

Automatic Induction of Lexical Inheritance Hierarchies

Harald LÜNGEN Caroline SPORLEDER

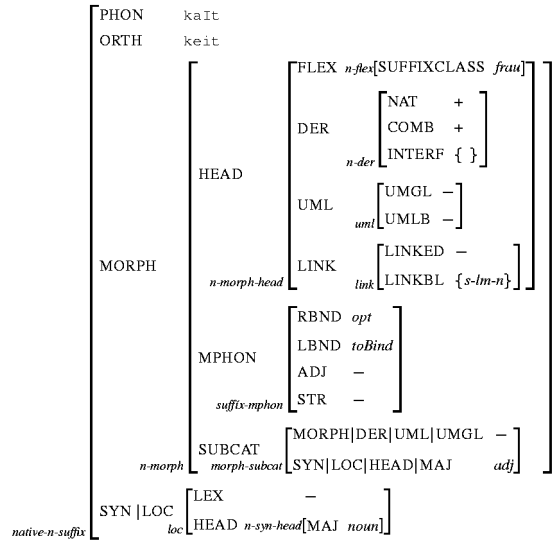
1. Introduction

Inheritance hierarchies in general and type hierarchies in particular have recently become a widespread means of representing lexical information. However, constructing inheritance hierarchies for a lexicon by hand can become fairly tedious, especially when the lexicon is very big. The best hierarchy may not be obvious in all details. Therefore it seems reasonable to investigate ways to construct hierarchies automatically. An automatically constructed hierarchy may also help to gain new insight into the underlying data set, revealing interesting interrelations between linguistic objects. In particular, a computer program can be strongly data oriented, constructing hierarchies which reflect the actual distribution of linguistic properties in a given lexicon.

In this paper, we introduce our lexicon of German morphemes used in a corpus analysis task and give an overview of how a lexical hierarchy of morph types was manually constructed. We then present an algorithm called *Top-Down Attribute Selection* for machine learning of single lexical inheritance hierarchies, which is based on decision tree learning methods.

2. A Lexical Hierarchy of Morph Types

Our morph lexicon is a component of a morphological analyser called MCLASS employed during the acquisition of a speech oriented lemma lexicon from a corpus of transcribed spoken dialogues (cf. LÜNGEN et al. 1998). The linguistic model of morphology underlying MCLASS is based on HPSG (POLLARD and SAG (1987, 1994)) and covers all areas of morphology, i.e. inflection, derivation, and composition. An attribute-value matrix (AVM) representing the feature structure that models the lexical entry of the nominalising derivational suffix *keit* is given in Figure 1. This feature structure is complex, i.e. values of features can again be feature structures, and it is well-typed, i.e. it contains only attributes that are appropriate for a morph type called *native-n-suffix*, and the values

Figure 1: AVM for the suffix *keit*

assigned are appropriate for the respective attributes.¹ We have a set of mutually exclusive base types underlying the definition of features and their appropriate values for the lexical entries of morphs. These base types are notated above the dotted edges leading to the lowest possible types (the instances) in Figure 3.

The 31 features used describe combinatorial morphophonological (right- and leftboundness, root-adjacency, potential stress), morphotactic (such as nativeness, suitability of interfixes and linking morphemes, umlauting and umlaut-causing properties, inflectional class), and morphosyntactic properties of morphs.²

The central question when manually constructing a type hierarchy is how should types be grouped into common supertypes, or conversely, how should a type be divided into subtypes. There are two possibilities: Firstly,

¹ For several reasons, we employ feature structures as introduced in POLLARD and SAG (1987) and neglect some of the modifications of POLLARD and SAG (1994). An independent MORPH level in HPSG feature structures was also introduced by KRIEGER (1993).

² For a detailed description see LÜNGEN (to appear).

