Inductive learning of lexical semantics with typed unification grammars

Dimitar Kazakov[1] and Simon Dobnik[2]*

[1] Department of Computer Science, University of York
Heslington, York YO10 5DD
`kazakov@cs.york.ac.uk`,
Web: `http://www.cs.york.ac.uk/~kazakov/`
[2] Computational linguistics, Oxford University
Centre for Linguistics and Philology
Walton Street, Oxford OX1 2HG
`simon.dobnik@clg.ox.ac.uk`,
Web: `http://www.clg.ox.ac.uk/people/sd.htm`

Abstract. In the last decade machine learning techniques based on logic such as Inductive Logic Programming (ILP) have started being used in learning grammars from corpora. While the first approaches were based on the translation of grammar into first-order predicate logic, an attempt has been made recently to adapt the ILP learning schema to the feature constraint logic of typed-unification grammars. In this framework, the learner applies in turn generalisation and specialisation to the typed feature structures representing the grammar in order to improve its coverage of the training corpus provided. In this paper we demonstrate how the lexical resource WordNet could be incorporated into an existing grammar learning tool and show how lexical semantic constraints could be learned on the basis of the ontology information.

1 Introduction

In the last decade machine learning techniques based on logic such as Inductive Logic Programming (ILP) (Muggleton and De Raedt, 1994) have started being used in learning grammars from corpora (Cussens and Džeroski, 2000). Ciortuz (2002) implements and documents a system known as *ilp*LIGHT which combines a logic-based learning approach with *typed feature structure unification grammars* to modify the initial grammar provided and adjust its coverage to the training corpus.

Typed Feature Structure Grammars (TFSG) present both grammar rules and lexical items as symbols or attribute-value matrices (which are also called feature structures), and thus lexical semantic information can be represented in an identical way, alongside the syntactic constraints. Furthermore, Feature Structures (FSs) are organised into hierarchies of *types* (represented as directed

* The second author's work has been funded by the EPSRC ROPA grant *Machine Learning of Natural Language in a Computational Logic Framework.*

acyclic graphs) where each type inherits the feature specifications of those types higher in the hierarchy. This is a very concise and convenient way of representing both lexical entries and grammar rules.

The present work complements and extends the work of Ciortuz (2002) by outlining a technique of incorporating lexical semantic information within the existing typed feature structure grammar and describes how to use his learning tool *ilp*LIGHT to induce lexical semantic generalisations for verbal predicates. The approach demonstrates that TFSGs are well suited for incorporating lexical information available in online ontologies such as WordNet since there the semantic information is organised in hierarchical structures that could be successfully incorporated within a FS hierarchy and referenced by the grammar rules. The approach also discusses the possibilities of reducing the ambiguity of grammars as additional lexical semantic constraints will make the parsing process (more) deterministic. While the present stage of research only offers a descriptive account, the future work will concentrate on its practical implementation within the *ilp*LIGHT system.

The article proceeds as follows: we start by giving a brief outline of the typed unification grammar formalism (Section 2) together with the details of the *ilp*LIGHT system (Section 3). We provide a short description of WordNet a large-scale resource of lexical semantic information (Section 4), and outline our proposal for interfacing the two (Section 6).

2 Typed feature structure grammars

The TFSG is a logic framework first studied by Carpenter (1992) and Smolka (1992), which is widely used in computational linguistics. Several formalisms have been developed: Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1982), Categorial Unification Grammar (Uszkoreit, 1986), and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994; Sag and Wasow, 1999). Finally, the Linguistic Knowledge Building (LKB) system described by Copestake (2002) is an open source TFSG development environment.

The linguistic objects (or *signs*) of TFSGs are represented as typed attribute value matrices (AVMs) or feature structures (FSs). For example, the simple grammar included with *ilp*LIGHT system includes the following information for the lexical item *girl* specified as a matrix of *attributes* or *features* and their values.[3]

```
girl:phrase_or_word
  [ PHON   <!''girl''!>,
    CAT    noun,
    SUBCAT <det> ]
```

One of the most important properties of the formalism is that it allows context-free grammar rules to be represented in an identical way: as *constraints*

[3] The following examples are taken from Ciortuz (2001a).

