An Example-Based Semantic Parser for Natural Language

Michel Généreux
Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1030, Vienna, Austria
michel@oefai.at

Abstract

This paper presents a method for guiding semantic parsers based on a statistical model. The parser is *example* driven, that is, it learns how to interpret a new utterance by looking at some examples. It is mainly predicated on the idea that similarities exist between contexts in which individual parsing actions take place. Those similarities are then used to compute the degree of certainty of a particular parse. The treatment of word order and the disambiguation of meanings can therefore be learned.

Mots-clefs – Keywords

Analyseur sémantique, Corpus, Langue naturelle
Semantic parser, Corpus, Natural language

1 Introduction

In order to achieve better results in *acquisition*, *coverage*, *robustness* and *portability*, corpus-based methods have been recently applied with success in areas like speech recognition (Rabiber, 1989), part-of-speech tagging (Charniak, Hendrickson, Jacobson and Perkowitz, 1993) and syntactic parsing (Manning and Carpenter, 1997). In *Semantic Parsing*, the process of mapping a natural language input to some structured meaning representation, Collins and Miller (Collins, 1998) describe a statistical model for extraction of *events*; Miller, Stallard, Bobrow and Schwartz describe an approach entirely based on a trained statistical model (Miller, Stallard, Robrow and Schwartz, 1996) and Thompson, Mooney and Tang (Thompson, Mooney and Tang, 1997) propose a novel and very interesting approach based on a bottom-up parser and a machine learning algorithm. I propose an approach in which a bottom-up parser is combined with a statistical model. Figure 1 shows the overall architecture of the system.

2 Overview of the Parsing Process

This section presents the various elements of the system. The parser used is a variant of a *Shift-Reduce* parser (Marcus, 1980). It actually comprises three different manageable actions that the parser uses to get to the final parse, which is a semantic interpretation (in first-order logic) of a natural language utterance. This left to right parsing makes the process relatively intuitive for humans.

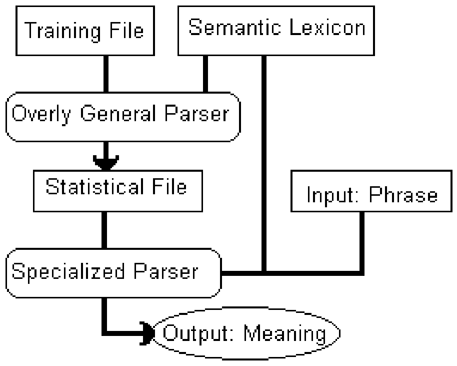

Figure 1: The Parser Architecture

The Input String The input string is a list of words to give an interpretation for. When no action is applicable and the input string is empty, then the parsing process is completed. For example[1]:

(1) [Ich,suche,einen,Artikel,über,Bush]

The Parse Stack Format: **[concept1:[context1],concept2:[context2],. . .]**

The parse stack is the actual parse state, the current interpretation of the input string found so far. It is a list of binary elements, each element representing a combination of the introduced predicate (or concept) with its context of introduction. Here is an example:

(2) [suche([],zeitung(_),zeit(_)):[suche,einen,Artikel],start:[Ich]]

The sHIFT action Syntax: **sHIFT(word_to_be_shifted)**

A *sHIFT* action simply puts the first word from the input string at the end of the context of the concept on the top of the parse stack. For example, the action *sHIFT(über)* on the parse stack 2 would result in the following new parse stack:

(3) [suche([],zeitung(_),zeit(_)):[suche,einen,Artikel,über],start:[Ich]]

The iNTRODUCE action Syntax: **iNTRODUCE(concept_to_be_introduced)**

The iNTRODUCE action takes a concept from the semantic lexicon and puts it on the top of the parse stack, initializing its context of introduction to the word (or list of words) that triggered this concept. For example, the action *iNTRODUCE(topic(1))* on the parse stack 2 would result in the following new parse stack:

(4) [topic(1):[topic(1)],suche([],zeitung(_),zeit(_)):[suche,einen,Artikel],start:[Ich]]

The dROP action Syntax: **dROP(source_term, target_term)**

The dROP action attempts to place a term from the parse stack as argument to another term of the parse stack. For example, the action *dROP(topic(1),suche([],zeitung(_),zeit(_)))* on the parse stack 4 would result in the following new parse stack:

(5) [suche([topic(1)],zeitung(_),zeit(_)):[suche,einen,Artikel],start:[Ich]]

A Parse State Format: **op(aCTION(arguments)#Parse_Stack#Input_String)**

It indicates in which context, i.e. how the Parse Stack and the Input String looked like, when the action took place. *Op* is simply a container[2] for all types of actions.

[1] Although not yet topicalized.
[2] A container is a term which embraces other terms.

A Final State Format: **final(Parse_Stack)**
It indicates the final aspect of a parse, i.e. the meaning found for an input string.

Semantic Lexicon Format: **lexicon(CONCEPT, [TRIGGERING_PHRASE])**
It comprises all the concepts and their triggering phrase(s) that we wish our parser to process. For example, *suche* or *brauche* would trigger the *suche* concept. Here is an example of a lexical entry:

(6) lexicon(suche([],zeitung(_),zeit(_)),[suche]).

