TRAVERSING THE LABYRINTH OF FEATURE LOGICS FOR A DECLARATIVE IMPLEMENTATION OF LARGE SCALE SYSTEMIC GRAMMARS[1]

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

Summary

Although systemic grammars are represented declaratively (as system networks), all successful implementations to date have been committed to the generation direction. The aim of this paper is, therefore, to re-represent systemic grammar resources in a typed feature logic in order to make it reusable in generation and analysis applications on the one hand, and to clarify the theoretical status of the mechanisms used in systemic grammar on the other hand. Three such representations in the feature logic formalisms ALE, TFS and CUF are compared. Implementations in TFS and CUF are given.

1 Introduction

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 one integrated model. Systemic grammars have been successfully employed especially in text generation as, for example, in the projects PENMAN ([Mann, 1983]), 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 solely committed to the generation direction. Some experiments with systemically based software for analysis remain in preliminary stages ([O'Donnell, 1993, O'Donoghue, 1991]). 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 primarily been a 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 the precondition for a more adequate implementation that supports bidirectionality.

[1]This work was partially supported by the Commission of the European Communities through the project EP 6665 "Dandelion"

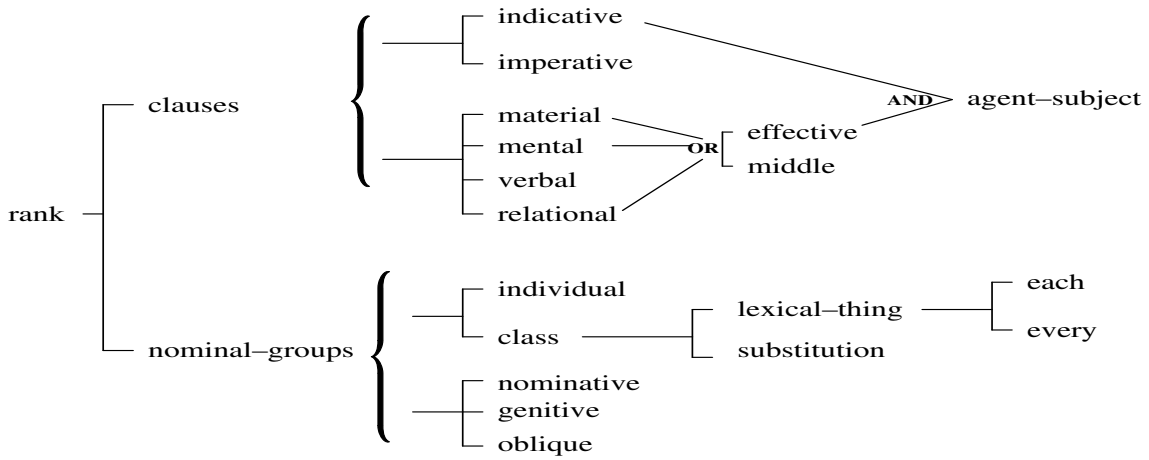

Figure 1: Example network

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. It 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), and in [Kasper and O'Donnell, 1990] with the help of the knowledge representation language LOOM. The experiment with FUG has the main disadvantage that a typing scheme for the control of the unification did not exist. The more recent experiment with LOOM has not resulted in an applicable resource either. An implementation of extra control strategies for the analysis and generation direction, originally planned by [Kasper and O'Donnell, 1990], has not been carried out.

The merit of the FUG representation by Kasper is that it relates systemic grammar to a feature logic for the first time. Contemporary feature logics, however, 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 type hierarchy. Their approach is the starting point for the present paper. The weakness of that approach is that it is implemented only for

[2] In the mentioned generation projects, however, 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 is beyond the scope of this paper.

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

Figure 2: Realization statements for the example

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 generalized to a general transformation procedure into TFS.

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 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 complexity of this pre- and post-compiling has not been clarified. 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 three feature logics (ALE, CUF, TFS) for representing systemic grammars will be compared. An automatic precompilation into TFS and into CUF notation based on the results of the theoretical comparison will be presented. Both representations tackle all deficiencies of Bateman et al.'s approach. Finally, a reimplementation in CUF of the large English systemic grammar NIGEL developed in the PENMAN project ([Mann and Matthiessen, 1983]) is described.

