# Sense and Deduction: The Power of Peewees Applied to the SENSEVAL-2 Swedish Lexical Sample Task

**Torbjörn Lager**† and **Natalia Zinovjeva**‡
†Department of Linguistics, Uppsala University
‡Hapax Information Systems AB, Stockholm

## Abstract

This paper describes our use of Prolog Word Experts (PWEs) in the SENSEVAL-2 competition. We explain how we specify our PWEs as sequences of transformation rules and how they can be trained on sense tagged corpus data. We give a semantics of PWEs by translating them into first order predicate logic, and we describe how PWEs can be compiled into Prolog procedures. We finally present our results for the Swedish lexical sample task: 63% (fine-grained score) for our best PWE, and a second place in the ranking.

## 1 Introduction

Word experts are small expert system-like modules for processing a particular target word based on neighboring words. Typically, a word expert uses rules that test the identity and relative position of words in the context in order to infer the role of the target word in the passage (Berleant, 1995). In this paper, we describe the development of various kinds of word experts in a logic programming framework, dealing with word sense disambiguation in the context of the SENSEVAL-2 competition.

In a logic programming framework, the task of engineering a word (sense) expert can be specified as follows. Given a suitable representation of a text, we want to define a predicate `sense/2` such that `sense(P,S)` is true iff the word at position `P` in the text has the sense `S`. In the remainder of the paper, we will refer to this kind of word expert as a Prolog Word Expert (or PWE for short – "Peewee" to its friends). This is to distinguish it from other kinds of word experts, and to emphasize the fact that it is 'programmed in logic'.

## 2 The Anatomy of a Peewee

### 2.1 Peewee Specifications

In the present paper, a word expert's knowledge will be expressed, not as Prolog clauses defining `sense/2` directly, but as a sequence of transformation rules. For example, here is how we specify a word expert which is able to disambiguate occurrences of *interest*:[1]

```
word_expert sense :=
  sense:add 6 <- word:interest@[0] o
  sense:6>1 <- word:in@[1] o
  sense:1>5 <- word:'%'@[-1] o
  end.
```

The first rule works as a default rule, which simply assigns the most frequent sense to the word *interest* (6 in this case). If no other rules apply, this is the tag that the word will eventually get. The other rules dictate when – based on the context – a word should have its tag changed. The second rule is to be read "replace the tag for sense 6 with the tag for sense 1, if the next word is *in*". The third rule says "replace the tag for sense 1 with the tag for sense 5, if the previous 'word' is '%'." The `o`-symbol is a composition operator, and `(R o Rs)` basically means that the output of applying the rule `R` forms the input to the application of the rules `Rs`. Thus, rules are strictly order-dependent. Note, for example, that the third rule is applicable only if the second rule is.

Needless to say, the above rules are not at all sufficient for the task of disambiguating all uses of *interest*. But the number of rules can be increased, and typically a word expert will

[1]This word was of course not used the Swedish task, but is used here for expository reasons. The sense tags are numbers: 1="readiness to give attention", 5="a company share", 6="money paid for the use of money", etc.

have access to anything between just a handful of rules and several hundred ones.[2]

### 2.2 Peewee Logic

Interestingly, a sequence of transformation rules can be translated into a set of axioms, expressed in first-order predicate logic, defining relationships between positions in a text, word forms, and senses (Lager, 2000; Lager & Nivre, 2001). For example, the meaning of the rules from the previous section can be spelled out as follows:

$$\forall p[w(p,interest) \rightarrow S_1(p,6)]$$

$$\forall p_0,p_1[S_1(p_0,6) \wedge p_1{=}p_0{+}1 \wedge w(p_1,in) \rightarrow S_2(p_0,1)]$$
$$\forall p_0,p_1,x[S_1(p_0,x) \wedge p_1{=}p_0{+}1 \wedge \neg w(p_1,in) \rightarrow S_2(p_0,x)]$$

$$\forall p_0,p_1[S_2(p_0,1) \wedge p_1{=}p_0{-}1 \wedge w(p_1,\%) \rightarrow S_3(p_0,5)]$$
$$\forall p_0,p_1,x[S_2(p_0,x) \wedge p_1{=}p_0{-}1 \wedge \neg w(p_1,\%) \rightarrow S_3(p_0,x)]$$

$$\forall x,p[S_3(p,x) \rightarrow S(p,x)]$$

The idea is that for each rule in the sequence a new predicate $S_i$ is introduced, where the subscript indicates where in the sequence the rule belongs. Semantically, $S_i$ relates a position to a sense, and the formulas define this predicate in terms of the predicate $S_{i-1}$ plus a number of other predicates. Each $S_i$ corresponding to a replacement rule is defined by two sentences – one stating the conditions under which a sense tag is replaced with another sense tag, the other one stating the conditions under which the old sense tag is kept.

