Inheritance-based models of the lexicon

James Kilbury, Wiebke Petersen, Christof Rumpf

1 Emergence of the present view of the lexicon

A rapid and remarkable development took place within computational linguistics in the years immediately following the introduction of unification-based models of language, in particular *Lexical Functional Grammar* (LFG) and *Generalized Phrase Structure Grammar* (GPSG), which employ feature structures to represent linguistic information. By the end of the 1980s a consensus had emerged, according to which the lexicon, which pairs word forms with feature structures, constitutes the main repository of information in a language. Furthermore, hierarchical structuring had come to be viewed as an essential aspect or perhaps even the most salient characteristic of the lexicon (cf. Briscoe et al. 1993).

GPSG, as conceived in Gazdar & Pullum (1982) and even in Gazdar et al. (1985), still largely represents the older, dichotomous view of the lexicon. Here major aspects of linguistic structure were encoded in *syntactic* rules, many of which later came to be regarded as stating the possible complement structures of verbs, i.e. *lexical* information. Later developments in the treatment of subcategorization are only alluded to in a footnote (cf. Gazdar et al. 1985: 107). While the question "How is a classification imposed on the content of the lexicon by the system of features" is raised (p. 13), the answer of GPSG does not explicitly model the hierarchical inheritance relations inherent in lexical classifications. Rather, these relations are captured in logical *feature co-occurrence restrictions* (FCRs) and *feature specification defaults* (FSDs), the latter of which, prophetically, are nonmonotonic.

From the start the *lexicalist* orientation was prominent in LFG (cf. Bresnan 1982, therein Kaplan & Bresnan 1982) and reached a peak in the radical lexicalism of Karttunen (1986), which uses the framework of categorial grammar to shift the entirety of linguistic description to the lexicon. The move toward the lexicalist view was independent of *hierarchical* modelling, which emerged in other work. In particular, Flickinger (1987) pioneered the explicit description of relations between English verb classes in terms of inheritance hierarchies. On a separate front, de Smedt (1984) initiated the use of inheritance-based representation formalisms to capture the structure of inflectional classes. Practical advantages of hierarchical lexica quickly became apparent and include the economical representation, integrity, homogeneity, and updating of data. The grammar formalism *PATR-II* (cf. Shieber et al. 1983) provides *templates* as an indispensable formal device for stating inheritance relations in a hierarchy, the consequences of which are clear to the authors (p. 62):

> But our notation does not only allow convenient abbreviations; it also plays an important role in the linguist's use of the formalism. [...] perhaps most importantly, grammar writers can use the notational tools to express generalizations they could not state in the "pure" unification notation of the formalism.

Head-driven Phrase Structure Grammar (HPSG), as presented in Pollard & Sag (1987) carries over much of GPSG but restructures the model in terms of the hierarchical lexicalist framework. The introduction of the *sign* as a uniform data structure for the representation of both lexical and phrasal information, together with the integration of all linguistic levels within the sign, provides the last means needed in order to state all information about a language within an inheritance hierarchy defining relations between signs.

Crucially, HPSG adopts no obvious counterpart for the FCRs of GPSG, although the *conditional feature structures* (Pollard & Sag 1987: 43) of HPSG could have been employed for this purpose. Instead, the HPSG strategy for avoiding lexical redundancy lies in the use of inheritance hierarchies: "Structuring the lexicon in terms of an inheritance hierarchy of types has made it possible to factor out information common to many lexical entries, thereby greatly reducing lexical redundancy" (Sag & Wasow 1999: 202). The informational domain consists of typed feature structures, where the types serve two functions: On the one hand they allow access to embedded feature structures appearing as values of features; this permits the formulation of generalizations about such substructures. On the other hand, the types bear appropriateness conditions which restrict the set of feature structures of this type; such statements are feature-type pairs. By ordering the types in an inheritance hierarchy, the so-called *type signature*, in which appropriateness conditions are inherited, further redundancies are avoided.

It was clear to linguists that HPSG had replaced the FCRs of GPSG with inheritance hierarchies of types, but the relations between these formal devices were misunderstood and hardly questioned. Clearly, the devices had to be related in some way, but no formal framework was available within which the relations could be made explicit. Gerdemann & King (1994, 1993) present a procedural method for transforming a type signature so that it expresses an FCR. With *Formal Concept Analysis* (FCA, cf. Ganter & Wille 1999) a general framework is now available which allows the equivalence of the devices to be explained in a transparent and declarative fashion, which we will show in §3.3.1 below.

The subsequent development of HPSG and its modelling of the lexicon is in some ways paradoxical, showing an accumulation of rather ad hoc and baroque devices like *functional values* (cf. Pollard & Sag 1994) on the one hand, and a powerful striving toward homogeneity on the other. The role of the type hierarchy in HPSG has recently been extended to deal with morphological patterns of word formation (cf. Riehemann 1998, König 1999), and the homogeneity of HPSG is also furthered by the efforts of Bouma et al. (2000) to replace the isolated formal device of lexical rules with constraints within the type hierarchy. The motivation for this move turns out to lie particularly in the desire of HPSG proponents to avoid the nonmontonicity of lexical rules, whereby "by default, information in the input [which is not changed by the rule] is assumed to be included in the output" (p. 55).

This highlights the role of nonmonotonicity as the principal point of contention remaining after the synthesis achieved by HPSG in 1987. The usefulness of nonmonotonic devices, like *overwriting* in the lexical rules of PATR-II (cf. Shieber 1986: 62 reporting earlier work) is apparent, but reservation is also noted (Shieber 1983: 63):

> The linguistic status of templates and lexical rules needs to be determined. One could adopt a simple view and use lexical rules every time the power of pure [i.e. monotonic] unification with a template does not suffice, i.e. whenever changing to the graph structure of lexical entries requires more than the simple addition of arcs and nodes. It would be more gratifying, though, if one had a clearer correspondence between the use of notational tools, on the one hand, and classes of linguistic regularities, on the other.

Indeed, nonmonotonic devices like *completeness* in LFG and FSDs in GPSG were already in wide linguistic use by the mid 1980s. Gazdar (1987) points to the pervasiveness of phenomena in natural language that call for nonmonotonic means to describe continua of regularities, subregularities, and exceptions. The language DATR for lexical knowledge representation (cf. Evans & Gazdar 1989, 1996) can be seen as a natural development in this context. While Kilbury et al. (1991) propose the use of DATR in conjunction with feature-based unification formalisms, Krieger & Nerbonne (1993) adamantly reject nonmonotonicity in such formalisms. Viewed in retrospect, neither of the two positions was truly successful, since DATR practitioners never produced a fully convincing integration of DATR with formalisms like HPSG that permitted tractable implementations, while the critics of DATR were unable to dissuade many HPSG followers from the attractiveness of nonmonotonicity for capturing linguistic generalizations. Symptomatic of this situation was the series of proposals for a synthesis in the form of some kind of default unification (cf. Lascarides & Copestake 1999 for a late position), which remains inefficient and has not been adopted in the canon of HPSG but has been integrated in the *Linguistic Knowledge Building* (LKB) system (cf. Copestake 2002).

Thus, the turn of the millenium reveals widespread consensus about the fundamental organization of the lexicon as an inheritance hierarchy accompanied by disagreement over the role of nonmonotonicity, which has remained attractive because of the stronger generalizations and simpler hierarchies that it permits. An issue which has been largely skirted centers on nonmonotonicity as a property of *unification* or *inheritance*. Of course, inheritance can be implemented with unification, but the nonmonotonicity of DATR involves only inheritance and is unrelated to unification. Regarding default unification the LKB implementation “assumes that defaults are only relevant with respect to an inheritance hierarchy” (Copestake 2002: 204).

In §2 below we introduce our framework QType, which provides for a restriction of nonmonotonicity to inheritance within type signatures; the operation of unification at parsetime is in turn entirely monotonic. This captures the expressiveness of nonmonotonicity for linguistic generalizations involving subregularities and exceptions while preserving the efficiency and transparency of monotonic unification for parsetime processing. We thereby seek to combine advantages of DATR with frameworks like HPSG within a single integrated formalism.

On an entirely separate front of computational linguistics, the past two decades have witnessed a rapid growth of corpus-based empirical work for lexicon development (cf. Boguraev & Pustejovsky 1996). Parallel to this there has been a strong emphasis on inductive procedures and learning algorithms (cf. Daelemans & Durieux 2000), but this has been almost entirely disjunct from theoretical work on feature-based formalisms, with no bridge that connects the two domains.

Here, too, in §3 below this paper aims at a synthesis. Modern HPSG-like descriptions of languages have become so complex that their type hierarchies can hardly be constructed manually, and this problem exists independently of the extraction of large bodies of data from corpora. Our work within the framework of Formal Concept Analysis presented below introduces essential techniques that help to overcome the difficulties of developing highly complex type hierarchies by hand. The advantages of such automatically induced hierarchies lie in their coherence, non-arbitrariness, and compatibility with algorithms for inserting new lexical items. These techniques of §3 can in turn be employed for the induction of type signatures like those of §2 in QType.

The rest of this paper will thus focus on our formalism QType (§2), its special use of nonmonotonicity (§2.11), and our use of FCA in particular to explicate the relations between FCRs and type signatures (§3.3.1) and in general for the automatic induction of inheritance hierarchies for lexical description (§3).

2 QType: A grammar-development environment

2.1 Survey of QType

QType is a grammar-development environment for constraint-based (i.e., unification-based) grammars. The core of QType consists of typed feature structures, type constraints, and relational constraints. These devices suffice for the development, e.g., of HPSG-grammars, and ‘parsing’ then reduces to constraint solving, where all the information of categories is encoded solely in their associated feature structures, which are unified with other feature structures. In order to permit the use of established parsing techniques like left-corner parsing with linking based on a context-free phrase-structure skeleton, QType also allows syntax rules and lexical entries to be specified. Lexical rules with copying via nonmonotonic unification supplements this inventory. The most important feature is the possibility of employing default inheritance in the type signature, i.e., the description of the type hierarchy. Type conflicts are resolved with a strategy according to which more specific information has precedence. The nonmonotonic type signature is compiled offline into a monotonic counterpart, so that only the efficient operation of monotonic unification needs to be computed at runtime. In the following, the monotonic component of QType will first be presented as a basis for the comparison with related formalisms, before the nonmonotonic extension is introduced.

2.2 The type signature

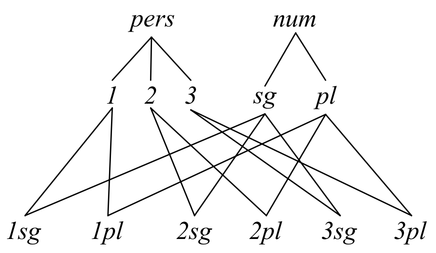
The type signature serves to define the classes of well-formed or possible feature structures of a grammar. A typed feature structure consists of a type and a set of attribute-value pairs, where the associated pairs must be specified explicitly for each type. Types and attributes are atomic identifiers, and values are feature structures. Atomic values are regarded as feature structures which are not associated with any attribute-value pairs and therefore consist only of a type. The type system of QType will now be described before attribute-value pairs and their inheritance are discussed in the subsequent section.

2.2.1 The type system

The type system of QType consists of a partial order on types which, in the monotonic component, is interpreted as subsumption. The order can be represented with a directed acyclic graph whose nodes are labelled with type names and whose edges denote subsumption. There is exactly one given node which has no predecessor, the node for the most general type *top*, which subsumes all other types. Furthermore, there can be any finite number of minimal types. These are all maximally specific in the sense that they subsume no other type. Between these extremes lie a finite number of nonminimal types which cover the set of minimal types. Disregarding redundant nodes in the graph, the total number of possible types is bounded by the power set (i.e., the set of all subsets) of the minimal types. The description of the type system consists of the definition of the relation 'immediate subtype', represented by '*supertype* >> *list_of_immediate_subtypes*'.

(1) A simple type hierarchy

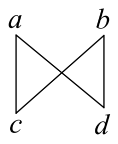
pers >> 1, 2, 3.
num >> sg, pl.
1 >> 1sg, 1pl.
2 >> 2sg, 2pl.
3 >> 3sg, 3pl.
sg >> 1sg, 2sg, 3sg.
pl >> 1pl, 2pl, 3pl.



The relation ‚subtype' is defined with the reflexive transitive closure of the graph induced by the relation ‚immeditate subtype'; it consequently consists of every type paired with itself as well as every pair of types which are connected by a directed path in the graph. Type unification is the binary operation on types $\sqcup$: $T \times T \rightarrow T$, where $t_1 \sqcup t_2 = t_3$ is defined when t_3 exists as the most general subtype of t_1 and t_2. The type t_3 is called the *most general unifier* (mgu). In (1) we have$1 \sqcup pl = 1pl$, but $sg \sqcup pl$ is undefined. In contrast to other formalisms (e.g., ALE; cf. Carpenter & Penn 1999) QType does not require the type system to form a *bounded* *complete partial order* (bcpo) and thus a semi-lattice. As a consequence, type unification is nondeterministic, i.e., for a pair of types there can be more than one most general unifier.