Because of the notational equivalence of system networks and the HPSG formalism, first mentioned by [Carpenter, 1992], the following considerations are also valid for HPSG implementations, especially for the lexical hierarchy where cross classification takes place.

```
<expression> ::= <feature> |
                 (AND <expression> <expression>+) |
                 (OR <expression> <expression>+)
```

Figure 3: Syntax of the entry condition

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* SN = [FEAT, S] is a set of systems S defined over a set of n grammatical features FEAT. Every system s = [e(s), out(s)] $\in$ S consists of an entry condition e(s) and a set of output features out(s) $\subseteq$ FEAT. No feature occurs in more than one system as output feature, i.e., the output feature sets form a partition of FEAT. The *output features* of one system represent alternative grammatical options. 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* e(s) is a propositional logic expression over features of FEAT with the connectives AND and OR. A pecise definition of the syntax of the entry condition in BNF notation is given in Figure 3. The entry conditions establish the connections between the systems. Cyclic connections are thereby disallowed.

A system network has one distinguished *start feature* 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.[3]

Figure 1 shows an example network with eight systems. A system is here graphically represented with the help of the choice connector ⊣, the output features are placed to its right-hand side, the entry condition is represented by the lines of the left-hand side which are connected via a logical connective in case of more than one line. The example is an extract from the NIGEL grammar ([Mann and Matthiessen, 1983]), developed within the PENMAN project, describing unmarked SP(O) sentences in active voice. The whole NIGEL grammar consists of about 700 systems with over 1200 features.

2.2 Realization statements

The system network represents the sentences of a language by specifying the allowable combinations of features. These features are paradigmatically based. The realization of the chosen features in the form of a surface string is controlled by an additional kind

[3]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.

of statements, called *realization statements*, which are associated with the paradigmatic features.

It is supposed 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, for the nominal-group Deictic, Classifier and Thing. These functions are first introduced independently of each other, but may later on be conflated (e.g. Subject = Actor, Goal = Medium) or ordered (Subject < Finite). 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*)

The order is interpreted as immediate precedence. 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 is constrained to be filled by a substructure possessing the preselected feature. The realization statements completing the example network of Figure 1 are given in Figure 2.

3 The capability of feature logics for expressing systemic grammars

For the intended re-representation, we have examined three 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 three formalisms ALE, TFS and CUF. These offer different kinds of a type discipline and can be seen as a representative candidate selection from currently available state-of-the-art formalisms.

3.1 System networks and type hierarchies

Since type hierarchies are increasingly used for structuring grammatical resources, one might ask if system networks are eventually 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 as assumption 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 and CUF.

Specificity
each strictly $\leq$ *class*,
agent-subject weakly $\leq$ *material*,
agent-subject strictly $\leq$ *clauses*
Inconsistency
material # *relational*,
individual # *every*
Compatibility
{*indicative, material*},
{*mental, agent-subject*}

Figure 4: Feature relations

τ_1 = {*rank, clauses, material, indicative, effective, agent-subject*}
τ_2 = {*rank, nominal-groups, individual, nominative*}

Figure 5: Selection expressions

According to the definition given in 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)

it has means for expressing

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

The numbers in brackets give the frequence of their occurence in NIGEL.

In preparing the construction of type hierarchies equivalent to a system network, we define between systemic features relations which are candidates for serving as subtype and unifiability relations in those type hierarchies. Although it might appear intuitively clear what a system network means, system networks do not in fact come with a precisely defined semantics. The representations of Kasper, Mellish and Carpenter then established a semantics for system networks indirectly by a mapping to formalisms with a well defined semantics. We do not want to make the detour over a third formalism to come to our candidate formalisms, however, but give the definitions directly in terms of the system network definition in 2.1 (see Figure 4 for examples):

- Specificity (= Subtype relation)

  Let p,q $\in$ FEAT. p is *more specific* than q (p $\leq$ q), iff p=q or if there is a system s with q occurs in e(s) and p $\in$ out(s)[5]. p is *strictly more specific* than q, if additionally q occurs as a conjunct in the conjunctive normal form of e(s). Both relations can be extended transitively. p and q are called mutually *dependent*, if p is strictly more specific than q or q is strictly more specific than p.

- Inconsistency

[4]Also called multidimensional inheritance (see [Erbach, 1994]).
[5]If we imagine the generation direction, the choice of q is a *precondition* for the choice of p.

p and q (p≠q) are called *inconsistent*, if they are output features of the same system or if there is a feature r, q strictly $\leq$ r, and p and r are output features of the same system.