inside FSs in the form of indices (or *tags*) which indicate structure sharing. For example, *ilp*Light has been used with a FS which encodes the following HPSG rules: the Head Feature Principle, the Subcategorisation Principle, and the Saturation Principle. The Head Feature Principle specifies the category of the phrase be token-identical (thus not simply share the same feature specifications) with the category of the head. The Subcategorisation Principles specifies the first member of the `SUBCAT` list to be identical with the category of the complement phrase. Finally, the Saturation Principle specifies that the value of `COMP.SUBCAT` is `nil`.

```
satisfy_HPSG_principles
  [ PHON   diff_list,
    CAT    #1:categ,
    SUBCAT #2:categ_list,
    HEAD   #4:phrase_or_word
     [CAT    #1
      SUBCAT #3|#2 ],
    COMP   #5:phrase_or_word
     [CAT    #3,
      SUBCAT nil ],
    ARGS   <#4, #5> ]
```

The FSs are *typed* or *sorted* (the types of the above feature structures are *phrase_or_word* and *satisfy_HPSG_principles* respectively) by the grammar which means that the grammar contains specifications or *appropriateness conditions* about which features are appropriate for each typed feature structure or *type/sort* and what are the possible values of these features.

Types can be atomic or non-atomic. A non-atomic type is one that has more specific instances. These are represented in a hierarchical relation to a more general non-atomic type, as a directed acyclic graph. The grammar used by *ilp*Light is based on the sort hierarchy in Figure 1 (the origins of which can be traced back to Shieber (1986)), and is a simplified version of the HPSG sort hierarchy found in Sag and Wasow's book (1999), p.386.

A more specific type contains more information in terms of the number and the value of features and thus constraints that apply for more than one type need only to be stated once for a parent type and are then inherited by its descendants. Stating this differently: the types higher in the type hierarchy *subsume* those below. This property enables the grammar to be constructed in a very concise way (see Figure 3). For example, the sample type hierarchy specifies that the type *lH_phrase* is more specific than *satisfy_HPSG_principles*. It adds the following constraint:[4]

[4] The `!` notation is used to represent difference lists. The constraint specifies that the head of the phrase is its leftmost element. *rH_phrase* specifies the opposite and is required to represent subjects as complements of right-headed phrases.

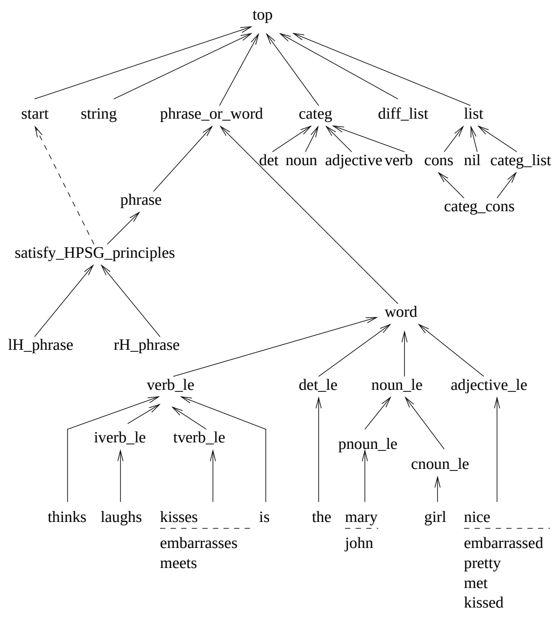

Fig. 1. A sample type/sort hierarchy

```
lH_phrase
[ PHON      #1!#3,
  HEAD.PHON #1!#2,
  COMP.PHON #2!#3 ]
```

Thus the knowledge specified in the type *lH_phrase* is as follows:

```
lH_phrase
[ PHON      #6!#8,
  CAT       #1:categ,
  SUBCAT    #2:categ_list,
  HEAD      #4:phrase_or_word
              [ PHON   #6!#7,
                CAT    #1,
                SUBCAT #3|#2 ],
  COMP      #5:phrase_or_word
              [ PHON   #7!#8,
                CAT    #3,
                SUBCAT nil ],
  ARGS      <#4, #5> ]
```

Additional types present in the grammar allow further expansion of the *lH_phrase* type until all the relevant features specified in the type hierarchy are present and their values are maximal or atomic types. Such feature structures are said to be *totally well typed* and *sort resolved* and as such are representations of linguistic signs. On the other hand, types can be partial and thus relating to a series of linguistic signs.

Finally, the grammar must allow two constraints (typed FSs) to be combined or *unified*. For example, the COMP feature in the *lH_phrase* must combine with the type *phrase_or_word* while satisfying certain constraints regarding the values of features CAT and SUBCAT. A unification of two feature structures F and F' is the greatest lower bound of F and F' in the collection of feature structures ordered by subsumption (Copestake, 2000). Thus, the result of the unification is the most general typed feature structure that is compatible with both input F and F' which is identical to the logical conjunction $F \wedge F'$. If no greatest lower bound of F and F' exists, i.e. the information specified in both feature structures is incompatible, the unification fails.

