

Inductive learning of attribute path values in typed-unification grammars

Liviu Ciortuz*

Department of Computer Science, University of Iasi, Romania
E-mail: ciortuz@infoiasi.ro

Abstract. This work is aiming to show that inductive logic programming (ILP) is a suitable tool to learn linguistic phenomena in typed-unification grammars, a class of attribute-value grammars increasingly used in natural language processing (NLP).

We present a strategy for generating hypothesis spaces for either generalisation or specialisation of attribute-path values inside type definitions. This strategy is the core of a prototype module, extending the LIGHT system for parsing with typed-unification grammars.

1 Introduction

The work on *typed-unification grammars* can be traced back to the seminal paper on the PATR-II system [18]. Basically, the design of such grammars is underlined by two simple ideas: *i.* context-free rules may be augmented with constraints, generalising the grammar symbols to attribute-value matrices (also called feature structures); *ii.* feature structures (FSS) may be organised into hierarchies of *types*, a very convenient way to describe in a concise manner classes of rules and lexical entries.

Different *logical perspectives* on such a grammar design were largely studied; see [4] for a survey. For the LIGHT parsing system [7] we adopted the Order-Sorted Feature (OSF) constraint logic framework [2]. In this way, typed-unification grammars may be seen as a generalisation of Definite Clause Grammars (DCG, [15]), similar to the way in which LOGIN generalised Prolog [1].²

* This paper was published in The Scientific Annals of the “Al.I. Cuza” University of Iasi, Romania, Computer Science Series, 2003, pages 105–125.

² As a matter of terminology, the ψ -terms in LOGIN, or OSF-terms as they were renamed in [2], correspond to Feature Structures in the Natural Language Processing (NLP) and Computational Linguistics (CL) literature.

OSF-terms generalise first-order terms in the following respects:

- i.* constant symbols (‘sorts’) are organised into a hierarchy, assumed to be an inferior semi-lattice;
- ii.* the fix order of subterms is replaced by identification via attributes/features;
- iii.* variables/tags can identify not merely leaf nodes into the tree representing a term, but whole sub-trees.

Unification on such terms is either OSF-unification or OSF-theory unification, the latter being known as ‘typed-unification’ to NLP/CL people.

The *aim* of our work is to explore the usefulness of a logic-based learning approach — namely Inductive Logic Programming (ILP [14]) — to improve the coverage of a given typed-unification grammar: we either generalise some rule or lexical type feature structures (FSs) in the grammar so to make it accept a given sentence/parse tree, or try to specialise a certain type in the grammar so to reject a (wrongly accepted) parse.

Among several *related approaches* to ILP-based learning of unification grammars, we mention first the work by Cussens and Pulman [9]. By generalising over ‘examples’ of chart-based parse actions which can build full parses starting from partial ones, they ‘invent’ new rules to be added to the grammar. Somehow complementary to theirs, our approach proposes new versions of existing rules, either through generalisation or specialisation. Moreover, our approach is not limited to rules, being capable of affecting any type FS in the hierarchy of types which defines the grammar. Concerning the conception, while the previously mentioned approach is closer to the parsing, we are closer to the logic underlying the grammar. An earlier approach [21] proposed learning grammars from treebanks (set of sentences with associated parses) and performed ILP-based lexical inferences in order to make the learned grammar act deterministically.

The *organisation* of this paper proceeds somehow top-down: Section 2 outlines our approach for ILP-based strategy for learning attribute path values in typed-unification grammars. Section 3 firstly presents the two procedures for learning generalisations and respectively specialisations of type FSs, and secondly formalises the *refinement operators* used by these learning procedures. Sections 4 and 5 go into low-level details to exemplify how the two learning procedures work on a simple but significant HPSG grammar, which will be given in Appendix. Section 6 briefly presents the strategies we designed to scale up the present learning approach to a large HPSG grammar for English.

The present paper is a revised and extended version of [8].

2 Overview

The *main idea* of ILP-based learning is to generate (and then evaluate) new clauses starting from those defined in the input program/grammar. The acceptance/rejection of new ‘hypotheses’ is done using a set of examples and an evaluation function, to rate the inferred hypotheses. Learning a new clause in the case of typed-unification grammars amounts to the creation of new type FSs and this is done either by generalisation or specialisation. Generalisation either relaxes or

We have to *note* that in lexicalized large-scale typed-unification grammars, due to the large number of exceptions in human languages, NLP/CL people prefer doing parsing in a bottom-up manner (rather than top-down as Prolog and DCG do).

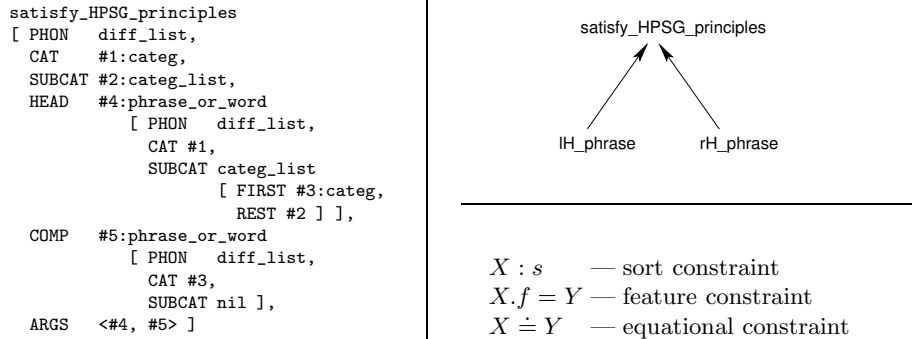


Fig. 1. A sample type feature structure, a simple sort/type hierarchy, and the atomic OSF-constraints for the logical description of feature structures.

removes one or more atomic OSF-constraints, while specialisation adds new such constraints to a type FS.

