

An Example-Based Semantic Parser for Natural Language

Michel Génèreux

Austrian Research Institute for Artificial Intelligence (ÖFAI)

University of Vienna

Freyung 6, A-1010 Vienna

E-mail: 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. Ways of treating word order and disambiguating meanings can therefore be learned.

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 [Rabiner, 1989], part-of-speech tagging [Charniak *et al.*, 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 and Miller, 1998] describe a statistical model for extraction of *events*; Miller, Stallard, Bobrow and Schwartz [Miller *et al.*, 1996] describe an approach entirely based on a trained statistical model and Thompson, Mooney and Tang [Thompson *et al.*, 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 is meant to give a flavor of the parsing process and provides a "light" introduction to the parser. All statistical considerations are for the moment deferred to section 4. The parser used is a variant of a *Shift-Reduce* parser. It actually comprises three different 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. I

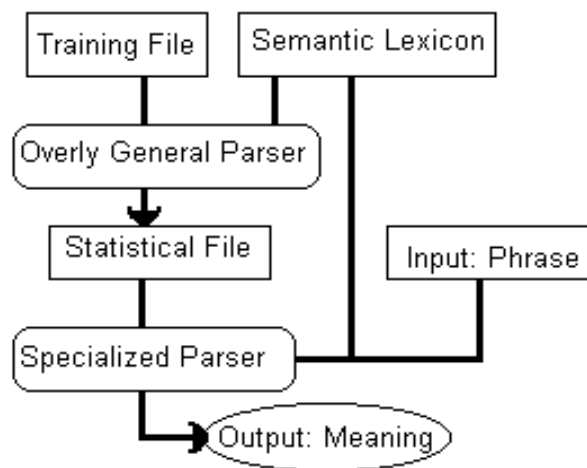


Figure 1: The Parser Architecture

first present the various elements making up our *specialized* parser and then go through a concrete parsing example.

2.1 The Input String

The input string is a list of words to give an interpretation for. When no actions are applicable and the input string is empty, then the parsing process is completed. Typically, one word is removed from the list for a *SHIFT* action, one or more for an *INTRODUCE* action and none for a *DROP* action. It also provides some contextual information while applying a parsing action. For example:

- (1) [Ich,suche,einen,Artikel,über,Bush]
is an input string¹.

2.2 The Parse Stack

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. The context gives

¹Although not yet topicalized, see section 2.10.

partial (but useful) information on the words following the concept at the time of introduction. Each concept must be in the semantic lexicon (see 2.8). Here is the general format of the parse stack followed by an example:

- ```
(2) [concept1:[context1],concept2:[context2],...]
(3) [suche([],zeitung(_),zeit(_)):suche,einen,Artikel],start:[Ich]]
```

Example 3 shows that the concept *suche* (which has three arguments) was introduced in the context of *suche einen Artikel* following it. The first argument (`[]`) will hold the list of topics (see 2.10) which are searched for. The second and third arguments are placeholders for the newspaper and the time period of the search. The types of arguments can be defined in the semantic lexicon (see 2.8). The *start* predicate is there only to provide room for words from the input string which would be shifted at the very beginning of the parse; of course, *start* does NOT contribute to the meaning of the input phrase.

### 2.3 The sHIFT action

Syntax:

```
sSHIFT(word_to_be_shifted)
```

A *sSHIFT* 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 *sSHIFT(über)* on the parse stack 3 would result in the following new parse stack:

- ```
(4) [suche([],zeitung(_),zeit(_)):suche,einen,Artikel,über],start:[Ich]]
```

2.4 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 (see 2.8) this concept. These concepts will then participate to the meaning representation. For example, the action *iNTRoDUCE(topic(1))* on the parse stack 3 would result in the following new parse stack:

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

### 2.5 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. The context of the source term is lost in the process. Some restrictions might be imposed in the semantic lexicon to prevent some undesirable combinations. For example, only newspaper names can be dropped into the *zeitung(\_)* argument of the concept *suche*. This action has no effect on the input string. For example, the action *dROP(topic(1),suche([],zeitung(\_),zeit(\_)))* on the parse stack 5 would result in the following new parse stack:

