COMPILING SYSTEMIC GRAMMAR INTO FEATURE LOGIC SYSTEMS[1]

Renate Henschel
GMD/IPSI, Dolivostr. 15, D-64293 Darmstadt, Germany
`henschel@darmstadt.gmd.de`

Summary

Although systemic grammars are represented declaratively (as system networks), the most important implementations to date have been committed to the generation direction, and seem more or less incomprehensible for the researcher not familiar with systemic descriptions and software. The aim of this paper is, therefore, to re-represent systemic grammar resources in a standard typed feature logic in order (1) to clarify theoretically the relation between systemic grammar and state-of-the-art feature logics, and (2) to serve as point of departure for bidirectional processing of systemic grammar. Four such representations in the feature logic formalisms ALE, TFS, CUF and TDL are compared. Bidirectional processing in TFS and CUF based on type deduction are discussed.

1 Introduction

The problem of the reusability of existing linguistic resources is one central issue in current computational linguistic research. One attempt to cope with this problem is the very active area of grammar migration – the automatic compilation of grammar resources implemented in a particular NLP system to another NLP system or programming language. One can find migration programs from HPSG-style grammars into relational programming languages ([Goetz and Meurers, 1995]), from open world systems into closed world systems ([Gerdemann, 1995]), and from HPSG grammars to TAG (see [Kasper *et al.*, 1995]).

This paper is another contribution to the current discussion in grammar migration. It discusses the transformation of systemic grammar into currently available feature logic systems. The recent shift of focus from syntactic to textual and pragmatic aspects in computational linguistics renders systemic grammars ([Halliday, 1985]) increasingly attractive because they treat these fields in a single integrated model. Systemic grammars have been successfully employed especially in text generation as, for example, in the projects PENMAN ([Mann and Matthiessen, 1985]), COMMUNAL ([Fawcett and Tucker, 1990]), TECHDOC ([Rösner and Stede, 1994]) and KOMET ([Bateman *et al.*, 1991]). In contrast to this front-line position in text generation, the implementation of systemic grammars remains rather old-fashioned. The implemented processing strategies of the mentioned projects (see, e.g., [Penman Project, 1989]) are committed solely to the generation direction. Some experiments with systemically-based software for

[1] This work was partially supported by the Commission of the European Union through the project EP 6665 "Dandelion", and by the DAAD through grant D/96/17139.

analysis, which will be discussed later, are either not in standard formalisms or not applicable for real systemic grammars. So, unfortunately the reusability of systemic grammars has been restricted to similar generation projects.

This is due to the fact that systemic grammar has not developed its own logic. It focusses on providing linguistic resources for a broad range of empirical phenomena for often non-formal purposes; the elaboration of a sound logical foundation has not been a primary concern. Hence, it is of particular interest to investigate existing logics (here: feature logics) with respect to how appropriate they are for modelling the ideas and data presented by systemic grammars. This would clarify the theoretical status of the means and mechanisms used in systemic grammar for the organization of grammatical resources. Additionally, it is a precondition for a more adequate implementation that supports bidirectionality.

Before proceeding, it should be noted that we will use the term "systemic grammar" to refer to a systemic grammar in the narrow, original sense – i.e., to the system network and realization statements. This will be precisely defined in section 2. The bidirectional representation and processing of such grammars is the topic of the present paper.[2] The main source of procedurality in systemic grammar implementations is the procedure for traversing the system network. The system network itself is already a declarative representation. What is missing is a tool allowing bidirectional processing of system networks.

There are a number of preliminary approaches concerning the bidirectional processing of systemic linguistic resources. In [Kasper, 1987] systemic grammar is represented with the help of Functional Unification Grammar (FUG). Kasper adopted here the original FUG ([Kay, 1979]) extended by a notion of implication.[3] The merit of this FUG representation is that it related systemic grammar to a feature logic for the first time. However, FUG did not include a typing facility to control unification, whereas contemporary feature logics involve the idea of typing. The obvious similarity between system networks and type hierarchies suggests that we raise once more the question of an appropriate feature logic representation for systemic grammar. This has been done by [Bateman *et al.*, 1992]. They have proposed a representation of systemic grammar in the Typed Feature System TFS ([Zajac, 1991]) where the system network functions directly as a type hierarchy. Their approach is the starting point for the present paper. The weakness of that approach is that it is implemented only for a very small example grammar which does not cover all theoretically possible connection possibilities inside a system network; nor does the example cover those connection possibilities that are in fact used in practical grammars. It cannot be

[2] In the generation projects mentioned, the decisions of the grammar are controlled by a linked semantic knowledge base. A consideration and re-representation of all three components – the grammar, the knowledge base, and the interface between them – is of high interest, but beyond the scope of this paper.

[3] A more recent experiment with the knowledge representation language LOOM ([Kasper and O'Donnell, 1990]) has not resulted in an applicable resource either. An implementation of extra control strategies for the analysis and generation direction, originally planned by the authors, has not been carried out.

generalized to a general transformation procedure into TFS.

Another approach close to our objective is presented in [O'Donnell, 1994]. O'Donnell represents systemic grammar in his own fully declarative knowledge representation language WAG-KRL, and was able to carry out generation and analysis for medium sized systemic grammars (ca. 500 features). Unfortunately, WAG-KRL has not a precisely defined semantics, so that the relation to other contemporary feature logics remains vague.

In this discussion, two further papers concerning the system network in its own right (without realization statements) should be mentioned. [Mellish, 1988] proposed a representation of system networks in PROLOG. The typical declarative processing inferences – subsumption and compatibility checking (unification) for systemic features – can indeed be elegantly simulated in the proposed lattice of PROLOG terms. The encoding, however, is so inscrutable that a translation from systemic features to PROLOG terms and back would be necessary for the user. The problem of the exponential complexity of this pre- and post-compiling has not been solved. Nevertheless, from the logical point of view, his construction is very valuable in that it is an exhaustive answer to the semantics of a system network. [Carpenter, 1992] has given another important contribution to the theoretical discussion about the nature of system networks. He presents a construction of type semilattices out of system networks. This construction, however, is only valid for system networks without disjunctions, a restriction which does not apply for any real systemic grammar. We will present a generalization of this construction below.