The shift-reduce parser I am now ready to present the variant of the shift-reduce parser I am using. The algorithm of the parser is as follows:

1. Try to *introduce* a new concept or *shift* a word.
2. If possible, make one *drop* action.
3. If there are more words in the input string, go back to Step 1. Otherwise stop.

Topic_extraction Typically, topic_extraction replaces relevant noun phrases or prepositional phrases in the input string by successive *topic(_)* terms. For our running example 1, after topic_extraction, this is the following phrase which is passed on to the parser:

[Ich,suche,einen,Artikel,topic(1)]

3 Training

The training phase records successful actions (called *op*), as well as the different final states. For each of them uniquely defined, it assigns a frequency measure, defined as follows:

$$Frequency = \frac{Occurence\_of\_a\_particular\_action\_in\_a\_specific\_context}{Total\_number\_of\_occurence\_of\_this\_action} \quad (7)$$

In the following are examples of sentences on which the parser was trained (*a test set*).

Written Korpus

Artikel über das Wiener Neujahrskonzert suche ich.
Etwas über das Konzert als festen Bestandteil des kulturellen Lebens suche ich.

Spoken Korpus

Ich möchte jetzt eine neue Suche beginnen.
Die Kosovo-Krise un und Bill Clinton.

Artificial data

Bitte geben Sie mir einen Text zum Thema Kosovo-Krise.
Aber bitte nur in der Zeitung Salzburger Nachrichten von vor einer Woche.

Training file The training file is the file in which training examples are stored. These examples have the following format:

training([topicalized_phrase], meaning).

Meaning is in the form of first order logic expressions. A training file example could be[3]:

```
training([Ich,moechte,jetzt,eine,neue,Suche,beginnen],neue_suche).
training([Ich,suche,einen,Artikel,topic(1)],suche([topic(1)],zeitung(_),zeit(_))).
```

[3]oe stands for ö, ue stands for ü and ae stands for ä

Note that the examples for which we wish to train for should be topicalized for a changing domain. The third example could be helpful in training for a sentence like:

Ich suche einen Artikel über Bush.

While training, an *overlyGeneralParser* is used. It is overly general in the sense that it tries any possible actions to get to the final parse, without considering any information (such as *statistics*) that could be helpful to guide the parsing process. In training, a *training beam* can be specified. This means that only a certain number of parses will be recorded in the *statistical file* for each training example.

File used by the Specialized Parser: the Statistical File The overlyGeneralParser parses the *training file* to generate the *statistical file*. Every step needed to go from the *topicalized_phrase* to the *meaning* is recorded, as well as final states themselves. Final states are simply the states of the parse stack themselves at the end of the parse. Each of them (actions and final states) are assigned a frequency measure as described previously. Each line has either one of the following format (recall that *op* is a container for any action):

```
op(ACTION#PARSE_STACK#INPUT_STRING#FREQUENCY).
final(FINAL_STATE#FREQUENCY).
```

Here is an example:

```
op(sHIFT(Ich)#[start:[]]#[Ich,suche,einen,Text,for,topic(1),topic(2),
  bitte,bearbeiten,Sie,meinen,Suchauftrag]#0.3333).
```

These lines are used by the *specializedParser* to compute the best parse.

4 Statistical Parsing

The actual parsing of the input phrase is done by a *specializedParser*. It is specialized in the sense that it uses a statistical model to process all the information available from the training phase in order to get the best possible parse (the one with the highest probability). This section presents a detailed description of the statistical model used. Weighting parameters are presented but not discussed.

The Search space Like in the training phase, the most obvious way to influence the parse is to tell the parser how many parses it should try before taking a decision. I call it the *search beam* parameter.

Measure of similarity between lists This is a crucial aspect of the specialized parser. When the parser tries to choose a suitable parse, it must compare list of words (to compare *Actions*, *Parse stacks* or *Input strings*). A good *similarity* measure between lists is essential, but because computing similarity is very demanding on computer resources, one must find a trade-off that preserves computational efficiency. At the top level, the similarity measure is simply a measure of the number of identical elements in both lists, divided by the size of the largest list. Therefore, we have:

$$Similarity = \frac{Number\_of\_identical\_elements}{Size\_of\_the\_biggest\_list} \tag{8}$$

For example, omitting case-sensitivity:

(9) similarity([Ich,suche,einen,Artikel],[Das,suche,ich]) = 2/4 = 0.5

Comparisons sometimes involves structures. Structures can be decomposed into list in PROLOG[4], and then compared by using equation 8:

$$predicate(arg1, arg2, ...) = [predicate, arg1, arg2, ...] \tag{10}$$

Therefore, we can now roughly compare structures as lists. Preliminary empirical results tend to show that the approximation is sensible .