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

2.6 A Parse State (no statistics)

A parse state is given by the following expression:

```
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 container² for all types of actions.

2.7 A Final State

A final state is given by the following expression:

```
final(Parse_Stack)
```

It indicates the final aspect of a parse, i.e. the meaning we have found for an input string.

2.8 Semantic Lexicon

We also need a semantic lexicon. It comprises all the concepts and their triggering phrase(s) that we wish our parser to process. A triggering phrase is simply a word (or group of words) in the input string that triggers some concept. For example, *suche* or *brauche* would trigger the *suche* concept. The format of a lexical entry is:

```
lexicon(CONCEPT, [TRIGGERING_PHRASE]).
```

Here are two examples of lexical entries:

- ```
(6) lexicon(topic(1),[topic(1)]).
(7) lexicon(suche([],zeitung(_),zeit(_)),[suche]).
```

The term *topic(1)* is a placeholder for the interpretation of successive topics (here it is the first one in the input); this means that if an input phrase holds the topic *der Krieg*, then *topic(1)* is a synonym for it during the time of parsing. *Der Krieg* is only reintroduced at the end of the (hopefully successful) parse. The term *zeitung(\_)* is a placeholder for newspapers and *zeit(\_)* for the time period of the search. The underscore (`_`) signifies that it is still not yet specified. The type of the arguments for `[]`, *zeit* and *zeitung* must be specified in the semantic lexicon file.

### 2.9 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.

### 2.10 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)]
```

<sup>2</sup>A container is a term which embraces other terms.

## 2.11 A parsing example

I now show a parse for *Ich suche einen Artikel über Bush*. I assume for simplicity that the parser always takes the best available action. The following trace presents the successive actions taken by the parser. The initial parse stack and input string states for *Ich suche einen Artikel über Bush* are:

```
[start:[]]
```

and

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

Here is the complete parse, using the two lexical entries 6 and 7 shown in section 2.8. Each line represents a Parse State (omitting *op*) (see 2.6):

```
SHIFT(Ich)#[start:[]]#[Ich,suche,einen,Artikel,topic(1)]
```

The word *Ich* is not in the semantic lexicon<sup>3</sup>, so the only action possible is to shift it on the parse stack.

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

The word *suche* is in the lexicon, so it can be introduced as a new predicate on the parse stack. Another possibility would be to shift it.

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

The word *einen* is not in the lexicon, it must be shifted.

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

The word *Artikel* is not in the lexicon (it is actually an *empty\_word* for the newspaper domain) and is therefore shifted.

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

*topic(1)* is in the lexicon, so it can be introduced.

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

I can drop the predicate *topic(1)* into the first argument of the *suche* predicate. The final parse stack<sup>4</sup> or final state (see section 2.7) is:

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

In the final stage, the parser simply puts back the meaning for *topic(1)* collected during the *topic\_extraction* phase and the final parse is, without contextual information:

```
suche([(Bush)],zeitung(_),zeit(_))
```

which indicates a search for the topic *Bush* with no specific newspaper or time frame.

## 3 Training

The training phase uses an overly general parser, which produces all possible path of actions (only limited by the *search beam*, see 3.1) to be taken in order to get from each training example utterance to its semantic representation. In the process, it records successful actions (called *op*), as well as the different final

<sup>3</sup>Of course *Ich* is not semantically relevant for interpreting newspaper browsing commands or search.

<sup>4</sup>To be valid, a final Parse Stack must have only two predicates (including *start*).

states. For each of them uniquely defined, it assigns a frequency measure, defined as follows:

$$\text{Frequency} = \frac{\text{Occurrence\_of\_a\_particular\_action\_in\_a\_specific\_context}}{\text{Total\_number\_of\_occurrence\_of\_this\_action}} \quad (8)$$

## 3.1 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<sup>5</sup>:

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

Note that the examples for which we wish to train for should be topicalized (see 2.10) 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* (see 3.2) for each training example.