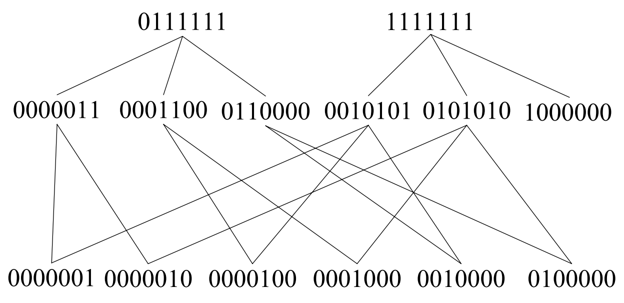
(2) A type hierarchy violating the bcpo condition

a >> *c*.
a >> *d*.
b >> *c*.
b >> *d*.



In (2) there is no single most general unifier of *a* and *b*, but rather two: *c* and *d*. Our implementation of the type system is based on a bit-vector encoding of the types (cf. Aït-Kaci et al. 1989, Bernhardt 2001), which opens an extensional perspective since the bit vector for a given type represents all the subsumed minimal types that can be regarded as the extension of that type. All types have bit vectors whose length is identical with the total number of minimal types, where each minimal type has a defined position in the vector. In the bit vector of a given type, all the bits of the subsumed minimal types are set to 1, and all others to 0. Accordingly, every minimal type has a bit vector in which exactly one bit is set to 1, whereas for the most general type *top* all bits are set to 1. In the case of unary branchings and more complex redundancies in graphs like examples (1) and (2) this encoding leads to the problem that types can no longer be distinguished by their bit vectors because the sets of subsumed minimal types are identical. We solve this problem through the addition of further, so-called 'dummy' types as the following example shows:

(1a) Bit-vector encoding of the type hierarchy in (1)



The type with bit vector 1000000 in (1a) is a dummy type which is necessary to distinguish the two most general types *num* and *pers*. Without the dummy type these would have identical bit-vector representations since they would cover the same set of minimal types. It is furthermore necessary to add an additional dummy type as an immediate subtype of *pers* to allow a common supertype of *num* and *pers* that has a bit vector distinct from that of *num*, but we have avoided this in (1a) for the sake of simplicity. The bit-vector coding of types permits simple and efficient computation of operations on types through logical operations on bit vectors.

(3) Let b_i be the bit-vector representation of type t_i ; then the following holds:

(3.1)	unification	$t_1 \sqcup t_2$	$\Leftrightarrow$	b_1 AND b_2
(3.2)	generalization	$t_1 \sqcap t_2$	$\Leftrightarrow$	b_1 OR b_2
(3.3)	complement	$\neg t$	$\Leftrightarrow$	NOT b
(3.4)	subsumption	$t_1 \sqsubseteq t_2$	$\Leftrightarrow$	b_2 =? (b_1 AND b_2)

The unification of two types is normally computed with the AND operation on the corresponding bit vectors, where only the bits remain which are set to 1 in the same positions of both vectors. This corresponds to the intersection of the sets of minimal types that constitute the extensions of the types to be unified. If the intersection is empty, a bit vector results which contains only zeros, and this is equivalent to failure of unification. For generalization an OR operation is computed, where all those bits remain which are set to 1 in a given position in one of the vectors. This corresponds to the union of the sets of minimal types. The complement of a type is built simply by switching all the bits: 0 is replaced with 1 and *vice versa*. If the result of unifying one type with another is identical to the second type, then the former subsumes the latter and is more general. The operator '=?' tests the identity of two bit vectors.

It has already been mentioned that unification in the type system of QType is nondeterministic. Nevertheless, the AND operation always produces a unique result. From this it follows that the operations on bit vectors may produce vectors that do not correspond to any defined type of the signature. We call such vectors 'virtual types' (Bernhardt 2001). They can be resolved in a disjunction of defined types by identifying the minimal sets of most general types that cover all the subsumed minimal types. The same phenomenon appears in the case of the complement, where the same solution strategy is employed.

2.2.2 Appropriateness conditions

The set of corresponding attribute-value pairs (henceforth 'AV pairs'), also called feature-value pairs, for a class of feature structures of a given type is defined by the partial function ‚Appropriateness': *Types* × *Attributes* → *Types*, which is introduced in the QType type signature with the symbol '::'.

(4) Description of a class of feature structures and its most general AV-matrix

agr :: num:numerus, pers:person, gen:genus.
numerus >> sg, pl.
person >> 1st, 2nd, 3rd.
genus >> masc, fem, neut.

$$\begin{bmatrix} \mathit{agr} \\ \mathit{num} : \mathit{numerus} \\ \mathit{pers} : \mathit{person} \\ \mathit{gen} : \mathit{genus} \end{bmatrix}$$

In (4) a class of feature structures of type *agr* is introduced for which the three AV pairs are appropriate, e.g., *agr* × *num* → *num*. Possible combinations give 2×3×3 = 18 different instances for this class, but there can be classes with an enumerably infinite number of instances:

(5) Description of a class *list* and an instance of length two

list >> elist, nelist.
nelist :: *head:top, tail:list.*

$$\begin{bmatrix} \mathit{nelist} & \\ \mathit{head} : 1 & \\ \mathit{tail} : & \begin{bmatrix} \mathit{nelist} \\ \mathit{head} : 2 \\ \mathit{tail} : \mathit{elist} \end{bmatrix} \end{bmatrix}$$

This recursive definition of the type *list* allows lists of arbitrary length and with elements of arbitrary type to be defined, including the type *list* itself. The possible instances for this class correspond to at least the set ℕ of natural numbers. Thus, the type signature defines a finite number of classes of feature structures, but these potentially may have indefinitely many instances.

2.2.3 Inheritance

The AV pairs of a type are inherited by its subtypes, which for their part may introduce further AV pairs or may sharpen the inherited values. A specificity hierarchy thus arises in which supertypes subsume their subtypes. In a framework with monotonic inheritance the AV pairs of subtypes must be consistent with those which are inherited; otherwise, one speaks of 'type clashes'. In QType type clashes are allowed, but (more) specific information takes precedence over inherited information. This point will be discussed in §2.7, but first the feature logic of QType in the context of monotonic inheritance will be introduced.

2.3 Feature logic

A feature logic is a description language for feature structures that has a syntax and a compositional semantics. It can be used to create instances for the classes which are defined in the type signature. The feature logic of QType has the following descriptive elements, where *Desc* denotes the set of descriptors:

(6) Syntax of QType's feature logic

(6.1)	variables	$Vars \subset Desc$
(6.2)	types	$Types \subset Desc$

$f \in Feat$, $d_i \in Desc$:

(6.3)	feature-value pairs	$f : d \in Desc$
(6.4)	conjunction	$d_1 \,\&\, d_2 \in Desc$
(6.5)	disjunction	$d_1 \,;\, d_2 \in Desc$
(6.6)	complement	$-(d) \in Desc$
(6.7)	relational constraints	$@r(d_1, \ldots, d_n) \in Desc$

The interpretation of this feature logic is given by a function $mgu : Desc \rightarrow 2^{FS}$. Because of disjunction and negation the function returns values from the power set 2^{FS} of those feature structures which are possible instances of the classes defined in the type signature.

(7) Semantics of QType's feature logic

$v \in Vars$, $t \in Types$:

(7.1) $mgu(v) = \{top\}$

(7.2) $mgu(t)$ is given directly by the definition of t in the type signature.

$f \in Feat$, $t_i \in Types$, $d_i \in Desc$:

(7.3) $mgu(f : d) =$

$\{FS_1 \mid FS_1 \in mgu(t) \text{ where } t \in lub\_pf(f : d)\} \sqcup_{\text{Set}}$

$\{[f : FS_2\,] \mid FS_2 \in mgu(d)\}$

(7.4) $mgu(d_1 \,\&\, d_2) = mgu(d_1) \sqcup_{\text{Set}} mgu(d_2)$

(7.5) $mgu(d_1 \,;\, d_2) = mgu(d_1) \cup mgu(d_2)$

(7.6) $mgu(-v) = \varnothing$

(7.7) $mgu(-t) = max(\bigcup mgu(t_i))$ for all t_i with $t \sqcup t_i = \varnothing$

(7.8) $mgu((\text{-}f) : d) = mgu(f : \text{-}d)$

(7.9) $mgu(\text{-}(f : d)) = mgu(f : \text{-}d)$

(7.10) $mgu(-(d_1 \,\&\, d_2)) = mgu(-d_1 \,;\, -d_2)$

(7.11) $mgu(-(d_1 \,;\, d_2)) = mgu(-d_1 \,\&\, -\, d_2)$

(7.12) $mgu(@r(d_1, \ldots, d_n)) = mgu(def_1(r(d_1, \ldots, d_n))) \cup \ldots \cup mgu(def_m(r(d_1, \ldots, d_n)))$

The interpretation of descriptions consisting of a variable (7.1) gives a set of feature structures with the special type *top* as its only element. Descriptions consisting of a type name give the feature structure of the type according to its definition in the type hierarchy.

(8) Interpretation of descriptions consisting of a single type

$$mgu(agr) = \left\{ \begin{bmatrix} agr \\ num : numerus \\ pers : person \\ gen : genus \end{bmatrix} \right\}$$

Feature-value pairs occur in descriptions as attribute-description pairs, whose interpretation (7.3) requires that the most general types be determined for which the pair is appropriate. The latter is achieved with the function *lub-pf* (least upper bound for polyfeatures). QType allows polymorphic features (‘polyfeatures’) such that a given feature can be appropriate for several disjoint types, possibly with different values. Other frameworks (e.g., ALE) require unique ‘feature introduction’. The value of the feature *f* of the candidates t_i that have been found must be compatible with the description *d*. This is guaranteed by the operation $\sqcup_{\text{Set}}$, which unifies the feature structures of the resulting set pairwise. Conjunctions of descriptions (7.4) return for each member of the conjunction a set of feature structures whose elements must be unified pairwise. Our implementation uses an incremental strategy for this which—according to our experience—leads to early failure in cases of inconsistency and thus results in greater efficiency. The incremental procedure is based on the principle that, for a series of conjunctions d_1 & …& d_n each interpretation $mgu(d_i)$ must be unified with the unifications of the interpretations $(mgu(d_1) \sqcup_{\text{Set}} \ldots) \sqcup_{\text{Set}} mgu(d_{i-1})$, i.e., intermediary results are accumulated and available as constraints for following computations. Disjunctions of descriptions (7.5) lead to union of the sets of feature structures given by the interpretations of the members of the disjunction.

The semantics of negated descriptions must be specified separately for each kind of description. The negation of a variable (7.6) corresponds to the complement of *top*, i.e., the empty set. The negation of a type *t* (7.7) consists of the maximal elements with respect to subsumption in the set of feature structures of all types t_i that are inconsistent with *t* and thus have no subtype in common with *t*. In feature-description pairs (7.8, 7.9) the negation is always shifted to the description. This excludes an interpretation according to which, for negated features, the set of all types is given which do *not* have this feature. The negation of complex descriptions (7.10, 7.11) is resolved by shifting the negation inward according to DeMorgan's laws. The semantics of relational constraints is the union of the feature structures corresponding to their definitions, where the definitions are nothing other than the descriptions just defined in (6) and (7) (cf. §2.8 for a more detailed discussion).

Aside from the differences in the type system, the interpretation of descriptions in QType in a way is like that in ALE since most general unifiers are computed. In CUF (cf. Dörre & Dorna 1993) and ConTroll (cf. Götz et al. 1997), in contrast, unifiers are computed which always are minimal types. A nonminimal type accordingly represents the disjunction of all minimal types that it covers, i.e., the most specific unifiers (*msu*). Pollard & Sag (1994) and Penn (2000) call these representations 'sort resolved', where 'sort' refers to types. Such differing definitions have grave consequences for grammar development, as the following example is intended to show. The interpretation of *mgu*(*agr*) results in a set with a single element (see (8)), whereas the interpretation of *msu(agr)* results in a set with 18 elements, because every value of every feature has to be a minimal type.

(9) Interpretation of types with the function *most specific unifier*

$$msu(agr) = \left\{ \begin{bmatrix} agr \\ num : sg \\ pers : 1st \\ gen : masc \end{bmatrix}, \begin{bmatrix} agr \\ num : pl \\ pers : 1st \\ gen : masc \end{bmatrix}, \begin{bmatrix} agr \\ num : sg \\ pers : 2nd \\ gen : masc \end{bmatrix}, \ldots, \begin{bmatrix} agr \\ num : pl \\ pers : 3rd \\ gen : neut \end{bmatrix} \right\}$$

2.4 Type constraints

The framework presented thus far consists of a type signature and an interpretation function for descriptions. See Rumpf (forthcoming) for an implementation of finite-state automata within this framework. Since variables are not used in the description language for the type signature, it follows that co-indices are excluded, although they are required for modelling HPSG schemata and subcategorization in the type signature. To solve this problem we extend the type signature with type constraints. The latter are partial functions *Desc* → *Types* for which we use the notation *t* ::= *d* in QType. With type constraints the classes of feature structures defined in the type signature can be refined to any desired degree and in such a way that the additional information of a type is inherited by its subtypes. However, the refinement must remain within bounds set by appropriateness conditions, i.e., no new feature-value pairs may be introduced.