In the present paper, the capability of four feature logics (ALE, CUF, TDL, TFS) for representing systemic grammars will be compared. An automatic compilation into TFS, into CUF and into TDL notation based on the results of the theoretical comparison will be presented. Both representations tackle all the deficiencies of Bateman et al.'s approach. Finally, a re-implementation in CUF of the large English systemic grammar NIGEL developed in the PENMAN project ([Mann and Matthiessen, 1985]) is described.

2 Systemic Grammar

2.1 System Network

Systemic grammar takes its name from its representation in the form of a system network. A system network $\mathcal{SN} = [\mathcal{F}, \mathcal{S}]$ is a network which brings into relation a set of n grammatical features $\mathcal{F}$. These grammatical features can be seen as types forming a large type hierarchy.

Features can form *feature terms* – propositional logical expressions with the connectives *and* and *or*:

- f $\in \mathcal{F}$ is a feature term.
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ are feature terms, iff ϕ_1 and ϕ_2 are feature terms.

The basic unit of a system network is the so-called system. A system represents something which can be interpreted as the immediate subtype relationship

Interpretation $[\mathcal{D}, \varepsilon]$ of a system network $\mathcal{SN} = [\mathcal{F}, \mathcal{S}]$

- $\mathcal{D}$ is a nonempty set of objects, the domain.
- ε is an extension function mapping from terms over $\mathcal{F}$ to $2^{\mathcal{D}}$.
- $\varepsilon(rank) = \mathcal{D}$.
- Term interpretation
  $\varepsilon(\phi_1 \wedge \phi_2) = \varepsilon(\phi_1) \cap \varepsilon(\phi_2)$
  $\varepsilon(\phi_1 \vee \phi_2) = \varepsilon(\phi_1) \cup \varepsilon(\phi_2)$
- Disjointness of output features of one system
  $\varepsilon(f_1) \cap \varepsilon(f_2) = \emptyset$ for all $f_1, f_2 \in s$, for all $s \in \mathcal{S}$
- Exhaustive partition of the entry condition
  $$\bigcup_{f \in out(s)} \varepsilon(f) = \varepsilon(entry(s)) \text{ for all } s \in \mathcal{S}$$

Fig. 1. Interpretation for system networks

usually found in type hierarchies. A *system* s $= [entry(s), out(s)] \in \mathcal{S}$ consists of an *entry condition* $entry(s)$ and a set of *output features* $out(s) \subseteq \mathcal{F}$. The output features are understood as subtypes of the entry condition. No feature occurs in more than one system as output feature, i.e., the output feature sets form a partition of $\mathcal{F}$. Every output feature can be annotated with one or more *realization statements* which define the consequences of the choice of this feature for the syntactic surface realization. An *entry condition* $entry(s)$ is a feature term. The entry conditions establish the connections between the systems. Cyclic connections are disallowed.

A system network has one distinguished start feature *rank* which serves as the entry condition for the most general system of that network. It is the only feature which does not occur as an output feature.[4]

Although it might appear intuitively clear what a system network means, system networks do not in fact come with a precisely defined semantics; only subsequently has the work of Kasper, Mellish, Brew and Carpenter established indirectly a semantics for system networks by providing translations into formalisms with a well defined semantics. The interpretation of a system network lying behind all this previous work is summarized here in Figure 1. The domain can be understood as a set of well-formed sentences of a language. Term conjunction and disjunction is modelled by intersection and union of the corresponding extensions respectively; the output features of one and the same system are understood as grammatical alternatives having mutually disjoint extensions.

Given this, we can define a subsumption and a compatibility relation between

[4] The given definition is equivalent to that of [Brew, 1991, Mellish, 1988] and that of [Carpenter, 1992], although the notion of *system* is here defined differently. The limitation of the term "system" to choice systems (and the exclusion of so-called and-systems, conjunctive and disjunctive systems) made in this paper is fully compensated by allowing the possibility of nested logical expressions in the entry condition. This view of a system is more in accordance with the systemic terminology.

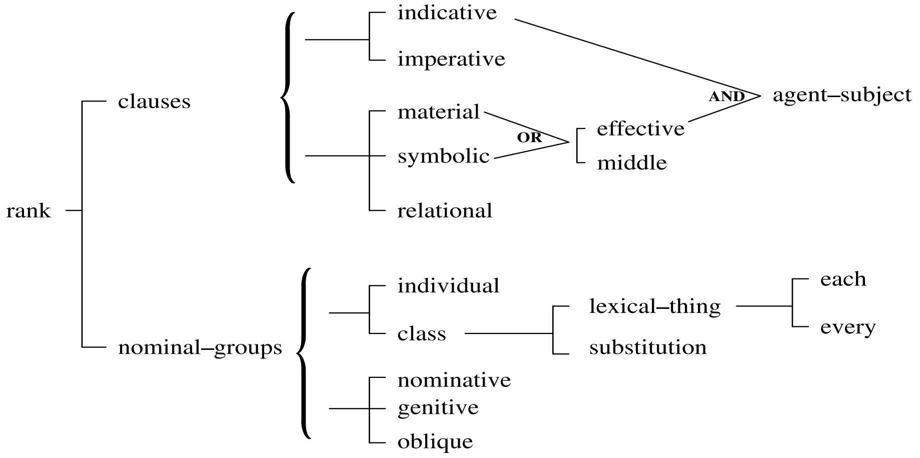

Fig. 2. Example network

systemic features:

Subsumption: f_1 is more specific than f_2 ($f_1 \leq f_2$) iff $\varepsilon(f_1) \subseteq \varepsilon(f_2)$.

Compatibility: $\{f_1, f_2, ...f_n\}$ are compatible iff $\varepsilon(f_1 \wedge f_2 \wedge ... \wedge f_n) \neq \emptyset$.