The *ilp*LIGHT system uses a somewhat different logic than typed feature structure theory (Carpenter, 1992; Copestake, 2000; Copestake, 2002). Instead, it is based on Order-Sorted Feature (OSF) logic common in Constraint Logic Programming (CLP) and elaborated by Aït-Kaci *et al.* (1994). These authors define the notion of OSF-theory unification on *order consistent* OSF-theories as more general than well-typed feature structure unification. Ciortuz (2001a; 2001b) complements the OSF notion of *order consistency* with *type consistency* and requires both conditions to be satisfied by the OSF theories used in his tool. The reader is referred to Ciortuz (2001b) for a detailed discussion on the scopes of each of these approaches and the benefits they bring in.

3 Inductive learning of typed unification grammars with *ilp*LIGHT

Inductive logic programming (ILP) (Muggleton and De Raedt, 1994) is a machine learning approach where the goal is to induce hypotheses starting from sample data (examples of one or more concepts) and background knowledge. In its most common form, ILP uses the representation formalism of logic programming, i.e., a subset of first-order logic.

Given two sets of positive examples E^+ and negative examples E^- and a background theory H, the following conditions have to hold for the hypothesis H:

$$\begin{aligned} &\textit{Prior Satisfiability} \quad B \wedge E^- \not\models \Box && (1)\\ &\textit{Posterior Satisfiability} \quad B \wedge H \wedge E^- \not\models \Box && (2)\\ &\textit{Prior Necessity} \quad B \not\models E^+ && (3)\\ &\textit{Posterior Sufficiency} \quad B \wedge H \models E^+ && (4) \end{aligned}$$

When 4 holds, H is said to be *complete*, and when 2—*consistent*.

The *ilp*LIGHT system (Ciortuz, 2002) consists of three components: the Expander, the ABC LIGHT[5] parser, which is a combination of a head corner parser (ABC), LIGHT, the OSF abstract machine for feature structure unification, and the *ilp* Learner component.

Note that the OSF formalism used by ILP Learner to represent the grammar is different from first order logic, traditionally used in ILP. In *ilp*LIGHT the inductive learning methods are adapted to typed unification grammars which are represented in the feature constraint logic. The architecture of the system is summarised in Figure 2.

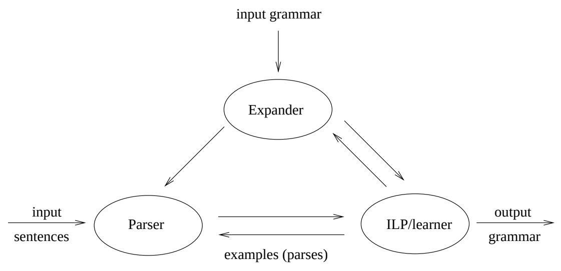

Fig. 2. The *ilp*LIGHT system for learning typed feature structure grammars

The framework can be used to process large-scale HPSG grammars such as LinGO (Copestake, Flickinger, and Sag, 1999) developed by the Center for Studies of Language and Information (CSLI) at the University of Stanford.[6] For the purposes of this article we will only be concerned with a small subset of this grammar described by Ciortuz (2002) which is given in Figure 3. The type hierarchy is as in Figure 1.

The grammar is fed to the Expander which expands the types with the features inherited from their ancestors in the type hierarchy. The expanded types are sent to the Parser which uses them to analyse the input sentences. Since the grammar may be incomplete, parsing may produce partial parses which are evaluated by the ILP learner module. This module suggests improved grammar rules that provide a better coverage of the input sentences, that is, to generate or succeed only on positive examples from the set of input sentences and to exclude the negative examples. Alternatively, the learning data may consist of a set of sentences which are only marked for the number of possible parses or of pairs of sentences and parses.

[5] ABC stands for Active Bottom-up Chart-based parser. LIGHT stands for Logic, Inheritance, Grammars, Heads and Types. It was developed at the Language Technology Lab of the German Research Centre for Artificial Intelligence (DFKI), Saarbrücken.

[6] Online resources are available at `http://lingo.stanford.edu`.

The parser and the expander operate bidirectionally as shown by the arrows. For example, the parser can take as input a parse[7] and then attempt to build a feature structure associated with that parse. If the process fails the grammar is incorrect and the information on failure is passed back to the learner in order to augment the grammar to accept the parse. The expander, on the other hand, can accept as an input an expanded type and a feature path inside this type and returns information on the unexpanded type which introduced that feature path/constraint.