(10) Simplified grammar fragment modelling the *head feature convention* of HPSG

headed-phrase :: *HEAD*:*sign*, *DTRS*:*dtrs*.
headed-phrase ::= *HEAD*:*$1* &
DTRS:*HEAD-DTR*:*HEAD*:*$1*.

$$\begin{bmatrix} \mathit{headed-phrase} \\ \mathit{HEAD}: \boxed{1} \\ \mathit{DTRS}: \mathit{HEAD-DTR}: \mathit{HEAD}: \boxed{1} \end{bmatrix}$$

The symbol '$' in (10) is used to distinguish variable names from types with the same name. The coindexation is made possible only through the use of type constraints with which one can use expressions in feature logic to address constituents embedded at arbitrary depth in feature structures; this would not be possible with a simple extension of the description language for type signatures with variables. Type constraints are comparable to implicational constraints like those in ConTroll (cf. Götz et al. 1997) but slightly weaker since the left-hand side of implicational constraints consists of arbitrary descriptions and every feature structure subsumed by it must be consistent with the arbitrary description on the right-hand side of the implicational constraint. Furthermore, type constraints provide a new means for representing nondeterminism since expressions in feature logic may contain disjunction and negation. Cyclic feature structures can be defined as well.

2.5 Relational constraints

Relational constraints (RCs) extend a constraint-based grammar-development environment with a general constraint-logic programming (CLP) language for typed feature structures. In other frameworks RCs are also called *definite clauses* (ALE, cf. Carpenter & Penn 1999) or *sorts* (CUF, cf. Dörre & Dorna 1993). An RC $r(d_1,\ldots,d_n) \in Desc$ is a relation r over feature-logic descriptions $d_1,\ldots,d_n$ which, like all descriptions, are mapped onto the set 2^{FS}. In contrast to other descriptive elements of our feature logic which refer exclusively to classes defined in the type signature, RCs are defined outside the type signature (although they conform to it) and thus constitute an independent system in which further inheritance relations can be defined. So one must distinguish between the definition and the use of a relational constraint as a feature-logic expression, where they are preceded by the symbol '@' to distinguish them from types with the same name. The syntax for definitions of RCs appears as follows:

(11) Syntax of relational constraint definitions

$r(d_1,\ldots,d_n)$ #> d_0. with $d_i \in Descr$, $r \in String$, $n \geq 0$

There can be more than one definition for a relation r with arity n, and these definitions are then interpreted disjunctively (see (7.12)). The practical use of the parameters $d_1,\ldots,d_n$ is manifold. They can trigger the matching of a specific constraint definition and transport information between a feature-logic expression, where the constraint appears, and its definition.

A subset of the RCs is comparable with the templates of PATR-II (cf. Shieber 1986), which may be regarded as abbreviations for complex feature structures. Since templates can themselves be used in the definition of templates, they provide a means — indeed, in the case of PATR-II the only means — of defining inheritance hierarchies. (12) shows the translation into QType of an example from Shieber (1986: 57):

(12) Relational constraints defined as templates

> *verb #> cat:v.*
> *finite #> @verb & head:form:finite.*

The constraint *finite* is defined with the constraint *verb*. In contrast to PATR-II templates, RCs can be parametrized with descriptions. Coupled with recursive data structures like lists, they thereby achieve a status that goes far beyond the abbreviatory character of templates. A constraint-logic programming language arises which is comparable with „Pure Prolog“ and with which programs involving infinite sets of feature structures can be defined, e.g., in order to implement Turing machines. This does not mean that relational constraints are essential in order to implement Turing machines with feature structures; they just make it simpler. Keller (1993) describes how „Negated Regular Rounds-Kasper Logic“ (NRRK Logic) can be used as an extension of PATR-II to do the latter. NRRK Logic is an extension of Rounds-Kasper Logic (Kasper & Rounds 1986), one of the first studies to provide a feature logic for untyped feature structures with a denotational semantics. Johnson (1988) shows an implementation of Turing machines with unary syntax rules as state transitions and lists as tapes. Nevertheless, relational constraints can be used to define arbitrary relations over lists:

(13) A relational constraint defined for lists of arbitrary length

> *append(elist, $L) #> $L.*
> *append((first:$H & rest:$T), $L) #>*
>   *first:$H & rest:@append($T, $L).*

Recall that ‘$’ distinguishes variables from other descriptions. The interpretation of the description *append($Prefix, $Suffix) & first:1 & rest: (first:2 & rest: (first:3 & rest:elist))* produces the disjunction of all decompositions of the list <1, 2, 3> into prefixes and suffixes as instances of the parameters *$Prefix* and *$Suffix*. Type constraints can be used to embed RCs in the inheritance hierarchy of the type signature, e.g., in order to permit constraint-based parsing as the following example shows:

(14) A type constraint using a relational constraint

> *sign >> phrasal_sign, lexical_sign :: phon:phonlist.*
> *phrasal_sign >> left:phonlist & right:phonlist.*
> *phrasal_sign ::= phon:@append($L, $R) & left:$L & right:$R.*

The type *phrasal_sign* is treated here as a binary tree, where the type constraint *phrasal_sign* ensures that the value of the feature *phon* is always the concatenation of the values of the features *left* and *right*. Since undirected constraint-based parsing cannot compete for efficiency with established parsing strategies, QType provides the possibility of specifying a grammar with context-free syntax rules and a lexicon, which can then be processed using a left-corner parser with linking.

2.6 Phrase-structure rules and lexical entries

As mentioned above, the phrase-structure rules and lexical entries of QType are provided to allow for efficient parsing. Phrase-structure rules are written as context-free phrase-structure rules which define sets of trees whose nodes are associated with feature structures.

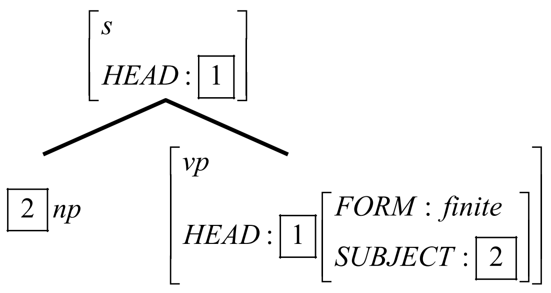
(15) Syntax of phrase-structure rules in QType

> d_0 => d_1 ^ ... ^ d_n. with $d_i \in$ *Descr*

The symbol ‚=>' denotes immediate dominance of the mother node d_0 over the daughter nodes d_1 ^ ... ^ d_n, while ‚^' denotes concatenation and therefore linear precedence among the daughters. Since the rules are processed using a left-corner parser with linking (cf. Covington 1993), left-recursive rules do not present a problem. The linking relation is computed automatically during compilation of the grammar. The following example shows the adaptation of a phrase-structure rule for sentences S $\rightarrow$ NP VP from Shieber (1986: 25):

(16) A phrase-structure rule for main clauses in QType and the corresponding syntax tree

> *s & HEAD:$H => np & $NP ^*
> *vp & HEAD:($H & SUBJECT:$NP & FORM:finite).*



The Head-Feature Principle of HPSG (cf. Pollard & Sag 1987, 1994) has been explicitly stated in the rule here. The application of rules follows the left-corner alogrithm: starting with a top-down prediction, a bottom-up step involving lexical access is triggered by a linking relation that determines which constituents are useful beginnings of a phrase dominated by the top-down prediction. So, parsing involves matching the strings of lexical items and unification of the feature structures at the nodes of trees that are adjoined.
We decided to distinguish syntax rules and lexical entries to make the parser more efficient and to facilitate the application of lexical rules to lexical entries. The format for lexical entries pairs strings with descriptions of feature structures and appears as follows:

(17) Syntax of lexical entries in QType

s -> d. where $s \in$ *String*, $d \in$ *Descr*

Two concrete examples of lexical entries are given here:

(18) Two lexical entries and a relational constraint used as a template

```
man    ->  cnoun.
sleep  ->  @vbase.
vbase  #>  v & HEAD:FORM:base.
```

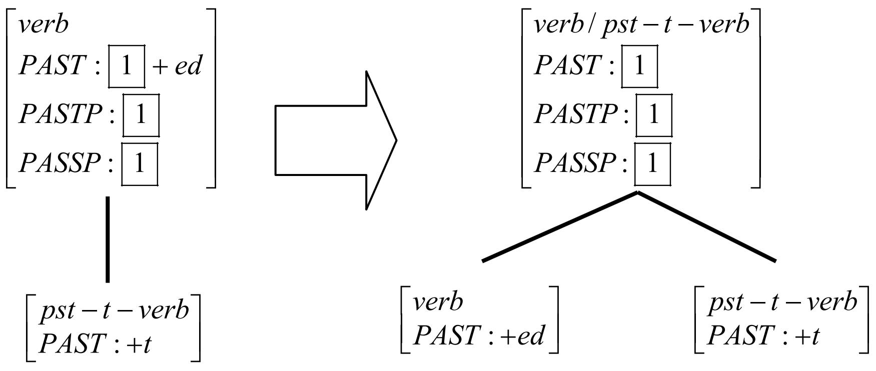
The word *man* has the type *cnoun* for common nouns in its description, while the description of the word *sleep* contains a relational constraint which functions here as an abbreviatory template and specifies the type *v* as well as the value *base* for the feature path *HEAD:FORM*. In order to take advantage of the power of inheritance in the type signature, good practice would be to define almost all information of lexical classes down to specific lexical entries in the type signature. This would lead to lexical entries in the format (17) that pair a string with a single type.

A further device in QType consists of lexical rules, which we have incorporated merely for historical reasons, since they can be eliminated with suitable techniques (cf. Bouma et al. 2000). They facilitate a compact description of the lexicon and define binary relations between lexical entries by generating new lexical entries as output from previously-defined entries as input to cover lexical processes like inflection and derivation. The generation process involves asymmetric nonmonotonic unification to copy all parts of the input into the output that are compatible with the output specification of the lexical rule. See Bricoe & Copestake (1999) for a general survey and Rumpf (forthcoming) for a detailed description of the implementation of lexical rules in QType.

2.7 Nonmonotonic inheritance

There are two reasons for employing nonmontonic inheritance of information in object hierarchies: on the one hand it helps to avoid redundancy and thus to achieve more compact representations; on the other hand it provides a way to model rule exceptions explicitly. Precisely the last point can be a decisive criterion for the adaquacy of a theory, but with monotonic means it remains elusive. Although the majority of linguists working within the framework of constraint-based methods have emphasized the advantages of monotonicity, approaches involving nonmonotonic processing of feature structures have a long tradition. Most previous attempts to integrate nonmonotonicity into constraint-based approaches use variants of nonmonotonic unification (e.g., Bouma 1990, Carpenter 1993), which is neither associative nor commutative, however. This leads to serious problems for the semantics of constraint-based frameworks since they thereby cease to be declarative. With YADU Lascarides & Copestake (1999) present a variant of default unification whose result is independent of the order of the computations and which therefore preserves the declarative character of the formalism. For this gain they pay a high price, however: so-called 'tails' store the history of unifications, and the computation of symmetric unifications is factorial in the worst case (Carpenter 1993) and thus may be intractable. We therefore propose an alternative which preserves declarativity in the framework of QType but is also tractable: nonmonotonic inheritance in the type signature. The basic idea is to allow conflicting information in type signatures; conflicts in the top-down inheritance are then resolved according to the principle that more specific information has precedence, whereby the subtype relation is suspended. A default hierarchy can be converted offline into a corresponding monotonic hierarchy so that only the monotonic hierarchy is needed at runtime.

(19) Conversion of a nonmonotonic to a monotonic hierarchy



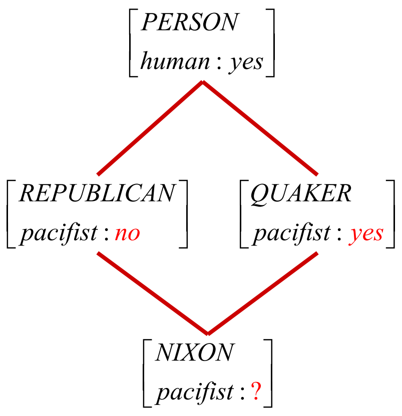
In (19) we see an example involving verb inflection in English (cf. Lascarides & Copestake 1999). Regular verbs like *push* have the ending +*ed* in the forms for past, past participle, and passive participle. Exceptionally, a group of verbs like *burn* has the ending +*t* for all three forms. In the nonmonotonic link shown to the left of the arrow, the type *pst-t-verb* inherits the structure sharing of the type *verb*, and the inheritance of +*ed* is blocked. The nonmonotonic inheritance can be converted to monotonic inheritance by computing the generalization of the types *verb* and *pst-t-verb*, which produces the abstract type *verb/pst-t-verb*. The latter inherits the compatible information from *verb* and *pst-t-verb* monotonically.

As the example clearly shows, the default relation between *verb* und *pst-t-verb* is lost since the types are disjoint and equally specific. In our system, however, a default connection can be reconstructed by referring back to the nonmonotonic inheritance hierarchy. For this purpose a mapping relation between the types of the two signatures is computed, with the help of which reference is made only to the nonmonotonic signature in output. Accordingly, a user of QType does not even notice that the computation takes place with a signature containing types that he has not defined.