Maximal compatible feature sets are called *selection expressions*. Maximal means that it is not possible to add any further feature without rendering the corresponding term inconsistent. The notion of "selection expression" is broadly used in the PENMAN implementation and is there procedurally defined.[5] It will serve later on to define closed world type hierarchies out of system networks.

Figure 2 shows an example network with nine systems. The example is an extract from the NIGEL grammar describing unmarked SP(O) sentences in active voice. The whole NIGEL grammar consists of about 700 systems with over 1200 features. A system is here graphically represented with the help of the choice connector -[; the output features are placed on its right-hand side, the entry condition is represented by the lines on the left-hand side and are connected via a logical connective in case of more than one line[6]. Seven systems have a

[5] For the reader more familiar with the traditional generation perspective on system networks, the following procedural definition of a selection expression might be convenient. A *selection expression* is a subset of $\mathcal{F}$ resulting from a complete network traversal beginning with the start feature. Imagine the network as a flow chart where features are activated in succession. A system is active if all features necessary to fulfill its entry condition are already active. In an active system exactly one of its output features can now be activated. Hereby, parallel paths of active features can occur. The set of all features activated during one traversal is a *selection expression*.

[6] However, in real systemic grammars, the entry conditions can be much more complex and often include deeper embedded feature terms.

single feature as entry condition: *rank, clauses* (2 systems), *nominal-groups* (2 systems), *class* and *lexical-thing*; one has a disjunctive, and one a conjunctive entry condition. The conjunction (AND *indicative effective*) enters a system with only one output feature; such systems are also allowed in a system network. The braces in the Figure indicate that one and the same feature is the entry condition for more than one system. The definitions given above can be illustrated by examples from the example grammar as follows:

Subsumption	**Compatiblity**
each $\leq$ *class*	{*indicative, material*}
agent-subject $\leq$ *clauses*	{*mental, agent-subject*}

Selection expressions

$\tau_1 = \{$*rank, clauses, material, indicative, effective, agent-subject*$\}$
$\tau_2 = \{$*rank, nominal-groups, individual, nominative*$\}$
$\tau_3 = \{$*rank, nominal-groups, class, oblique, lexical-thing, every*$\}$

τ_1 has the sentence "Kim devours every cookie." in its extension, τ_2 and τ_3 the phrases "Kim" and "every cookie" respectively.

2.2 Realization Statements

The system network represents the sentences of a language by specifying the allowable combinations of features, the selection expressions. These features are paradigmatically based. They do not provide a syntactic structure. The realization of the chosen features in the form of a syntactic structure is controlled by an additional means, the so-called *realization statements*, which are associated with the paradigmatic features.

It is assumed that a sentence and every smaller phrase consists of a number of appropriate grammatical functions. Grammatical functions for the clause are, among others, Subject, Directcomplement, Actor, Goal, Theme, Medium, and for the nominal-group Deictic, Classifier and Thing. These functions are first introduced independently of each other, but may later on be conflated or ordered. Functions are the main means by which constituency is constructed in systemic grammar. Realization statements concerning grammatical functions are:

(INSERT *function*)
(CONFLATE *function1 function2*)
(ORDER *function1 function2*)
(ORDERATFRONT *function1*)
(ORDERATEND *function1*)
(PRESELECT *function feature*)

Conflation of two grammatical functions means that both functions are realized by one and the same substructure (e.g. Subject = Theme, Goal = Medium). The order is interpreted as immediate precedence (e.g. Subject < Finite). Front and final position in a phrase is declared by the one-argument realization statements ORDERATFRONT and ORDERATEND. Preselection expresses immediate dominance in systemic function structures: An inserted grammatical function

clauses	(insert process) (insert actor) (preselect actor nominal-groups) (insert theme) (orderatfront theme)	*indicative*	(insert subject) (conflate subject theme) (preselect subject nominative) (insert finite) (conflate process finite)
middle	(insert medium) (conflate medium actor)	*imperative*	(insert nonfinitive) (conflate nonfinitive process) (conflate nonfinitive theme)
effective	(insert medium) (insert goal) (preselect goal oblique) (conflate medium goal)	*agent-subject*	(conflate subject actor)
		nominal-groups	(insert thing)
		lexical-thing	(insert deictic) (order deictic thing)

Fig. 3. Realization statements for the example

is constrained to be filled by a substructure which itself is licensed by a selection expression containing the preselected feature. The realization statements completing the example network of Figure 2 are given in Figure 3.

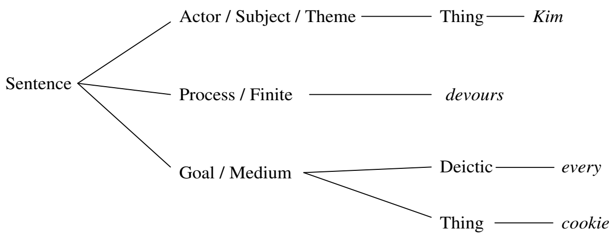

Fig. 4. Systemic function structure

The functioning of the realization statements is illustrated in Figure 4. Here the systemic function structure of the sentence "Kim devours every cookie." is shown as it would be generated by the example grammar. It reflects the syntagmatic realization of the selection expression τ_1 with regard to the clause level, and the realization of the selection expressions τ_2 and τ_3 for the Subject and Goal substructure respectively. For a detailed explanation of the generation of sentences with a systemic grammar the reader is referred to [Matthiessen and Bateman, 1991].

3 The Capability of Feature Logics for Expressing Systemic Grammars

For the intended compilation of systemic grammar we have examined four feature logic systems with type discipline to determine their capability to represent a systemic grammar and to serve as processing tool for it. Specifications have been carried out for the four formalisms ALE, TFS, CUF and TDL. These offer different kinds of type discipline and can be seen as a representative candidate selection from currently available state-of-the-art formalisms. The investigation is divided into two parts: First the representation of the system network will be considered, second the representation of the realization statements.

