Type Signature Induction with FCA$_{\text{Type}}$

Wiebke Petersen

Institute of Language and Information
Heinrich-Heine-Universität Düsseldorf
petersew@uni-duesseldorf.de

Abstract. Type signatures are common in modern linguistic theories. Their construction and maintenance is intricate, and therefore, an automatic induction method is desirable. In the present paper we present FCA$_{\text{Type}}$, a module of our system FCA$_{\text{Ling}}$, that automatically induces type signatures from sets of untyped feature structures. The induction procedure is based on so-called *decomposition semilattices* which serve as a basis for initial type signatures. These signatures can be folded up to result in compact and restrictive type signatures which adequately specify the input structures.

1 Introduction

The primary task in grammar engineering is to construct a grammar which generates exactly those phrases which are well-formed in the target language. The purpose of the lexicon is to provide the basic units of the language. Modern linguistic theories tend to express more and more grammatical information in the lexicon. Hence, "lexical entries have evolved from simple pairings of phonological forms with grammatical categories into elaborate information structures, in which phonological forms are now paired with more articulated feature structure descriptions.", [1, p.173]. *Feature structures* (FSs) are recursive attribute-value structures which are known as frames in other disciplines, e.g. [2].[1] An example lexicon with small 'toy' FSs taken from [4] is shown in Fig. 1.

As depicted in Fig. 1, FSs can be written as recursive *attribute-value matrices* (AVMs). The AVMs are constructed as follows: FSs are enclosed in square brackets. Each first-level attribute is followed by a colon and its value. The values are either atomic (i.e., not specified by further attributes) or complex FSs. Restricting a FS to one of its paths yields the value of the path in the FS, e.g.,

$$\begin{bmatrix} \text{CAT}: & \mathsf{np} \\ \text{HEAD}: & \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{PERS}: & \mathsf{third} \\ \text{NUM}: & \mathsf{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix} @_{\text{HEAD AGR}} = \begin{bmatrix} \text{PERS}: & \mathsf{third} \\ \text{NUM}: & \mathsf{sing} \end{bmatrix}.$$

Ordered by subsumption the FSs form a semilattice where the *generalization* of two FSs is the most specific FS which subsumes both FSs.

[1] Due to space limits we decided to omit all formal definitions and to concentrate on why FCA$_{\text{Type}}$ is useful and how it works in principal. A detailed description of FCA$_{\text{Type}}$ and a formal proof that the described procedures are well-defined can be found in [3]. FCA$_{\text{Type}}$ can be obtained from the author on request.

$$\text{Uther} = \begin{bmatrix} \text{CAT} : & \mathsf{np} \\ \text{HEAD} : & \begin{bmatrix} \text{AGR} : & \begin{bmatrix} \text{PERS} : \mathsf{third} \\ \text{NUM} : \mathsf{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix} \qquad \text{sleeps} = \begin{bmatrix} \text{CAT} : & \mathsf{vp} \\ \text{HEAD} : & \begin{bmatrix} \text{FORM} : \mathsf{finite} \\ \text{SUBJ} & \begin{bmatrix} \text{CAT} : & \mathsf{np} \\ \text{HEAD} : & \begin{bmatrix} \text{AGR} : & \begin{bmatrix} \text{PERS} : \mathsf{third} \\ \text{NUM} : \mathsf{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\text{knights} = \begin{bmatrix} \text{CAT} : & \mathsf{np} \\ \text{HEAD} : & \begin{bmatrix} \text{AGR} : & \begin{bmatrix} \text{PERS} : \mathsf{third} \\ \text{NUM} : \mathsf{plur} \end{bmatrix} \end{bmatrix} \end{bmatrix} \qquad \text{sleep} = \begin{bmatrix} \text{CAT} : & \mathsf{vp} \\ \text{HEAD} : & \begin{bmatrix} \text{FORM} : \mathsf{finite} \\ \text{SUBJ} & \begin{bmatrix} \text{CAT} : & \mathsf{np} \\ \text{HEAD} : & \begin{bmatrix} \text{AGR} : & \begin{bmatrix} \text{PERS} : \mathsf{third} \\ \text{NUM} : \mathsf{plur} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Fig. 1: Example lexicon with small untyped feature structures

In order to organize the lexicon, avoid redundancy, and capture generalizations, a strict type discipline has been developed [5]. Types are assigned to FSs and their restrictions and they are organized in a *type hierarchy*, that is, in a finite semilattice. In the AVM representation of a FS types are represented as small indices. In order to restrict the class of admissible FSs, plain type hierarchies are typically enriched by appropriateness conditions [5, 6]. They regulate which features are appropriate for FSs of a special type and restrict the values of the appropriate features. A type hierarchy enriched by appropriateness conditions is called a *type signature*. Fig. 3 (top) shows a small type signature. The appropriateness condition 'CAT : np' at type t_3 means that the attribute CAT is appropriate for structures of type t_3 and its value is restricted to structures of type np or subtypes of np. Appropriateness conditions are inherited downwards. Hence, the subtype t_4 of t_3 inherits the condition 'CAT : np' from t_3. It also inherits the condition 'HEAD : t_9' from t_3, but tightens it up to 'HEAD : t_{10}'.