Although the example just presented involves structure sharing, it still assumes a simplified scenario since only local conflicts at the level of the relation ‘immediate subtype’ need to be solved, whereas in general conflicts occur within the transitive closure of this relation. The supertype t_0, which introduces a feature-value pair into the hierarchy via an appropriateness specification, and the subtype t_n containing a conflicting value may be arbitrarily far apart. All conflict-free types t_i along the path to t_0 should, of course, remain in the monotonic inheritance relation to the supertype t_0. This requires unfolding the signature at t_0 by inserting a common immediate supertype t_0-t_n for t_0 and t_n which represents their generalization. However, this also suspends the inheritance relation between t_i and t_n, which is unacceptable since the types t_i can introduce additional feature-value pairs which are inherited by t_n. Consequently, a complete procedure must also compute the generalizations of all t_i with t_n and insert the corresponding supertypes t_i-t_n. This raises the question of where all the new inserted types should be attached. Clearly, t_0-t_n must be an upper bound for the new generalizations.

The procedure just described is always applied to a single conflict; in order to resolve several conflicts it must be iterated until all the conflicts are removed. It remains to be seen whether the order of conflict resolution leads to different monotonic signatures and whether a strategy can be found that is most efficient.

(20) The Nixon diamond



We see no plausible solution for conflicts arising from multiple inheritance. The Nixon diamond (cf. Touretsky 1986) represents such a scenario: The type *nixon* immediately inherits the property *pacifist:yes* from *quaker* as well as *pacifist:no* from *republican*. Which property has precedence can only be decided with the help of a given specificity weighting. We are inclined to reject solutions with *nixon* as a nondeterministic type. The procedure is nevertheless applicable to signatures with multiple inheritance, but conflicts may arise in the way just described; otherwise, due arbitrarily to the order of computation, the value would survive which is computed last. Signatures are consequently tested for such inconsistencies before the procedure is applied.

A further problem involves embedded structures. In the previous examples, all values occurred at the same depth, which in general is a gross simplification of the possibilities, unless one first considers type signatures without type constraints. Here the situation is in fact relatively simple since the appropriateness specifications always have the same basic form *Types* × *Attributes* → *Types* and, thus, neither embedded nor shared structures can be addressed. Besides these extensions, type constraints also introduce disjunction and relational constraints, however. QType requires type constraints to be locally consistent, i.e. no type constraint may conflict with the type for which it is defined. On the other hand, the information introduced by type constraints is inherited by all subtypes. Since the conversion of nonmonotonic signatures into monotonic ones takes place offline, the type constraints should also be computed offline so that conflicts introduced by them arising through inheritance may be taken into account. Unfortunately, this is not always possible since there is a class of relational constraints that cannot be computed offline, namely, recursive relational constraints, which are defined for partial data structures like lists of indefinite length. In such cases the solution set is nonfinite, so we have chosen partial evalutation (cf. Pereira & Shieber 1987) of type constraints as a remedy: whenever possible, the computation is carried out offline, but it is delayed till runtime for recursive relational constraints. As a consequence, no conflicts may be introduced into the hierarchy through recursive relational constraints in connection with type constraints, since they can be resolved neither offline nor during parsing. In practice we are aware of no cases where this is important.

The model of nonmonotonic inheritance presented here is significantly more restrictive than that suggested by Lascarides &

Copestake (1999). Nevertheless, we believe that it entails no essential limitations on the specification of default relations in grammar development. In the following section, we present a method by which type hierarchies of the kind we have discussed can be induced automatically using Formal Concept Analysis.

3 Induction of inheritance hierarchies

3.1 General remarks

Technical progress in recent decades has led to an enormous growth in the quantity of data that can be obtained through corpus-based empirical studies. It is no longer possible to evaluate such data manually for the construction of lexica in the form required by modern grammar formalisms. The use of automatic methods is therefore unavoidable.

In principle there are two different approaches for obtaining appropriate inheritance-based lexica from collected linguistic data on words and their properties:

> Incremental and dynamic procedures imitate human behavior (cf., e.g., Barg 1996). A relatively small data set is used to produce an initial inheritance hierarchy, which then, on the basis of additional data, is refined. A 'pruning' at all intermediate stages ensures that the resulting hierarchy is not too detailed. The ultimate result is a compact representation of the complete data set in an inheritance hierarchy which depends to a great degree, however, on the initial data, the order of the subsequently processed data, and the pruning algorithm employed. With this procedure it normally is only possible to obtain results that are locally optimal.

The alternative procedure is to obtain a single hierarchy in one step; this hierarchy logically corresponds to the data in a defined form and may be extremely large. In a second step it can be reduced to an acceptable size through statistical or semi-automatic pruning. The disadvantage of this procedure is that the insertion of new words constitutes an independent problem, the solution of which is not given directly by the induction procedure.

The method presented here, based on Formal Concept Analysis, although belonging to the second group of approaches, results however in hierarchies which are so detailed that new, unknown words normally can directly inherit appropriate information from a node already present in the hierarchy. If such a node does not exist, the new, extended hierarchy can be computed from the current one without starting over again from scratch.

3.2 Formal Concept Analysis

Formal Concept Analysis (FCA) is a mathematical theory that models the notion of the 'concept' set theoretically and that orders concepts in concept lattices (cf. Ganter & Wille 1999). These concept lattices can be regarded as inheritance hierarchies. Ganter & Wille (1998) characterize FCA as follows:

> The sophisticated name of Formal Concept Analysis needs to be explained. The method is mainly used for the analysis of data, i.e. for investigating and processing explicitly given information. Such data will be structured into units which are formal abstractions of concepts of human thought allowing meaningful and comprehensible interpretation. We use the prefix formal to emphasize that these formal concepts are mathematical entities and must not be identified with concepts of the mind. The same prefix indicates that the basic data format, that of a formal context, is merely a formalization that encodes only a small portion of what is usually referred to as a context.

3.2.1 Formal contexts

In order to be analyzed with FCA, data must be structured as a formal context. A *formal context* (FC) is a triple (G,M,I) consisting of a set of objects G, a set of attributes M, and a binary incidence relation $I \subseteq G \times M$ between the two sets. If an object g is in a relation I with an attribute m, we write $(g,m) \in I$, and read it as "the object g has the attribute m". A formal context can — at least in finite cases — be represented in a table like that in (21).

(21) Example of a formal context[1]

	gender: masc	gender: fem	gender: neut	sing nom:*	sing gen:*	sing gen:*_s	sing gen:*_n	sing gen:*_ns	sing dat:*	sing dat:*_n	sing acc:*	sing acc:*_n	plur nom:*	plur nom:*_n	plur gen:*	plur gen:*_n	plur dat:*_n	plur acc:*	plur acc:*_n
Herr	x			x			x			x		x		x		x	x		x
Friede	x			x				x		x		x		x		x	x		x
Staat	x			x		x			x		x			x		x	x		x
Hemd			x	x		x			x		x			x		x	x		x
Farbe		x		x	x				x		x			x		x	x		x
Onkel	x			x		x			x		x		x		x		x	x	
Ufer			x	x		x			x		x		x		x		x	x	

Since “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” (Sag &Wasow 1999: 173), data must be collected in empirical studies that fill these feature-structure descriptions. Such data build ternary relations between phonological forms, attributes, and values of the attributes. The data must be put in the form of a suitable FC, i.e. a binary relation, through the choice of a suitable scaling for each many-valued attribute. It often is enough just to regard each attribute-value pair as an independent attribute, as has been done in (21). This transformation process is called “scaling with the nominal scale”. Sometimes, however, e.g., in the analysis of subject-verb agreement in English, a more differentiated scaling is better suited. (22a) shows a small example of data about English inflected verb forms. In order to permit explicit reference to non-third values the scale in (22b) can be chosen,

[1] The context covers the inflectional paradigms and the gender information of the following seven German nouns: *Herr* ‘mister’, *Friede* ‘peace’, *Staat* ‘state’, *Hemd* ‘shirt’, *Farbe* ‘color’, *Onkel* ‘uncle’, *Ufer* ‘bank/shore’. In the attribute names ‘*’ stands for the stem of the noun.

which introduces the values *local* and *nonlocal*, to refer to the status of the subject. The resulting formal context is shown in (22c). The choice of suitable scales is normally not a problem in linguistic contexts since only discrete values appear; these can always be scaled without information loss using the nominal scale.

(22) Possible scaling of the attribute PERS

	PERS	NUM
sleep_1	1.	sing
sleep_2	2.	sing
sleeps	3.	sing

(a)

	1.	2.	3.	local	nonlocal
1.	x			x	
2.		x		x	
3.			x		x

(b)

	PERS: 1.	PERS: 2.	PERS: 3.	PERS: local	PERS: nonlocal	NUM: sing
sleep_1	x			x		x
sleep_2		x		x		x
sleeps			x		x	x

(c)

3.2.2 Formal concepts

In FCA concepts are conceived in terms of sets as in the classical definition of concepts. They are defined through their extension as well as their intension. A *formal concept* of a context is thus a pair consisting of a set of objects, its *extent*, and a set of attributes, its *intent*. The extent consists exactly of the objects in the context for which all the attributes of the intent apply; the intent consists correspondingly of all attributes of the context which the objects from the extent have in common. In order to formally express the strong connection between the extent and the intent of a formal concept given by the binary relation I, two derivational operators are defined between the set of objects G and the attribute set M of a formal context: If $A \subseteq G$ is a set of objects, then the set of the common attributes of A is

$$A' := \{m \in M \mid \forall g \in A: (g,m) \in I\},$$

and if, analogously, $B \subseteq M$ is a set of attributes, then the set of objects that have B in common is

$$B' := \{g \in G \mid \forall m \in B: (g,m) \in I)\}.$$

A formal concept is thus a pair (A,B) with $A \subseteq G$, $B \subseteq M$ where $A = B'$ and $B = A'$. Furthermore, the definition of the derivational operators guarantees that (A'',A') is a formal concept for any set of objects $A \subseteq G$ and (B',B'') is a formal concept for any set of attributes $B \subseteq M$.[2] Formal concepts from (21) are, e.g.,

({Staat, Farbe}'' , {Staat, Farbe}') = ({Hemd, Staat, Farbe},{sing nom:*, sing dat:*, sing acc:*, plur nom:*_n, plur gen:*_n, plur dat:*_n, plur acc:*_n}),

and

({sing acc:*}', {sing acc:*}'') = ({Ufer, Onkel, Hemd, Staat, Farbe},{sing nom:*, sing dat:*, sing acc:*, plur dat:*_n}).

3.2.3 Concept lattices

The subconcept-superconcept relation on the set of all formal concepts of a context defines a partial order: $(A_1,B_1) \leq (A_2,B_2) \Leftrightarrow A_1 \subseteq A_2 \Leftrightarrow B_1 \supseteq B_2$. This order relation corresponds to our intuitive notion of super- and subconcepts. Superconcepts are more general, encompass more objects, and are characterized by fewer attributes. It can be shown that the set of all formal concepts of a formal context ordered with respect to the subconcept-superconcept relation constitutes a complete lattice, which is called the *concept lattice* of the formal context.[3] (23) shows the concept lattice corresponding to the formal context in (21).

Concept lattices are normally represented by Hasse diagrams as in (23). Superconcepts are located above subconcepts and are a linked to them

[2] A detailed introduction of Formal Concept Analysis is given by Ganter & Wille (1999).

[3] Furthermore, the Basic Theorem on Concept Lattices states that every complete lattice is the concept lattice of a formal context.

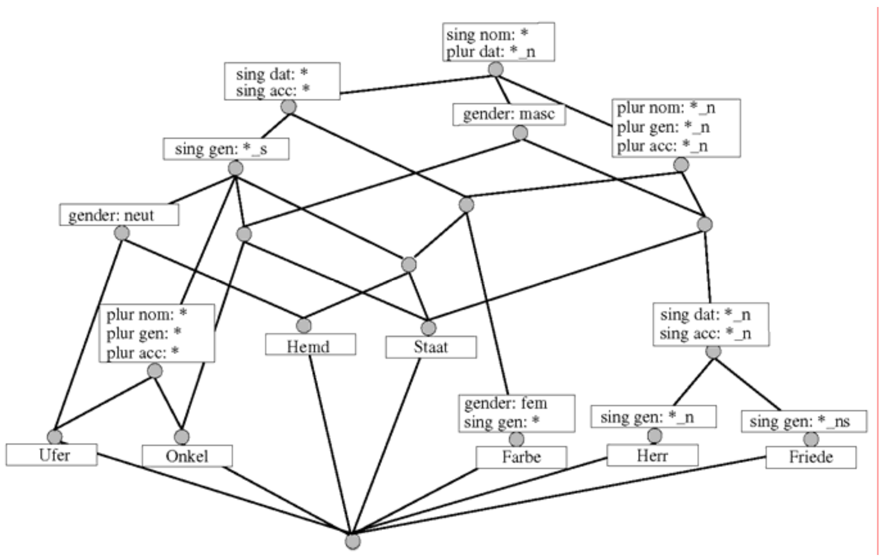
by a path. The definition of a concept lattice allows an especially economical labelling: instead of labelling every concept with its complete extent and intent, only the object and attribute concepts are labelled. The *object concept* γg of an object g is the smallest concept whose extent includes g, i.e. $\gamma g = (g'',g')$. The *attribute concept* μm of an attribute m is analogously the largest concept whose intent includes m, i.e. $\mu m = (m',m'')$.[4] In this way a concept lattice becomes an inheritance hierarchy: Any concept of the lattice inherits all objects which are labelled with subconcepts as its extent, and it inherits all attributes with which superconcepts are labelled as its intent. Inheritance hierarchies based on concept lattices are completely nonredundant, i.e. each attribute and each object appears exactly once in the hierarchy.

