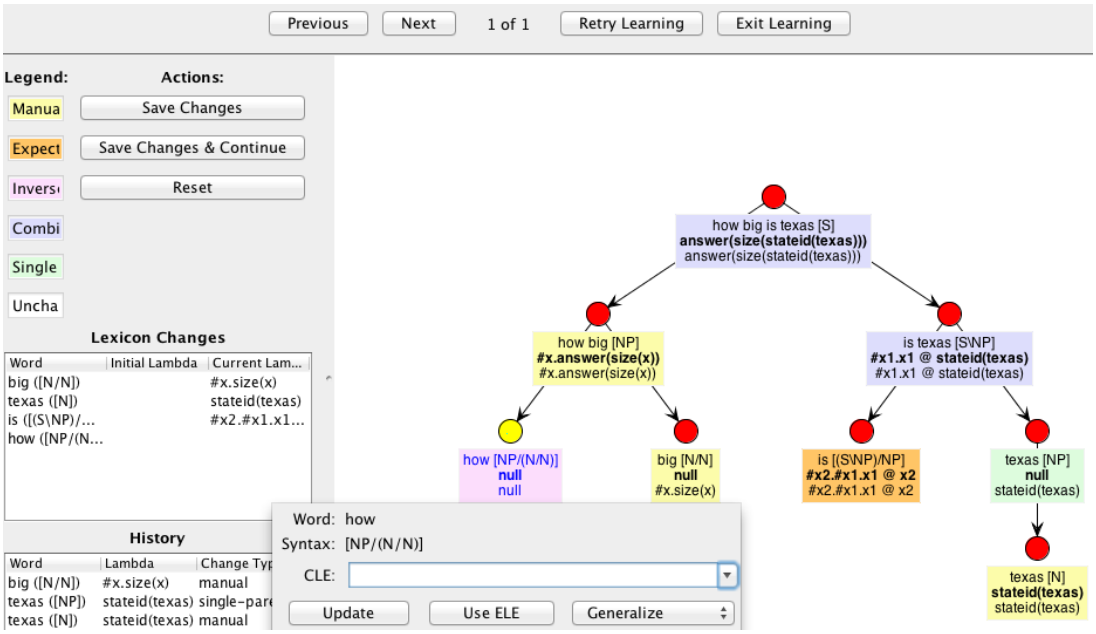

Figure 4: Interactive learning GUI. The box under each node show: the corresponding phrases [CCG category], the expected meanings and the current meanings. Click on the red node will show the window to change the current meaning (CLE)

Algorithm 2 BottomUp-TopDown (BT) algorithm

1: **function** BT($sentence, lexicon, sentsMeanings$)
2: $parseTrees \leftarrow$ CCGPARSER($sentence$)
3: **for all** $tree \in parseTrees$ **do**
4: $t \leftarrow$ BOTTOMUP($tree$,$lexicon$)
5: TOPDOWN(t,$sentsMeanings$)
6: **end for**
7: **end function**

word(s). Once satisfied, the user can switch back to the automated learning mode.

Example Let us consider the question "How big is texas?" with meaning $answer(size(stateid(texas)))$ (see Figure 4).

Let us assume that the initial dictionary has the following entries: *how* := $NP/(N/N) : \lambda x.\lambda y.answer(x@y)$, *big* := $N/N : \lambda x.size(x)$ and *texas* := $NP : stateid(texas)$. The algorithm then proceeds as follows.

First, the meanings of "*how*" and "*big*" are combined to compute the current meaning of "*how big*" := $NP : \lambda x.answer(size(x))$ in the "bottom up" process. Since the meaning of "*is*" is unknown, the current meaning of "*is texas*" still remains unknown.

It then starts the "top down" process where it knows the expected meaning of "*How big is texas*" := $S : answer(size(stateid(texas)))$ and the current meaning of "*how big*". Using them in the Inverse-λ algorithm, it then compute the meaning of "*is texas*" := $S\backslash NP : \lambda x1.x1@stateid(texas)$. Using this expected meaning and current meaning of "*texas*" := $NP : stateid(texas)$, it then calculates the expected meaning of "*is*" as "*is*" := $(S\backslash NP)/NP : \lambda x2.\lambda x1.x1@x2$. This newly learned expected meaning is then saved into the lexicon.

Since the meaning of all the words in the question are known, the learning algorithm stops here and the *Interactive Learning* is never called.