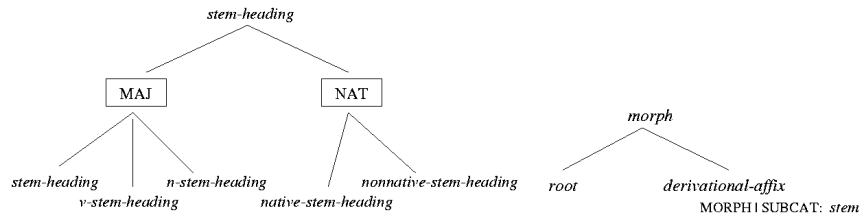


Figure 2: Two partial type hierarchies, for stem-heading and morph

a type T may be divided into the subtypes T_1, T_2, \dots, T_n according to different value types of one of its attributes. Such an attribute may then be called a dimension and the resulting subtypes are said to partition the type T , since they stand in an exclusive disjunction relation to each other and exhaust T , i.e. an instance of type T must either be of type T_1, T_2, \dots , or of type T_n . Usually, a type can be partitioned along several dimensions at once, leading to multiple orthogonal inheritance. The type-partitioning dimension is sometimes notated as a node in the tree representing the type hierarchy (though it is not a type itself) and set in a box (Figure 2, left). Secondly, the types T_1, T_2, \dots and T_n may constitute subtypes of one type T , if different sets of attributes are appropriate for them in addition to the common set of attributes appropriate for type T . In Figure 2 (right), the *derivational-affix* is a subtype of *morph* because in addition to the feature specifications of *morph*, the feature MORPH|SUBCAT is appropriate for it. Thus, the partitioning of the type *morph* into *root* and *derivational-affix* is along the criterion of absence vs. presence of the attribute MORPH|SUBCAT. Both principles of subdividing a type may occur jointly and multiply; in most cases there will be one outstanding feature which suffices as the type discriminating dimension but whose values may automatically entail type restrictions on some other features and the introduction of several new features. The type of the value of MORPH|HEAD, for example, is always co-varying with the (atomic) value of SYN|LOC|MAJ.³

³ For remarks on the principles of subdividing types and constructing type hierarchies

In Figure 3, a more elaborate version of our manually constructed morph type hierarchy is given. Subtypes introduce constraints in the fashion described above. The type *stem-heading* expresses the fact that root and derivational-suffix share a large set of features, namely the morphological head features. The hierarchy is not totally free of redundancy since multiple inheritance was not employed wherever possible (we consider an inheritance hierarchy of feature structures as being totally free of redundancy if each feature-value specification is stated exactly once) as such a strategy would have yielded numbers of fairly abstract types with no meaning in the morphotactics such as *nonnative-stressed-derivational-nominal-umlauting-Noun_VI-suffix*.

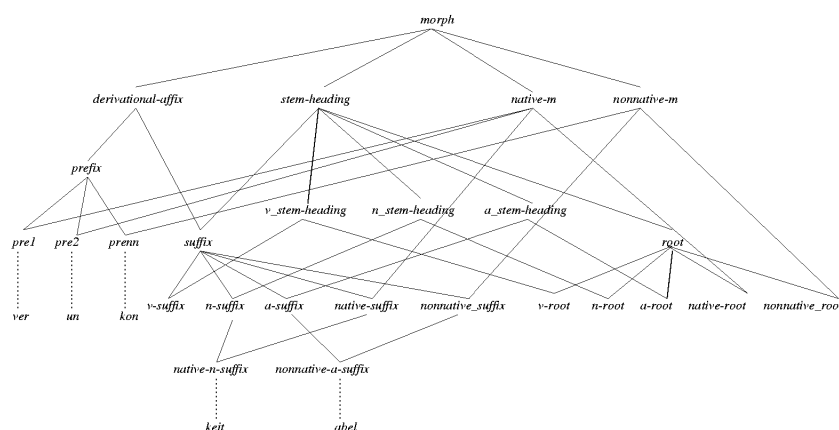


Figure 3: Another partial type hierarchy for *morph*

3 Learning Single Inheritance Hierarchies

The actual database of morphs used for the machine learning task contains flat sets of attribute-value pairs, the attribute paths in Figure 1 are collapsed into single attributes. Information about the typing of values is not incorporated either. Thus the derived hierarchies are “untyped”, in so

in the HPSG literature see e.g. (POLLARD and SAG, 1987, p.198ff), (FLICKINGER, 1987, p.17ff), RIEHEMANN (1998).

far as they only contain atomic value types and not value types partially ordered by subsumption.