Concept lattices visualize structural connections between the data of a flat data list. Ganter & Wille (1998) stress*:* "It is our belief that the potential of Formal Concept Analysis as a branch of Applied Mathematics is just beginning to show. A typical task that concept lattices are useful for is to unfold given data, making their conceptual structure visible and accessible, in order to find patterns, regularities, exceptions, etc."

---

[4] For example, the attribute concept of the attribute *sing acc:** from (21) is ({sing acc:*}', {sing acc:*}'') = ({Ufer, Onkel, Hemd, Staat, Farbe},{sing nom:*, sing dat:*, sing acc:*, plur dat:*_n}).

(23) Concept lattice from the context in (21)



3.2.4 Complete theories and concept lattices

The mutual dependencies between the data of a context can also be shown in another way, namely, with a complete theory that describes the data. Such a complete theory is a set of clauses over the attributes of the context which are compatible with the data of the context and from which all the compatible clauses can be derived.

The clauses are of the form $\forall x \in G : \phi \rightarrow \psi$, where the set of attributes are treated as one-place atomic predicates and ϕ and ψ are terms built inductively from $\wedge$ and $\vee$ and the atomic predicates.[5] Quantors and predicates no longer need to be specified explicitly; instead, we write *PERS:first* $\rightarrow$ *PERS:nonthird* rather than $\forall x$: *PERS:first(x)* $\rightarrow$ *PERS:nonthird(x).*

[5] Standard transformations can be used to show that a negation operator would introduce no additional expressivity.

As has been shown in Osswald & Petersen (2003), it frequently makes sense to restrict the form of clauses, e.g. by dispensing with the disjunctive or conjunctive operators. Three principal classes of complete theories result: theories with neither disjunctive nor conjunctive operators are termed *simple inheritance theories*, theories without disjunctive operators are *Horn theories*, general theories without restrictions are termed *observational theories* (for a thorough discussion see Osswald & Petersen 2002 and Osswald & Petersen 2003).

Every theory determines a hierarchical classification, the *information domain* of the theory. Its classes are the attribute sets maximally consistent with the theory and are ordered by the subset relation. The set of intents of all formal concepts of a context ordered by the subset relation correspond precisely to the information domain of a complete Horn theory of the context. Due to the convention in FCA we call Horn clauses *(attribute) implications*. The implications can be read off directly from the concept lattice: An implication $m_1 \wedge m_2 \wedge ... \wedge m_k \rightarrow m$ holds exactly when the greatest lower bound of the attribute concepts $\mu m_1, ... , \mu m_k$ is a subconcept of the attribute concept μm. Hence, the implication *sing dat:* ∧ gender: masc → sing gen:*_s* holds in (21), since in (23), the greatest lower bound of the nodes labelled with *sing dat:** and *gender: masc* is a lower neighbor of the node labelled with *sing gen:*_s.* Since the union of every premise and the maximal corresponding conclusion forms an intent of one of the concepts of the context, the structure of the concept lattice is determined by the set of valid attribute implications up to isomorphism.

FCA thus occupies an intermediate position with respect to the complexity of the underlying theories. In the following section we will show that precisely this intermediate position makes FCA an outstanding tool for the induction of lexical hierarchies. More complex theories (those with disjunctive operators) tend to overfit, the induced hierarchies are too flat, and overgeneralizations are made. On the other hand, simpler theories (those without conjunctive operators) fail to capture some interesting linguistic generalizations, which can be seen from the fact that many linguistic formalisms include implications with conjunctions.

It is noteworthy that many linguistic formalisms can be situated in this intermediate position. For example, feature-structure descriptions in the framework of HPSG are based on conjunction, whereas, e.g., the semantics of disjunction of types is a widely discussed problem.

3.3 Examples using FCA

3.3.1 Strategies for avoiding redundancy in the lexicon: feature co-occurrence restrictions in GPSG versus hierarchical type signatures in HPSG

Lexicalist linguistic theories encode a great deal of grammatical information in the lexicon and employ various strategies in order to avoid unnecessary redundancy. This section will introduce two of these strategies and show how FCA can be used to translate lexica built according to one strategy into corresponding ones for another.

GPSG uses so-called *feature co-occurrence restrictions* (FCRs) to restrict the distribution of features and their values. A pair consisting of a feature and a feature value is called a *feature specification*. Whereas GPSG features are atomic symbols, feature values are either atomic symbols or categories,[6] i.e., set of feature specifications (cf. Gazdar et al. 1985: 22). The FCRs are part of a grammatical theory and restrict the set of possible categories and their extensions in the theory. A typical FCR is [+INV] ⊃ [+AUX, FIN], which is [<INV,+>] ⊃ [<AUX,+>, <VFORM, FIN >] when written out fully.[7] The condition stated here is that in English the feature specification <INV,+> implies <AUX,+> and <VFORM:FIN >: if a verb occurs initially in a sentence containing a subject, then this verb must be a finite auxiliary.

As noted above, HPSG adopts no obvious counterpart for the FCRs of GPSG and instead employs inheritance hierarchies, whose informational domain consists of typed feature structures. HPSG further reduces redundancy by ordering the types in a type hierarchy, the so-called *type signature*, in which appropriateness conditions are inherited (for more details see §2).

[6] Categories of GPSG correspond to the untyped features structures of other unification-based formalisms.

[7] The feature specification <INV,+> marks sentence-initial verbs, <AUX,+> marks auxiliary verbs and <VFORM, FIN> specifies, that the verb is finite.

HPSG thus replaced the FCRs of GPSG with inheritance hierarchies of types, but the relations between these formal devices were not well understood. With FCA a general framework is now available which allows the equivalence of the devices to be explained in a transparent and declarative fashion.

The main idea for showing the convertability of FCRs and type signatures is to construct a suitable formal context in order to employ the methods of FCA. (24) shows a lexical fragment consisting of 10 lexemes classified with respect to some of the features proposed in Gazdar et al. (1985). The chosen features are exactly those which play a role in the first 4 FCRs given in Gazdar et al. (1985); these FCRs regulate the distribution of the features *v, n, vform, nform, pform, inv, aux,* and their values (see (25)).[8] (24) is formed by taking each feature-value pair as an independent attribute of the context. GPSG also allows FCRs of the form [VFORM] ⊃ [+V, -N] (see FCR 2), which encode not only restrictions on feature-value pairs but also on the admissibility of certain features in the first place. Because of this, further attributes of the form *feature:VAL* have been added in (24), where such an attribute applies to a word if there is some value *value* such that *feature:value* is an feature specification of the word. Feature structures with embedded structures can be represented in a formal context by *flattening* them and viewing the path-value pairs as attributes (cf. Sporleder 2003; Petersen 2004b, Petersen forthcoming). We call such a formal context obtained from feature structures a *feature structures context*.

[8] The feature *vform* distinguishes parts of the verb paradigm (*bse*, base-form; *fin*, finite; *pas*, passive participle) and *nform* analogously distinguishes the special expletive pronoun *it* from normal nouns (*norm*).

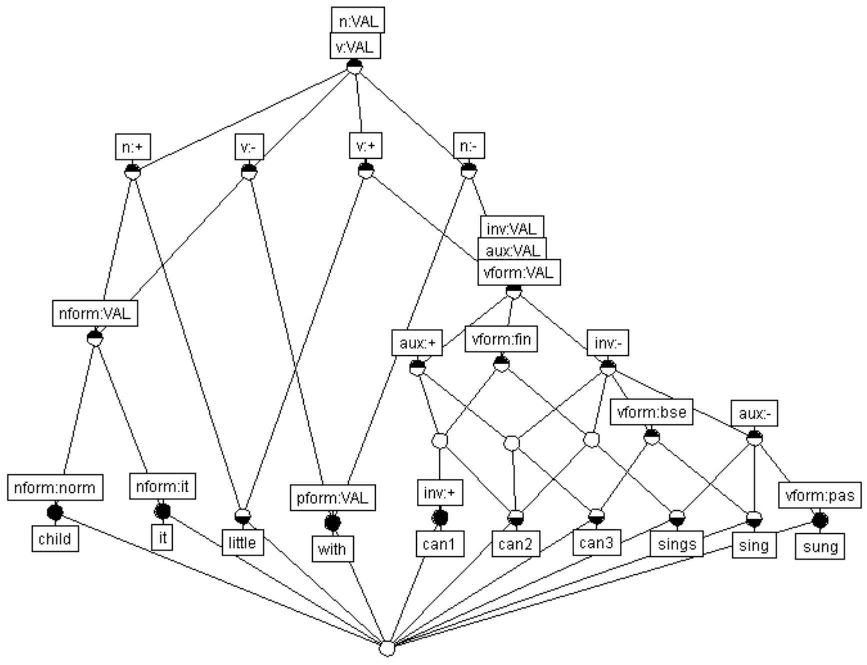
(24) Lexemes classified with respect to their feature specifications in Gazdar et al. (1985)

	v:+	v:−	n:+	n:−	vform:bse	vform:fin	vform:pas	aux:+	aux:−	inv:+	inv:−	nform:norm	nform:it	v:VAL	n:VAL	vform:VAL	aux:VAL	inv:VAL	nform:VAL	pform:with	pform:VAL
sing	×			×	×				×		×			×	×	×	×	×			
sings	×			×		×			×		×			×	×	×	×	×			
sung	×			×			×		×		×			×	×	×	×	×			
can1	×			×		×		×		×				×	×	×	×	×			
can2	×			×		×		×			×			×	×	×	×	×			
can3	×			×	×			×			×			×	×	×	×	×			
child		×	×									×		×	×				×		
it		×	×										×	×	×				×		
little	×		×											×	×						
with		×		×										×	×					×	×

(25) The first 4 FCRs from Gazdar et al. (1985)

FCR 1 :	[+INV]	$\supset$	[+AUX, FIN]
FCR 2 :	[VFORM]	$\supset$	[+V, −N]
FCR 3 :	[NFORM]	$\supset$	[−V, +N]
FCR 4 :	[PFORM]	$\supset$	[−V, −N]

(26) The concept lattice for the feature structure context of (24), which can be regarded as a type signature[9]



To illustrate the relationship between FCRs and type signatures we will proceed as follows: First we show how a type signature can be derived from a feature structure context, and then we do the same for a system of FCRs. Finally, we demonstrate how a feature structure context can be constructed from either a type signature or a set of FCRs.[10]

[9] This figure was created using the program ConExp (http://www.sourceforge.net/projects/conexp).

[10] A detailed description of our approach can be found in Petersen & Kilbury (2005).

(26) shows the concept lattice for the feature structure context in (24), which can be directly interpreted as a type signature of HPSG if a unique type is assigned to each node of the lattice. The feature labels then encode the appropriateness conditions associated with each type, and the subconcept relation corresponds to the subtype relation in the type signature. Only the lowest node of the concept lattice has no direct counterpart in the type signature; it expresses the failure of unification and often is represented with $\perp$, the symbol for bottom. If one reads the hierarchy in (26) as a type signature it is apparent that the hierarchy makes extensive use of multiple inheritance, which is possible to only a limited degree in some varieties of HPSG. Due to the definition of formal concepts, however, it can never be the case that a type inherits incompatible information from its upper neighbors.[11]

The hierarchy in (26) encodes so much information about the distribution of atomic values that it is sufficient to pair phonological forms with types in the lexicon in order to obtain adequate feature structures. For realistic lexica such a procedure leads to enormous type signatures since every lexical feature structure must have its own type. Moreover, it conflicts with the idea of types when they, e.g. in the case of *with*, cover only a single object. On this issue Pollard & Sag (1987: 192) write:

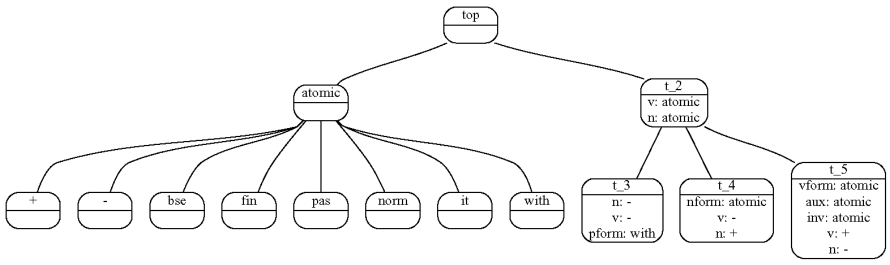
> [...] lexical information is organized on the basis of relatively few — perhaps several dozen — word types arranged in cross-cutting hierarchies which serve to classify all words on the basis of shared syntactic, semantic, and morphological properties. By factoring out information about words which can be predicted from their membership in types (whose properties can be stated in a single place once and for all), the amount of idiosyncratic information that needs to be stipulated in individual lexical signs is dramatically reduced.