3.1 System Networks and Type Hierarchies

Since type hierarchies are increasingly used for structuring grammatical resources, one might ask if system networks are after all just type hierarchies. The obvious similarity between a system network and a type hierarchy at first sight suggests such a claim. But this approach has its obstacles. It is necessary to compare carefully the different definitions of the term "type hierarchy" as they can be found in different theoretical frameworks and assumed for the input specification of feature logic systems. In the following we compare the definition of a system network with the properties assumed for an input type hierarchy in ALE, TFS, CUF and TDL.

According to the definition given in section 2.1, a system network is a particular kind of classification device classifying the syntactic realization statements into a hierarchy. Besides the classical

(1) subtype relation (1400) and
(2) conjunctive multiple inheritance (400)

a system network has means for expressing

(3) disjunctive multiple inheritance (200),
(4) partition of classes (700),
(5) cross classification (350)[7].

The numbers in brackets give the frequencies of occurence in NIGEL.

Analogously to logic programming languages, feature logic systems can have open world or closed world semantics. Following [Gerdemann, 1995], systems with closed world semantics employ the *exhaustive species condition* and moreover the *disjoint species condition*. The exhaustive species condition asserts that if an object is of type t than the object is of at least one maximally specific type subsumed by t. The disjoint species condition asserts that the object is of no more than one maximally specific type subsumed by t. Hence, in a closed world system the unification of two types t_1, t_2, which share no common subtype, is always bottom (the inconsistent type). Deviating a little from [Gerdemann, 1995], we will apply the term *open world semantics* to a system in which two types t_1 and t_2 which share no user defined common subtype are always assumed to be

[7] Also called multidimensional inheritance (see [Erbach, 1994a]).

unifiable (the result could be written as $t_1 \wedge t_2$), as long as no explicit statement of their inconsistency is present. ALE and TFS are closed world systems maintaining the disjoint species condition.[8] In TDL one can choose between open or closed world semantics while CUF is an open world system. A system network is also a particular kind of an open world system in which the exhaustive species condition holds – but the disjoint species condition does not. Compiling a system network into ALE or TFS therefore means compiling the disjoint species (maximally specific types) which in a system network are left implicit.

Closed World Systems (ALE, TFS). The Attribute Logic Engine ([Carpenter and Penn, 1994]) requires as input a type hierarchy in the sense of [Carpenter, 1992]. This is a bounded complete partial order (= semilattice) where every set of compatible types has a unique common subtype. This strong kind of closed world semantics is also called GLB semantics. A system network does not meet this semilattice condition. It is not a type hierarchy in the sense of Carpenter. Hence, in order to represent a system network in ALE one has to precompile all common upper bounds. [Carpenter, 1992] has already proposed a construction for this based on the equivalence between system networks and ISA networks with ISNOTA links. But this construction is only valid for system networks without disjunctive entry conditions. Since in real systemic grammars disjunctions play a central role, this construction cannot be used for our purpose. Extending Carpenter's idea, we provide here a construction valid for general system networks (including disjunctions):

$$\textsc{type1} = \{\sigma |\ \sigma \subseteq \mathcal{F} \text{ and } \sigma \text{ is a subset of a selection expression containing only pairwise incomparable features}\}^{9}$$

TYPE1 includes all singleton feature sets with a systemic feature as element and moreover all compatible sets of features. This construction is equivalent to the *restricted powerset construction* for object-oriented programming languages given in [Aït-Kaci *et al.*, 1989]. A subset of a selection expression containing only pairwise incomparable features is called a *cochain* in this approach. Aït-Kaci points out that $[\textsc{type1}, \sqsubseteq, \sqcup\,]$ is a semilattice. The subtype relation $\sigma \sqsubseteq \tau$ holds iff for all $f_1 \in \sigma$ there exists an $f_2 \in \tau$ with $f_1 \leq f_2$. Unification $\sqcup$ is given by set union, removing all those features for which a more specific feature is contained in the union. Moreover, there exists a function $g : \mathcal{F} \longrightarrow \textsc{type1}$ which maps every feature $f \in \mathcal{F}$ into the singleton $g(f) = \{f\}$. g is a homomorphism with the nice property of preserving the ordering of $\mathcal{F}$ and the GLBs if they exist in $\mathcal{F}$. Hence, $[\textsc{type1}, \sqsubseteq, \sqcup\,]$ is the GLB preserving lower semilattice extension of $\mathcal{F}$. And it can be taken as input hierarchy for ALE.

Although TFS assumes the disjoint species condition, typing in TFS differs from ALE in the following way. TFS expresses compatibility by the existence of a common subtype for every set of compatible types. This must not necessarily be

[8] For ALE this is true as long as we consider the bare type hierarchy without feature constraints.

[9] If we restrict ourselves to networks without disjunctions, this construction is equivalent to the one given by Carpenter.

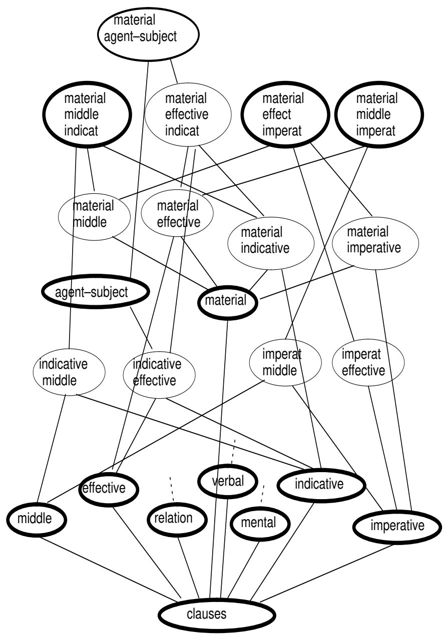

Fig. 5. ALE type hierarchy

unique. The input type hierarchy is thus meant to be a bounded partial order. The semilattice condition can be dropped. As input specification for reasoning with TFS, the partial order consisting of all selection expressions extended by all atomic types (singleton feature sets) – a subhierarchy of TYPE1 – is sufficient. In fact, we do not need the complete selection expressions because they include redundant information. If we define the *maximal restriction* of a feature set $\sigma \in 2^{\mathcal{F}}$ as the set of all maximal elements in σ [10], then the input type set for TFS can be defined as

$$\text{TYPE2} = \{\{\mathrm{f}\} \mid f \in \mathcal{F}\} \cup \{\sigma \mid \sigma \subseteq \mathcal{F} \text{ and } \sigma \text{ is the maximal restriction of a selection expression}\}$$

The subtype relation and join operation are similarily defined as for TYPE1.