We can consider a type signature as a *specification* of a set of FSs, namely the set of its totally well-typed FSs. We call a FS *totally well-typed* with respect to a type signature if all its attributes are licensed by the type signature and their values are at least as specific as demanded by the appropriateness conditions. Additionally, all attributes which are prescribed by the appropriateness conditions need to be present. For example, $\begin{bmatrix} \text{CAT} : & \mathsf{np} \\ \text{HEAD} : & \begin{bmatrix} \text{AGR} : & \begin{bmatrix} \text{PERS} : \mathsf{third} \\ \text{NUM} : \mathsf{sing} \end{bmatrix}_{\mathsf{t}_{14}} \end{bmatrix}_{\mathsf{t}_{11}} \end{bmatrix}_{\mathsf{t}_5}$ is totally well-typed w.r.t. the type signature in Fig. 3 (top), but neither $\begin{bmatrix} \text{PERS} : \mathsf{third} \\ \text{NUM} : \mathsf{num} \end{bmatrix}_{\mathsf{t}_{14}}$ nor $\left[\text{CAT} : \mathsf{np}\right]_{\mathsf{t}_5}$ are.

FCAType is a system for the automatic induction of a type signature from a set of untyped FSs. Generally, in the grammar engineering process, the type signature is constructed simultaneously with the rest of the grammar starting with a small grammar covering only a few linguistic phenomena. However, FSs which encode all the necessary phonological, morphological, syntactic, and semantic information of a lexical entry are huge, and type signatures which cover generalizations about such FSs become so complex that a purely manual construction and maintenance is intricate. Therefore, an induction of type signatures which is at least semiautomatic would be most welcome.

The following three grammar engineering tasks are particularly supported by our induction method: (1) corpus-driven grammar development, (2) reuse of grammar resources, and (3) grammar maintenance: Today, we are in the lucky position that we are provided with huge, corpus-extracted lexica. Usually, the entries of these lexica can be seen as untyped FSs, but the manual hierarchical organization of them is not feasible and must be automated. For economical reasons, the reusability of grammatical resources is desirable [7] and should be supported by automatic systems like FCAType. Transferring a grammar from one formalism into another one may also unveil new theoretical insights, cf. [8, 9]. Finally, [10] discusses how error and consistency checking of a large scale untyped grammar can be facilitated by adding an appropriate type signature. However, constructing an appropriate type signature for an already existing grammar is usually a difficult task which requires deep insight into the structural design of the grammar. Therefore, we propose that an expert should intellectually investigate the automatically induced type signature in order to detect errors and inconsistencies in the given grammar which can be an easier task than to build up an appropriate type signature from scratch by hand.

The key idea of our system for the induction of type signatures is to construct the *decomposition semilattice* (DSL) from the untyped input FSs which can be seen as a *featureless* type signature [6]. A similar method is used by Sporleder [11] for a different task, namely for the automatic induction of lexical inheritance hierarchies, i.e. hierarchies of untyped FSs: she reduces the task to a classification problem where the search space is defined by a concept lattice.

2 FCAType Approach

Our aim is to automatically induce an *adequate* type signature from a set of untyped FSs. The type signature is adequate if it specifies the input data, i.e., if for each untyped input structure a totally well-typed, typed version exists (a *typed version* of an untyped FS is identical to the untyped structure, except that types are assigned to each restriction). Therefore, we need types for the input structures themselves and for each restriction of them. Since it is required that the type signature expresses generalizations, we also need types for all possible generalizations about these structures. Moreover, we ask that the generalizations about restrictions of the input structures can be naturally order embedded (via their typed versions) into the ordered set of totally well-typed FSs of the induced type signature.

Further quality criteria for the induced type signatures are *restrictiveness* and *compactness*. Improving the restrictiveness of an induced type signature means reducing the set of totally well-typed FSs, and improving the compactness means reducing the number of types in the type hierarchy.