- Compatibility

  The intended semantics of compatibility in system networks can be captured in the notion of 'selection expression' as used in the PENMAN implementation. Assuming that the entry conditions are given in disjunctive normal form, a *selection expression* is inductively defined as a subset σ of FEAT with

  - The start feature $\in \sigma$.
  - If p $\in \sigma$, for each system s where p occurs in e(s) and p is not conjunctively bound, exactly one of the output features $\in$ out(s) is in σ also.
  - If $\tau \subseteq \sigma$, then for each system s where the features of τ occur as the features of a conjunction in e(s), exactly one of the output features of that system is in σ also.[6]

  A subset σ of FEAT is called *compatible*, if there exists a selection expression τ with $\sigma \subseteq \tau$.Figure 5 gives selection expressions for the example network.

ALE type hierarchy

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. A system network does not meet this semilattice condition. It is not a type hierarchy in the sense of Carpenter. Therefore [Carpenter, 1992] sets out how to construct such a type hierarchy out of a system network. The construction is 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):

TYPE1 = $\{\sigma |\ \sigma \subseteq$ FEAT and σ is a subset of a selection expression$\}$

[TYPE1, $\subseteq, \cup$] forms a semilattice with the subset relation as subtype relation and set union as join.[7]

[6]For the reader more familiar with the traditional generation perspective of system networks, the following informal definition of a selection expression might be convenient. A *selection expression* is a subset of FEAT 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 fulfil 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*.

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

It is a type hierarchy in the sense of [Carpenter, 1992] and can be taken as input type hierarchy for ALE. Types including mutually dependent features are redundant. We can restrict ourselves to feature sets containing only pairwise independent features. τ_1 is represented then by {*material, agent-subject*}, τ_2 by {*individual, nominative*}. Thus the type set is reduced by a factor equal to the depth of the network. The subtype relation and the join operation have to be modified accordingly: σ subsumes τ ($\sigma \sqsubseteq \tau$), iff for all features s $\in \sigma$ there exists a feature t $\in \tau$ which is strictly more specific than s. Unification $\sqcup$ is given by set union, removing all those features for which a strictly more specific feature is contained in the union.[8] We denote the reduced semilattice by [TYPE1′, $\sqsubseteq, \sqcup$].

Based on a mapping g : FEAT $\mapsto$ TYPE1′ with $g(f) := \{f\}$, exactly the same inferences concerning the subtype and compatibility relation in [FEAT, SN] and [TYPE1′, .., ..] can be drawn:

(1) Subtype relation
For all $f1, f2 \in$ FEAT, it holds that:
$f1$ strictly $\leq f2$ iff $g(f2) \sqsubseteq g(f1)$.
(2) Compatibility
For all $f1, f2, ..., fn \in$ FEAT, it holds that:
$\{f1, f2, ...fn\}$ are compatible in [FEAT, SN] iff
$g(f1), g(f2)...g(fn)$ have a common upper bound in [TYPE1′,$\sqsubseteq, \sqcup$].

Proof: (1) Let $f1$ be strictly$\leq f2$ for any $f1, f2 \in$ FEAT. According to the definition of the subtype relation in [TYPE1′, $\sqsubseteq, \sqcup$], this is identical with $\{f2\} \sqsubseteq \{f1\}$ (one should take $\tau = \{f1\}$ and $\sigma = \{f2\}$). Thus $g(f2) \sqsubseteq g(f1)$.

(2) Let $\sigma = \{f1, f2, .., fn\}$ be any set of compatible features of FEAT. Obviously, $\sigma \in$ TYPE1. The hypothesis is that $g(f1), g(f2), ...g(fn)$ have a common upper bound in TYPE1′. Let us construct a subset $\tau \subseteq \sigma$ which is obtained by removing all those features from σ for which a strictly more specific feature is in σ, too. This construction guarantees that τ is a type of the reduced type set TYPE1′. We will show now that τ is an upper bound such as we require. For all features $f \in \tau$ holds $\{f\} \sqsubseteq \tau$ by the subtype relation definition. Let f' be any feature occuring in the difference $\sigma \backslash \tau$. f' has been removed from σ because there exists a certain feature $f'' \in \tau$ with f'' strictly $\leq f'$. Per (1) holds $\{f'\} \sqsubseteq \{f''\} \sqsubseteq \tau$. Thus all singleton sets $g(f1), g(f2), ..g(fn)$ have an upper bound in TYPE1′. All proof steps can also be performed in the reverse direction. □

Figure 6 shows a subhierarchy of the ALE semilattice for the example network. Cross classification and disjunctive inheritance are represented in TYPE1′ by the simultaneous occurence of not mutually dependent features in a type. Cross classification makes output features of different systems which share a feature in their entry condition cooccur (e.g. {*material, indicative*}, {*material, imperative*}). Output features of disjunctive systems cooccur with the disjuncts in their entry condition. (e.g. {*material, middle*}, {*material, effective*}).