If initially, the dictionary contains only two meanings: "*big*" := $N/N : \lambda x.size(x)$ and "*texas*" := $NP : stateid(texas)$, NL2KR tries to first learn the sentence but fails to learn the complete sentence and switches to *Interactive Learning* which shows the interactive GUI (see Figure 4). If the user specifies that "*how*" means $\lambda x.\lambda y.answer(x@y)$, NL2KR combines its meaning with the meaning of "*big*" to get the meaning "*how big*" := $NP : \lambda x.answer(size(x))$. It will then use Inverse-λ to figure out the meaning of "*is texas*" and then the meaning of "*is*". Now all the meanings are combined to compute the current meaning $answer(size(stateid(texas)))$ of "*How big is texas*". This meaning is same as the expected

meaning, so we know that the sentence is successfully learned. Now, the user can press *Retry Learning* to switch back to automated learning.

2.5 Parameter Estimation

The Parameter Estimation module estimates a weight for each word-meaning pair such that the joint probability of the training sentences getting translated to their given representation is maximized. It implements the algorithm described in Zettlemoyer et. al.(2005).

2.6 Translation

The goal of this module is to convert input sentences into the target formalism using the lexicon previously learned. The algorithm used in Translation module (Algorithm 3) is similar to the bottom-up process in the learning algorithm. It first obtains several weighted CCG parse trees of the input sentence. It then computes a formal representation for each of the parse trees using the learned dictionary. Finally, it ranks the translations according to the weights of word-meaning pairs and the weights of the CCG parse trees. However, test sentences may contain words which were not present in the training set. In such cases, Generalization is used to guess the meanings of those unknown words from the meanings of the similar words present in the dictionary.

Algorithm 3 Translation algorithm

```
1: function TRANSLATE(sentence, lexicon)
2:   candidates ← ∅
3:   parseTrees ← CCGPARSER(sentence)
4:   for all tree ∈ parseTrees do
5:     GENERALIZE(tree);
6:     t ← BOTTOMUP(tree)
7:     candidates ← candidates ∪ t
8:   end for
9:   output ← VERIFY-RANK(candidates)
10:  return output
11: end function
```

3 Experimental Evaluation

We have evaluated NL2KR on two standard corpora: GeoQuery and Jobs. For both the corpus, the output generated by the learned system has been considered correct if it is an exact replica of the logical formula described in the corpus.

We report the performance in terms of precision (percentage of returned logical-forms that are correct), recall (percentage of sentences for which the correct logical-form was returned), F1-measure (harmonic mean of precision and recall) and the size of the initial dictionary.

We compare the performance of our system with recently published, directly-comparable works, namely, FUBL (Kwiatkowski et al., 2011), UBL (Kwiatkowski et al., 2010), λ-WASP (Wong and Mooney, 2007), ZC07 (Zettlemoyer and Collins, 2007) and ZC05 (Zettlemoyer and Collins, 2005) systems.

3.1 Corpora

GeoQuery GeoQuery (Tang and Mooney, 2001) is a corpus containing questions on geographical facts about the United States. It contains a total of 880 sentences written in natural language, paired with their meanings in a formal query language, which can be executed against a database of the geographical information of the United States. We follow the standard training/testing split of 600/280. An example sentence meaning pair is shown below.

Sentence: *How long is the Colorado river?*
Meaning: *answer(A,(len(B,A),const(B, riverid(colorado)), river(B)))*

Jobs The Jobs (Tang and Mooney, 2001) dataset contains a total of 640 job related queries written in natural language. The Prolog programming language has been used to represent the meaning of a query. Each query specifies a list of job criteria and can be directly executed against a database of job listings. An example sentence meaning pair from the corpus is shown below.

Question: *What jobs are there for programmers that know assembly?*
Meaning: *answer(J,(job(J),title(J,T), const(T,'Programmer'),language(J,L), const(L,'assembly'))))*

The dataset contains a training split of 500 sentences and a test split of 140 sentences.

3.2 Initial Dictionary Formulation

GeoQuery For GeoQuery corpus, we manually selected a set of 100 structurally different sentences from the training set and initiated the learning process with a dictionary containing the repre-

	GUI Driven	Initial Dictionary	Learned Dictionary
♯ <word, category >	31	118	401
♯ <word, category, meaning>	36	127	1572
♯ meaning	30	89	819

Table 1: Comparison of Initial and Learned dictionary for GeoQuery corpus on the basis of the number of entries in the dictionary, number of unique <word, CCG category> pairs and the number of unique meanings across all the entries. "GUI Driven" denotes the amount of the total meanings given through interactive GUI and is a subset of the Initial dictionary.