Our approach to learning inheritance hierarchies is based on top-down tree induction (TDTI) methods, which are used to learn decision trees (cf. QUINLAN's (1986) ID3). Starting with the complete set of objects, at every level of the tree, one attribute is selected which best splits the object set into subsets. Single inheritance hierarchies also constitute trees but while a decision tree expresses differences between classes of objects, an inheritance hierarchy expresses generalisations. Thus, in a decision tree, the discriminating properties occur towards the top, i.e. they are preferred by the attribute selection. In an inheritance hierarchy, the discriminating properties occur towards the bottom, constituting idiosyncrasies. For learning inheritance hierarchies, the attributes therefore have to be selected in the inverse order. In addition, while in a decision tree only discriminating properties are represented, an inheritance hierarchy has to list all properties of a class, that is apart from the discriminating attribute, other attribute-value pairs shared by the members of the class are listed at the class node. Because of these differences, we chose to name our algorithm *Top-Down Attribute Selection* (TDAS).

Algorithms which base a classification on one attribute are more efficient than those which base a classification on a set of attributes, like MICHALSKI & STEPP's Conceptual Clustering (1983), since in the latter, theoretically, every possible conjunction of attribute-value pairs has to be considered. The disadvantage of TDAS is that it can only derive a subset of the classifications derivable by Conceptual Clustering, namely those which are not based on conjunctions of attribute-value pairs. It seems, however, that classifications based on conjunctions are not necessary. As we have seen above, there are two main principles of dividing a type into subtypes: either the subtypes differ with respect to the value of an attribute or they differ with respect to the attribute sets appropriate for them. In the former case, TDAS is clearly adequate, since the classification is based on *one* attribute. In the latter case, knowledge about the presence or absence of one of the attributes in a set is also enough to distinguish the classes. Thus with some amendments to the algorithm, the

second case can also be dealt with.⁴ However, this has not yet been implemented. So far an attribute can only be selected if it is contained in all objects under consideration. The only kind of classification that *cannot* be derived by the algorithm is the one in Figure 4 (left). However, these classifications seem to be rare in manually constructed hierarchies, possibly because the information they contain can also be expressed by means of multiple inheritance. Thus, in Figure 4, the fact that three types of prefixes can be distinguished depending on whether they are native (NAT) or stressed (STR), may as well be expressed by cross-classifying prefixes according to nativeness and stress. Alternatively, it is possible to introduce an additional level (Figure 4, right).

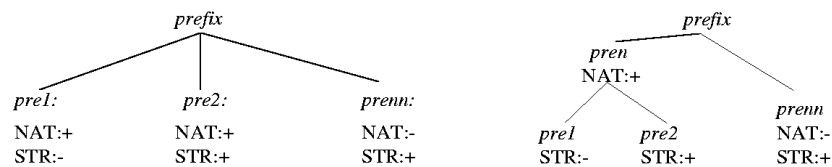


Figure 4: Two type hierarchies for prefix

Several variations of the basic algorithm have been implemented and tested on the morph database, which contained 1690 entries. The variations differed mainly with respect to the attribute selection and the hierarchy building component (cf. SPORLEDER (1999)). One attribute selection module (called *NOC*, Number of Occurrence) chooses the attribute of the most frequent attribute-value pair, the other attribute selection module takes an information theoretic approach, similar to Quinlan's ID3, and selects an attribute according to its inverse information gain (*IIG*). Hierarchy building can be simple or n-best, where n-best means that at every level of the hierarchy n classifications are built. The classification which makes use of the most attribute-value pairs is then chosen. The idea behind this was that if a class is a plausible one then its

⁴ For example, it is possible to treat the absence of an attribute as a special value (e.g. NIL).

members will share several attribute-value pairs. In addition, a variation of the algorithm for learning non-monotonic hierarchies was implemented.