To illustrate a *type feature structure*, Figure 1 presents `satisfy_HPSG_principles`, the parent of the two (binary) rules used in a simple Head-driven Phrase Structure Grammar (HPSG, [16]) appearing in [19]. The `satisfy_HPSG_principles` type encodes three HPSG principles: the Head Feature Principle, the Subcategorization Principle, and the Saturation Principle.³ The knowledge embodied in the `satisfy_HPSG_principles` will be inherited into two rules: `IH_phrase` and `rH_phrase`.^{4 5}

The *architecture* of the *ilp*LIGHT prototype system we implemented for learning attribute path values in typed-unification grammars is designed in Figure 2. The initial grammar is submitted to an Expander module, which propagates the appropriate constraints down into the type hierarchy. The Parser module uses the expanded form of (syntactic) rules and lexical descriptions to analyse input sen-

³ The feature constraint encoding of the three principles is respectively:

$$\begin{aligned}
\Psi.CAT &\doteq \Psi.HEAD.CAT, \\
\Psi.HEAD.SUBCAT &\doteq \Psi.COMP.CAT \mid \Psi.SUBCAT, \\
\Psi.COMP.SUBCAT &\doteq \text{nil}.
\end{aligned}$$

where $\Psi = \text{satisfy_HPSG_principles}$, as given in Figure 1.

⁴ One can easily imagine classical `np` and `vp` rules as derived from `rH_phrase` and respectively `IH_phrase`.

⁵ Mainly, the constraints specific to these rules, as shown in Appendix, impose that the *head* argument is on the left, respectively the right position inside a phrase which represents a particular value of the `PHON` feature. Hence the names of the two rules, standing respectively for left-headed phrase and right-headed phrase. The constraints imposed on the `SUBCAT` values — `cons` and respectively `nil` — for these two rules are meant to rule out phrases like “girl the” and “nice is” which otherwise would be accepted by the grammar.

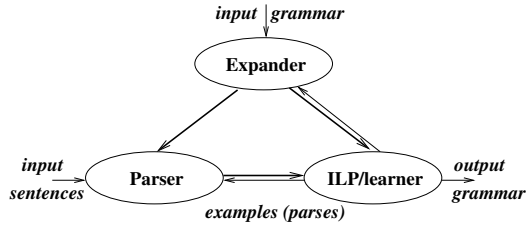


Fig. 2. The *ilp*LIGHT architecture for learning typed-unification grammars.

tences. The ILP/learner module receives (maybe partial) parses produced by the Parser, and — by calling one of two hypothesis generation procedures — it infers either more specific or more general type descriptions for the considered grammar, such that the new grammar will provide a better coverage of the given sample sentences.

The bidirectional arrows in the diagram in Figure 2 are due to the (double) functionality of the Parser and Expander modules. When asked to act in ‘reverse’ mode, the Parser takes as input a parse and tries to build up the FS associated to that parse. If the construction fails, then the LIGHT’s tracer component is able to deliver an ‘explanation’ for the failure.⁶ This failure ‘explanation’ will be analysed by the ILP/learner module to propose ‘fixes’ to the grammar so that parse get accepted. The Expander can also work in ‘reverse’ mode: given as input a type and an atomic constraint contained in the description of that type, the expander will indicate which type in the input (unexpanded) grammar is responsible for the introduction of that constraint in the expanded form of the designated type.

We conducted a series of *experiments* during which the *ilp*LIGHT prototype system was able to learn by specialisation each of the three HPSG principles, if the other two were present and a (rather small) set of sample sentences was provided. Equally, the system learned by specialisation lexical descriptions, and was able to recover (by generalisation) the definition of the HPSG principles if they were provided in an over-restricted form. In all cases mentioned above, learning took place in a few minutes amount of time.⁷

⁶ This explanation assembles informations on: *i.* the parse operation at which the ‘derivation’ of the FS associated to the input parse was blocked; *ii.* the feature path along which the last (tried) unification failed; *iii.* the (atomic) unification steps which lead to the sort clash causing the unification failure.

⁷ For instance, learning the Subcategorization Principle as presented in section 4 took 4min. 25 sec. on a Pentium III PC at 933MHz running Linux Red Hat 7.1, using the sample HPSG grammar in [18] and a set of 14 training sentences. Generalising the over-constrained form of `satisfy.HPSG.principles` presented in section 5 took 2min. 23sec.

We suggest that this speed can be substantially improved in a system which avoids recompilation of the grammar (despite the fact that the compilation speeds up parsing).

3 Learning attribute values

3.1 Algorithms

Two *procedures*, which are in charge with grammar generalisation, respectively type specialisation, build up our strategy for learning typed-unification grammars. These procedures are presented in detail in Figure 3. The *key idea* behind the two procedures is simple: given a type FS, we generalise and respectively specialise it by removing/relaxing and respectively adding one or more atomic OSF-constraints inside the given FS.

The *operators* for creation/deletion of atomic OSF-constraints invoked by the two procedures will be given in detail in section 3.2, Figure 5 and 7. The names of these operators — *relax-sort*, *remove* (feature or equation constraint), *equation introduction*, *sort-specialisation*, *type-unfolding*, *hypothesis-combination*, — are enough intuitive to let the reader go through the text of this subsection, without needing to refer here to low-level technical details.

A *design difference* between the two procedures: The generalisation procedure picks up automatically — by running the parser in ‘reverse’ mode, taking as input a parse which is (wrongly) rejected by the grammar — which type FS it has to generalise. Instead, the specialisation procedure, in the form given in Figure 3, requires to be specified (as part its input) which are the type and the path (inside that type) from which it has to start building sub-type specialisations.

Otherwise said, the specialisation procedure works only on a single type FS, inferring different FSs which are subsumed by the value of the specified path (inside that type). But once called, it never looks for another type (to specialise). Instead, the generalisation procedure may proceed with identifying different types that can make the grammar accept the (wrongly) rejected parse/sentence.

Note that if generalisation finds a hypothesis which reaches this goal, but the new form of the grammar over-parses some of the given sample sentences, then the generalisation procedure may call the specialisation procedure to refine that hypothesis/type.⁸

Both procedures work iteratively: the main operations are performed inside a while loop. At each iteration, a set of new hypotheses is added to the search space. The new hypotheses are evaluated one by one, and then the algorithm decides either to stop or to further search for other, better hypotheses.

For both generalisation and specialisation procedures, the acceptance or rejection of a hypothesis is based — as inspired by ILP — on comparing the parsing results with a given *annotation* on the set of sample sentences. In the current prototype of the *ilpLIGHT* system, we annotate each sentence with (the number of) its correct parses. Any other parse will be considered a negative example.