FCAType is based on the construction of a DSL from a set of untyped FSs. The DSL consists of (1) the FSs themselves, (2) their restrictions, and (3) all gener-

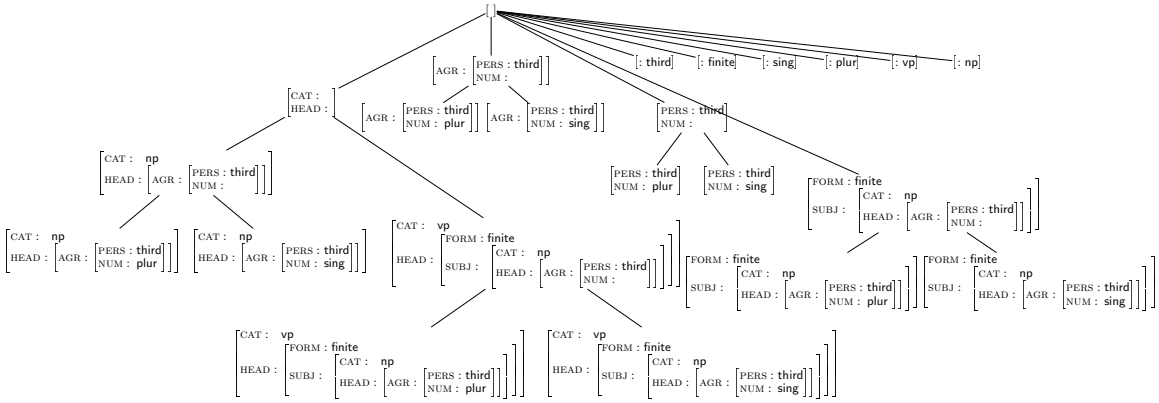

Fig. 2: The decomposition semilattice for the structures of Fig. 1

alizations about structures from (1) and (2). All those structures are partially ordered by subsumption as in Fig. 2.[2]

By assigning a type to each element of the DSL one gains a type hierarchy which provides the required types. That a DSL can be straightforwardly transformed into a well-formed, adequate type signature by inferring adequate appropriateness conditions from the DSL can be seen by comparing the DSL in Fig. 2 with the inferred type signature in Fig. 3 (top) (for details see [3]).[3]

However, the type signature in Fig. 3 (top) still has two undesirable properties: First, the type signature is not very *compact* since some types are unnecessary (the set of totally well-typed FSs would not change substantially if the types $\mathtt{t}_4$,$\mathtt{t}_5$,$\mathtt{t}_7$,$\mathtt{t}_8$,$\mathtt{t}_{10}$,$\mathtt{t}_{11}$,$\mathtt{t}_{16}$, and $\mathtt{t}_{17}$ were deleted). Second, the induced appropriateness conditions are not *restrictive* enough (e.g., the appropriateness condition 'NUM:$\mathtt{t}_1$' permits that the attribute NUM takes a complex FS of type $\mathtt{t}_2$ as value). The first problem is solved by *folding up* the signature and the second one by adding additional types to control the values:

We *fold up* a signature at a type $\mathtt{t}$ by deleting all proper subtypes of $\mathtt{t}$ under the condition that this deletion does not affect the set of well-typed FSs of our signature (up to typing). Hence, folding up a type signature results in a

[2] Actually, instead of computing the set of all generalizations about restrictions of the input structures and ordering them by subsumption, FCAType implements an alternative approach: It generates the *decomposition context* of the input structures and computes its concept lattice which is isomorphic to the DSL (except for the bottom element). Its object set corresponds to the set of restrictions and its attribute set encodes information about paths, path equations, and path values (for details see [3]). This approach enables us on the one hand to use our FCA-submodule which is used by other modules of FCALing, too. On the other hand, we found it useful to have direct access to the decomposed properties of the input structures when it comes to infer appropriateness conditions and to determine folding opportunities. Another alternative would have been to use pattern concepts as described in [12].

[3] If decomposition contexts are employed (see footnote 2), the required appropriateness conditions can be immediately read off from the attribute concepts.

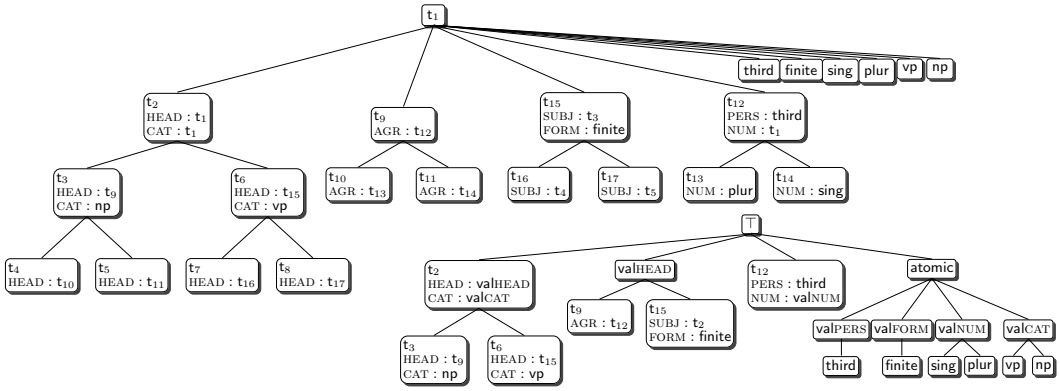

Fig. 3: Unfolded type signature (top) and maximally folded, value-controlled type signature (bottom) for the structures of Fig. 1 (each box shows a type label in the first line followed by the appropriateness conditions)