Open World Systems (CUF, TDL). The type discipline in CUF ([Dörre *et al.*, 1996]) and TDL ([Krieger, 1994]) differs from the previous systems in that all types which are not explicitly declared as inconsistent can unify. Thus compatible types do not need a user-defined common upper bound. This is the

[10] This reduction is taken from [Carpenter, 1992], and can also be found in [Aït-Kaci *et al.*, 1989]

same interpretation of compatibility as is found in system networks. This kind of open world semantics makes it possible to represent a system network in CUF and in TDL directly as type hierarchy. Every feature is a type:

TYPE3 $= \mathcal{F}$.

However, there are distinctions between the CUF and the TDL representation. CUF is closer to the systemic representation style. It includes a theorem prover, and allows the expression of arbitrary type axioms including the definition of partitions. Hence, compiling into CUF means that every system is a type axiom:

```
clauses = indicative | imperative.
clauses = material | symbolic | relational.
```

In TDL, the types are defined as 'subtype $<$ supertype' statements. Partitions can only be stated once for a type, that means, partitioning of a type along different dimensions is not possible. Here, inconsistency statements for the ouput features of one and the same system have to be added:

```
begin :type.
   indicative :< clauses.
   imperative :< clauses.
   material :< clauses.
   symbolic :< clauses.
   relational :< clauses.
end :type.
```

```
begin :declare.
   nil = indicative & imperative.
   nil = material & symbolic.
   nil = material & relational.
   nil = symbolic & relational.
end :declare.
```

This representation is not, however, fully equivalent to the logic of a system network.

Let us compare the four input type hierarchies concerning the number of types. The ALE input specification [TYPE1, $\sqsubseteq, \sqcup$] has the structure of a type hierarchy according to Carpenter, but, for any real grammar, an intuitively ungraspable number of types of the order of 2^n. The TFS specification is located near to the ALE semilattice. The number of types here is of the order of $\binom{n}{k}$, where k is the depth of the network.[11] The source for the explosion of the type number is the lack of appropriate means for expressing disjunction and cross classification.[12] On the other hand, the CUF and the TDL specifications (and similarly the system network) bring in relation a smaller number of items – the n grammatical features – with the help of the five structuring means: subtype relation, conjunctive inheritance, disjunctive inheritance, partition (only in CUF)

[11] It should be remarked that the number of arguments in Mellish's PROLOG terms is exactly the number of possible selection expressions. Every feature is represented by a certain covering assignment of the PROLOG arguments with 0 or 1. This assignment mirrors exactly in what selection expression the feature occurs. Hence, our program for the transformation of a system network into TYPE2 can serve as well as an implementation of the translation into Mellish's PROLOG encoding.

[12] A reduction of the type set can be achieved by introducing features to represent parallel subtype partitions. This is possible only in the case when the subtypes do not have constraints attached which is not valid for real systemic grammars.

(INSERT function)	[function: rank]
(CONFLATE fun1 fun2)	[fun1: X, fun2: X]
(PRESELECT fun feature)	[fun: feature]
(ORDER fun1 fun2)	[fun1: X, fun2: Y, order: append(_,[X, Y | _])]
(ORDERATFRONT fun)	[fun: X, order: [X | _])]
(ORDERATEND fun)	[fun: X, order: append(_,[X])]

Fig. 6. Realization statements as feature structures

and cross classification. The comparison is illustrated in Figure 5. It shows a subhierarchy of the ALE semilattice for the example network. The bold types are the necessary types for a TFS input. The corresponding CUF and TDL input includes only the ten types consisting of exactly one feature.

Other current formalisms relevant to the discussion are PROFIT and CL-ONE. The PROFIT system ([Erbach, 1994b]) – particularly developed for the treatment of cross classification – has not yet been included in the comparison. Presumably it occupies a position between TFS and CUF in that it provides means for conjunctive and multidimensional, but not for disjunctive, inheritance. CL-ONE ([Erbach *et al.*, 1995]) in its current relase is comparable to ALE except that the GLB semantics can be released in a similar way as in TFS. The significant easier precompilation for open-world systems may be an indicator of a closer relationship between terminological logics and systemic grammars.

3.2 Realization Statements and Feature Structures

The representation of realization statements as feature structures (except for the ORDER statements) can be taken straightforwardly from [Bateman *et al.*, 1992]. Insertion is a feature introduction, preselection is a type constraint for a certain feature, conflation can be modelled by coreference. The realization statements associated with one output feature *feature* form together a feature structure which is associated with the singleton type {*feature*}. The complex types in TYPE1 and TYPE2 have no feature assignments. They inherit all appropriate features from their supertypes. Figure 6 shows the encoding.

ALE, TFS, CUF and TDL provide mainly the same means for expressing feature structures. However, there are some minor distinctions which play a role in our task: (1) In ALE features have to be introduced only once at a most general type. (2) CUF's and ALE's type systems do not support coreferences.

The linear precedence of inserted grammatical functions is controlled by the three kinds of order statements given above. To represent this in a feature algebra, we use list unification. However, it has turned out that this direct encoding of the systemic order statements with the help of list unification is a source of severe inefficiency because of the uninstantiated start and tail of the lists.