⁸ Of course, the specialisation procedure may also proceed independently, as in fact will be exemplified in the next section.

Generating typed-unification *grammar generalisations*

Input: a grammar \mathcal{G} already expanded (all types being normalised),
 $E = E_+ \cup E_-$ – a set of sample grammatical (E_+)
and un-grammatical (E_-) sentences, and
a parse δ for a grammatical sentence rejected by \mathcal{G} .

Output: \mathcal{G}' , more general grammar than \mathcal{G} (logically: $\mathcal{G} \models \mathcal{G}'$), such that \mathcal{G}' accepts δ .

Procedure (Generalisation):

1. $\mathcal{G}' := \mathcal{G}$;
2. While δ is not accepted by \mathcal{G}' do:
 - a. Find a failing unification in δ – *unify*(φ, ψ) – and an associated failure path π .
 - b. Eliminate the sort clash $\varphi.\pi \wedge \psi.\pi = \perp$ by using the rule

$$\frac{X : s \ \& \ Y : s' \ \& \ X \doteq Y \ \& \ s \wedge s' = \perp}{\text{relax-sort}(X, LUB(s, s')) \ \underline{\text{or}} \ \text{relax-sort}(Y, LUB(s, s')) \ \underline{\text{or}} \ \text{remove}(X \doteq Y)}$$

- c. Take \mathcal{G}' one of the grammar versions produced by the *relax-sort* and *remove*.
3. For the pairs (Ψ, π) of ‘faulting’ types Ψ and failure paths π identified successively for \mathcal{G}' at the point 2.a, in case of over-generation run the *specialisation* procedure (see below).

Generating a *type specialisation hypothesis space*

Input: a grammar $\mathcal{G} = \{\Psi(s)\}_{s \in \mathcal{S}}$ already expanded (and all types being normalised),
a type $\Psi(t)$ (with $t \in \mathcal{S}$), and a feature path π in $\Psi(t)$;
 $E = E_+ \cup E_-$ – a set of sample sentences.

Output: a rooted direct acyclic graph Γ of *specialisation hypotheses* (ψ, ρ) :
 $\psi \sqsubseteq \Psi(t)$, with π a prefix of ρ , and
 (ψ_1, ρ_1) precedes (ψ_2, ρ_2) in Γ iff $\psi_1 \sqsubseteq \psi_2$ and ρ_1 is a prefix of ρ_2 .

Procedure (Specialisation):

```

 $(\psi, \rho) := (\Psi(t), \pi)$ ;    % initialise (the root hypothesis of)  $\Gamma$ 
while  $(\psi, \rho)$  is not a convenient hypothesis    % cf. expand + parse + evaluate
  and the specialisation hypothesis space was not yet exhaustively searched
{
  generate specialisation hypotheses for  $(\psi, \rho)$  by
    equation introduction, sort specialisation, type unfolding, or
    hypothesis combination;
  take  $(\psi, \rho)$  the next not-yet-analysed hypothesis
  % (“next”: in the sense of the above stated “precedence” relation)
}
```

Fig. 3. Procedures for *hypothesis space generation* in inductive learning of attribute-path values in typed-unification grammars.

$relax-sort(X, s) \equiv$
 if $X = \Psi.\pi$, and $X : _$ is a genuine constraint, then
 replace the original sort constraint on $\Psi.\pi$ with $\Psi.\pi : s$
 else (i.e. $X : (s_1 \wedge s_2)$ using *sort refinement*)
 $(relax-sort(X, s) \text{ \underline{and} } relax-sort(Y, s)) \text{ \underline{or} } remove(X \doteq Y)$.

$remove(Y.f = Z) \equiv$
 if $Y.f = Z$ is a genuine constraint ($Y = \Psi.\pi$), then
 delete it from Ψ ;
 else, (i.e. if $remove(Y.f = Z)$ comes as result of feature carrying)
 $remove(X \doteq Y) \text{ \underline{or} } remove(X.f = U) \text{ \underline{or} } remove(Y.f = V)$,
 where $X \doteq Y$, $X.f = U$ were used in the premise of *feature carrying*;

$remove(U \doteq V) \equiv$
 if $U \doteq V$ is a genuine constraint ($U = \Psi.\pi$, $V = \Psi.\pi'$, $U \neq V$), then
 delete it from Ψ (either a rule, or a type (used in type-checking)), and
 propagate the *dis-equation* $U \neq V$ in Ψ ;
 else, i.e. $U \doteq V$ was synthesised by (combined) application of *feature carrying* and *coreferencing variables*,
 $remove(X \doteq Y) \text{ \underline{or} } remove(X.f = U) \text{ \underline{or} } remove(Y.f = V)$,
 where $X \doteq Y$, $X.f = U$ were used in the premise of *feature carrying*,
 and
 $Y.f = V$ in *coreferencing variables*.

Fig. 5. The (basic) operators used in generalising typed-unification grammars.

learning strategy is detailed in Appendix . This grammar is capable of analysing sentences like:

Mary laughs.
John meets Mary.
John embarrasses the girl.
The girl is nice.
John thinks Mary is nice.
Mary thinks John thinks Mary is pretty.
Mary thinks John embarrasses the girl.
John thinks Mary thinks the girl thinks John is embarrassed.

4 Learning generalisations of attribute values: exemplification

This section will exemplify how to learn/generate grammar generalisations using the first procedure in Figure 3.

relax-sort(X, s):
 if $X : _$ comes with type-checking, then
 eliminate (that) *type-checking* or *relax-sort*($X/Y, _$) so to imply $X : s$.
remove($Y.f = Z$):
 if $Y.f = Z$ comes with type-checking, then
 eliminate (that) *type-checking* or *relax-sort*($X/Y, _$) so not to involve
 $Y.f = Z$.
remove($U \doteq V$):
 if $U \doteq V$ comes with type-checking, then
 eliminate (that) *type-checking* or *relax-sort*($X/Y, _$) so not to involve
 $U \doteq V$.
eliminate type-checking \equiv
 relax-sort($Y, _$) so that $s \wedge s' = s$ or *remove* all feat constraints $X.f = Z$
 or *remove*($X \doteq Y$), where $X \doteq Y$ was used in the premise of *sort*
 refinement

Fig. 6. The type-checking extensions to generalising operators

Consider that the type `satisfy_HPSG_principles` in our training grammar was further constrained such that `CAT` \doteq `COMP.CAT`.⁹ Let us denote by Ψ_0 this (over-constraint) form of the type `satisfy_HPSG_principles`. As a consequence, now the grammar will recognise at most sequences of words having the same category (like *noun noun*), and therefore no sentence from the initial set of examples will be accepted. How will the *ilp*LIGHT system have to proceed to recover the initial form of the grammar (or a form closed to it) so to parse in an (acceptably) correct manner the initial set of examples?