Consequently, type signatures are frequently constructed in such a way that they primarily encode information about the general structure of feature structures. This is accomplished with appropriateness conditions which regulate the distribution of the feature, but restrict their values only to

[11] The principle of unique feature introduction is never violated because the features are introduced on their own, unique feature concepts.

general types. (27) shows such a type signature which was generated with our system FCALING. The restricted distribution of the atomic values from the FCRs in (25) can be enforced with the introduction of additional type constraints.

(27) A typical type signature for the data in (24)



The data of our example are rather atypical for an HPSG analysis since they are based on feature structures containing paths with no more than one feature. In the following we show that FCA is able to induce type signatures from sets of more deeply embedded feature structures of the sort shown in (28).

(28) A small example lexicon with untyped feature structures from Shieber (1986)

$$\text{Uther}=\begin{bmatrix} \text{CAT} & \text{np} \\ \text{HEAD} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PERS} & \text{third} \\ \text{NUM} & \text{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

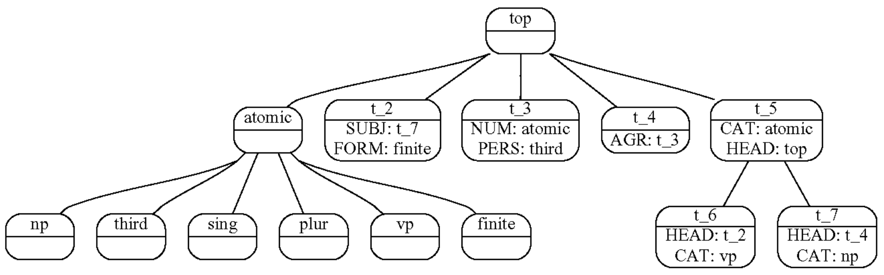
$$\text{knights}=\begin{bmatrix} \text{CAT} & \text{np} \\ \text{HEAD} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PERS} & \text{third} \\ \text{NUM} & \text{plur} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

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

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

A module of our system FCALING computes such inductions (details are given in Petersen 2004b, Petersen forthcoming). The following approach was chosen for the implementation: In an initial step the feature structures are translated into a feature structure context; in the course of this translation all atomic values are replaced by the marker ‘av_’, however. In addition to the large matrix feature structures we also take all the embedded structures and encode them in the same feature structure context. In a procedure reverse to that of Penn’s *unfolding* (see above) this concept lattice is then *folded*, which results in a type signature that encompasses all structural aspects of the feature structures. Finally, information about the atomic values is added and the appropriateness conditions are sharpened correspondingly. When applied to the data in (28) the type signature of (29)[12] results. Type signatures induced with FCALING can be used directly as a starting point for developing grammars in QType (see §2).

(29) Type signature induced from the untyped feature structures in (28)



Sporleder (2003) proposes another approach for the induction of type signatures. Here a lattice like that of (26) is defined as the search space for an induction task. A maximum entropy model is applied to this lattice which classifies nodes into plausible and implausible generalizations. This maximum entropy model is the result of a machine learning process which was trained on plausible manually constructed hierarchies.

[12] The graphic output was realized with the help of the program GraphViz (http://www.graphviz.org/).

Having constructed type signatures from formal concept lattices, we will now show how to derive FCRs from a feature structure context as given in Table (24). The FCRs of GPSG are nothing other than implications that are compatible with the data of Table (24). In order to restrict the set of admissible categories sufficiently, the FCRs must reflect the mutual dependencies between the data of the context. Hence, the data of the context must respect the FCRs and the object concepts of the context must be derivable from the FCRs. The latter ensures that the FCRs license no feature structure with features which do not co-occur in the input structures.

(30) shows a basis of the complete Horn theory of (24);[13] some abbreviations have been used: A $\leftrightarrow$ B denotes the two implications A $\rightarrow$ B and B $\rightarrow$ A and $\perp$ is the (inconsistent) set of all attributes of the context in (24). Because of the eqivalence (A $\rightarrow$ C)$\wedge$(B $\rightarrow$ C) $\Leftrightarrow$ A$\vee$B $\rightarrow$ C, we summarize implications with equal conclusions in one clause (e.g. clause 17).

If one compares the implications in (30) with the corresponding FCRs from Gazdar et al. (1985) in (25), it is apparent that all FCRs can be derived from the implications in the basis of the Horn theory: FCR 1 corresponds to implication 8; FCR 2 is contained in equivalence 5, FCR 3 in equivalence 4, and FCR 4 in equivalence 2.

(30) Horn theory for the feature structure context in (24)

	$\rightarrow$	v : VAL $\wedge$ n : VAL	(1)
v : $-$ $\wedge$ n : $-$	$\leftrightarrow$	pform : VAL	(2)
	$\leftrightarrow$	pform : with	(3)
v : $-$ $\wedge$ n : $+$	$\leftrightarrow$	nform : VAL	(4)
v : $+$ $\wedge$ n : $-$	$\leftrightarrow$	vform : VAL	(5)
	$\leftrightarrow$	aux : VAL	(6)
	$\leftrightarrow$	inv : VAL	(7)
inv : $+$	$\rightarrow$	vform : fin $\wedge$ aux : $+$	(8)
aux : $-$	$\rightarrow$	inv : $-$	(9)
vform : pas	$\rightarrow$	aux : $-$	(10)
vform : bse	$\rightarrow$	inv : $-$	(11)
nform : it	$\rightarrow$	nform : VAL	(12)
nform : nor	$\rightarrow$	nform : VAL	(13)
inv : $-$	$\rightarrow$	inv : VAL	(14)
aux : $+$	$\rightarrow$	aux : VAL	(15)
vform : fin	$\rightarrow$	vform : VAL	(16)
(n : $+$ $\wedge$ n : $-$) $\vee$ ... $\vee$(inv : $+$ $\wedge$ inv : $-$)	$\rightarrow$	$\perp$	(17)

However, (30) includes further implications, some of which bear no real information in the sense of GPSG since they either result from the sparse input data (cf. equivalence 3), from the special role of the feature value VAL (cf. implications 12–16), or from the fact that no knowledge about the exclusivity of features is implemented in FCA (cf. implication 17). Implication 9 follows from FCR 1 on the condition that, first, whenever *inv* is specified then *aux* is also specified and vice versa (implication 7) and, second, *inv* is restricted to the values + and -; implication 11 follows analogously from FCR 1.

The implications 1, 6, and 7 are missing in the FCRs and moreover only one direction of the equivalences 2, 4, and 5 is stated in the FCRs. These implications regulate when categories necessarily must be specified with respect to certain features without saying anything about the concrete feature values. A surprising gap in the list of FCRs is evident in implication 10, according to which passive verbs are not auxiliaries. This fact cannot be derived from the FCRs in (25). The GPSG grammar given in Gazdar et al. (1985) thus allows the feature-specification bundle {[PAS], [+AUX], [-INV]}, which encodes a non-inverted auxiliary in passive. This demonstrates that the automatically extracted complete Horn theory is more explicit than the manually formulated FCRs, which arise from linguistic intuition and miss statements which were probably too obvious for the investigators. Hence, FCA can be a powerful tool for tracking down gaps in theories that experts have constructed manually.

However, the FCRs of GPSG are not restricted to attribute implications (Horn clauses). Some of the FCRs in Gazdar et al. (1985: 246), make use of negation and disjunction. The following consideration shows that such FCRs are implicitly encoded in the concept lattice, too. As we saw before, a clause with a disjunctive premise and conjunctive conclusion can be directly transformed into a set of attribute implications. Hence, we can focus on the derivation of clauses with disjunctive conclusions from concept lattices. The information domain of a complete observational theory of a formal context consists exactly of the intents of the object concepts. These considerations open a way to derive a complete observational theory and hence a complete set of FCRs from a concept lattice:

Each formal concept (A,B) of the lattice which is not an object concept corresponds to an observational statement whose premise is the conjunction of the elements of the intent B and whose conclusion is the disjunction of the conjunctions of its subconcept intents minus B. Adding the corresponding statement of a concept to the theory amounts to removing the concept intent from the informational domain of the theory. For example, the statement corresponding to the attribute concept of *nform:val* is

$$\text{n:VAL} \wedge \text{v:VAL} \wedge \text{v:-} \wedge \text{n:+} \wedge \text{nform:VAL} \rightarrow \text{nform:norm} \vee \text{nform:it}$$

which can be simplified by (30) to

$$\text{nform:VAL} \rightarrow \text{nform:norm} \vee \text{nform:it}$$

This states that if an object bears the feature *nform*, then the latter must be specified for the value *norm* or *it*.[14]

As argued in the preceding section, in order to describe the data of (24) a complete observational theory or a simple inheritance theory could be employed instead of a complete Horn theory. Since the FCRs of GPSG do not allow disjunctions, observational theories cannot be used for the induction of FCRs.[15] Simple inheritance theories are not adequate for the induction of FCRs, even though conjunctions occur only in the conclusions of the FCRs in (25) and thus can be resolved through the introduction of new FCRs. But among the FCRs in Gazdar et al. (1985) there are also some with conjunctive premises (e.g., FCR 10: [+ INV, Bar 2] $\supset$ [+SUBJ]), which in general cannot be resolved.

The necessary FCRs can be generated fully automatically with FCA, however. The task of the grammar developer is thus restricted to the selection of suitable features and the construction of a feature-structure context like that of (24).

---

[14] Ganter and Krause (1999) present an alternative procedure for systematically constructing a complete observational theory of a formal context.

[15] The disjunctive operators in implication 17 result from an abbreviatory notation. Disjunctions in the premisses can always be resolved by introducing a separated implication for each member of the disjunction.

In this section we have shown so far how a lexicon in pure list form, as in (24), can be used to produce either GPSG FCRs or an HPSG type signature. It remains to show how feature structure contexts can be derived either from a type signature or from FCRs. Given a type signature, the adequate feature structure context can be directly constructed from the set of totally well-typed and sort-resolved feature structures. A detailed description of the construction method (even for type signatures with co-references) is given in Petersen (forthcoming). Given a set of FCRs, the corresponding formal context is equivalent to the information domain of the FCRs (see Osswald & Petersen, 2003). Hence, FCA enables us to switch directly from the format of a type signature to FCRs and back.

3.3.2 Inherent lexical features versus inflectional classes

This section will present a further example for the applicability of FCA in linguistics, and, especially, in the lexicon. In particular we wish to show how different approaches to the analysis of German noun inflection can be reconciled with each other.

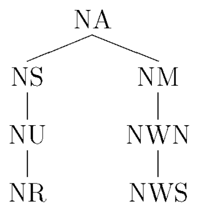
Classical approaches to inflectional morphology as seen, e.g., in Wahrig (1966) assign word stems an inflectional class in the lexicon; the inflectional class itself determines the inflectional paradigm of a stem. Minimalist Morphology, however, argues that "the membership in an inflectional subclass should not be arbitrarily assigned, but rather follow from features that can be memorized: either on the basis of a substantial property of the stem itself (such as gender or phonological shape), or on the basis of an additional lexical entry." (Wunderlich 1999).

Different inflectional paradigms typically reveal numerous similarities so that it is appropriate to model hierarchical relations between the classes in order to avoid unnecessary redundancy. Kilbury (2001) analyzes the inflectional classes of German nouns and proposes the hierarchical ordering shown in (31).[16] From the class **NS** of strong nouns without umlaut in plural the class **NU** with umlaut in plural inherits all the singular forms as well as the fact that the plural forms end in a schwa-

[16] Further examples of hierarchical analyses of the nominal inflection classes are given in Cahill & Gazdar (1999).

syllable. According to Minimalist Morphology it is necessary to analyze such relations in terms of lexically inherent, memorizable features that determine the class membership of individual nouns.

(31) Analysis of the inflection classes of German nouns and their hierarchical relations in Kilbury (2001)



NA	regular nouns (i.e., with *s*-plural)
NS	strong nouns (i.e., with a schwa-syllable in plural)
NU	strong nouns with umlaut in plural
NR	strong nouns with *r*-plural
NM	mixed nouns (with *n*-plural)
NWN	weak nouns without *s* in gen sg
NWS	weak nouns with *s* in gen sg

(32) shows a list of representative German nouns together with their inflectional classes and the features proposed in Kilbury (2001) as determining the class membership; clearly, this is a formal context. In (33) we see the corresponding concept lattice for this context, in which the gender features have been omitted for better legibility. The nodes in the hierarchy, which correspond to an inflectional class, are labelled accordingly.