The resultant hierarchies were then evaluated. Ideally, the evaluation criteria should be precise enough to be implemented and permit automatic assessment, because the size of the hierarchies renders complete manual evaluation impractical. The evaluation should take into account the structure (e.g. minimal redundancy) as well as the content of a hierarchy (e.g. linguistic plausibility of the classes). Structural evaluation can often be done automatically by implementing a set of well defined evaluation criteria and testing automatically in how far a hierarchy fulfils these criteria. Content evaluation is much more difficult to deal with. Thus, it is hard to think of (and implement) a set of criteria which measure the linguistic plausibility of a class. To get some idea of the content of the derived hierarchies, the top-levels of one of them were compared to a manually constructed one (see section 4).

With respect to structural evaluation, several criteria may be employed. Among other criteria, we used “number of features” (Feat.) (to be minimised, high priority), “average path length” (APL) (to be maximised, medium priority), and “number of nodes” (to be minimised, low priority).⁵ Some results are shown in table 1.

	NOC			IIG		
	Feat.	APL	Nodes	Feat.	APL	Nodes
simple, monotonic	10,884	10.44	2,757	9,906	11.03	2,851
2-best, monotonic	10,003	10.28	2,849	9,968	10.22	2,843
3-best, monotonic	10,245	9.04	2,852	10,832	8.78	2,895
simple, non-mon.	9,962	10.50	2,862	9,834	11.20	2,847

Table 1: Results

⁵ Some structural criteria are a bit problematic, because, on their own, they do not say very much about the quality of the hierarchy. For a discussion see BARG (1996) and SPORLEDER (1999).

The best hierarchy reduced the amount of redundancy (measured by the number of features) to one third compared with the flat lexicon, which contained 30,857 features. 1,160 new nodes were introduced and the hierarchy contained an average number of 11 levels. The information theoretic attribute selection method in combination with simple search yielded the best results.

4. Manually vs. Automatically Constructed Hierarchies

The first four levels of the linguistically most plausible of our hierarchies are displayed in Figure 5. It was derived with NOC attribute selection and simple search,⁶ and it has a lot more nodes (2862 total) than the manually constructed one. This is because no multiple inheritance was learned and subclasses were also constructed at very low levels in the hierarchy.

On the fourth level in Figure 5 we can see the features LBND or RBND (left- and right-bound) to be the class-discriminating features in most subtrees. In one of them, the feature RBND led to the implausibility that some verbal roots are in a class together with the prefixes because they are RBND=*toBind*. For the manually constructed type hierarchy in Figure 3, LBND and RBND were not considered to define classes needed at the top of a morph type hierarchy used for our immediate dominance morphotactics, as they encode surface-oriented (linear precedence) properties of morphs. This demonstrates that linguistic background knowledge is applied when deciding which attributes should be considered first for building a hierarchy. Attribute selection methods, such as NOC and IIG, do not incorporate this kind of background knowledge. Both of them are biased to attributes which are specified for all or many morphs and will thus select RBND and LBND fairly early.

⁶ The nodes stand for the classes learned and are labelled with the feature-value specifications that the algorithm regarded as the property shared by all members of the respective class.

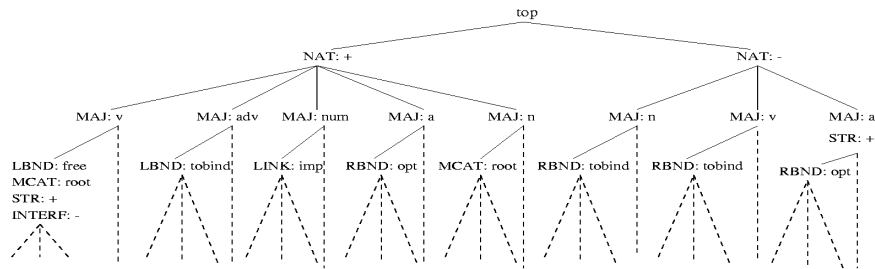


Figure 5: An Automatically Derived Hierarchy

It is interesting that there is a certain kind of balance in the derived hierarchy in that the program tends to choose the same attribute (e.g. MAJ) at the same level of different subtrees. Hierarchies like this hint strongly at the necessity of multiple inheritance. In this case, it is possible to cross-classify morphs according to the two dimensions MAJ and NAT, as was done in the manually constructed hierarchy given in Figure 3. We are confident that it is possible to extend the algorithm to transform single hierarchies that are balanced in such a way into multiple inheritance hierarchies. But it remains doubtful whether inheritance hierarchies totally free of redundancy are desirable, in which for every feature value specification that occurs more than once, a class node is introduced from which the specification subsequently is inherited, i.e. in which feature-value specifications are replaced by inheritance links. The consistent application of multiple inheritance seems to be useful only down to a certain level in a lexical hierarchy. An algorithm automatically inducing such hierarchies would have to take this into account.