We run the new (expanded) grammar using the LIGHT parser in ‘reverse’ mode, giving as input a simple grammatical parse like `rH_phrase(the, girl)`. The information provided by the tracer module of the LIGHT unifier — which gets stopped at the moment when that parse fails to be recognised — is that after having accepted the key argument `girl` for the rule `rH_phrase`, the unification with the complement argument `the` fails. The unification failure involves two incompatible sorts constraints, $X:\text{noun}$ and $Y:\text{det}$, found on the path values $X = \text{girl.CAT}$, and respectively $Y = \text{girl.SUBCAT.FIRST}$ ($\doteq \text{the.CAT}$).

The LIGHT unification tracer module finds this conflict as a sort-inconsistent pair of paths: (`girl.CAT`, `girl.SUBCAT.FIRST`). How will the *ilp*LIGHT system use this information to arrive (if possible) at the removal of the constraint `CAT` \doteq `COMP.CAT` (or equivalently `HEAD.CAT` \doteq `COMP.CAT`) in Ψ_0 ?

There are two recovery possibilities:

⁹ Adding this constraint corresponds to equating the tags/variables `#1` and `#3` in the expanded form of the `satisfy_HPSG_principles` type in Figure 1.

equation introduction \equiv
 if $root(\psi.\rho) = s$, then
 for all paths ρ' in ψ such that $root(\psi.\rho) \wedge root(\psi.\rho') \neq \perp$,
 generate the hypothesis (ψ', ρ) , where
 ψ' is obtained from ψ by unifying/equating $\psi.\rho$ with $\psi.\rho'$.

sort specialisation \equiv
 if $root(\psi.\rho) = s$ and s_1, s_2, \dots, s_n are all the immediate subsorts of s in \mathcal{S} , then
 generate the hypothesis (ψ_i, ρ) , where
 ψ_i is obtained from ψ by replacing the root sort of $\psi.\rho$ with s_i .

(eventual) unfolding \equiv
 If $\psi.\rho$ is an atomic FS, $s = root(\psi.\rho)$ and the type $\Psi(s)$ is non-atomic,
 then
 generate the hypotheses (ψ', ρ_i) , where
 ψ' is obtained from ψ by unifying $\psi.\rho$ with (a copy of) $\Psi(s)$, and
 $\rho_i = \rho.f_i$, for each feature f_i defined at root level in $\Psi(s)$.

If $\psi.\rho$ is non-atomic, then
 generate the hypothesis (ψ, ρ_i) , where $\rho_i = \rho.f_i$,
 for each feature f_i defined at root level in $\psi.\rho$, provided that
 the hypothesis (ψ, ρ_i) was not already generated.

hypotheses combination \equiv
 if (ψ, ρ) was created before (ψ', ρ') ,
 neither ρ' is a prefix of ρ nor ρ is a prefix of ρ' ,
 and $unify(\psi, \psi')$ doesn't fail, then
 generate the hypothesis $(\psi \& \psi', \rho)$;

Fig. 7. The operators used in generating the *type specialisation* hypothesis.

1. Relax (or even remove) either one of the two constraints $X:noun$ and $Y:det$ involved in the sort clash.

Suppose that we run the grammar with the sort value of either one of `girl.CAT` or `girl.SUBCAT.FIRST` relaxed to `categ` (or removed). When the grammar will be re-expanded — as it has to be done, for the sake of logical soundness — the old, clashing constraints will be re-instated from Ψ_0 . Therefore, the removal or relaxation of such a sort constraint will have to be reversely propagated in the un-expanded grammar.

To reach this aim, reverse expansion firstly suggests to relax/remove the constraint `CAT:noun` from the `noun_le` type. Alternatively, the constraint on `det_le.CAT` can be relaxed to `categ`. However, in all of these ways the grammar will accept many incorrect sentences (like, “*the laughs*”, if the `CAT` value for `det_le` was relaxed). (This is why these three alternatives were represented as compacted into a single search path in Figure 9.) Therefore the second generalisation path must be

sort specialisation:

it is enough to consider directly those subsorts s_i (of s) for which $s_i :$
 $LUB(S)$,

where

$S = \{ \text{root}(\varphi.\rho) \mid \varphi \text{ is substructure in the FS associated to a parse}$
for a positive example, with $\text{root}(\varphi) = \text{root}(\Psi(t))\}$;

equation introduction:

consider $X \doteq Y$, where $\psi.\rho = X$ and $\psi.\rho' = Y$, only if
for φ chosen as above, $\varphi.\rho$ and $\varphi.\rho'$ are identical/equated;

eventual unfolding:

do not apply it for attribute paths ρ for which
 $\varphi.\rho$ is undefined for any φ taken as above.

Fig. 8. Refinements of the specialisation operators — using positive examples to guide the search.

pursued:

2. Remove the “inner” source of sort clash, namely the equation `rH_phrase.HEAD.CAT` \doteq `rH_phrase.HEAD.SUBCAT.FIRST`. The LIGHT’s unification tracer will reveal that this equation — identified by analysing backwards the coreference chain between the heap cells representing variables — has lead to sort clash for `rH_phrase.HEAD.CAT` = `girl.CAT` and `rH_phrase.HEAD.SUBCAT.FIRST` = `girl.SUBCAT.FIRST` (= `the.CAT`).

Actually, the equation `rH_phrase.HEAD.CAT` \doteq `rH_phrase.HEAD.SUBCAT.FIRST` will be removed not (only) from the `rH_phrase` type but from `satisfy_HPSG_principles`, i.e. Ψ_0 , as dictated by reverse expansion.

