Experiments in Reusability of Grammatical Resources

Doug Arnold[°], Toni Badia[⊙], Josef van Genabith[◇], Stella Markantonatou[°], Stefan Momma[◇], Louisa Sadler[°], Paul Schmidt[□]

[°]Dept of Language and Linguistics, University of Essex, Colchester CO4 3SQ, UK
[⊙]Universitat Pompeu Fabra, La Ramba 32, 08002 Barcelona, Spain
[◇]IMS-CL, Azenbergstraße 12, University of Stuttgart, D-W7000 Stuttgart, Germany
[□]IAI, Martin-Luther-Straße 14, D-W6600 Saarbrücken 3, Germany
doug;marks;louisa@essex.ac.uk, tbadia@upf.es,
steff;josef@ims.uni-stuttgart.de, paul@iai.uni-sb.de

Abstract

Substantial formal grammatical and lexical resources exist in various NLP systems and in the form of textbook specifications. In the present paper we report on experimental results obtained in manual, semi-automatic and automatic migration of entire computational or textbook descriptions (as opposed to a more informal reuse of ideas or the design of a single "polytheoretic" representation) from a variety of formalisms into the ALEP formalism.[1] The choice of ALEP (a comparatively lean, typed feature structure formalism based on rewrite rules) was motivated by the assumption that the study would be most interesting if the target formalism is relatively mainstream without overt ideological commitments to particular grammatical theories. As regards the source formalisms we have attempted migrations of descriptions in HPSG (which uses fully-typed feature structures and has a strong 'non-derivational' flavour), ETS (an un-typed stratificational formalism which essentially uses rewrite rules for feature structures and has run-time non-monotonic devices) and LFG (which is an un-typed constraint and CF-PSG based formalism with extensions such as existential, negative and global well-formedness constraints).

[1]The work reported in this paper was supported by the CEC as part of the project ET10/52.

1 Introduction

Reusability of grammatical resources is an important idea. Practically, it has obvious economic benefits in allowing grammars to be developed cheaply; for theoreticians it is important in allowing new formalisms to be tested out, quickly and in depth, by providing large-scale grammars. It is timely since substantial computational grammatical resources exist in various NLP systems, and large scale descriptions must be quickly produced if applications are to succeed. Meanwhile, in the CL community, there is a perceptible paradigm shift towards typed feature structure and constraint based systems and, if successful, migration allows such systems to be equipped with large bodies of descriptions drawn from existing resources.

In principle, there are two approaches to achieving the reuse of grammatical and lexical resources. The first involves storing or developing resources in some theory neutral representation language, and is probably impossible in the current state of knowledge. In this paper, we focus on reusability through *migration*—the transfer of linguistic resources (grammatical and lexical descriptions) from one computational formalism into another (a target computational formalism). Migration can be completely manual (as when a linguist attempts to encode the analyses of a particular linguistic theory in some computationally interpreted formalism), semi-automatic or automatic. The starting resource can be a paper description or an implemented, runnable grammar.

The literature on migration is thin, and practical experience is episodic at best. Shieber's work (e.g. [Shieber 1988]) is relevant, but this was concerned with relations between formalisms, rather than on migrating grammars *per se*. He studied the extent to which the formalisms of FUG, LFG and GPSG could be reduced to PATR-II. Although these stud-

ies explored the expressivity of the different grammar formalisms (both in the strong mathematical *and* in the functional sense, i.e. not only which class of string sets can be described, but also what can be stated directly or naturally, as opposed to just being encoded somehow or other), the reduction was not intended to be the basis of migration of descriptions written in the formalisms. In this respect the work described below differs substantially from Shieber's work: our goal has to be to provide grammars in the target formalisms that can be directly used for further work by linguists, e.g. extending the coverage or restructuring the description to express new insights, etc.

The idea of migration raises some general questions.

- What counts as *successful* migration? (e.g. what properties must the output/target description have and which of these properties are crucial for the reuse of the target description?).
- How conceptually close must source and target be for migration to be successful?
- How far is it possible to migrate descriptions expressed in a richer formalism (e.g. one that uses many expressive devices) into a poorer formalism? For example, which higher level expressive devices can be directly expressed in a 'lean' formalism, which ones might be compiled down into a lean formalism, and which ones are truly problematic? Are there any general hints that might be given for any particular class of higher level expressive devices? When should effort be put into finding encodings for richer devices, and when should the effort go into simply *extending* the target formalism?
- How important is it that the source formalism have a well-defined semantics? How far can difficulties in this area be off-set if the grammars/descriptions are well-documented?
- How does the existence of non-monotonic devices within a source formalism effect migratability, and is it possible to identify, for a given source grammar, uses of these mechanisms that are not truly non-monotonic in nature and could thus still be modelled inside a monotonic description?
- To what extent are macros and preprocessors a useful tool in a step-wise migration from source to target?