Given a suitable logical representation of a text, such as

$$w(1,Sue)\ \ w(2,developed)\ \ w(3,an)\ \ w(4,interest)$$
$$w(5,in)\ \ w(6,computers)\ \ w(7,and)\ \ w(8,bought)$$
$$w(9,an)\ \ w(10,11.5)\ \ w(11,\%)\ \ w(12,interest)$$
$$w(13,in)\ \ w(14,Microsoft)$$

and given a suitable constructive proof method, the exact identity of the sense of an occurrence of the word *interest* – say the word at position 12 – will follow as a logical consequence of the theory formed by taking the union of the previous two sets of formulas. For example, the formula $\exists x[S(12,x)]$ is a theorem, for which we can construct (only) the example $x \rightarrow 5$, and we have thus formally proved that this particular occurrence of *interest* means "a share in a company".[3]

What we have here is something that we like to think of as *word sense disambiguation as deduction*, in analogy to the ideas of *parsing as deduction* due to Pereira and Warren (1983).

### 2.3 The Peewee Compiler

Since the above formulas have already logic programming form, it is straightforward to translate them into Prolog. For example, the second and the third formulas can be translated as follows:[4]

```
s2(P0,1) :- s1(P0,6), P1 is P0+1, w(P1,in).
s2(P0,X) :- s1(P0,X), P1 is P0+1, \+ w(P1,in).
```

To write Prolog procedures such as these by hand for many rules would be tedious and prone to errors. Fortunately, since the formalism for transformation rules is compositional, it was straightforward to write a compiler[5] that generates word expert procedures from word expert specifications automatically.

### 2.4 Peewee Training

There is an obvious choice of learning method for training Prolog Word Experts, namely Transformation-Based Learning (Brill, 1995). Of course, the fact that transformation rules can be learned from tagged corpora was a major reason for using them in the first place. The $\mu$-TBL system – described in detail in (Lager, 1999) – uses the search and database capabilities of the Prolog programming language to implement a generalized form of transformation-based learning. Through its support of a compositional rule/template formalism and 'pluggable' algorithms, the $\mu$-TBL system can easily be tailored to different learning tasks.[6]

Rules that can be learned in Transformation-Based Learning are instances of rule templates. For example, the second of the rules in our example PWE specification is an instance of the following template:

```
sense:A>B <- word:C@[1].
```

[2]A demo of a more potent PWE is available at: `http://www.ling.gu.se/~lager/Home/pwe_ui.html`

[3]The theory can be used in other ways too. *Searching* for a word token with a particular sense (say 5) becomes a matter of constructively proving $\exists p[S(p,5)]$.

[4]There are equivalent but more efficient ways to represent these clauses in Prolog (cf. Lager, 2000).

[5]Download the compiler from the PWE homepage at: `http://www.ling.gu.se/~lager/pwe.html`

[6]The $\mu$-TBL system is available from: `http://www.ling.gu.se/~lager/mutbl.html`

The template is to be read "replace the tag for sense A with the tag for sense B if the word immediately to the right is C", where A, B and C are variables. Learning is a matter of repeatedly instantiating rule templates in training data, scoring rules on the basis of counts of positive and negative evidence of them, selecting the highest scoring rule on the basis of this ranking, and applying it to the training data.

## 3 Peewees at SENSEVAL-2

The lexical sample task for Swedish in SENSEVAL-2 involved 40 lemmas: 20 nouns, 15 verbs and 5 adjectives. Together they represented 145 senses and 304 sub-senses. 8,718 annotated instances were provided as training material and 1,525 unannotated instances were provided for testing. Furthermore, a lexicon – the GLDB (Gothenburg Lexical Database) – complete with morphological information, definitions, language examples, etc. was available.

Our team explored three approaches. For each lemma, we trained:

- PWE-smpl: a simple PWE capable of arriving at a single sense for each instance of that lemma in the testing material.
- PWE-disj: a committee of PWEs (i.e. a set of PWEs) capable of arriving at (possibly) multiple senses for each instance of that lemma, by collecting the individual results into a set.
- PWE-vote: a committee of PWEs capable of arriving at a single sense for each instance of that lemma, by applying a simple voting procedure.

As it turned out, the second of these approaches produced a rather unimpressive result, and we will therefore spend very little time discussing it. Indeed, had we been able to run the scoring software ourselves (which we were not), we would have left them outside the competition altogether.

### 3.1 The Simple Peewees

For the training of our simplest form of sense disambiguation expert, the following set of seven templates was used:

```
sense:A>B <- word:C@[-1].
sense:A>B <- word:C@[-1,-2].
sense:A>B <- word:C@[1].
sense:A>B <- word:C@[1,2].
sense:A>B <- word:C@[1] & word:D@[2].
sense:A>B <- word:C@[-1] & word:D@[-2].
sense:A>B <- word:C@[-1] & word:D@[1].
```

The idea was to exploit a fact noted by many researchers in the field: that the sense of an occurrence of a word can fairly successfully be determined from just looking at the two previous words and the two following words (cf. Ide & Véronis, 1998). The choice of the above set of templates is based on a fairly thorough trail-and-error process and works well for most words that we have tried.