Removing an equation constraint $U \doteq V$ requires the creation of a copy U of the feature structure V shared in Ψ_0 by the paths `HEAD.CAT` and `HEAD.SUBCAT.FIRST`. The reader should notice that in our example, removing the equation $\Psi_0.\text{HEAD.CAT} \doteq \Psi_0.\text{HEAD.SUBCAT.FIRST}$ does not generate a single FS generalising Ψ_0 , but four! This is because the value shared (before equation elimination) by the two paths inside Ψ_0 was shared also with the paths `CAT` and `COMP.CAT`. Removing the equation `HEAD.CAT` \doteq `HEAD.SUBCAT.FIRST` from Ψ_0 entails the creation of all possible permutations corresponding to the potential values (U and V) for the paths `CAT` and `COMP.CAT`. This is why/how $\Psi_1, \Psi_2, \Psi_3, \Psi_4$ are generated, as shown in Figure 9 and Figure 10. The evaluation of the parsing results in each case will show that one of them (Ψ_3) is the right generalisation.

5 Learning specialisations of attribute values: exemplification

This section will exemplify how the procedure for generating type specialisations presented in Figure 3 works for learning the Subcategorization Principle encoded

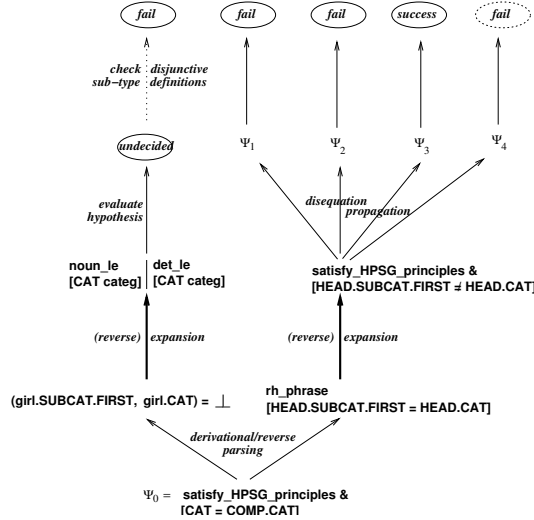


Fig. 9. A sample (grammar) search space when generalising a type.

in the type `satisfy_HPSG_principles` already presented in Figure 1.

Note that the Subcategorization Principle is directly linked to the nature of the parsing itself in lexicalized typed-unification grammars like HPSG, Lexicalized Functional Grammars (LFG, [12]), Categorical Unification Grammars (CUG, [20]): the (few) rules only provide very general principles/constraints about how phrases or words may be combined, while information about specific combinatorial valances of words is provided in the (many) lexical descriptions.

Basically, as formalised here on right, the Subcategorization Principle says that when applying a rule — in order to build a *mother* FS out of a *head* daughter and a *complement* daughter —, the complement daughter “consumes” the first element of the head daughter’s SUBCAT list, and “transmits” the rest of that list to the mother FS.

```
satisfy_HPSG_principle
[ SUBCAT #2,
  HEAD phrase_or_word
    [ SUBCAT #3|#2 ]
  COMP phrase_or_word
    [ CAT #3 ] ]
```

So, can we learn this very nature of the parsing in such lexicalized typed-unification grammars? How should we proceed? We outline below how we apply our learning strategy for type specialisation. Figure 11 shows the specialisation hypothesis search space explored/created when applying the second procedure in Figure 3.

1. To eliminate the Subcategorization Principle from our grammar, we delete the value of the HEAD.SUBCAT feature path in the unexpanded definition of the `satisfy_HPSG_principles` type in Figure 14. Then, the expansion of this type would come

```

satisfy_HPSG_principles [ HEAD.CAT ≠ HEAD.SUBCAT.FIRST ]

Ψ1 = satisfy_HPSG_principles
[ CAT #1:categ,
  SUBCAT #2:categ_list,
  HEAD #3:phrase_or_word
    [ CAT categ,
      SUBCAT categ_cons
        [ FIRST #1,
          REST #2 ] ],
  COMP #4:phrase_or_word
    [ CAT #1,
      SUBCAT nil ],
  ARGS <#3, #4> ]

Ψ2 = satisfy_HPSG_principles
[ CAT #1:categ,
  SUBCAT #2:categ_list,
  HEAD #4:phrase_or_word
    [ CAT #3:categ,
      SUBCAT categ_cons
        [ FIRST #1,
          REST #2 ] ],
  COMP #5:phrase_or_word
    [ CAT #3,
      SUBCAT nil ],
  ARGS <#4, #5> ]

Ψ3 = satisfy_HPSG_principles
[ CAT #1:categ,
  SUBCAT #2:categ_list,
  HEAD #4:phrase_or_word
    [ CAT #1,
      SUBCAT categ_cons
        [ FIRST #3:categ,
          REST #2 ] ],
  COMP #5:phrase_or_word
    [ CAT #3,
      SUBCAT nil ],
  ARGS <#4, #5> ]

Ψ4 = satisfy_HPSG_principles
[ CAT #1:categ,
  SUBCAT #2:categ_list,
  HEAD #3:phrase_or_word
    [ CAT #1,
      SUBCAT categ_cons
        [ FIRST categ,
          REST #2 ] ],
  COMP #4:phrase_or_word
    [ CAT #1,
      SUBCAT nil ],
  ARGS <#3, #4> ]

```

Fig. 10. Propagating a dis-equation in generalising a typed-unification grammar

up with the constraints `HEAD.SUBCAT : categ_list`, `CAT : categ`, and `COMP.CAT : categ`. The expanded form of the `satisfy_HPSG_principles` type will be now Ψ_0 , as shown in Figure 12.

All sentences which have been parsed with the initial grammar are parsable now too — because the associated parses are entailed by the new grammar, which is less restrictive —, but also many ungrammatical input sentences become (incorrectly) acceptable. (The grammar over-generates.) For instance, the phrase *the laughs* is acceptable since the two words, taken respectively as complement and head for the (under-constrained) `rH_phrase` make it satisfiable. How should one proceed to fix this over-parsing issue?