4 Implementation

Compilation. Implementations of an automatic compilation transforming arbitrarily complex systemic grammars written in the PENMAN style into an equivalent type signature with associated feature structures have been carried out for TFS, for CUF and for TDL. The implementations do not include the treatment of word order. The compilation procedure into TFS compiles all selection expressions from a given system network. Starting from the trivial feature set $\{rank\}$ (and the set of already passed systems $passed = \emptyset$), compatible feature sets are expanded in a breadth first network traversal. Every expansion step generates a new set of feature sets which have to be considered for further expansion. In the following sketch of the algorithm, it is assumed that for all $s \in \mathcal{S}$, $entry(s)$ is given in disjunctive normal form.

expand-one-selection-expression($featureset, passed$):
1 **for all** $f_i \in featureset$ **do** $in_i :=$ feature-expansion($f_i, featureset, passed$)
2 **if** $in_1 \cup in_2 \cup ... \cup in_n = featureset$
  then $featureset$ is type in TFS
  else $newpassed := relevants_1 \cup relevants_2 \cup ... \cup relevants_n \cup passed$
    for all $set \in in_1 \times in_2 \times ... in_n$
    do expand-one-selection-expression($set, newpassed$).

feature-expansion($f, featureset, passed$):
1 $fsystems := \{s | s \in \mathcal{S}$ and f occurs in $entry(s)\}$
2 **if** $fsystems = \emptyset$
  then return $\{\{f\}\}$
  else let $ors := \{s | s \in fsystems$ and there is a disjunction in $entry(s)\}$
    $ands := fsystems \setminus ors$
    $ands1 := \{s | s \in ands$ and the features in $entry(s)$ form a subset of $featureset\}$
    $diss := \{s | s \in ors$ and f is not conjunctively bound in $entry(s)\}$
    $cons := \{s | s \in ors$ and f is conjunctively bound in $entry(s)\}$
    $cons1 := \{s | s \in cons$ and the conjunction, in which f occurs, is a subset of $entry(s)\}$
    $relevants := (diss \cup cons1 \cup ands1) \setminus passed$
   if $ors \neq \emptyset$ **or** $ands \setminus ands1 \neq \emptyset$
    then return $\{f\} \times out(s_1) \times ... \times out(s_n)$ for $s_i \in relevants$
    else return $out(s_1) \times ... \times out(s_n)$ for $s_i \in relevants$.

Because of the one-to-one-correspondence between systemic features and types, compilation into TDL and into CUF is straightforward and fast. We dispense with a presentation of the algorithm; it is obvious from comparing the mood distinction system (*indicative/imperative*) in Figure 2 with its encoding in Figure 7.

The TDL representation is organized as a *subtype* $<$ *supertype* hierarchy. It should be remarked that the partition property of the output features of one sys-

```
clauses = indicative | imperative.

indicative :: subject: nominative,
              finite: rank.
imperative :: nonfinitive: rank.

indicative_sort := indicative &
                     finite: X1 & subject: X2 &
                     theme: X2 & process: X1.
imperative_sort := imperative &
                     theme: X1 & nonfinitive: X1.
```

```
nil = indicative & imperative.

indicative := clauses & [subject #s & nominative,
                          finite #fin & rank,
                          theme #s,
                          process #fin].
imperative := clauses & [nonfinitive #non & rank,  theme #non].
```

Fig. 7. CUF encoding, TDL encoding

tem has to be delared extra as pairwise inconsistency declarations, and because TDL does not allow identification of type terms without feature structures, those cases (systems with only one output feature and without realizations) have to be modelled as templates in TDL.

The compilation into CUF notation is even more straightforward. But it requires a special treatment of the systemic conflations because their representation as coreferences is not possible with CUF types. So, in CUF every systemic feature is represented twice: once as type and second as sort (see Figure 7). The types serve to express the logical connections between the features and are especially necessary for expressing the disjoint subtyping and cross classification. The sorts are employed to model the realization statements and are necessary to express coreferences. The sort hierarchy is organized in a top-down fashion.

Processing. Inferences concerning the bare system network – subtype and compatibility checking – are possible in all three representations without any restriction. Possible inferences with respect to the complete grammar formalism, including the realization statements, are the generation of the syntactic structure out of given systemic features and the classification of a given syntactic structure in terms of features of $\mathcal{F}$ (sometimes also called type inference). The inferential behavior differs between the three systems as is shown in Figures 8 – 10: where the results of two identical generation and classification queries taken from our example grammar are given.

Processing with TFS provides a behavior which is called 'total sort resolving'

```
tfs> ? MATERIAL & INDICATIVE.       tfs> ? [Subject: [Thing: LEX-KIM],
                                            Process: LEX-DEVOUR,
                                            Goal: [Thing: LEX-COOKIE]].
KB DUB_region(5):

AGENT-SUBJECT*MATERIAL
 [Finite:  #3=LEX-DEVOUR
              [spelling: <"devour">],
  Goal:    #1=EVERY*OBLIQUE
              [Deictic: LEX-EVERY
                        [spelling: <"every">],
               Thing:   LEX-COOKIE
                        [spelling: <"cookie">]],
  Medium:  #1,
  Subject: #2=INDIVIDUAL-NAME*NOMINATIVE
             [Thing: LEX-KIM[spelling: <"Kim">]],
  Theme:   #2,
  Actor:   #2,
  Process: #3]
```

Fig. 8. TFS example

(see [Pollard and Sag, 1994]) where the partial information given in a query is completed by all consistent information found in the type hierarchy including the specialization into most specific subtypes. Corresponding analysis and generation queries provide one and the same completed feature structure. The TFS feature structure in Figure 8 occurs as the result of both queries above it.

Assuming the given CUF-encoding, queries consisting of partial information are completed without specialization into more specific types than the asked ones (in the generation direction) and without respect to conflations (in the analysis direction). The partial character of the reasoning is no problem as long as the generation direction is considered – at least as long as one does not want to handle underspecified input. Figure 9 shows a generation and an analysis inference of our example. In principal, total sort resolving in CUF can be achieved by adding two special sorts to the sort hierarchy as suggested by Jochen Doerre (personal communication) which control the inheritance of the conflations bottom-up and recursively.

```
conflations := (~indicative ; indicative_sort)&(~imperative ; imperative_sort) & ...
con_recursion := conflations & (~(subject: top) ; subject: con_recursion) &
            (~(finite: top) ; finite: con_recursion) & ...
```

The TDL encoding is only applicable to generation because TDL is not designed to perform type inference (classification). Figure 10 gives an example.