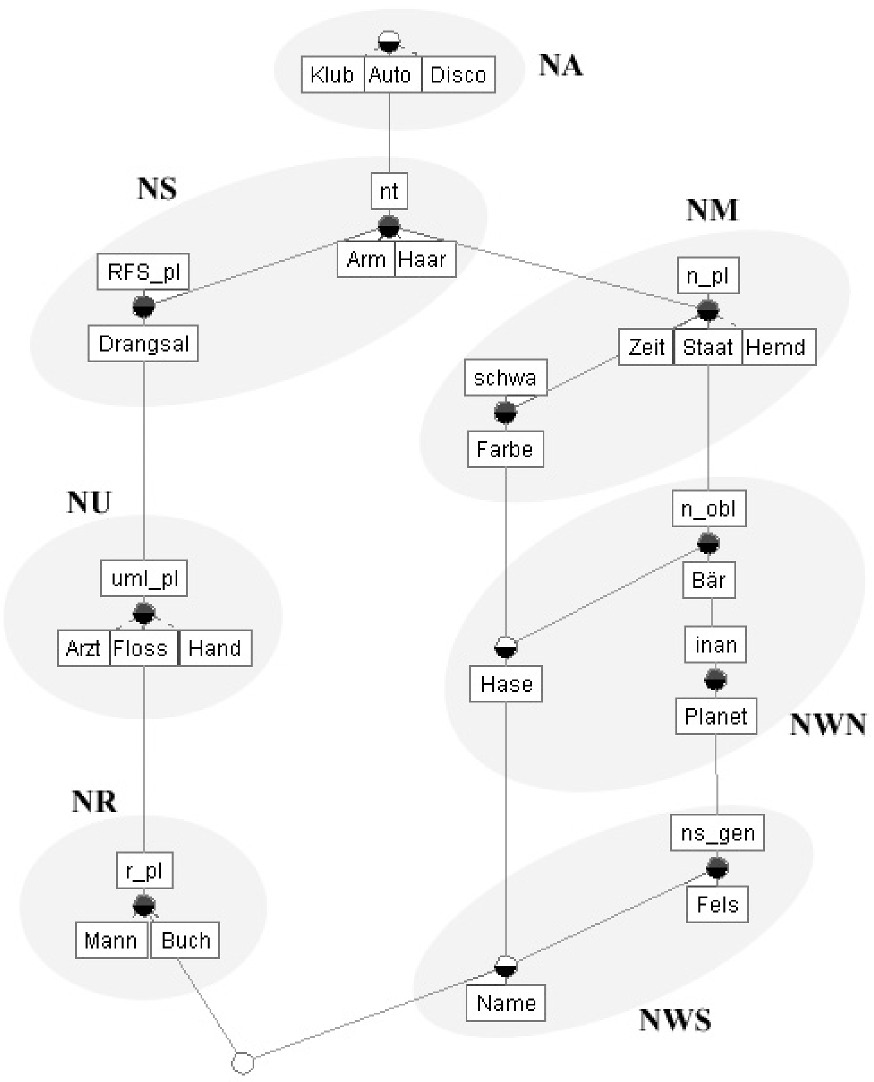
(32) German nouns together with their inflectional classes and the features that determine these classes according to Kilbury (2001)[17]

		[nt]	[f]	[m]	[schwa]	[inan]	[RFS_pl]	[uml_pl]	[r_pl]	[n_pl]	[n_obl]	[ns_gen]
NA	Klub			×		×						
	Auto					×						
	Disco		×			×						
NS	Arm	×		×		×						
	Haar	×				×						
	Drangsal	×	×			×	×					
NM	Zeit	×	×			×				×		
	Farbe	×	×		×	×				×		
	Staat	×		×		×				×		
	Hemd	×				×				×		
NWN	Hase	×		×	×					×	×	
	Bär	×		×						×	×	
	Planet	×		×		×				×	×	
NWS	Name	×		×	×	×				×	×	×
	Fels	×		×		×				×	×	×
NU	Arzt	×		×			×	×				
	Floss	×				×	×	×				
	Hand	×	×			×	×	×				
NR	Mann	×		×			×	×	×			
	Buch	×				×	×	×	×			

[17] The following abbreviations for features are used:

nt	typical noun
f	feminine
m	masculine
schwa	stem-final -e (schwa)
inan	inanimate
RFS_pl	reduced final syllable in plural
uml_pl	umlaut in plural
r_pl	suffix -r in plural
n_pl	suffix -n in plural
n_obl	suffix -n in singular nonnominative (i.e., oblique)

(33) Concept lattice from the context in (32)



ns_gen -n-s in genitive singular

It is not necessary to know all the features of a noun in order to inflect it since, normally, some follow from others. For each noun, Kilbury (2001) therefore distinguishes between the features that are *distinctive*, *redundant* (i.e. those that follow from others), or *irrelevant* for inflection. The previously unsolved problem of (semi-)automatically separating the distinctive and redundant features is one for which the method we have proposed lends itself. In particular, the representation of feature distributions in the form of feature implications makes it possible to characterize the interdependencies between the features explicitly and to depict them visually.

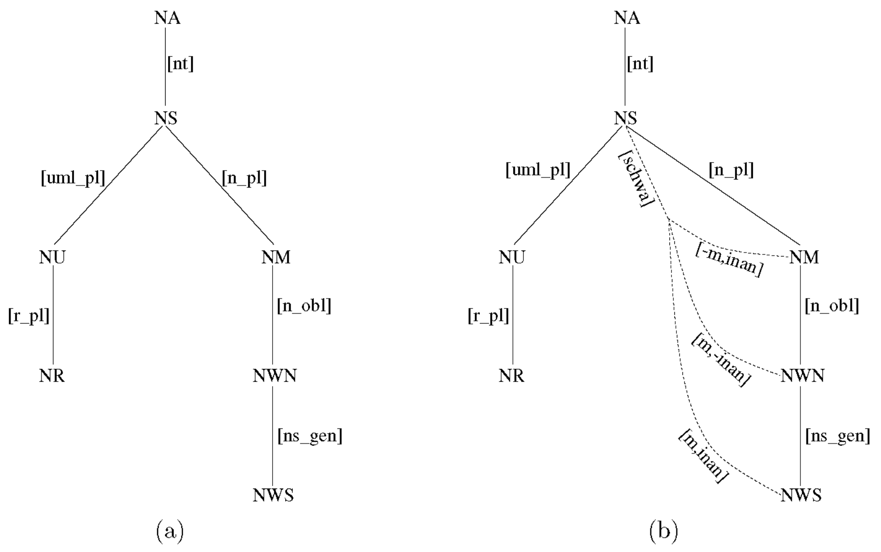
On the basis of (32) in the following we will show a procedure with which the membership of nouns in inflectional classes can be determined. To do this we first augment the context with features for the inflectional classes. We then consider the set of valid feature implications for the dichotomous context, i.e., the context, which also contains the negation of each feature. We obtain seven independent implications in which a class feature constitutes the conclusion, as shown in (34).

(34) Dependency of the inflectional classes on the features of (32)

`-nt`	$\rightarrow$	NA
`ns_gen`	$\rightarrow$	NWS
`r_pl`	$\rightarrow$	NR
`nt` ∧ `-uml_pl` ∧ `-n_pl`	$\rightarrow$	NS
`n_pl` ∧ `-n_obl`	$\rightarrow$	NM
`uml_pl` ∧ `-r_pl`	$\rightarrow$	NU
`n_obl` ∧ `-ns_gen`	$\rightarrow$	NWN

Minimalist Morphology assumes a principle of *underspecification* according to which unspecified features are interpreted as being negatively specified. Given this assumption the seven implications can be depicted in a hierarchy as in (35a). Edges are labelled with the distinctive features that determine class membership, and these features are percolated down along the edges.

(35) Hierarchical representation of the dependency relationship of the inflectional classes from features of (32)



The features in (32) obviously differ with respect to their memorizability. The first five are more easily memorized and thus more plausible than the last six features. In order to take this into account we have recalculated the valid implications of the context disregarding the last six features. As shown in (36), this results in three new implications with a class feature which are not in (34).

(36) Additional dependencies of the inflectional classes on the features of (32) which are easier to memorize

$$\begin{array}{lcl}
\texttt{schwa} \wedge \texttt{-m} \wedge \texttt{inan} & \rightarrow & \mathrm{NM} \\
\texttt{schwa} \wedge \texttt{m} \wedge \texttt{-inan} & \rightarrow & \mathrm{NWN} \\
\texttt{schwa} \wedge \texttt{m} \wedge \texttt{inan} & \rightarrow & \mathrm{NWS}
\end{array}$$

When these implications are added, the hierarchy in (35b) results. Note that dotted edges, in contrast to normal ones, do not denote inheritance. This hierarchy strongly resembles the manually constructed hierarchy of Kilbury (2001).

3.4 General remarks on Formal Concept Analysis as a tool for automatic induction

> Most other techniques of data analysis have the aim to drastically reduce the given information and to obtain a few significant parameters. By contrast, a concept lattice does not reduce complexity since it contains all the details of the data represented by the formal context. [...] Nevertheless it may be exponential in size, compared to the formal context. Complexity therefore is, of course, a problem, even though there are efficient algorithms and advanced program systems. A formal context of moderate size may have more concepts than one would like to see individually. (Ganter & Wille 1998)

This quotation expresses the particular strength of concept lattices as well as the problems that emerge when they are employed for the induction of hierarchies. FCA is superior to many alternative approaches to induction inasmuch as it is neutral with respect to the analyzed data and that it describes them completely. The relations between formal contexts and the corresponding concept lattices is absolutely transparent since there is exactly one of the latter for each context. When examination of the valid attribute implications reveals strange statements, this points either to new discoveries[18] or else to an incomplete or incorrect context. A semi-automatic attribute exploration can be carried out in order to complete a context (cf. Stumme 1996; Ganter & Wille 1999).[19]

We saw that FCA provides three different ways to display data: in tabular form, as a hierarchy, and as a set of implications. Each of these representations can be useful as a tool for various tasks in an analysis. Data

[18] Remember implication 10 in (30).

[19] The program ConImp can be used for feature explorations.

frequently are available in tabular form, which facilitates input. Information about a single object can most easily be read from the context, while the hierarchical concept lattice shows information about connections which are only implicit in the context. Furthermore, the concept lattice gives a more compact and less redundant representation of the data in comparision with the context. Finally, attribute implications express dependencies between the attributes.

In the following we will address two problems involving the application of FCA and possible solutions.

3.4.1 Pruning the hierarchies

Concept lattices explicitly include all possible generalizations since each set of common attributes induces a corresponding node, so the induced hierarchies can become very large. In the case of the lexical data base CELEX[20] the concept lattice corresponding to the context that describes German lexemes and their derivational potential contains more than 72,000 concepts. The basic context, which likewise was extracted from CELEX using our system FCALING, has 9,567 objects and 2,032 attributes.

One possibility for avoiding such large hierarchies is, as Sporleder (2003) discusses, to take the highly differentiated hierarchies as a point of departure for pruning; here one may have to accept a high cost for the computation of large hierarchies, but the approach has the advantage that the pruning algorithms can be specially tailored to the individual task. Sporleder achieves this by training her pruning algorithm on manually constructed hierarchies. The resulting hierarchies, however, are dependent on the learned pruning parameters. In Sporleder's approach this is intended, of course, since the pruning parameters reflect linguistic expert knowledge and are aimed at inducing hierarchies which are linguistically plausible.

Here we present a possibility for reducing the size of hierarchies that has already been described in Petersen (2004a). Instead of starting with the concept lattice of a context we take just the partially ordered set (poset)

[20] CELEX, which is compiled by the Dutch Center for Lexical Information, consists of three large electronic databases and provides users with detailed English, German, and Dutch lexical data.

of object and attribute concepts and then add unique maximal and minimal elements; in this way one obtains a hierarchy, the *AOC-poset* (attribute-object-poset), which normally is much more compact than the lattice.[21] In the case of the CELEX context mentioned above the hierarchy is reduced to one with less than 4,000 nodes. The AOC-poset, just like the concept lattice, is directly defined by the context. If the AOC-poset is labelled as an inheritance hierarchy as before, there again results a redundancy-free inheritance hierarchy, from which the context can be reconstructed.[22]

The shift to AOC-posets also reduces memory requirements. In computing an AOC-poset only the attribute and object concepts must be computed, which can be done independently of the others, so that the procedure is very efficient. Thus, in comparision to concept lattices AOC-posets offer a very simple method for the induction of redundancy-free inheritance hierarchies from large data sets. Inference tasks, however, are better supported by concept lattices, due to the explicit representation of shared attributes. Concept lattices are also better when the obtained hierarchies must be visualized, since AOC-posets normally are less legible due to their numerous crossing edges. Furthermore, valid attribute implications can be read off directly only in the lattice.

3.4.2 The problem of nonmonotonic hierarchies

In the preceding sections we have seen how FCA can be used to induce monotonic inheritance hierarchies. The relationship between a context and the corresponding monotonic hierarchy is completely transparent.

In contrast, induction of nonmonotonic hierarchies introduces special problems since there can be no such transparency between the contexts and hierarchies. Information in nonmonotonic hierarchies can be

[21] If a context consists of g objects and m features, then the maximum number of concepts is $2^{\min\{g,m\}}$, while that of nodes in an AOC-poset is at most g + m + 2.

[22] In section 3.3.1 we presented a method to induce non-Horn clauses from a formal context, which can be applied to construct the complete theory describing the AOC-poset.