We can provide some answers in advance of experimentation. In particular, *successful* migration implies that the target description must be practically usable—that is, understandable and extensible. There is one exception to this, which is where a large grammatical resource is migrated solely to test the (run-time) capabilities of a target formalism. Practically, usability implies at least I/O equivalence with the source grammar but should ideally also imply the preservation of general properties such as modularity, compactness and user-friendliness of the specification.

This paper reports on and derives some lessons from a series of on-going experiments in which we have attempted automatic, semi-automatic and manual migration of implemented grammatical and lexical resources and of textbook specifications, written in various 'styles', to the ALEP formalism (see below). The choice of ALEP was motivated by the assumption the study would be most interesting if the target formalism is relatively mainstream.[2] As regards the 'style' and expressivity of source formalisms, we have carried out migrations from HPSG, which uses fully-typed feature structures and a variety of richly expressive devices, from ETS grammars and lexicons[3] (ETS is an untyped stratificational formalism essentially using rewrite rules for feature structures), and from an LFG grammar[4] (LFG is a standard untyped AVS formalism with some extensions, with a CFG backbone).

2 The Migration Experiments

2.1 The Target Formalism

The target formalism, ALEP, is a first prototype implementation of the formalism specified in the ET-6 design study (the ET-6 formalism [Alshawi *et al.* 1991]). ET-6 was intended to be an efficient, mainstream CL formalism without ideological commitments to particular grammatical theories and suitable for large-scale implementations. It is declarative, monotonic and reversible, although in ET-6 and in ALEP it is possible to model certain non-monotonic operations (e.g. getting some treatment of defaults out of parametrised macros). ALEP is CF-PSG rule based and supports feature structures which are typed and simple inheritance between types. Type information and inheritance is effective only at compile time. ALEP provides atoms, lists, booleans and terms as basic types. Complex structured types and simple inheritance relations are defined by the user in a type system specification. In addition to standard grammar rules which are effective during a parse (generation) the formalism provides refinement rules which operate on the output of the parser and specify values which are still undefined after parsing by using only unification. Although the core formalism is rather conservative, for reasons of efficiency, it is intended to support the eventual inclusion of a periphery including external constraint processing

[2] Of course, for practical purposes one might want to migrate resources to a non-standard formalism, provided it is relatively easy to understand.

[3] Developed at Saarbrücken, Essex and UMIST during the EUROTRA project.

[4] Developed at Stuttgart as part of the EUROTRA accompanying research, see [Meier 1992].

modules. Similarly, it does not (yet) directly provide potentially computationally expensive expressive devices such as e.g. set-valued features and operations on sets, functionally dependent values, separation of ID and LP statements, multiple inheritance or membership and concatenation constraints on lists. The idea is that such extensions should be provided, probably as external modules, as and when they are found to be necessary.[5]

2.2 Manual Migration from HPSG

Although both HPSG and ALEP use typed feature structures and support type inheritance, they differ crucially in that HPSG specifications are consciously non-derivational and strongly modularised in terms of sets of principles, immediate dominance schemata and linear precedence statements operating as constraints on typed feature structures. To achieve this, HPSG employs a number of powerful descriptive devices, including list and set operations (often expressed as functionally dependent values), and multiple type inheritance. The focus for the HPSG $\longrightarrow$ ALEP conversion, then, is to what extent can the latter, rather lean formalism support in a reasonable way the style of linguistic specification found in HPSG (the source specifications for this experiment was the description of English provided in [Pollard & Sag 1992]).

Various approaches to conversion are possible. For example, it would be possible to define a user language permitting the expression of principles (in much the same way as some formalisms permit feature percolation principles to be separately stated) and a compiler into ALEP allowing their effects to be expanded into the rules. In this spirit, following the work of Mellish [Mellish 1988] the technique of encoding boolean combinations of atomic feature values so that satisfaction can be checked by unification is adopted in the ET-6 formalism [Alshawi *et al.* 1991].

Since there were open questions as what could be directly expressed in ALEP, in this conversion experiment we first took a direct approach, essentially employing ALEP as a feature term rewriting system for HPSG specifications. The focus of this conversion was mainly on exploring the limits of the expressivity of ALEP and thus identifying which higher level expressive devices could not be treated.

The resulting translation is not as perspicuous, modular, compact and maintainable as the original HPSG specification. Migration results in a fragmentation and particularisation of the linguistic information encoded in the original specification. This is because (i) HPSG principles and schemata have to be compiled out into (possibly large) sets of ALEP phrase-structure rules; and (ii) some descriptions cast in a richly expressive formalism have to be simulated and can often only be approximated in ALEP.

For example, ID-2 and the valence principle as it applies to ID-2, (1) has to be approximated with sets of ALEP rules of the form in (2), because of the lack of the functional constraint `val_append`.