	GUI Driven	Initial Dictionary	Learned Dictionary
♯ <word, category>	58	103	226
♯ <word, category, meaning>	74	119	1793
♯ meaning	57	71	940

Table 2: Comparison of Initial and Learned dictionary for Jobs corpus.

sentation of the nouns and question words. These meanings were easy to obtain as they follow simple patterns. We then trained the translation system on those selected sentences. The output of this process was used as the initial dictionary for training step. Further meanings were provided on demand through interactive learning. A total of 119 word meanings tuples (Table 1, ♯ <*word, category, meaning* >) were provided from which the NL2KR system learned 1793 tuples. 45 of the 119 were representation of nouns and question words that were obtained using simple patterns. The remaining 74 were obtained by a human using the NL2KR GUI. These numbers illustrate the usefulness of the NL2KR GUI as well as the NL2KR learning component. One of our future goals is to further automate the process and reduce the GUI interaction part.

Table 1 compares the initial and learned dictionary for GeoQuery on the basis of number of unique <*word, category, meaning*> entries in dictionary, number of unique <*word, category*> pairs and the number of unique meanings across all the entries in the dictionary. Since each unique <word, CCG category> pair must have at least one meaning, the total number of unique <*word, category*> pairs in the training corpus provides a lower bound on the size of the ideal output dictionary. However, one <*word, category*> pair may have multiple meanings, so the ideal dictionary can be much bigger than the number of unique <*word, category*> pairs. Indeed, there were many words such as "of", "in" that had multiple meanings for the same CCG category. Table 1 clearly describes that the amount of initial effort is substantially less compared to the return.

Jobs For the Jobs dataset, we followed a similar process as in the GeoQuery dataset. A set of 120 structurally different sentences were selected and a dictionary was created which contained the representation of the nouns and the question words from the training corpus. A total of 127 word meanings were provided in the process. Table 2 compares the initial and learned dictionary for Jobs. Again, we can see that the amount of initial effort is substantially less in comparison to the return.

3.3 Precision, Recall and F1-measure

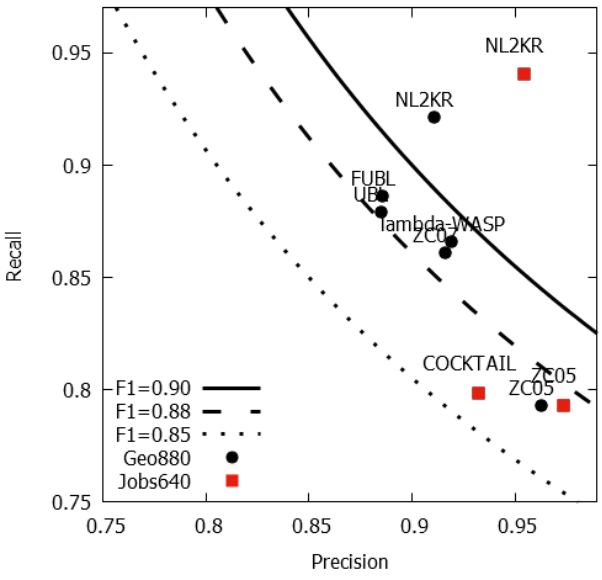

Figure 5: Comparison of Precision, Recall and F1-measure on GeoQuery and Jobs dataset.

Table 3, Table 4 and Figure 5 present the comparison of the performance of NL2KR on the GeoQuery and Jobs domain with other recent works. NL2KR obtained 91.1% precision value, 92.1%

System	Precision	Recall	F1
ZC05	0.963	0.793	0.87
ZC07	0.916	0.861	0.888
λ-WASP	0.9195	0.8659	0.8919
UBL	0.885	0.879	0.882
FUBL	0.886	0.886	0.886
NL2KR	0.911	0.921	0.916

Table 3: Comparison of Precision, Recall and F1-measure on GeoQuery dataset.

System	Precision	Recall	F1
ZC05	0.9736	0.7929	0.8740
COCKTAIL	0.9325	0.7984	0.8603
NL2KR	0.9543	0.9403	0.9472

Table 4: Comparison of Precision, Recall and F1-measure on Jobs dataset.

recall value and a F1-measure of 91.6% on GeoQuery (Figure 5, Geo880) dataset. For Jobs corpus, the precision, recall and F1-measure were 95.43%, 94.03% and 94.72% respectively. In all cases, NL2KR achieved state-of-the-art recall and F1 measures and it significantly outperformed FUBL (the latest work on translation systems) on GeoQuery.