## 3.2 File used by the Specialized Parser: the Statistical File

The *overlyGeneralParser* parses the *training file* (see 3.1) to generate the *statistical file*. Every step needed to go from the *topicalized\_phrase* to the *meaning* (see 3.1) is recorded, as well as final states themselves. Final states (see 2.7) 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 (see section 3). 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 are two examples:

```
op(SHIFT(Ich)#[start:[]]#[Ich,suche,einen,Text,for,topic(1),topic(2),
bitte,bearbeiten,Sie,meinen,Suchauftrag]#0.3333).
final([bestaetigen(neue_suche):[bearbeiten,Sie,meinen,Suchauftrag],
start:[Ich,moechte,jetzt,eine]]#0.2).
```

These lines are used by the *specializedParser* to compute the best parse. The next section describes the statistical parsing process.

## 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

<sup>5</sup>oe stands for ö, ue stands for ü and ae stands for ä

best possible parse (the one with the highest probability). This section presents a detailed description of the statistical model used. In particular, it shows how different parameters in the model can be adapted to influence the parsing phase outcome. Some default values for the parameters are discussed.

#### 4.1 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.

#### 4.2 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 greatest list. Therefore, we have:

$$\text{Similarity} = \frac{\text{Number\_of\_identical\_elements}}{\text{Size\_of\_the\_biggest\_list}} \quad (9)$$

For example, omitting case-sensitivity:

$$(10) \text{ similarity}([\text{Ich, suche, einen, Artikel}], [\text{Das, suche, ich}]) \\ = 2/4 = 0.5$$

Comparisons sometimes involves structures. Structures can be decomposed into list in PROLOG<sup>6</sup>, and then compared by using equation 9:

$$\text{predicate}(\text{arg1}, \text{arg2}, \dots) = [\text{predicate}, \text{arg1}, \text{arg2}, \dots] \quad (11)$$

Therefore, we can now roughly compare structures as lists. The situation gets a little more complicated when we have a list of lists. Each sublist has its own measure of similarity. The problem is when to consider two sublists as similar enough (a *threshold* of similarity) so that they won't be compared further with other sublists. By doing so, I choose efficiency (by not comparing all elements of each list) at the cost of some approximation of similarity. Besides the advantage of computational efficiency, always underestimating lists similarity counterbalances overestimating in some cases, specifically when assimilating natural language to lists:

- Assimilating natural language utterances to *sets* instead of *lists*. Natural languages utterances are in general best assimilated to lists, except maybe for free word order languages.
- Assimilating first-order expressions to sets.

However, it is not clear whether or not these assumptions are correct; but it is also rather difficult to prove that *a priori*, they are not. Preliminary empirical results tend to show that they are sensible.

<sup>6</sup>The programming language used.

#### 4.3 Parametrizing 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:

$$P = \max_i P_i \quad (12)$$

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 19). These two components are weighted by *Pop* and *Pfinal*. Those weighting values must be chosen in such a way that translates the importance of the steps needed to get to a final parse compared to the final state itself. In short, the weighting of actions taken together must be high enough to discriminate among similar final states (in term of probability), should that case arises. Multiplying by 100 gives a more readable value between 0 and 100.

$$P_i = (\text{Pop} * (\prod_k a_k) + \text{Pfinal} * \text{ProbF}) * 100 \quad (13)$$

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 15) as well as the frequency of this *op* (*Frequency*). These two components are also weighted by *Pop\_sim* and *Pop\_occ*. Default values are chosen with respect to how one would want to consider the respective importance of *similarity* over *frequency*.

$$a_k = \max_m (\text{Pop\_sim} * P_m + \text{Pop\_occ} * \text{Frequency}) \quad (14)$$

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*).

$$P_m = \max (\text{sim\_A}(\text{op\_action}, t_{op}) * \text{sim\_PS}(\text{op\_ps}, t_{ps}) * \text{sim\_I}(\text{op\_input}, t_{input})) \quad (15)$$