Learning in the *ilp*LIGHT set up is implemented as search through the space of possible generalisations and specialisations of one of the elements of the OSF theory encoding the grammar. Generalisation can be achieved through (1) replacing a sort (unexpanded type) with a more general one, (2) removing an equation unifying two variables (indices) in the OSF type or (3) removing a feature from that type. Specialisation is based on four refinement operators, which are to a large extent complementary to the generalisation steps. One can (1) replace a sort with another, subsumed by the former, (2) introduce a new equation unifying two indices or (3) unfold a sort, i.e., replace it with the corresponding OSF type and propagate the relevant constraints through the root features of this type. The remaining, fourth refinement operator merges two independently produced refinements of the same term if they are unifiable. In the present implementation, the generalisation step considers all elements of the grammar in turn, whereas the specialisation step has to be provided with the grammar item that is to be refined (Ciortuz, 2002).

4 WordNet

WordNet is an on-line lexical database which contains various syntactic and semantic information for a large number of words and idioms. Originally developed for English (Miller et al., 1993), WordNet is also implemented for a number of other languages as EuroWordNet, BalkanWordNet, etc.[8] The central building element of WordNet is called a *synset*, or *lexicographer's entry.* A synset is a set of words or idioms which share a common meaning. For instance: {*(to) shut, (to) close*}. To simplify the internal representation, each synset is assigned a large integer used as a unique identifier. For instance, {*monetary resources, funds*} is Synset 109616555 in WordNet1.6. WordNet uses a set of rules and lists of exceptions to map word-forms to all relevant lexical entries. Figure 4 shows the word-form 'funds' which is recognised by WordNet as corresponding to two lexical entries, 'fund' and 'funds'. The lexical entry 'fund' appears in three synsets: {*store, fund*}, {*fund, monetary fund*}, and {*investment company, fund*}, respectively. The lexical entry 'funds' only appears in the synset {*monetary resource, funds*}. WordNet describes several semantic relations between synsets, such as

[7] Complete parses are of the form `(rH_phrase 0 2 (john 0 1 ("john" 0 1)) (laughs 1 2 ("laughs" 1 2)))`.

[8] Information on these resources is available at `http://www.globalwordnet.org`.

```
cons
[ FIRST top,
  REST  list ]

diff_list
[ FIRST_LIST list,
  REST_LIST  list ]

categ_cons
[ FIRST categ,
  REST categ_list ]

phrase_or_word
[ PHON   diff_list
  CAT    categ,
  SUBCAT categ_list ]

phrase
[ HEAD #1:phrase_or_word,
  COMP #2:phrase_or_word,
  ARGS <#1, #2> ]

satisfy_HPSG_principles
[CAT    #1,
 SUBCAT #2,
 HEAD   top
        [ CAT    #1,
          SUBCAT #3|#2 ],
 COMP   top
        [ CAT    #3,
          SUBCAT nil ] ]

det_le
[ CAT    det,
  SUBCAT nil ]

noun_le
[ CAT noun ]

pnoun_le
[ SUBCAT nil ]

cnoun_le
 [ SUBCAT <det> ]

adjective_le
[ CAT    adjective,
  SUBCAT nil ]
```

```
iverb_le
 [ CAT    verb,
   SUBCAT <noun> ]

tverb_le
 [ CAT    verb,
   SUBCAT <noun, noun> ]

lH_phrase
 [ PHON      #1!#3,
   HEAD.PHON #1!#2,
   COMP.PHON #2!#3 ]

rH_phrase
 [ PHON      #1!#3,
   HEAD.PHON #2!#3,
   COMP.PHON #1!#2 ]

the [ PHON <!''the''!> ]
girl [ PHON <!''girl''!> ]
john [ PHON <!''john''!> ]
mary [ PHON <!''mary''!> ]
nice [ PHON <!''nice''!> ]
fed [ PHON<!''fed''!> ]
pretty [ PHON <!''pretty''!> ]
met [ PHON <!''met''!> ]
kissed [ PHON <!''kissed!''> ]
is [ PHON   <!''is''!>,
     CAT    verb,
     SUBCAT <adjective, noun> ]
laughs [ PHON <!''laughs''!> ]
kisses [ PHON <!''kisses''!> ]
thinks [ PHON   <!''thinks''!>,
          CAT    verb,
          SUBCAT <verb, noun> ]
meets [ PHON <!''meets''!> ]
feeds [ PHON <!''feeds''!> ]
```