The toy examples serve here to illustrate the differences. However, more interesting is the application of the obtained compilation procedures to real systemic grammar resources. This will be discussed here with respect to the most

```
| ?- cuf(material_sort & indicative_sort).

%%% used time (msec): 309
%%% result:

  +-- (indicative & material)
  |actor: (1) +-- (class_name ; individual_name)
  |           |thing: (2) start
  |           +--
  |finite: (3) disposal_verb
  |process: (3)
  |subject: (4) nominative
  |theme: (4)
  +--

  (1) & individual_name --> (2) & proper_noun
  (1) & class_name --> (2) & common_noun
  (1) & individual_name --> (2) & proper_noun
  (1) & class_name --> (2) & common_noun

| ?- | ?- cuf(process: word(lex_devour) &
              subject: thing: word(lex_kim) &
              goal: (deictic: word(lex_every) &
                     thing: word(lex_cookie))).

%%% used time (msec): 354
%%% result:

  +-- (effective & indicative & ~symbolic)
  |goal: +-- (every & oblique)
  |      |deictic: +-- every_form
  |      |         |spelling: ["every"]
  |      |         +--
  |      |thing: +-- common_noun
  |      |       |spelling: ["cookie"]
  |      |       +--
  |      +--
  |process: +-- disposal_verb
  |         |spelling: ["devour"]
  |         +--
  |subject: +-- individual_name
  |         |thing: +-- proper_noun
  |         |       |spelling: ["kim"]
  |         |       +--
  |         +--
  +--
```

Fig. 9. CUF example

```
<EXA:TYPE> test := material & indicative.
<EXA:TYPE> expand-type 'test.
<EXA:TYPE> fgp 'test.
```

$$\begin{bmatrix} \textit{test} & \\ \text{THEME } \boxed{1} & \begin{bmatrix} \textit{nominative} \\ \text{THING } \textit{start} \end{bmatrix} \\ \text{PROCESS } \boxed{2} & \textit{disposal-verb} \\ \text{FINITE } \boxed{2} & \\ \text{SUBJECT } \boxed{1} & \\ \text{ACTOR} & \begin{bmatrix} \textit{nominal-groups} \\ \text{THING } \textit{start} \end{bmatrix} \end{bmatrix}$$

Fig. 10. TDL example

elaborated systemic grammar, NIGEL. NIGEL is a grammar for English. It covers in terms of the recently proposed EAGLES standard: sentence types (main and subordinated clauses, imperative, yes-no questions and tagging, wh-questions), diathesis, modality and tense, clause complementation (including prop subject, complementizer, predicate constructions, infinitival complements), clause modification, NP complementation, NP modification, agreement and coordination for clauses and NPs. The sentence length depends on the semantic input and is not restricted.

The application of the TFS precompilation for NIGEL is limited because the type number explodes combinatorically with the number of disjunctions and cross classifications. We have so far only succeeded in representing in TFS internally coherent subnetworks of NIGEL consisting of up to 150 features. So the TFS implementation (and all the more any ALE implementation) is not suited to serve as the processing tool for large scale systemic grammars. We have succeeded in providing a CUF implementation of the whole NIGEL grammar, however. For this it was necessary to install a CUF system with maximally 6000 PROLOG variables, instead of 1000 as allowed in the standard release. The specification includes the realization statements (excluding order statements). The following table presents an overview over the average run times with the CUF-NIGEL for different phrase types – called *ranks* in systemic terminology. The generation starts from a complete selection expression for one rank and returns the corresponding function structure for this rank. This is a partial structure lacking detailed substructures. Hence, run times for a recursively fully specified function structure are obtained by adding the run times of all its subconstituents to the given time. The classification starts from a partially specified function structure (as might be the result of a conventional parser) and returns the set of systemic features which can be deduced from the structure. The result is an often very incomplete selection expression which unfortunately tends to contain the less significant systemic features.

		time in ms
Generation	PP	20
	NP	600
	clauses	2000-7000
Classification	PP	80
	NP	80-500
	clauses	600-1000

The run-time tests have been carried out on a SPARC 20 with an example set delivered with the current NIGEL release. The sentence length ranges from 5 to 30 words.

5 Conclusion

The compilation experiments have shown the bias between using a closed world system (TFS) which requires a huge, and for large systemic grammars an unmanageable precompilation and using an open world system (CUF, TDL) which is characterized by an easy precompilation and by notational transparency, so that every systemic linguist can make the transition to the new representation.

For the time being, we would state that CUF offers the most appropriate means for the representation of systemic grammars so far because it has been the only approach succeeding in compiling NIGEL at all. The presented implementation is superior to Mellish's PROLOG approach in that it includes the realization statements. It differs from the FUG work of Kasper because it makes use of a type hierarchy to limit the search space for unification. It is superior to that of [Bateman *et al.*, 1992] in that it is applicable to arbitrarily complex system networks, and in that it is generated automatically. The implementation of systemic grammar in CUF is an essential contribution to the current research discussion concerning grammar migration. It is theoretically valuable because it is a fully declarative representation for systemic grammar in a standard feature logic system thus enabling a more direct comparison with other grammar models and the exchange and the combination with linguistic resources of other research streams. The implementation of systemic grammar in CUF is practically of interest at least for two reasons: First, it provides a good environment for testing the internal consistency of large systemic grammars. This is illustrated by the fact that CUF has detected inconsistent systems with contradictionary entry conditions and inappropriate realization statements in NIGEL which the PENMAN software did not determine. Second, smaller unification tasks (see below) can be carried out in the CUF representation. On the other hand, the compiled CUF-NIGEL is not practically useful for parsing. Parsing on the basis of type deduction only is not efficient even for more common grammars, all the more for systemic grammars. It is typical that all previous attempts with systemic parsers (except for [O'Donnell, 1994]) are based on a separate phrase structure grammar. To come closer to a real systemic parser, we plan the automatic extraction of well-formedness conditions and structural commonalities between systemic structures scattered around the system network of a given systemic grammar. For this, a first step is an automatic compilation of all SUBCAT lists licensed by the grammar. This requires a unification based implementation of systemic grammar, namely the CUF implementation.