For both GeoQuery and Jobs corpus, our recall is significantly higher than existing systems because meanings discovered by NL2KRs learning algorithm is more general and reusable. In other words, meanings learned from a particular sentence are highly likely to be applied again in the context of other sentences. It may be noted that, larger lexicons do not necessarily imply higher recall as lambda expressions for two phrases may not be suitable for functional application, thus failing to generate any translation for the whole. Moreover, the use of a CCG parser maximizes the recall by exhibiting consistency and providing a set of weighted parse trees. By consistency, we mean that the order of the weighted parse tree remains same over multiple parses of the same sentence and the sentences having similar syntactic structures have identical ordering of the derivations, thus making Generalization to be more effective in the process of translation.

The sentences for which NL2KR did not have a translation are the ones having structural difference with the sentences present in the training dataset. More precisely, their structure was not identical with any of the sentences present in the training dataset or could not be constructed by combining the structures observed in the training sentences.

We analyzed the sentences for which the translated meaning did not match the correct one and observed that the translation algorithm selected the wrong meaning, even though it discovered the correct one as one of the possible meanings the sentence could have had in the target formal language. Among the sentences for which NL2KR returned a translation, there were very few instances where it did not discover the correct meaning in the set of possible meanings.

It may be noted that even though our precision is lower than ZC05 and very close to ZC07 and WASP; we have achieved significantly higher F1 measure than all the related systems. In fact, ZC05, which achieves the best precision for both the datasets, is better by a margin of only 0.019 on the Jobs dataset and 0.052 on the GeoQuery dataset. We think one of the main reasons is that it uses manually predefined lambda-templates, which we try to automate as much as possible.

4 Conclusion

NL2KR is a freely available[2], user friendly, rich graphical platform for building translation systems to convert sentences from natural language to their equivalent formal representations in a wide variety of domains. We have described the system algorithms and architecture and its performance on the GeoQuery and Jobs datasets. As mentioned earlier, the NL2KR GUI and the NL2KR learning module help in starting from a small initial lexicon (for example, 119 in Table 2) and learning a much larger lexicon (1793 in Table 2). One of our future goals is to reduce the initial lexicon to be even smaller by further automating the NL2KR GUI interaction component .

Acknowledgements

We thank NSF for the DataNet Federation Consortium grant OCI-0940841 and ONR for their grant N00014-13-1-0334 for partially supporting this research.

[2]More examples and a tutorial to use NL2KR are available in the download package.

References

Yoav Artzi and Luke Zettlemoyer. 2013. UW SPF: The University of Washington Semantic Parsing Framework. *arXiv preprint arXiv:1311.3011*.

Chitta Baral, Juraj Dzifcak, Marcos Alvarez Gonzalez, and Jiayu Zhou. 2011. Using inverse λ and generalization to translate english to formal languages. In *Proceedings of the Ninth International Conference on Computational Semantics*, pages 35–44. Association for Computational Linguistics.

Chitta Baral, Juraj Dzifcak, Marcos Alvarez Gonzalez, and Aaron Gottesman. 2012. Typed answer set programming lambda calculus theories and correctness of inverse lambda algorithms with respect to them. *TPLP*, 12(4-5):775–791.

Alonzo Church. 1936. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58(2):345–363, April.

James Curran, Stephen Clark, and Johan Bos. 2007. Linguistically Motivated Large-Scale NLP with C&C and Boxer. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 33–36, Prague, Czech Republic, June. Association for Computational Linguistics.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1223–1233. Association for Computational Linguistics.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523. Association for Computational Linguistics.

Yuliya Lierler and Peter Schüller. 2012. Parsing combinatory categorial grammar via planning in answer set programming. In *Correct Reasoning*, pages 436–453. Springer.

Richard Montague. 1974. English as a Formal Language. In Richmond H. Thomason, editor, *Formal Philosophy: Selected Papers of Richard Montague*, pages 188–222. Yale University Press, New Haven, London.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing with Compositional Vector Grammars. In *ACL (1)*, pages 455–465.

Mark Steedman. 2000. *The syntactic process*, volume 35. MIT Press.

Lappoon R Tang and Raymond J Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Machine Learning: ECML 2001*, pages 466–477. Springer.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*.

Yuk Wah Wong and Raymond J Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Annual Meeting-Association for computational Linguistics*, volume 45, page 960. Citeseer.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *UAI*, pages 658–666. AUAI Press.

Luke S Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-2007)*.