2. The first possibility: try to equate the `HEAD.SUBCAT` value — whose sort is `categ_list` — to a substructure inside the type `satisfy_HPSG_principles`, which has a compatible root sort. There are two such substructures: the `SUBCAT` attribute value whose sort is also `categ_list`, and the `COMP.SUBCAT` value which is `nil`. Either one of the equational constraints (denoted as `HEAD.SUBCAT ≐ SUBCAT`, respectively `HEAD.SUBCAT ≐ COMP.SUBCAT`) makes the new (expanded) form of the grammar reject all sample sentences, therefore it is not acceptable.

3. Alternatively, we may try to refine the sort constraint on the `HEAD.SUBCAT` path value (in Ψ_0). Choosing to make it `nil` — one of the two descendents of `categ_list`

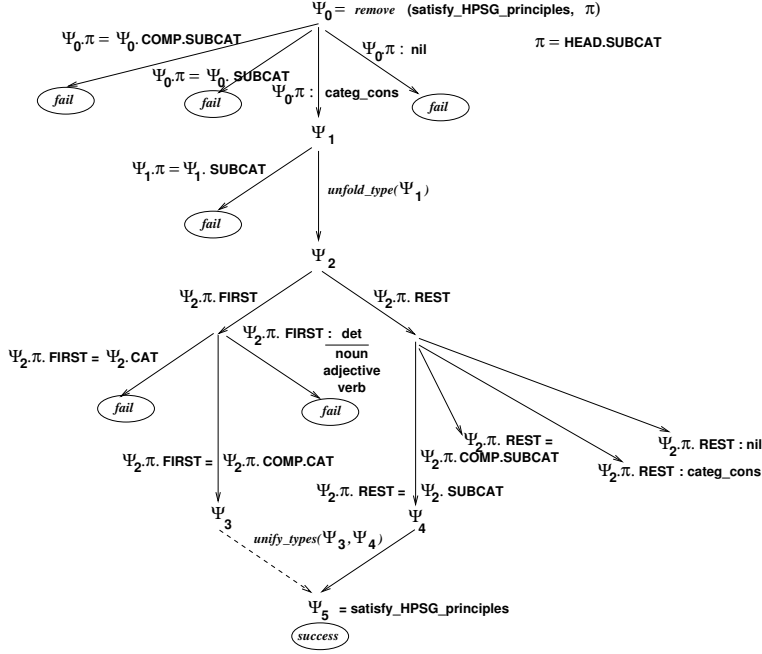


Fig. 11. A sample search space in specialising a typed attribute value.

found in the grammar’s sort hierarchy — would block any parse; we have to abandon this alternative, and consider to refine the HEAD.SUBCAT path value in Ψ_0 to `categ_cons`. Let Ψ_1 denote this new form of `satisfy_HPSG_principles`. The new grammar over-generates again.

The outcome of parsing is still not (yet) significantly improved w.r.t. the above point 1: all sentences which were grammatical w.r.t. the initial grammar get accepted — since our most recent grammar is still less specific than the original grammar — and most (if not all) ungrammatical sentences tested before are still accepted. Therefore more constraints must be pushed onto Ψ_0 .HEAD.SUBCAT. In order to do so, we have to unfold it, using the definition of its type value, namely `categ_cons`. Let Ψ_2 denote the newly obtained form of `satisfy_HPSG_principles`.

4. We study the possibility of equating the value for the path HEAD.SUBCAT.FIRST to compatible sub-types in the type current form (Ψ_2) of `satisfy_HPSG_principles`. The potential candidates are: `CAT`, `HEAD.CAT`, and `COMP.CAT`.

The first two possibilities — which are in fact one and the same, due to the equation constraint `CAT ≐ HEAD.CAT` in the type `satisfy_HPSG_principles` — are discarded when the result of parsing with the new form of the grammar is compared

```

 $\Psi_0 = \text{satisfy\_HPSG\_principles}$ 
[ PHON  diff_list,
  CAT   #1:categ,
  SUBCAT categ_list,
  HEAD  #4:phrase_or_word
    [ PHON  diff_list,
      CAT   #1:categ,
      SUBCAT categ_list ],
  COMP  #5:phrase_or_word
    [ PHON  diff_list,
      CAT   categ,
      SUBCAT nil ],
  ARGS  <#4, #5> ]

 $\Psi_3 = \text{satisfy\_HPSG\_principles}$ 
[ PHON  diff_list,
  CAT   #1:categ,
  SUBCAT categ_list,
  HEAD  #4:phrase_or_word
    [ PHON  diff_list,
      CAT   #1,
      SUBCAT cons
        [ FIRST #2:categ,
          REST  categ_list ] ],
  COMP  #5:phrase_or_word
    [ PHON  diff_list,
      CAT   #2,
      SUBCAT nil ],
  ARGS  <#4, #5> ]

```

Fig. 12. Specialisation types created during learning the Subcategorization Principle.

with the parsing results for the original grammar.

The provided sample sentences will support the equation

$$\text{satisfy_HPSG_principles}[\text{HEAD.SUBCAT.FIRST} \doteq \text{COMP.CAT}].$$

which make the the current version of `satisfy_HPSG_principles` become Ψ_3 as shown in Figure 12. The number of ungrammatical sentences accepted by the current form of the grammar is significantly reduced, but still sentences like *the embarrasses mary* are not rejected.

5. We can try to further constraint the value of the path `HEAD.SUBCAT.FIRST`. But replacing its actual sort, `categ`, to anyone of its descendants would reject some of the positive examples, i.e. grammatical sentences.¹⁰

6. Try to equate/coreference `HEAD.SUBCAT.REST`. The (type-guided) search subspace is represented by the attribute paths `SUBCAT`, and `COMP.SUBCAT`. Examples shall support the equation

$$\text{satisfy_HPSG_principles}[\text{HEAD.SUBCAT.REST} \doteq \text{SUBCAT}].$$

7. The new, expanded form of `satisfy_HPSG_principles` (Ψ_4), tested via parsing the given examples, proves to be perfectly convenient after combination/unification with the previously obtained specialisation (Ψ_3). (In fact it coincides with the form in Figure 1.) At this point the Subcategorization Principle as described in the initial grammar is learnt!