[8] This reduction of the type set is taken from [Carpenter, 1992].

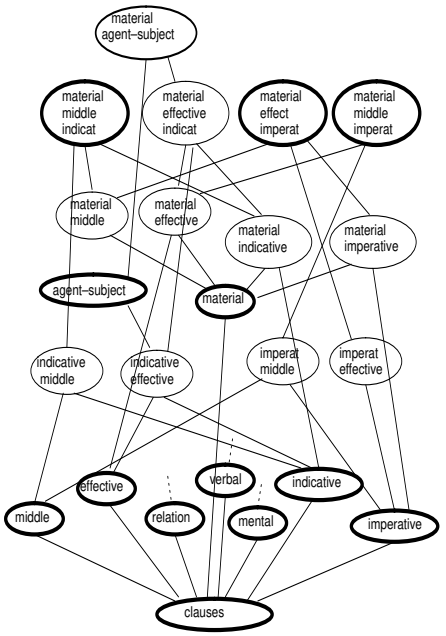

Figure 6: ALE type hierarchy

TFS type hierarchy

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. It must not necessarily be a unique one. 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:

$$\text{TYPE2} = \text{FEAT} \cup \{\sigma |\ \sigma \subseteq \text{FEAT} \text{ and } \sigma \text{ is a selection expression}\}$$

Subtype relation and join operation are similarily defined as for TYPE1. A TFS type hierarchy for the example network is given by the bold types in Figure 6.

CUF type hierarchy

The type discipline of the Comprehensive Unification Formalism ([Dorna and Dörre, 1993]) 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 same interpretation of compatibility as is used in system networks. This kind of open-world semantics together with an incorporated theorem prover makes

it possible to represent a system network in CUF directly as type hierarchy. The theorem prover provides means for dealing with disjunctive inheritance as well as with partitions and cross classification. Hence, using CUF, a system network can be straightforwardly encoded without precompiling compatible feature sets. Every feature is a type, every system is a type axiom:

TYPE3 = FEAT .

Comparison

We now compare the three input type hierarchies in terms of degree of structure and number of types. The ALE input specification [TYPE1, $\sqsubseteq, \sqcup$] has the very simple structure of a type hierarchy according to Carpenter (allowing only the subtype relation and multiple inheritance), but, for any real grammar, an intuitively ungraspable number of types at 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.[9] The source for the explosion of the type number is the lack of appropriate means for expressing cross classification.[10] How this problem arises similarily in HPSG is discussed in [Erbach, 1994]).

On the other hand, the CUF specification (and similar the system network) brings 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 and cross classification. Besides the small number of types (and therefore easier precompilation), the CUF specification offers the main advantage of preserving transparancy for the user. The one-to-one correspondence between systems and type axioms enables the linguist familiar with the PENMAN-style resources to make the transition to the feature unification account.

Nevertheless the ALE type hierarchy is a valuable construction because every implementation needs, if only internally, the complexity of this type hierarchy. Whether one needs the precompiled powerset construction as input specification depends, however, on the chosen formalism; when the formalism itself provides for disjunction, partition and cross classification., this is, unnecessary.

Other current formalisms relevant to the discussion are STUF, TDL and PROFIT. STUF ([Dörre and Seiffert, 1991]) is released in a new version which is nearly identical with CUF. TDL ([Krieger and Schäfer, 1993]) has the same position on the scale as CUF. The PROFIT system – particularly developed for the treatment of cross classification – has not been included in the comparison until now because it is relatively new. However, it is a candidate for detailed evaluation. Presumably it occupies a position between TFS and CUF

[9] 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.

[10] Another solution for dealing with cross classification in ALE and TFS - the use of features – has not been investigated. Carpenter's construction itself does not make use of this idea.

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

Figure 7: Realization statements as feature structures

in that it provides means for conjunctive and multidimensional, but not for disjunctive inheritance.