Acknowledgements

I would like to thank John Bateman for originally suggesting and encouraging this topic of research. I would also like to thank Jochen Dörre for the exceptionally good communication concerning this application of the CUF system, Stephan Oepen and Hans-Ullrich Krieger for the installation and useful hints with regard to the TDL system, and one anonymous reviewer for his very constructive and detailed recommendations.

References

[Aït-Kaci *et al.*, 1989] Hassan Aït-Kaci, Robert Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115 – 146, 1989.

[Bateman *et al.*, 1991] John A. Bateman, Elisabeth A. Maier, Elke Teich, and Leo Wanner. Towards an architecture for situated text generation. In *International Conference on Current Issues in Computational Linguistics*, Penang, Malaysia, 1991.

[Bateman *et al.*, 1992] John A. Bateman, Martin Emele, and Stefan Momma. The nondirectional representation of Systemic Functional Grammars and Semantics as Typed Feature Structures. In *Proceedings of COLING-92*, volume III, pages 916 – 920, 1992.

[Brew, 1991] Chris Brew. Systemic Classification and its Efficiency. *Computational Linguistics*, 17(4):375 – 408, December 1991.

[Carpenter and Penn, 1994] Bob Carpenter and Gerald Penn. *The Attribute Logic Engine, User's Guide.* Computational Linguistics Program, CMU Pittsburgh, 1994. Version 2.0.

[Carpenter, 1992] Bob Carpenter. *The Logic of Typed Feature Structures.* Cambridge University Press, Cambridge, England, 1992.

[Dörre *et al.*, 1996] Jochen Dörre, Michael Dorna, and Jörg Junger. *The CUF User's Manual.* Institut für maschinelle Sprachverarbeitung (IMS), Universität Stuttgart, Germany, 1996.

[Erbach *et al.*, 1995] G. Erbach, M. van der Kraan, S. Manandhar, H. Ruessink, and C. Thiersch. Extending unification formalisms. In *Proceedings of the 2nd Language Engineering Convention*, London, 1995.

[Erbach, 1994a] Gregor Erbach. Multi-dimensional inheritance. In H. Trost, editor, *KONVENS '94*, pages 102 – 111, Vienna, 1994.

[Erbach, 1994b] Gregor Erbach. *ProFIT 1.05 user's guide.* Computerlinguistik, Universitaet des Saarlandes, 1994.

[Fawcett and Tucker, 1990] Robin P. Fawcett and Gordon H. Tucker. Demonstration of GENESYS: a very large, semantically based systemic functional grammar. In *13th. International Conference on Computational Linguistics (COLING-90)*, volume I, pages 47 – 49, Helsinki, Finland, 1990.

[Gerdemann, 1995] Dale Gerdemann. Open and closed world types in NLP systems. In *Proceedings of 5. Fachtagung der Sektion Computerlinguistik der DGfS*, pages 25 – 30, Duesseldorf, 1995.

[Goetz and Meurers, 1995] Thilo Goetz and Walt Detmar Meurers. Compiling HPSG type constraints into definite clause programs. In *Proceedings of ACL-95*, 1995.

[Halliday, 1985] Michael A.K. Halliday. *An Introduction to Functional Grammar*. Edward Arnold, London, 1985.

[Kasper and O'Donnell, 1990] Robert T. Kasper and Michael O'Donnell. Representing the Nigel grammar and semantics in LOOM. Technical report, USC/Information Sciences Institute, Marina del Rey, California, 1990.

[Kasper *et al.*, 1995] Robert Kasper, Bernd Kiefer, Klaus Netter, and K. Vijay-Shanker. Compilation of hpsg to tag. In *Proceedings of ACL-95*, 1995.

[Kasper, 1987] Robert T. Kasper. Systemic grammar and functional unification grammar. In James D. Benson and William S. Greaves, editors, *Systemic Perspectives on Discourse, Volume 1*. Ablex, Norwood, New Jersey, 1987.

[Kay, 1979] Martin Kay. Functional grammar. In *Proceedings of the 5th meeting of the Berkeley Linguistics Society*, pages 142 – 158. Berkeley Linguistics Society, 1979.

[Krieger, 1994] Hans-Ulrich Krieger. *TDL - A Type Description Language for HPSG. Part 2: User Guide*. DFKI, Saarbruecken, 1994. Research Report D-94-14.

[Mann and Matthiessen, 1985] William C. Mann and Christian M.I.M. Matthiessen. Demonstration of the Nigel text generation computer program. In James D. Benson and William S. Greaves, editors, *Systemic Perspectives on Discourse, Volume 1*, pages 50–83. Ablex, Norwood, New Jersey, 1985.

[Matthiessen and Bateman, 1991] Christian M.I.M. Matthiessen and John A. Bateman. *Text generation and systemic-functional linguistics: experiences from English and Japanese*. Frances Pinter Publishers, London, 1991.

[Mellish, 1988] Christopher S. Mellish. Implementing systemic classification by unification. *Journal of Computational Linguistics*, 14(1):40–51, 1988.

[O'Donnell, 1994] Michael O'Donnell. *Sentence analysis and generation: a systemic perspective*. PhD thesis, University of Sydney, Department of Linguistics, Sydney, Australia, 1994.

[Penman Project, 1989] Penman Project. PENMAN documentation: the Primer, the User Guide, the Reference Manual, and the Nigel manual. Technical report, USC/Information Sciences Institute, Marina del Rey, California, 1989.

[Pollard and Sag, 1994] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, Chicago, Illinois, 1994.

[Rösner and Stede, 1994] Dietmar Rösner and Manfred Stede. Generating multilingual documents from a knowledge base: the TECHDOC project. In *Proceedings of the 15th. International Conference on Computational Linguistics (*COLING 94*)*, volume I, pages 339 – 346, Kyoto, Japan, 1994.

[Zajac, 1991] Remi Zajac. *Notes on the Typed Feature System*. Project Polygloss, IMS-CL /IfI-AIS, 1991. version 4.