¹⁰ As a matter of fact, this alternative (or: path in the specialisation search space) can be pruned by analysing the parses eventually provided as annotation to the grammatical sentences (positive examples): the least upper bound (*LUB*) for the `CAT` values in the lexical descriptions (which unify with the `HEAD` value of `satisfy_HPSG_principles` during parsing) is `categ`, therefore no sort refinement is acceptable here. See Figure 8 in Section 3.2 for different improvements to sub-type specialisation using positive examples.

6 Scalability issues

We have shown in the preceding sections that ILP learning of attribute path values is both feasible and interesting on reduced-scale grammar. The important question is how can it be scaled up to large grammars, since it is obvious that in such cases the search space may easily grow beyond acceptable limits. In this section we will give to answers to this question, reflecting on our recent and respectively current work with LinGO [10], the wide coverage HPSG grammar for English developed at the Center for the Study of Language and Information (CSLI), University of Stanford. The LinGO grammar — 2.5Mb of source code, going to 40.4Mb in expanded version — has 15059 types, among which 62 are rules and 6897 lexical entries.

1. The ILP-based generalisation of typed unification grammars presented in Section 3 and exemplified in Section 4 inspired us to elaborate another learning technique, which we called Generalised Reduction (GR) and presented in [6]. A couple of issues differentiate these two techniques:

— GR eliminates only as much as possible of atomic feature constraints, as long as the parsing results on a training corpus are not affected. The equation constraints and sort constraints associated to “reduced”/eliminated feature values are implicitly discarded.

— this form of unification grammar generalisation through successive elimination of atomic feature constraints is applied only to the rule FSs. Lexical FSs and other type FSs are not affected.

Applying our Generalised Reduction technique on LinGO to an interesting result: more than 59% of feature constraints in the rule FSs can be eliminated, thus downsizing the rule FSs and yielding a significant reduction factor for memory consumption (62%) and the average parsing time (22%). As training corpus for applying the GR learning technique on LinGO, we used the CSLI test-suite which has 1348 sentences with an average length of 6.38 tokens and an ambiguity of 1.5 parses per sentence. For the test, we used the a test-suite made of 96 sentences, the average length of which is 8.5 and the ambiguity 14.14. Following the learning of the GR-reduced form of LinGO, the average parsing precision we obtained on this second test-suite was 83.79%.

2. Coming back to our (general) ILP-based procedures presented in Section 3, we tested the *ilp*LIGHT prototype extension to the LIGHT system on a number of cases (similar to those presented in Sections 4 and 5) on the LinGO grammar. We saw that the potentially very large search space to be explored (especially) during generalisation can be effectively controlled by parameterising that procedure with a set \mathcal{P} of types to which the search is limited.¹¹ For instance, when limiting the search space for the example presented in Section 4 to the `satisfy_HPSG_principle`,

¹¹ For the implementation, that means that when running the parser in ‘reverse’ mode, its output will be limited to failure paths inside \mathcal{P} .

the tree in Figure 9 is pruned to its right-branch subtree. Similarly, for specialisation we can opt to limit the newly proposed feature paths (through unfolding) to a certain depth.

We make the *important remark* that restricting the application of the generalisation procedure through the above proposed (type FSs set) parameter is not only justified by efficiency reasons; it corresponds to the naturally modularised development of unification grammars, supported directly by their hierarchical organisation.

We are currently doing engineering work aiming to replace the *ilp*LIGHT prototype with a much faster version, suitable for more extensive tests. We appreciate that after this new version will become stable, the parallelisation of tasks for checking the validity of the hypotheses proposed by the ILP learner will be possible without much difficulty, by using multiple instances of the LIGHT parser distributed on a cluster of PCs.

Conclusion: We presented an framework for logic-based inductive learning of attribute path values in typed-unification grammars. The implemented prototype — *ilp*LIGHT — extended the LIGHT parsing system with an ILP/learner module whose core is made of a pair of procedures for generating/inferring hypotheses. Driven by simply annotated sample sentences, the creation of these hypotheses either generalises (one or more types automatically identified in) the input grammar or specialises a designated type FS definition. We tested both generalisation and specialisation on both a simple but relevant HPSG grammar and a wide-coverage grammar for English.

We expect that our ILP-based learning technique will first serve as a debugging tool assisting the users in the elaboration of new (extensions to) typed-unification grammars.¹² Moreover, in combination with shallow parsing techniques and lexical semantics resources like WordNet [13], our approach may also serve to perform lexical inferences. We will explore test this idea while extending the LIGHT system (and the LinGO grammar) so to parse abstracts of bio-medical reports.¹³

Acknowledgements: This work was primarily supported by the EPSRC ROPA grant “Machine Learning for Natural Language Processing in a Computational Logic Framework”. The LIGHT system was developed at the Language Technology Lab of The German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany. Many thanks go to Ulrich Callmeier who implemented the first versions of the *tracer* and *derivation* functions for the LIGHT system. I am grateful also to Steven Muggleton, Dimitar Kazakov, Suresh Manandhar, James Cussens and Ross King who, in different ways, made possible my work on this subject.

¹² See [3] for a recent proposal of methodological, concerted development of such grammars.

¹³ Note that existing approaches to such a task [5] [11] do not tackle the issue of building specific lexical descriptions for words not found in the grammar’s lexicon. They simply associate such words only word-independent/categorical characterization.

Appendix: A sample HPSG grammar

We give here the grammar that we adapted from [19] for exemplification of our learning strategy for typed-unification grammars, as it was presented in Section 3. This grammar is made of two parts: the sort/type hierarchy and the type definitions. For conciseness reasons, the sort hierarchy is given in Figure 13, while the types are presented in Figure 14 using the LIGHT syntax:

Strings are surrounded by quotes. The sorts `start`, `list` and the list subsorts `cons` and `nil` are special (reserved LIGHT) types, and so is the difference list sort, `diff_list` too. The notation `<!>` in the grammar's code is a syntax sugar for difference lists, just as `<>` is used for lists. Also, `!` is a constructor for difference lists, playing a similar role to that played by the constructor `|` for non-nil (i.e. `cons`) lists.

Just for convenience, in Figure 13 the sorts situated under a dashed line are derived (i.e. have the same parents) as the sort found immediately above that line. For instance, the adjective `pretty` is derived from `adjective_le`, similarly to the adjective `nice`. The grammar has two rules, `lH_phrase` and `rH_phrase` (see section program), and 15 lexical entries (see section query).