Fig. 3. A sample typed feature structure grammar

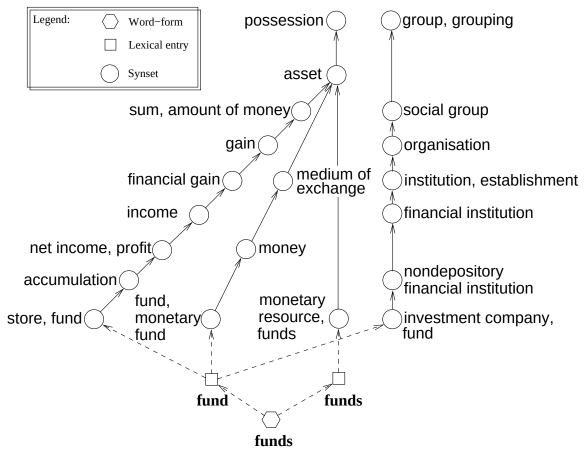

Fig. 4. Mapping from word-forms and lexical entries to synsets and their hypernyms in WordNet

meronymy (part-of), hypernymy or hyponymy. The latter are shown in Figure 4.

5 Related Work

There are a number of related approaches where grammars or the equivalent parsers have been automatically modified to best represent the properties of a corpus.

Learning Fast LR Parsers Samuelsson (1994) describes a technique for the development of an efficient LR parser based on Explanation-Based Learning (EBL) (Mitchell, 1997) and entropy-related information measures. The method is based on partial lexicalisation of grammar rules and expansion of RHS non-terminals. The new rules do not cover parts of the grammar, which are only marginally represented in the treebank. As a result, the grammar is less ambiguous at the price of a certain loss of coverage. Using that grammar also results in considerably faster parsing.

Zelle and Mooney (1993) have developed a method for learning semantic grammars from a treebank containing syntactic trees with semantically tagged non-terminal nodes. The treebank is used to construct an over-general shift-reduce parser covering the sentences in the treebank. The parser is then specialised and made deterministic by using ILP. When the semantic tags assigned to non-terminals are not sufficient to remove the ambiguity from the grammar, lexical semantic classes are automatically defined in order to achieve deterministic parsing.

Lapis (Kazakov, 1999) is a system which builds on Zelle and Mooney's research on the induction of shift-reduce parsers and extends it to learning LR parsers, while changing at the same time the focus of the learning task. The existence of sources of lexical semantic knowledge such as WordNet (Miller et al., 1993) makes the learning of lexical semantic classes done by Zelle and Mooney less attractive. Also, treebanks annotated with phrasal semantic tags are not commonly available. Instead, the system LAPIS constructs LR parsers from treebanks annotated with lexical semantic tags. LAPIS aims at the reduction of nondeterminism in the parsers it creates by the means of lexicalisation and partial unfolding of the underlying grammar rules, in combination with the use of lexical semantic constraints.

Cussens and Pulman (2000) combine ILP with a chart parser to find the missing rules in a grammar that would allow parsing all training sentences provided. When a sentence cannot be parsed, the parser suggests 'needed edges', i.e., those necessary to complete a parse. Examples of these are stored and used to learn a more general pattern corresponding to a grammar rule; additional linguistic constraints may be provided to prune the search for new grammar rules and ensure their plausibility.

6 Learning lexical semantics

This section describes methods of incorporating lexical semantic information from WordNet into type hierarchy and grammar rules.

6.1 Extending the type hierarchy

A type hierarchy is a finite bounded complete partial order (Copestake, 2000). This is also the case for the WordNet hierarchy of synsets, if a dummy bottom ($\perp$) synset that is subsumed by all others is added. Figure 5 represents a simplified WordNet hierarchy for some common and proper nouns.

Remember that WordNet distinguishes between words and lexical entries. Words are strictly not a part of the hierarchy but there exists a function which maps them to the hierarchy members. To distinguish the two we represent words below a dashed line.

The type hierarchy in Figure 1 and the WordNet hierarchy have to be merged so that the types are properly expanded and the lexical information is properly propagated within the same framework.

The lexical information in Figure 5 further restricts the categorial information for a given class of grammatical objects, nouns in our case. For example, most generally, the lexical item 'book' belongs to the category of *nouns*, but also to a number of its lexical sub-categories: it is an *entity*, an *object*, an *artefact*, and a *creation*. These semantic classes are gradual refinements of the two extremes: the grammatical category and the lexical item itself. Thus, the simplest solution to incorporate the lexical hierarchy into the existing type hierarchy is to embed