The following three equations show how smoothing of similarities is carried out. For example, the minimum value of the similarity between two actions *smoothingAction* (see 16) is a fraction *smoothing\_Action* of the minimum similarity *min\_Action* found by random testing on a representative set of *ops* and *final* states. Those minimum values found are:

- $\text{min\_Action} = 0.5$
- $\text{min\_PS} = 0.1111$
- $\text{min\_Input} = 0.0667$

and the default fractions used are:

- $\text{smoothing\_Action} = 0.1$
- $\text{smoothing\_PS} = 0.5$
- $\text{smoothing\_Input} = 0.5$

Those default values reflects the idea that care should be taken not to overestimate the similarity of an action, while it is relatively safe to assign half of the

minimum similarity of the *parse stack* or the *input string* in the case of a computed similarity of zero. Note that similarity between actions is actually only between arguments, while the actions compared *must* of course be the same.

$$\text{smoothingAction} = \text{smoothing\_Action} * \text{min\_Action} \quad (16)$$

$$\text{smoothingParseStack} = \text{smoothing\_PS} * \text{min\_PS} \quad (17)$$

$$\text{smoothingInput} = \text{smoothing\_Input} * \text{min\_Input} \quad (18)$$

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<sub>f</sub>* (see 20) and the frequency of this final state *Frequency*:

$$\text{ProbF} = \max_f(P_{\text{final\_sim}} * P_f + P_{\text{final\_occ}} * \text{Frequency}) \quad (19)$$

$$P_f = \max_n(\text{sim}(t_n, F)) \quad (20)$$

$$\text{smoothingFinal} = \text{smoothing\_Final} * \text{min\_Final} \quad (21)$$

For smoothing, we have:

- $\text{smoothing\_Final} = 0.5$
- $\text{min\_Final} = 0.3333$

#### 4.4 A full parse with statistics

I am now going to parse the example shown in section 2.11 by showing how the specialized parser uses statistics (and some rules to constraint the format of the semantics) to find the best parse. Suppose we have the following two training examples:

```
training([suche, im, Standard, Informationen, topic(1)],
suche([topic(1)], zeitung(standard), zeit(_))).
training([bitte, einen, Artikel, topic(1), von, gestern],
suche([topic(1)], zeitung(_), zeit(tag(-1)))).
```

which lead, after training (using a training beam of one parse per example), to the following excerpt of a statistical file (once again, *op* represents a container for any action):

```
(22) ...
(23) op(sHIFT(bitte)#[start:[]]
#[bitte,einen,Artikel,topic(1),von,gestern]#1.0).
(24) ...
(25) final([suche([topic(1)],zeitung(standard),zeit(_))
:[suche,Informationen],start:[]]#0.5).
(26) ...
```

Note that three lexical entries had to be added to the semantic lexicon of section 2.8 for the overlyGeneralParser to train properly:

```
lexicon(suche([], zeitung(_), zeit(_)), [Informationen]).
lexicon(zeit(tag(-1)), [von, gestern]).
lexicon(zeitung(standard), [im, Standard]).
```

We now ask the *specializedParser* to parse *Ich suche einen Artikel über Bush* using a search beam of 5 parses. Once again, I show the calculations for the *best* parse only (the probability is the last value on each line).

```
sHIFT(Ich)#[start:[]]#[Ich,suche,einen,
Artikel,topic(1)]#0.5125
```

Among all the sHIFT actions in the statistical file, line 23 was the best suited (or most similar). So, using equation 14 for this action<sup>7</sup>:

$$a_k = \max_m(Pop_{sim} * P_m + Pop_{occ} * Frequency) \quad (27)$$

$$a = (0.5 * P + 0.5 * 1.0) \quad (28)$$

and equation 9 to compute individual similarities:

```
sim_A(sHIFT(Ich),sHIFT(bitte)) = 0.1 * 0.5
= 0.05 (smoothed)
sim_PS([start:[]],[start:[]]) = 1.0
sim_I([Ich,suche,einen,Artikel,topic(1)],
[bitte,einen,Artikel,topic(1),von,gestern])
= 0.5
```