Parameterizing the model I introduce all the equations for the model, explaining their context of use. The best parse P is found by taking the highest probability P_i among the possible parses (limited by the search beam) available:

$$\mathsf{P} = \max_i P_i \tag{11}$$

Each of these parses P_i have a probability that amounts to combining the probability of the individual *op* or actions together ($\prod_k a_k$) and adding the probability of the final state ($ProbF$, see equation 15). These two components must be appropriately weighted by Pop and $Pfinal$. Multiplying by 100 gives a more readable value between 0 and 100.

$$P_i = (Pop * (\prod_k a_k) + Pfinal * ProbF) * 100 \tag{12}$$

The way each *op* a_k is assigned a probability is by taking into account its similarity with one of the *ops* in the statistical file (see equation 14) as well as the frequency of this *op* ($Frequency$). These two components are also weighted by Pop_sim and Pop_occ.

$$a_k = max_m(Pop\_sim * P_m + Pop\_occ * Frequency) \tag{13}$$

While looking for a suitable *op* in the statistical file, the parser looks for similarity. The similarity of an *op* P_m with one in the statistical file is measured by multiplying together the similarity of the *op* as such, the similarity of the *parse stack* and the similarity of the *input string* (*t* stands for *training*).

$$P_m = max(sim\_A(op_{action}, t_{op}) * sim\_PS(op_{ps}, t_{ps}) * sim\_I(op_{input}, t_{input})) \tag{14}$$

Computing the probability of a final parse state is similar to computing the one for actions. A final state probability $ProbF$ is the weighted sum of the most similar final state in the statistical file P_f (see 16) and the frequency of this final state $Frequency$:

$$ProbF = \max_f(Pfinal\_sim * P_f + Pfinal\_occ * Frequency) \tag{15}$$

$$P_f = max_n(sim(t_n, F)) \tag{16}$$

When needed, smoothing is carried out in a very conservative manner.

5 Results

I have conducted an experiment to test the performance of the parser. After training with only 80 examples, the parser averages 62% correctness while parsing a new sentence. *Recall* is therefore slightly lower than the other approaches mentioned at the beginning of the paper. I believe there are mainly four reasons to that:

1. The very low number (80) of training examples, compare to 560, 225 and 4000 sentences of other approaches.
2. The lack of extensive testing on what would be the best setting for default values of weighting parameters. Only a set of rather intuitive values were used.
3. The assimilation of natural language utterances to sets instead of lists.
4. A measure of similarity that sacrifices precision for computational efficiency.

[4]The programming language used.

6 Conclusion

In this paper, a new probabilistic framework for semantic parsing is presented. The combination of a *shift-reduce* parser and a purely statistical model makes it unique. More precisely, the parser learns efficient ways of parsing new sentences by collecting statistics on the context in which each parsing action takes place. It computes probabilities on the basis of the similarities of those contexts and their frequencies. The result is a simple and robust parser. Its configuration can be change in many ways, to fit different types of corpus or domains. At this point, the system has not yet been fully tested on very large corpora, to see if the statistical model remains as efficient. It does not include the treatment of variables, which means that there is no treatment of questions. However, testing with few examples, the parser shows promising results.
Compare to similar systems using some machine-learning techniques, ours offers an approach in which linguistics can play a decisive role; we have a more direct influence on the role of contexts in evaluating the probability of a parse. One crucial aspect of the parser, the computation of similarities between context, relies on a good interpretation of linguistic patterns found in phrases, and how those configurations may determine the particular meaning of a word or group of words. This is essential to interpret, and maybe *understand*, natural language utterances.

Acknowledgments

This work has been sponsored by the Fonds zur Förderung der wissenschaftlichen Forschung (FWF), Grant No. P13704. The Austrian Research Institute for Artificial Intelligence (ÖFAI) is supported by the Austrian Federal Ministry of Education, Science and Culture.

References

E. Charniak, C. Hendrickson, C. Jacobson and M. Perkowitz (1993), *Equations for part-of-speech tagging*, Proceedings of the 11th National Conference on AI, 784-789.

M. Collins and S. Miller (1998), *Semantic Tagging using a Probalistic Context Free Grammar*, In Proceedings of the Sixth Workshop on Very Large Corpora.

M.J. Collins (1997), *Three generative, lexicalised models for statistical parsing*, Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, 16-23.

S. Miller, D. Stallard, R. Bobrow and R. Schwartz (1996), *A Fully Statistical Approach to Natural Language Interfaces*, Proceedings of the 34th Annual Meeting of the ACL, Morgan Kaufmann Publishers, San Francisco, Arivind Joshi and Martha Palmer, 55-61.

L.R. Rabiner (1989), *A tutorial on hidden Markov models and selected applications in speech recognition*, Proceedings of the IEEE, Vol. 77, No. 2, 257-286.

C. A. Thompson, R. J. Mooney and L. R.Tang (1997), *Learning to Parse Natural Language Database Queries into Logical Form*, Proceedings of the ML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition.

C.D. Manning and B. Carpenter (1997), *Three generative, lexicalised models for statistical parsing*, Proceedings of the 5th Int. Workshop on Parsing Technologies, 147-158.

M.P. Marcus (1980), *A Theory of Syntactic Recognition for Natural Language*, MIT Press.

Ulf Hermjakob and Raymond J. Mooney (1997), *Learning Parse and Translation Decisions from Examples with Rich Context*, Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics, 482–489.