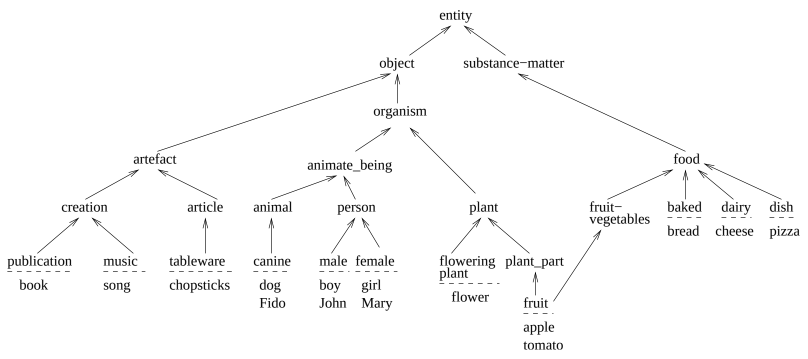

Fig. 5. A simplified WordNet hierarchy of synsets

the lexical *top* node *entity* in Figure 5 underneath the *noun* node of Figure 1. The *noun* node and thus all its daughters are now subsumed by the *categ* node (which is subsumed by the *top* node). This means that the system will now not only allow to infer that a certain lexical item is a noun but also that it is a 'creation-kind-of-noun'.

6.2 Adding lexical semantic types to the grammar

The *categ* type and the types that it subsumes do not introduce any new features that would interact with the existing grammatical constraints. Examining the sample grammar in Figure 3 the categorial information on the noun *word*s is introduced in the type *noun_le*. However, restricting this category with a more specific subtype such as *organism* is of no use since this would mean that our grammar would only be able to deal with 'organism' noun phrases. In our grammar, lexical information is as specific as phonological information and thus must be restricted at the same level (which is also the lowest level). Thus, the 'lexical types' in Figure 3 will be extended to the following unexpanded representation:

```
girl [ PHON <!''girl''!>,
       CAT  female ]

flower [ PHON <!''flower!''!>,
         CAT  flowering-plant ]
```

Note that now the specification of the feature `CAT` in the type *noun_le* is not necessary and can be removed. Its presence does not affect type expansion since a unification of `CAT noun` and `CAT flowering-plant` will always result in `CAT flowering-plant`.

6.3 Lexically enriched grammar: what can be done with it?

We shall discuss some of the possible benefits of introducing a lexically enriched grammar as shown in the examples below.

Learning the semantic restrictions of verbal predicates. Allowing a sufficiently large set of positive and negative data and a sufficiently fine-grained hierarchy of lexical relations it is theoretically possible to use *ilp*LIGHT to learn the lexical semantic restrictions of individual verbal predicates. Presently, the verbal predicates of the grammar in Figure 3 are only restricted in the most general way: they are specified for categorial information:

```
kisses [ PHON <!''kisses''!>,
         CAT  verb,
         SUBCAT <noun, noun> ]
```

Let our training corpus for *ilp*LIGHT consist of the following sentences.

```
John kissed Mary.
Fido kissed John.
The girl kissed the boy.
*Mary kissed the flower.
*The book kissed the castle.
*Mary kissed the book.
```

It is possible for the learner component of *ilp*LIGHT to find that the verbal predicate *kisses* requires both arguments to belong to the lexical class *animate being* through a number of specialisation steps.

```
kisses
   [ PHON <!''kisses''!>,
     CAT verb,
     SUBCAT <animate_being, animate_being> ]
```

Lexical semantic ambiguity Words that correspond to more than one meaning (synset) can be handled in this framework by adding separate lexical entries for each of their meanings to the grammar. Similarly, separate entries may have to be learned to cover the various semantic roles of a word where the grammar originally contained one lexical entry only specifying syntactic categorial information. To our knowledge, *ilp*LIGHT cannot handle this without a further modification.

Reducing ambiguity in parsing Lexically enriched grammars can be used to reduce non-determinism in the output of parsers. The grammar in Figure 3 does not distinguish between *complements* of verb and noun phrases and *adjuncts* which here are optional phrases modifying the verb phrase as a whole. The following is a typical set of sentences that introduces parsing ambiguity to any grammar which distinguishes the two.

```
John eats pizza with cheese.
John eats pizza with chopsticks.
```

The prepositional phrase *with cheese* is a complement of the noun *pizza* (it further specifies the pizza)[9] and thus should be analysed as its argument. On the other hand, *with chopsticks* is an adjunct of the entire verb phrase *eats pizza*. Because both phrases belong to the same grammatical category the parser would give two parses for each sentence, but only one of which is correct. The choice of the parse depends on the lexical semantics of the complement noun inside the prepositional phrase and important generalisations can be made to reduce the ambiguity. If the noun belongs to the lexical semantic class *food* the entire prepositional phrase is unambiguously analysed as the complement of the noun *pizza* which also belongs to this lexical category (or is subsumed by this type), otherwise the prepositional phrase is an adjunct of the verb phrase. In this case it is an *instrument* further specifying the pizza eating event.