The hierarchy in Figure 5 is actually non-monotonic. We are not arguing that non-monotonicity is in general necessary or desirable for lexical inheritance hierarchies, but we found that allowing a certain amount of it improved the quality of the derived hierarchies.⁷ One important reason for this is that the input database contained a small amount of noise in the form of faulty attribute-value specifications. If non-monoton-

⁷ In the implementation, it is possible to adjust the maximum amount of non-monotonic classifications by means of a parameter.

icity is allowed the program is permitted to treat noise as exceptions and still derive the “right” classes. To ensure the overall monotonicity of such a hierarchy, the cases in which non-monotonic inheritance was used would then have to be checked semi-automatically to see whether there are parts of the hierarchy where the non-monotonicity was not due to noise. In fact, this procedure also proved to be a good way of detecting remaining errors in our database.

5. Conclusion

In this paper, we have explored the principles of manually constructing lexical type hierarchies which are a kind of inheritance hierarchies. We have provided the Top-Down Attribute Selection algorithm for learning single lexical inheritance hierarchies. The classes learned with the help of our algorithm can be interpreted as some kind of lexical types forming a type hierarchy. Unlike typed feature structures as used in HPSG, each attribute has only atomic values and can only be introduced in a class description when all objects in the class have the same atomic value for it. For learning HPSG-style type hierarchies, the algorithm should be able to learn multiple inheritance and to deal with attributes underspecified for particular values.

The TDAS algorithm is more efficient than common clustering methods since only individual attributes have to be considered. Thus, even for big lexica, containing many entries and attributes, a hierarchy can be found in reasonable time. Connected with the high efficiency, however, is the limitation that only a subset of the classes learnable by e.g. Conceptual Clustering can be derived. Classes based on conjunctions or disjunctions of attribute-value pairs cannot be learned. However, the classes derivable by the TDAS algorithm seem to be those that are used by linguists.

References

- BARG, P. (1996): *Automatischer Erwerb von linguistischem Wissen. Ein Ansatz zur Inferenz von DATR-Theorien*. Tübingen: Niemeyer

- FLICKINGER, D. (1987): Lexical Rules in the Hierarchical Lexicon. Ph.D. thesis, Stanford University.
- KRIEGER, H.-U. (1993): Derivation without lexical rules. DFKI Research Report RR-93-27. Saarbrücken: DFKI.
- LÜNGEN et al. (1998): H. L., Karsten EHLEBRACHT, D. GIBBON and A.P. QUIRINO SIMÕES, Bielefelder Lexikon und Morphologie in VERBMOBIL Phase II. VERBMOBIL Report 233. Universität Bielefeld.
- LÜNGEN, H. (to appear): MCLASS. HPSG-based morphological analysis for the acquisition of a spoken language lexicon. Universität Bielefeld.
- MICHALSKI, R.S. and STEPP, R.E. (1983): Learning from observation: Conceptual clustering. In: R.S. MICHALSKI, J.G. CARBONELL and T.M. MITCHELL (eds.), *Machine Learning: An Artificial Intelligence Approach*, vol. 1, 331-363. Los Altos, CA: Morgan Kaufmann.
- POLLARD, C. and SAG, I. (1987): *Information-Based Syntax and Semantics*. Menlo Park, CA: CSLI International.
- POLLARD, C. and SAG, I. (1994): *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press.
- QUINLAN, J.R. (1986): Induction of decision trees. In: *Machine Learning* 1, 81-106.
- RIEHMANN, S.Z. (1998): Type-based derivational morphology. In: *Journal of Comparative Germanic Linguistics* 2, 49-77.
- SPORLEDER, C. (1999): Learning Lexical Generalisations. An Operational Evaluation of Current Machine Learning Methods. Master's thesis, Universität Bielefeld.