3.2 Realization statements and feature structures

The representation of realization statements as feature structures (except for the ORDER statements) can straightforwardly be taken from [Bateman *et al.*, 1992]. 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 7 shows the encoding.

ALE, TFS and CUF provide the same means for expressing feature structures. The only peculiarity is that CUF's type system does not support the expression of reentrancies (variables in feature structures). Hence, it is not possible to model the systemic conflations with the help of CUF types. But CUF provides a second data type – sorts – which can do this. The CUF specification therefore requires a double representation for every feature. A feature is represented once as type and second as sort. 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 they are necessary to express reentrancies. A CUF encoding of the mood distinction system (*indicative/imperative*) in the example grammar is shown in Figure 8.

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 (see Figure 7). The construction of the ordered list of grammatical constituents by inheritance involves two problems. A multiple occurence of one grammatical function must be excluded. And, fully specified phrase structures should not contain noninstantiated parts in their order list. Both can be achieved with the help of a dynamic construction of a daughters list (list of inserted functions). For that, however, set unification is necessary. Unfortunately, set unification is not supported in all mentioned formalisms. This problem shows a property of systemic grammar which distinguishes it from more structural oriented grammar models such as HPSG, where the number of daughters is known from the head's SUBCAT list.

```
indicative | imperative < clauses.

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

indicative_sort := indicative & clauses_sort &
                           finite: X1 & subject: X2 &
                           theme: X2 & process: X1.

imperative_sort := imperative & clauses_sort &
                           theme: X1 & nonfinitive: X1.
```

Figure 8: CUF encoding

4 Implementation

Implementations of an automatic procedure transforming arbitrarily complex systemic grammars written in PENMAN-style into an equivalent type signature with associated feature structures have been carried out both for TFS and for CUF. The implementations do not include the treatment of word order. Inferences concerning the bare system network – subtype and compatibility checking – are in both representations possible 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 FEAT. In Figures 9 and 10 TFS and CUF responses with respect to the example grammar are shown. The inferential behavior differs concerning the completeness of the result.

Processing with TFS provides a behavior which is called 'total sort resolving' (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 9 occurs as the result of both queries above it. The application of the TFS precompilation, for large scale systemic grammars, however, is limited because the type number explodes combinatorically with the number of cross classifications. The English systemic grammar NIGEL for instance has about 1200 features. We have so far only succeeded in representing in TFS internally coherent subnetworks of NIGEL, which typically include a number of cross classifications, 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. But nevertheless it does provide a precise semantics for systemic grammars.

CUF offers the most appropriate means for the representation of systemic grammars.

```
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]
```

Figure 9: TFS example

We succeeded in providing a CUF implementation of the whole NIGEL grammar. For that it was necessary to install a CUF system with maximally 6000 PROLOG variables, instead of 1000 which are allowed in the standard release. The specification includes the realization statements (excluding order statements). The price for the possibility of the large scale implementation seems to be that total sort resolving has not been achieved. In CUF, queries consisting of partial information are completed without specialization into most specific types (in the generation direction) and without respect to conflations (in the analysis direction). In Figure 10 a generation and an analysis inference is given. The partial character of the CUF reasoning is no problem as long as the generation direction is considered. On the contrary, the CUF answers are an appropriate means for handling those large type hierarchies. What shall a user do with more than one hundred answers to a question? The exclusion of the conflations in the analysis direction is a serious disadvantage which still calls for a better solution.

Figure 11 gives a CUF output example for a query to the whole NIGEL grammar. The question is to find the partial syntactic structures associated with the grammatical features *mental* (describing mental processes) and *passive-process*. The first two answers of altogether 12 answers are shown in the figure.

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

%%% used time (msec): 640
%%% result:

  +-- (indicative & material)
  |actor: (1) +-- (individual_name ; nominal_groups)
  |           |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

%%% other results? (y/n) n

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

%%% used time (msec): 550
%%% result:

  +-- (effective & indicative & ~mental & ~verbal)
  |goal: +-- (lexical_thing & oblique & ~each)
  |      |deictic: +-- every_form
  |      |         |spelling: ["every"]
  |      |         +--
  |      |thing: +-- common_noun
  |      |       |spelling: ["cookie"]
  |      |       +--
  |      +--
  |process: +-- disposal_verb
  |         |spelling: ["devour"]
  |         +--
  |subject: +-- (nominal_groups & ~class_name)
  |         |thing: +-- proper_noun
  |         |       |spelling: ["kim"]
  |         |       +--
  |         +--
  +--
```

Figure 10: CUF example

```
| ?- cuf(mental & passive_process).