Before setting up a learning task, the grammar in Figure 3 needs a modification since it does not recognise prepositional phrases, nor does it distinguish between complements and adjuncts.

Adding a new grammatical category is straightforward: a new node/type such as *prep* can be added to the type hierarchy so that it is subsumed by the *categ* node. The category of prepositions takes a noun phrase as its complement. We add this constraint in the form of a *prep_le* type subsumed by the *word* node.

```
prep_le
  [ CAT    prep,
    SUBCAT <noun> ]
```

To accommodate noun phrases with prepositional complements we create a new type *cnoun-compl_le* which is subsumed by the *noun_le* node. The *cnoun-compl_le* adds the following constraints:

```
cnoun-comp_le
 [ SUBCAT <prep> ]
```

We treat prepositional adjuncts of verbs as parts of their subcategorisation frame.[10] We add another type *tverb-pp_le* which is subsumed by the *verb_le* node and is specified for the following features:

```
tverb-pp_le
 [ CAT verb,
   SUBCAT <noun, prep, noun> ]
```

The learning task here is significantly more complex than the previous one since here the learner is trying to specialise phrasal templates rather than words.

[9] Notionally there is a very strong distinction between complements and adjuncts, yet no formal definition exists.

[10] This is the standard HPSG treatment (Sag and Wasow, 1999).

For example, to learn that a prepositional phrase is a complement of the noun rather than the verb phrase (in which case it is an adjunct), a relation must be established between the head noun (*pizza*) and the complement (*cheese*) of its complement (*with*) which leads to a specialisation of the *categ* value of *pizza* and *cheese* from *noun* to *food*. In the present design of the grammar, which distinguishes between words and phrases, words only contain information stated in the `SUBCAT` feature on the category of their *immediate* complements but not their internal structures. For example, an expanded type that constrains nouns that take prepositional complements *cnoun-comp_le* is as follows:

```
cnoun-comp_le
 [ PHON   diff_list,
   CAT    noun,
   SUBCAT <prep> ]
```

Therefore, the specialisation must be done at the level of expanded phrases *lH_phrase* and *rH_phrase* where *cnoun-comp_le* is the head of the phrase.[11] The learner should look for phrasal templates such as the following:

```
food-with-food
 [ PHON      #8!#10,
   CAT       #1:food,
   SUBCAT    #2:nil,
   HEAD      #4:cnoun-comp_le
              [ PHON   #8!#9,
                CAT    #1:food,
                SUBCAT #3:with|#2:nil ],
   COMP      #5:prep_le
              [ PHON   #9!#10,
                CAT    #3:with,
                SUBCAT nil
                COMP   #6:cnoun_le,
                         [ CAT #7:food ]],
   ARGS      <#4, #5> ]
```

The template is a specialisation of the *lH_phrase*. It adds the following constraints to the expanded *lH_phrase*:

```
food-with-food
 [ HEAD cnoun-comp_le
        [ CAT #1:food ],
   COMP prep_le
```

[11] Another apparent solution would be to merge the types *phrase_or_word* and *phrase* into a single type. This would result in a grammar which only accounts for phrases which is incorrect. Nouns such as *cheese* in *pizza with cheese* would be left unaccounted for.

```
        [ CAT  with,
          COMP cnoun_le,
            [ CAT #1:food ]]
```

It follows that this new specialised type, a partially unfolded and lexicalised template, is added to the type hierarchy so that it is subsumed by the *lH_phrase*.

A similar specialisation is needed for the case where the prepositional phrase is an adjunct.

```
verb-food-with-tableware
 [ PHON   #11!#13,
   CAT    #1:verb,
   SUBCAT #2:noun,
   HEAD   #4:tverb-pp_le
           [ PHON   #11!#12,
             CAT    #1:verb,
             SUBCAT #3:prep|#2:noun,
             HEAD   #6: tverb-pp_le
                      [ CAT #1:verb,
                        SUBCAT #7:food|#8:prep,noun,
                        COMP #9:cnoun_le,
                          [ CAT #7:food ]],
   COMP  #5:prep_le
          [ PHON #12!#13,
            CAT  #3:with,
            COMP #10:cnoun_le
                   [ CAT tableware ]]]
   ARGS  <#4,#5>
```

The previous examples show two implications of this approach for the learning procedures. Firstly, due to an increased level of depth at which the search operates (`HEAD.HEAD.COMP.CAT food`) the search space or the time to find a solution will be considerably increased. Secondly, learning does not only specialise or generalise over the existing types but creates new types which must be added to the type hierarchy.