more compact type signature (the terminology of *folding* is taken from [6]). In principle we have to consider two different folding opportunities. *Atomic folding opportunities* result from the distribution of the atomic types in the FSs. All folding opportunities of the type signature in Fig. 3 (top) are atomic. *Structural folding opportunities* are rare and therefore not discussed here (for details see [3]). In FCAType, we have chosen an easy way to take advantage of all atomic folding opportunities. The key idea is that manually constructed type signatures mainly encode information about the general structure of the lexical FSs. Hence, we have chosen to simplify the input structures in a first step by replacing each atomic value with a generic marker $av\_$. Starting from the DSL of these simplified structures, we construct the corresponding type signature. It covers all structural aspects of the untyped FSs, and it lays the foundation for our target signature. In the next step, the atomic values are taken into account and the appropriateness conditions are tightened up, wherever possible. Finally, all atomic values are arranged under a type 'atomic' and meanwhile ordered by the features they can be values of. In the resulting type signature, no atomic folding opportunities are left, thanks to the preceding simplification of the input structure. The fact that the rigorous simplification of the input structures can theoretically result in type signatures which are too heavily folded up can be captured by additionally induced *feature co-occurrence restrictions* [3, 9].

The deficient restrictiveness of type signatures of DSLs is caused by appropriateness conditions which do restrict values of an attribute to structures of the most common type. In such cases, the type signature has at least one recursive type and thus the set of totally well-typed FSs is infinite [6]. Therefore, we have decided to introduce an artificial type whenever such a situation would occur and to adjust the affected appropriateness conditions. By inserting those artificial types our type signatures become more restrictive.

Fig. 3 (bottom) shows the maximally folded, value-controlled type signature induced by FCAType from the input data of Fig. 1. A detailed description of the induction process is given in [3].

3 Conclusion

It would be interesting to combine our approach with that of Sporleder and to use our DSLs as input for her classification problem, since they encode much more detail than her lattices, and they can be used as basis for the construction of type signatures.

But also from a theoretical point of view, our observations are interesting: The set of typed FSs corresponding to a type signature is well understood [5, 6]. However, a lot of work has still to be done to answer the question which type signature models a set of untyped FSs best. In our opinion, a closer investigation of DSLs and the related type signatures can provide answers. In [3] these questions are discussed in greater detail: A number of alternative type signatures induced from DSLs are presented and their properties are compared. Additionally, type constraints are induced which either restrict the admissible path-equation relations or express feature co-occurrence restrictions.

References

1. Sag, I., Wasow, T.A.: Syntactic Theory: A Formal Introduction. CSLI, Stanford (1999)
2. Minsky, M.: A framework for representing knowledge. In Winston, P.H., ed.: The Psychology of Computer Vision. McGraw-Hill (1975) 211–277
3. Petersen, W.: Induktion von Typsignaturen mit Mitteln der Formalen Begriffsanalyse. PhD thesis, University of Düsseldorf (in submission process)
4. Shieber, S.M.: An Introduction to Unification-Based Approaches to Grammar. CSLI, Stanford (1986)
5. Carpenter, B.: The Logic of Typed Feature Structures. Cambridge Tracts in Theoretical Computer Science 32. CUP (1992)
6. Penn, G.: The Algebraic Structure of Attributed Type Signatures. PhD thesis, School of Computer Science, Carnegie Mellon University (2000)
7. Arnold, D.J., Badia, T., van Genabith, J., Markantonatou, S., Momma, S., Sadler, L., Schmidt, P.: Experiments in reusability of grammatical resources. In: Proceedings of 6th EACL. (1993) 12–20
8. Kilbury, J., Petersen, W., Rumpf, C.: Inheritance-based models of the lexicon. In Wunderlich, D., ed.: Advances in the Theory of the Lexicon. Mouton de Gruyter (2006) 429–477
9. Petersen, W., Kilbury, J.: What feature co-occurrence restrictions have to do with type signatures. In: Proceedings of FG/MOL-05, Edinburgh. (2005) 125–139
10. Wintner, S., Sarkar, A.: A note on typing feature structures. Computational Linguistics **28**(3) (2002) 389–397
11. Sporleder, C.: Discovering Lexical Generalisations. A Supervised Machine Learning Approach to Inheritance Hierarchy Construction. PhD thesis, School of Informatics, University of Edinburgh. (2003)
12. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. LNCS **2120** (2001) 129–142