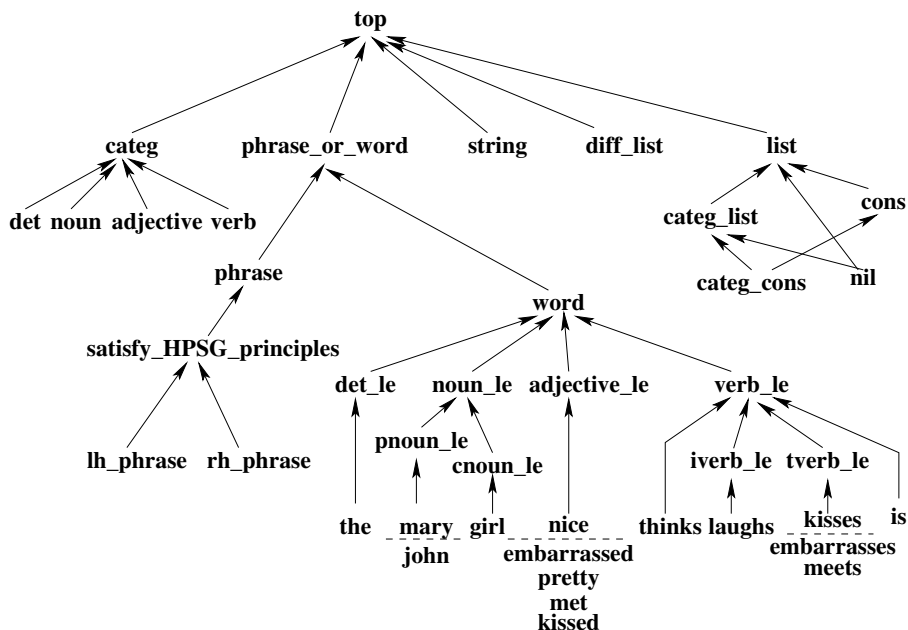


Fig. 13. The sort signature for the grammar in Figure 14.

```

types:

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> ]

program: // rules

lH_phrase
[ SUBCAT cons,
  PHON #1#3,
  HEAD.PHON #1#2,
  COMP.PHON #2#3 ]
rH_phrase
[ SUBCAT nil,
  PHON #1#3,
  HEAD.PHON #2#3,
  COMP.PHON #1#2 ]

query: // lexical entries

the[ PHON <!"the!"> ]
girl[ PHON <!"girl!"> ]
john[ PHON <!"john!"> ]
mary[ PHON <!"mary!"> ]
nice[ PHON <!"nice!"> ]
embarrassed[ PHON <!"embarrassed!"> ]
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!"> ]
embarrasses[ PHON <!"embarrasses!"> ]

```

Fig. 14. A sample typed-unification grammar (adapted from [17]), in LIGHT format.

References

1. H. Ait-Kaci and R. Nasr. Login: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.
2. H. Ait-Kaci and A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16:195–234, 1993.
3. E. Bender, D. Flickinger, and S. Oepen. The Grammar Matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proceedings of COLING 2002 Workshop on Grammar Engineering and Evaluation*, Taipei, Taiwan, 2002.
4. B. Carpenter. *The Logic of Typed Feature Structures – with applications to unification grammars, logic programs and constraint resolution*. Cambridge University Press, 1992.
5. J. Carroll and E. Briscoe. High precision extraction of grammatical relations. In *Proceedings of the 7th ACL/SIGPARSE International Workshop on Parsing Technologies*, pages 78–89, Beijing, China, 2001.
6. L. Ciortuz. Learning attribute values in typed-unification grammars: On generalised rule reduction. In D. Roth and A. van den Bosch, editors, *Proceedings of the 6th Conference on Natural Language Learning (CoNLL-2002)*, pages 70–76, Taipei, Taiwan, 2002. Morgan Kaufmann Publishers and ACL.
7. L. Ciortuz. LIGHT — a constraint language and compiler system for typed-unification grammars. In *KI-2002: Advances in Artificial Intelligence*, volume 2479, pages 3–17, Berlin, Germany, 2002. Springer-Verlag. Proceedings of the 25th Annual German Conference on AI, Aachen, Germany.
8. L. Ciortuz. Towards inductive learning of typed-unification grammars. In *Electronic Proceedings of the 17th Workshop on Logic Programming*, 2002. Dresden Technical University, (http://www.computational-logic.org/local/wlp2002/WLP_schedule.html).
9. J. Cussens and S. Pulman. Experiments in inductive chart parsing. In *Learning Language in Logic, volume 1925 of LNAI*. Springer, 2000.
10. Daniel P. Flickinger, Ann Copestake, and Ivan A. Sag. HPSG analysis of English. In Wolfgang Wahlster, editor, *VerbMobil: Foundations of Speech-to-Speech Translation, Artificial Intelligence*, pages 254–263. Springer-Verlag, 2000.
11. C. Grover, M. Lapata, and A. Lascarides. A comparison of parsing technology for the biomedical domain. 2002. Submitted for publication to The Journal of Natural Language Engineering. (<http://www.ltg.ed.ac.uk/disp/papers.html>).
12. R. M. Kaplan and J. Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–381. MIT Press, 1983.
13. G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.
14. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
15. F.C.N. Pereira and D. Warren. Definite Clause Grammars for Language Analysis – a survey of the formalism and a comparison with Augmented Transition Networks. *Artificial Intelligence*, 13:231–278, 1983.

16. C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar*. CSLI Publications, Stanford, 1994.
17. S. Shieber. *An introduction to unification-based approaches to grammars*. CSLI Publications, Stanford, 1986.
18. S. M. Shieber, H. Uszkoreit, F. C. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In J. Bresnan, editor, *Research on Interactive Acquisition and Use of Knowledge*. SRI International, Menlo Park, Calif., 1983.
19. G. Smolka and R. Treinen, editors. *The DFKI Oz documentation series*. German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, Sarrbrücken, Germany, 1996.
20. H. Uszkoreit. Categorical Unification Grammar. In *International Conference on Computational Linguistics (COLING'92)*, pages 498–504, Nancy, France, 1986.
21. J. Zelle and R. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 817–822. AAI Press/MIT Press, 1993.