7 Discussion and Future Work

We have described how WordNet, a standard resource of lexical semantic information, could be incorporated in a TFSG without any extension of the formalism used, and how an existing inductive grammar learner may use this information to add semantic constraints to a grammar in order to optimise its coverage of a training corpus. Further work should focus on implementing the approach outlined here in *ilp*LIGHT and testing its performance, and independently, on extending the initial grammar to include additional linguistic phenomena. In a study combining these two dimensions, the issue of pruning the search space of the ILP learner will become central. Here the research should concentrate on a systematic enumeration of the range of possible modifications of the grammar and the elimination of those constraints that are linguistically implausible, cf. (Cussens and Pulman, 2000).

References

Aït-Kaci, Hassan, Andreas Podelski, and Seth Copen Goldstein. 1994. Order-sorted feature theory unification. *The journal of logic programming*, 19(20):1–25.

Carpenter, Bob. 1992. *The logic of typed feature structures: with applications to unification grammars, logic programs and constraint resolution.* Number 32 in Cambridge tracts in theoretical computer science. Cambridge University Press.

Ciortuz, Liviu. 2001a. Expanding feature-based constraint grammars: Experience on a large-scale HPSG grammar for English. In *Proc. of the Workshop on modelling and solving problems with constraints*, Seattle, USA. `www.lirmm.fr/~bessiere/proc_wsijcai01.html`.

Ciortuz, Liviu. 2001b. Light AM - another abstract machine for FS unification. In Stephan Oepen, Daniel Flickinger, Jun-Chi Tsujii, and Hans Uszkoreit, editors, *Efficiency in Unification-Based Processing.* CSLI Publications, Stanford, pages 1–27.

Ciortuz, Liviu. 2002. Towards inductive learning of typed-unification grammars. In the (electronic) *Proceedings of the 17th Workshop on Logic Programming.* Dresden Technical University, Germany, 11–13 December.

Copestake, Ann. 2000. Definitions of typed feature structures. *Natural Language Engineering*, 1(6). Appendix to special issue on efficient processing with HPSG.

Copestake, Ann. 2002. *Implementing typed feature structure grammar.* Number 110 in CSLI Lecture notes. CSLI Publications, Stanford.

Copestake, Ann, Daniel Flickinger, and Ivan Sag. 1999. *A grammar of English in HPSG: Design and implementation.* CSLI Publications, Stanford.

Cussens, James and Sašo Džeroski, editors. 2000. *Learning Language in Logic.* Lecture Notes in Artificial Intelligence. Springer-Verlag.

Cussens, James and Stephen Pulman. 2000. Experiments in inductive chart parsing. In James Cussens and Sašo Džeroski, editors, *Learning Language in Logic*, Lecture Notes in Artificial Intelligence. Springer-Verlag.

Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The mental representation of grammatical relations.* Cambridge, Mass.: MIT Press, pages 173–381.

Kazakov, Dimitar. 1999. Combining LAPIS and sc wordnet for the learning of LR parsers with optimal semantic constraints. In Sašo Džeroski and Peter Flach, editors, *9th International Workshop on Inductive Logic Programming ILP-99*, number 1634 in Lecture Notes in Artificial Intelligence, pages 140–151, Bled, Slovenia. Springer-Verlag.

Miller, George A., Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. 1993. Introduction to WordNet: An on-line lexical database. Technical report, University of Princeton. `ftp://ftp.cogsci.princeton.edu/wordnet`.

Mitchell, Tom M. 1997. *Machine Learning*. McGraw-Hill.

Muggleton, S. and L. De Raedt. 1994. Inductive logic programming. Theory and methods. *Journal of Logic Programming*, 19(20):629–679.

Pollard, Carl and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Chicago: Chicago University Press.

Sag, Ivan A. and Tom Wasow. 1999. *Syntactic theory: a formal introduction*. Number 92 in CSLI Lecture notes. CSLI Publications, Stanford.

Samuelsson, Ch. 1994. *Fast Natural–Language Parsing Using Explanation–Based Learning*. Ph.D. thesis, The Royal Institute of Technology and Stockholm University.

Shieber, Stuart M. 1986. *An introduction to unification-based approaches to grammar*. Number 4 in CSLI Lecture notes. CSLI Publications, Stanford.

Smolka, Gert. 1992. Feature-constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87.

Uszkoreit, Hans. 1986. Categorial unification grammar. In *International conference on computational linguistics (COLING-92)*, pages 440–446, Nantes, France.

Zelle, John M. and Raymond J. Mooney. 1993. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of AAAI-93*, pages 817–822. AAI Press/MIT Press.