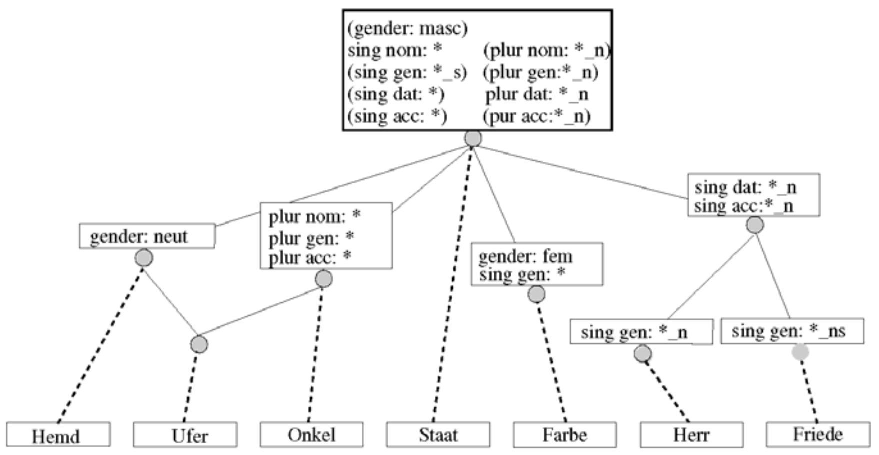
overwritten arbitrarily often, so there is an unlimited number of possibilities for representing data in such hierarchies.

The most important decision in constructing a nonmonotonic hierarchy is about what information should be regarded as regular. We will now show how monotonic lattices can be employed as a point of departure for the transformation into nonmonotonic hierarchies.

For a given context we first find the set of attributes which describe a *standard object* of the context. Three conditions are placed on such a default set of attributes (cf. Petersen 2004a): it must be internally consistent, its use must result in a more compact representation of the data, and it must be closed with respect to the valid attribute implications of the context. Such sets are just the intents of concepts with nonempty extents, and this connection allows such possible sets to be computed efficiently.

Since the choice of a plausible default attribute set is only possible with the aid of expert knowledge, we propose a semi-automatic procedure. Our system FCALING presents the user with all theoretically possible default attribute sets ordered according to the size of the corresponding extents. After the selection of a set, FCALING calculates the corresponding nonmonotonic hierarchy. This procedure can be applied interactively in order to represent subregularities as well. (37) shows a nonmonotonic inheritance network constructed with our method for the data from the context in (32). Note that defeasible attributes appear in parentheses. The hierarchy is based on the default attribute set {gender: masc, sing nom:*, sing gen:*_s, sing dat:*, sing acc:*, plur nom:*_n, plur gen:*_n, plur dat:*_n, plur acc:_n}.

(37) Example of a nonmonotonic inheritance network for the context in (32)



4 Conclusion

It is clear that the lexicon has become a major focus of linguistic investigation in the past quarter century. Research in computational linguistics has increasingly been aimed at modelling hierarchical relations within the lexicon and their formal representation. In particular, much recent work has concentrated on inheritance-based formalisms for typed feature structures, the choice between monotonic and nonmonotonic versions of these formalisms, and the automatic induction of classifications in general. Nonmonotonic inheritance is important as a means of capturing the continuum of regularity, subregularity, and irregularity in the lexicon.

These topics have been at the center of our attention in the present paper. We have presented our formalism QType, which combines major elements of typed unification-based formalisms with nonmonotonic inheritance hierarchies. Crucially, we confine nonmonotonicity to the static type hierarchy and compile the latter into a strictly monotonic counterpart for efficient processing.

We have described a technique using Formal Concept Analysis that aids in the construction of type signatures such as those of QType and other related formalisms; see Petersen (forthcoming) for an elaboration of this technique. At the same time we have shown how FCA provides an adequate formal basis for explicating the relationship between the type signatures of HPSG and the feature co-occurrence restrictions of GPSG.

Future work will undoubtedly seek to clarify further the empirical linguistic motivation for nonmonotonic type hierarchies as well as the formal constraints governing their automatic induction.

5 References

Aït-Kaci, Hassan, Robert Boyer, Patrick Lincoln & Roger Nasr
1989 Efficient implementation of lattice operations. *Programming Languages and Systems* 11: 115–146.

Barg, Petra
1996 *Automatischer Erwerb von linguistischem Wissen. Ein Ansatz zur Inferenz von DATR-Theorien.* Tübingen: Niemeyer.

Bernhardt, Wolfram
2001 Möglichkeiten der Effizienzsteigerung bei der Verarbeitung getypter Merkmalstrukturen. Master's thesis, Heinrich-Heine-Universität Düsseldorf.

Bock, Hans-Hermann & Wolfgang Polasek (eds.)
1996 *Data Analysis and Information Systems. Statistical and Conceptual Approaches* (Proceedings of the 19th Annual Conference of the Gesellschaft für Klassifikation e.V.,University of Basel, 1995–03–8/10). Berlin: Springer.

Boguraev, Branimir & James Pustejovsky (eds.)
1996 *Corpus Processing for Lexical Acquisition.* Cambridge, Mass.: MIT Press.

Bouma, Gosse, Frank Van Eynde & Dan Flickinger
2000 Constraint-based lexica, 43–71, in Van Eynde & Gibbon (2000).

Bouma, Gosse
1990 Defaults in unification grammar. *Proceedings of the Annual Meeting of the Association for Computational Linguistics* 28: 165–173.

Bresnan, Joan (ed.)
1982 *The Mental Representation of Grammatical Relations.* Cambridge, Mass.: MIT Press.

Briscoe, Ted, Valeria de Paiva, & Ann Copestake (eds.)
1993 *Inheritance, Defaults, and the Lexicon.* Cambridge: CUP.

Briscoe, Ted & Ann Copestake
1999 Lexical rules in constraint-based grammars. *Computational Linguistics* 25: 487–526.

Cahill, Lynne & Gerald Gazdar

1999 German noun inflection. *Journal of Linguistics* 35: 1–42.

Carpenter, Bob & Gerald Penn

1999 ALE: The Attribute Logic Engine User's Guide. Murray Hill, NJ: Bell Laboratories, Lucent Technologies.

Carpenter, Bob

1993 Sceptical and credulous default unification with application to templates and inheritance, 13–37, in Briscoe, de Paiva & Copestake (1993).

Copestake, Ann

2002 *Implementing Typed Feature Structure Grammars.* Stanford: CSLI.

Covington, Michael

1993 *Natural Language Processing for Prolog Programmers.* Englewood Cliffs, NJ: Prentice Hall.

Daelemans, Walter & Gert Durieux

2002 Inductive lexica, 115–136, in Van Eynde & Gibbon (2000).

De Smedt, Koenraad

1984 Using object-oriented knowledge representation techniques in morphology and syntax programming. *Proceedings of the European Conference on Artificial Intelligence 1984*: 181–184.

Dörre, Jochen & Michael Dorna

1993 CUF - A Formalism for Linguistic Knowledge Representation. DYANA 2 deliverable R.1.2A, Universität Stuttgart.

Evans, Roger & Gerald Gazdar

1989 Inference in DATR. *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics* 4: 66–71.

1996 DATR: a language for lexical knowledge representation. *Computational Linguistics* 22: 167–216.

Flickinger, Dan

1987 Lexical rules in the hierarchical lexicon. PhD thesis, Stanford University.

Ganter, Bernhard & Rudolf Wille

1998 Applied lattice theory: Formal concept analysis, 591–605, in Grätzer (1998).

1999 *Formal Concept Analysis. Mathematical Foundations.* Berlin: Springer.

Gazdar, Gerald

1987 Linguistic applications of default inheritance mechanisms, 37–67, in Whitelock (1987).

Gazdar, Gerald & Geoffrey Pullum

1982 Generalized Phrase Structure Grammar: A Theoretical Synopsis. Bloomington, Indiana: Indiana University Linguistics Club.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum & Ivan Sag

1985 *Generalized Phrase Structure Grammar.* Oxford: Blackwell.

Gerdemann, Dale & Paul John King

1993 Typed feature structures for expressing and computationally implementing feature cooccurence restrictions. *Proceedings of 4. Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft*: 33–39.

1994 The correct and efficient implementation of appropriateness specifications for typed feature structures. *Proceedings of the 15th Conference on Computational Linguistics*: 956–960.

Götz, Thilo, Detmar Meurers & Dale Gerdemann

1997 The ConTroll Manual. Manuscript, Universität Tübingen.

Grätzer, George

1998 *General Lattice Theory*. Basel: Birkhäuser Verlag.

Grosz, Barbara J. & Mark Stickel (eds.)

1983 *Research on Interactive Acquisition and Use of Knowledge.* Menlo Park, Calif.: SRI.

Hopcroft, John & Jeffrey Ullman

1979 *Introduction to Automata Theory, Languages and Computatio*n. Reading, MA: Addison-Wesley.

Kaplan, Ronald & Joan Bresnan

1982 Lexical-functional grammar: a formal system for grammatical representation, 173–281, in Bresnan (1982).

Karttunen, Lauri

1986 Radical Lexicalism (= Report No. CSLI-86–68). Stanford: CSLI.

Keller, Bill

1993 *Feature Logics, Infinitary Descriptions and Grammar.* Stanford: CSLI.

Kilbury, James

2001 German noun inflection revisited. *Journal of Linguistics* 37: 339–353.

Kilbury, James, Petra Naerger [Barg] & Ingrid Renz

1991 DATR as a lexical component for PATR. *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics* 5: 137–142.

Koenig, Jean-Pierre

1999 *Lexical Relations*. Stanford: CSLI.

Krieger, Hans-Ulrich & John Nerbonne

1993 Feature-based inheritance networks for computational lexicons, 90–136, in Briscoe et al. (1993).

Krieger Hans-Ulrich, Hannes Pirker & John Nerbonne

1993 Feature-based allomorphy. *Proceedings of the Annual Meeting of the Association for Computational Linguistics* 31: 140–147.

Lascarides, Alex & Ann Copestake

1999 Default representation in constraint-based frameworks. *Computational Linguistics* 25: 55–106.

Osswald, Rainer & Wiebke Petersen

2002 Induction of classifications from linguistic data. *Proceedings of the ECAI-Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases.*

2003 A logical approach to data-driven classification. *Lecture Notes in Computer Science* 2821: 267–281.

Penn, Gerald

2000 The algebraic structure of attributed type signatures. PhD thesis, School of Computer Science, Carnegie Mellon University.

Pereira, Fernando & Stuart Shieber

1987 *Prolog and Natural Language Analysis.* Stanford: CSLI.

Petersen, Wiebke

2004a A set-theoretic approach for the induction of inheritance-hierachies. In Proceedings of the Joint Conference on Formal Grammar and Mathematics of Language (FG/MOL-01), *Electronic Notes in Theoretical Computer Science* 53: 296–308.

2004b Automatic induction of type signatures. Unpublished manuscript.

forthcoming Induktion lexikalischer Vererbungshierarchien mit Mitteln der formalen Begriffsanalyse. PhD thesis (in preparation), Institute for Language and Information, University of Düsseldorf.

Petersen, Wiebke & James Kilbury

2005 What feature co-occurrence restrictions have to do with type signatures. *Proceedings of the Joint Conference on Formal Grammar and Mathematics of Language (FG/MOL-05)*.

Pollard, Carl & Ivan Sag

1987 *Information-based Syntax and Semantics, Vol. 1.* Stanford: CSLI.

1994 *Head-driven Phrase Structure Grammar.* Stanford: CSLI.

Riehemann, Susanne

1998 Type-based derivational morphology. *Journal of Comparative Germanic Linguistics* 2: 49–77.

Rumpf, Christof

forthcoming Default inheritance in constraint-based frameworks. PhD thesis (in preparation), Institute for Language and Information, University of Düsseldorf.

Russell, Graham, John Carroll & Susan Warwick-Armstrong

1991 Multiple default inheritance in a unification-based lexicon. *Proceedings of the Annual Meeting of the Association for Computational Linguistics* 29: 215–221.

Russell, Graham, Afzal Ballim, John Carroll & Susan Warwick-Armstrong

1992 A practical approach to multiple default inheritance for unification-based lexicons. *Computational Linguistics* 18: 311–337.

Sag, Ivan & Thomas Wasow

1999 *Syntactic Theory: A Formal Introduction.* Stanford:CSLI.

Shieber, Stuart

1986 *An Introduction to Unification-Based Approaches to Grammar.* Stanford: CSLI.

Shieber, Stuart, Hans Uszkoreit, Fernando Pereira, Jane Robinson & Mabry Tyson

1983 The formalism and implementation of PATR-II, 39–79, in Grosz & Sticker (1983).

Sporleder, Caroline

2003 Discovering lexical generalisations. A supervised machine learning approach to inheritance hierarchy construction. PhD thesis, Institute for Communicationg and Collaborative Systems, University of Edinburgh.

Stumme, Gerd

1996 Attribute exploration with background implications and exceptions, 457–569, in Bock & Polasek (1996).

Touretsky, David

1986 *The Mathematics of Inheritance Systems.* London: Pitman.

Van Eynde, Frank & Dafydd Gibbon (eds.)

2000 *Lexicon Development for Speech and Language Processing.* Dordrecht et al.: Kluwer.

Wahrig, Gerhard

1966 *Das Große Deutsche Wörterbuch*. Gütersloh: Bertelsmann Lexikon-Verlag.

Whitelock, Peter (ed.)

1987 *Linguistic Theory and Computer Applications.* London: Academic Press.

Wunderlich, Dieter

1999 German noun plural reconsidered*. Behavioral and Brain Sciences* 22: 1044–1045.