### 3.2 The Peewee Committees

The idea here was to train five different PWEs for each lemma, and then to use a simple voting mechanism to arrive at a final decision. The PWEs were different only in that they used different sets of templates during the training. Templates looking forwards only, templates looking backwards only, and templates looking both forwards and backwards. Furthermore, one member in each committee was trained for using a bag-of-words approach to disambiguation, based on templates of the following form:

```
sense:A>B <- inBag:W@[0].
sense:A>B <- inBag:W1@[0] & inBag:W2@[0].
```

Finally, one PWE in each committee had access to a list of words extracted from the language examples provided by the GLDB.

### 3.3 The Procedure

In this section we describe the actions that we took in order to submit our entry in the competition.

- In a preparatory step, the XML formatted training data was parsed and subsequently converted into the format required by the $\mu$-TBL system.
- The training was performed, and resulted in one PWE specification per lemma. Training took between 5 seconds and a couple of minutes per lemma, depending on the amount of training data available for the lemma in question.
- The PWE specifications were compiled into a set of PWE procedures, by means of the PWE compiler.

- Simple procedures were written to print the results to a file in the prescribed format, and the PWEs were then run on the test data. This took only a couple of seconds for the whole test corpus.

### 3.4 Results

In the following table we show the results of our entry in the competition, copied from the SENSEVAL-2 homepage.[7]

| System | Evaluation | Accuracy (%). |
|---|---|---|
| PWE-smpl | Fine | 61.1 |
| | Mixed | 66.8 |
| PWE-vote | Fine | 63.0 |
| | Mixed | 68.6 |

Five groups and altogether eight systems participated in the Swedish lexical sample task. In terms of ranking, our PWE-vote came in second, after Yarowski's JHU system, and before the Göteborg team's best entry. However, we hasten to add that the step from Yarowski's (nearly 70%, fine grained evaluation) to our results is a very significant 7%, and that the step down to Göteborg's result is very small and probably statistically insignificant. Our simple Peewees shared the fourth place with Resnik et al.'s UMD-SST.

As can be seen from the table, the PWE committees did slightly better than a single simple PWE. It is however dubious whether the small difference was really worth the trouble. It is quite possible that training a single PWE on the *combination* of corpus data and the examples from the GLDB would have lead to a result almost as good, and with less work.

## 4 Conclusion

It seems we can conclude that an approach to word sense disambiguation based on Transformation-Based Learning is competitive with approaches based on Memory-Based Learning as used by the Göteborg team, and support vector machine (SVM) learning, used by the University of Maryland team. This is good news for those aiming at building NLP systems in which transformation rules play a major role.

As we have seen, there is meaning in the life of Peewees, and sound mathematical meaning at that! Also, given the link between first order logic and a logic programming language such as Prolog, the implementation follows very directly from the specification. The existence of a compiler from Peewee specifications into Prolog procedures makes Peewees very convenient to work with in a Prolog environment.

[7]Note that the coarse-grained evaluation was not applicable to the Swedish task. Also, it should be noted that our results in the first round of evaluation were slightly worse than the results reported here. However, this was due to a spelling error which could be corrected by the conference organizers and thus did not involve any resubmission of test results.

## 5 Acknowledgements

We thank the SENSEVAL-2 organizers for making all this possible, and in particular Jerker Järborg and Dimitrios Kokkinakis in Göteborg for their work on preparing for the Swedish lexical sample task.

## References

Berleant, D. (1995) Engineering "Word Experts" for Word Disambiguation. *Natural Language Engineering*, 1(4).

Brill, E. (1995) Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics* 21.

Ide, N. and Véronis, J. (1998) Introduction to the Special Issue on Word Sense Disambiguation: The State of the Art. *Computational Linguistics* 24(1).

Lager, T. (1999) The $\mu$-TBL System: Logic Programming Tools for Transformation-Based Learning. In *Proceedings of CoNLL'99*, Bergen, Norway.

Lager, T. (2000) A Logic Programming Approach to Word Expert Engineering. In *Proceedings of ACIDCA 2000: Workshop on Corpora and Natural Language Processing*, Monastir, Tunisia, March 22-24 2000.

Lager, T. and Nivre, J. (2001) Part of Speech Tagging from a Logical Point of View. In de Groote, P., Morrill, G., Retor, C. (eds.) *Logical Aspects of Computational Linguistics.* Springer-Verlag, LNAI. VOL. 2099.

Pereira, F. and Warren, D. H. D. (1983) Parsing as Deduction, In *Proceedings of the 21th Meeting of the ACL.*