Now equation 15 gives us the overall similarity:

$$P = (0.05 * 1.0 * 0.5) = 0.025 \quad (29)$$

Putting it all back to 28, we have:

$$a = (0.5 * 0.025 + 0.5 * 1.0) = 0.5125 \quad (30)$$

The parser will do that for each of the parsing actions, leading to the following results:

```
iNtRODUCE(suche([], zeitung(_), zeit(_)))
#[start:[Ich]]#[suche,einen,Artikel,
topic(1)]#0.2583
sHIFT(einen)#[suche([], zeitung(_), zeit(_))
:[suche],start:[Ich]]#[einen,Artikel,topic(1)]
#0.6
sHIFT(Artikel)#[suche([], zeitung(_), zeit(_))
:[suche,einen],start:[Ich]]#[Artikel,topic(1)]
#0.5042
iNtRODUCE(topic(1))#[suche([], zeitung(_), zeit(_))
:[suche,einen,Artikel],start:[Ich]]#[topic(1)]
#0.4167
dRop(topic(1),suche([], zeitung(_), zeit(_)))
#[topic(1):[topic(1)],suche([], zeitung(_), zeit(_))
:[suche,einen,Artikel],start:[Ich]]#[#0.6389
```

The parser has now reached a suitable parse or final state,

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

and looks in the statistical file to assign it a probability. The most suitable (similar) final state in the statistical file is 25, and equation 20 is used to compute similarity *P* between:

```
[suche([topic(1)],zeitung(standard),zeit(_)):
[suche,Informationen],start:[]]
```

and

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

which is 0.3333. Now equation 19

$$\text{ProbF} = (0.5 * 0.3333 + 0.5 * 0.5) = 0.4167 \quad (31)$$

Using equation 13 to compute *P*, we get:

$$\begin{aligned} &= (.5 * (.5125 * .2583 * .6 * .5042 * .4167 * .6389) + .5 * .4167) * 100 \\ &= (0.5 * 0.0107 + 0.5 * 0.4167) * 100 \\ &= (0.0053 + 0.2084) * 100 \\ &= (0.2137) * 100 \\ &= 21.37 \end{aligned}$$

the rating of this parse.

In the final stage, the parser uses equation 12 to collect the best parse *P* (and possibly discard those with an invalid semantics) and simply puts back the meaning for *topic(1)* collected during the *topic\_extraction* phase and the final parse is, without contextual information:

```
suche([(Bush)], zeitung(_), zeit(_))
```

<sup>7</sup>In the following computation, *Pop*, *Pfinal*, *Pop<sub>sim</sub>*, *Pop<sub>occ</sub>*, *Pfinal<sub>sim</sub>* and *Pfinal<sub>occ</sub>* are assigned the value 0.5.

## 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 previous approaches introduced in section 1. 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.

## 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, this system 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

- [Charniak *et al.*, 1993] E. Charniak, C. Hendrickson, C. Jacobson, and M. Perkowski. Equations for part-of-speech tagging. *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 784–789, 1993.
- [Collins and Miller, 1998] Michael Collins and Scott Miller. Semantic tagging using a probabilistic context free grammar. *In Proceedings of the Sixth Workshop on Very Large Corpora*, 1998.
- [Manning and Carpenter, 1997] C.D. Manning and B. Carpenter. Three generative, lexicalised models for statistical parsing. *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 147–158, 1997.
- [Miller *et al.*, 1996] Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. A fully statistical approach to natural language interfaces. *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 55–61, 1996.
- [Rabiner, 1989] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [Thompson *et al.*, 1997] Cynthia A. Thompson, Raymond J. Mooney, and Lappoon R.Tang. Learning to parse natural language database queries into logical form. *Proceedings of the ML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*, 1997.