%%% used time (msec): 9070
%%% result:

  +-- (clause_simplex_typ & mental_typ & nonagentive_typ & passive_process_typ &
 transitivity_unit_typ)
  |lexverb: (1) +-- words_typ
  |              |effective: +
  |              |experienceverb: +
  |              +--
  |predicator: start_typ
  |process: (1)
  |voice: +-- words_typ
  |       |beaux: +
  |       +--
  |voicedependent: +-- words_typ
  |                |edparticiple: +
  |                +--
  +--

%%% other results? (y/n) y

%%% used time (msec): 7610
%%% result:

  +-- (clause_simplex_typ & cognitive_ranged_typ & cognitive_typ & mental_typ &
middle_typ & nonmediated_typ & passive_process_typ & transitivity_unit_typ)
  |lexverb: (1) +-- words_typ
  |              |cognition: +
  |              |experienceverb: +
  |              |middle: +
  |              +--
  |predicator: start_typ
  |process: (1)
  |range: (2) top
  |subject: (2)
  |voice: +-- words_typ
  |       |beaux: +
  |       +--
  |voicedependent: +-- words_typ
  |                |edparticiple: +
  |                +--
  +--

%%% other results? (y/n) |: n
```

Figure 11: Partial syntactic structures for passive mental clauses computed by CUF

5 Conclusion

The implementation of systemic grammar in CUF is an essential contribution to the current research discussion concerning bidirectionality. It not only opens up the way for a state-of-the-art architecture for systemic grammars, but also enables a more direct comparison with other grammar models, the exchange and the combination with linguistic ressources of other research streams, and the incorporation into up-to-date linguistic workbenches.

Besides grammar sharing between generation and analysis components, declarative processing offers other advantages. This is illustrated by the fact that CUF has detected inconsistent systems (systems with contradictionary entry conditions) in NIGEL which the PENMAN software did not determine. So, CUF also provides a good environment for testing the internal consistency of large systemic grammars.

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.

On the other hand, Bateman *et al.*'s approach includes TFS representations for the semantic knowledge base and the interface between grammar and knowledge base (the chooser and inquiry semantics) which are however not proved for large scale grammars. The treatment of the knowledge base and the interface in CUF will be the next steps in our research, as well as a deeper investigation of the role of the order statements and their eventual necessary precompilation so long as set unification is not available.

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.

References

[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. Also available as technical report of GMD/Institut für Integrierte Publikations- und Informationssysteme, Darmstadt, Germany.

[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.

[Dorna and Dörre, 1993] Michael Dorna and Jochen Dörre. *CUF – Version 2.28d.* Institut für maschinelle Sprachverarbeitung (IMS), Universität Stuttgart, 1993.

[Dörre and Seiffert, 1991] Jochen Dörre and Roland Seiffert. Sorted feature terms and relational dependencies. Technical Report IWBS report 153, IWBS, IBM Deutschland, Stuttgart, 1991.

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

[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, 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. Also available as USC/Information Sciences Institute, Reprint Report ISI/RS-87-179, 1987.

[Krieger and Schäfer, 1993] Hans-Ulrich Krieger and Ulrich Schäfer. TDL - A type description language for unification-based grammars. Technical report, DFKI, Saarbrücken, 1993.

[Mann and Matthiessen, 1983] William C. Mann and Christian M.I.M. Matthiessen. Nigel: A systemic grammar for text generation. Technical Report ISI/RR-85-105, Information Sciences Institute, February 1983. 4676 Admiralty Way, Marina del Rey, California 90292-6695.

[Mann, 1983] William C. Mann. An overview of the PENMAN text generation system. Technical Report ISI/RR-83-114, USC/Information Sciences Institute, Marina del Rey, CA, 1983.

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

[O'Donnell, 1993] Michael O'Donnell. Reducing complexity in a systemic parser. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 203 – 217. Tilburg, the Netherlands, August 10-13 1993.

[O'Donoghue, 1991] T. F. O'Donoghue. A semantic interpreter for systemic grammars. In *Proceedings of the ACL workshop on Reversible Grammars*, Berkeley, California, 1991. Association for Computational Linguistics.

[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.