(1) ID-2 and Valence Principle (simplified):

```
[SYNSEM|LOC|CAT|COMPS @1
 DRTS|HDTR|SYNSEM|LOC|CAT|COMPS val_append(@1,@2)
 COMPDTRS @2]
```

(2) ALEP rules for ID2:

```
id_2_0 = sign:{.... comps => [] ....} ->
       [sign:{.... comps => [] ....}] head 1.

id_2_1 = sign:{.... comps => [] ....} ->
       [sign:{.... comps => [X] ....},
        sign:{.... synsem => X ....}] head 1.

id_2_2 = sign:{.... comps => [] ....} ->
       [sign:{.... comps => [X,Y] ....},
        sign:{.... synsem => X ....},
        sign:{.... synsem => Y ....}] head 1.

id_2_3 = ..............
```

Of course, by adopting binary branching trees and altering the ID and Subcategorisation principles it would be possible to avoid some of this verbosity, but for the purposes of our experiment we considered it important to investigate the migration of the source formalism as is.

Note that the resulting ALEP specification in (2) is as compact, perspicuous and maintainable as in any rule based grammar formalism, although it compares badly with HPSG in these terms. While initially it seemed that it was possible to create a usable, extensible and understandable ALEP grammar on the basis of HPSG specifications, there is one feature of HPSG which remains problematic, that of set-valued features and set operations. The difficulty comes in modelling principles such as the HPSG Quantifier Inheritance Principle (QIP), which relies on the operations such as set union and complementation.

In ALEP set union can be approximated to a certain extent in terms of list concatenation in a difference list based threading approach. However, since the current implementation of ALEP does not even provide membership constraints on list representations, element and set difference constraints can only be approximated in terms of a multitude of minimally differing rules naming elements in set representations. This approach is only safe if the following two conditions hold:

- the sets involved are finite
- elements in the difference list representations of sets are unique

Even for small sets, however, any exhaustive implementation of set difference in terms of naming el-

[5] Apart from investigating issues involved in migration of descriptions, one motivation for these experiments is to explore just which devices are essential for expressing linguistically motivated grammatical descriptions.

$$\left[\begin{array}{l}\text{SYNSEM:}\left[\text{LOC:}\left[\text{CONTENT:}\left[\text{QUANTS:}\mathit{RETR} \cup \mathit{HQUANTS}\right]\right]\right] \\ \text{QSTORE:}\mathit{(HQSTORE} \cup \mathit{QUANTS}_1 \cup \ldots \cup \mathit{QUANTS}_n) - \mathit{RETR} \\ \text{RETRVD:}\mathit{RETR} \\ \text{DTRS:}\quad \left[\begin{array}{ll}\text{HDTR:}\left[\text{SYNSEM:}\left[\text{LOC: }\left[\text{CONTENT:}\left[\text{QUANTS:}\mathit{HQUANTS}\right]\right]\right]\right] \\ \text{DTR}_1 \ \left[\text{QSTORE:}\mathit{QUANTS}_1\right] \\ \ldots \quad [\ldots\ldots] \\ \text{DTR}_n \ \left[\text{QSTORE:}\mathit{QUANTS}_n\right]\end{array}\right]\end{array}\right]$$

Figure 1: Quantifier Inheritance Principle (simplified)

ements in the representation results in an unacceptable number of rules and associated parse time. In some cases we were able to avoid this problem by relegating e.g. quantifier retrieval to sets of refinement rules which operate on parse objects which are effectively underspecified for quantifier scope.

It soon became clear that sets of refinement rules are not a general solution for the modelling of element or set complement constraints in HPSG because they operate on the *output* of a parse and hence cannot decide about the 'phrase' structure of a sign. Introducing and filling gaps, however, is central to the structure of a sign. The Nonlocal Feature Principle (NFP) which is at the heart of the HPSG treatment of unbounded dependency constructions (UDCs) ensures that SYNSEM|NONLOC|INHER values are discharged in terms of a set difference specification which cannot be implemented in terms of sets of refinement rules since it directly decides about the well-formedness of strings in terms of the phrase structure of the sign.

$$\left[\begin{array}{l}\text{SYNSEM:}\left[\text{NONLOC:}\left[\text{INHER:}(S_1 \cup \ldots S_n) - S\right]\right] \\ \text{DTRS:}\left[\begin{array}{ll}\text{HDTR:}\left[\text{SYNSEM:}\left[\text{NONLOC:}\left[\text{BIND:}S\right]\right]\right] \\ \text{DTR}_1\text{:}\left[\text{SYNSEM:}\left[\text{NONLOC:}\left[\text{INHER:}S_1\right]\right]\right] \\ \ldots \quad [\ldots\ldots] \\ \text{DTR}_n\text{:}\left[\text{SYNSEM:}\left[\text{NONLOC:}\left[\text{INHER:}S_n\right]\right]\right]\end{array}\right]\end{array}\right]$$

Figure 2: Nonlocal Feature Principle (simplified)

Furthermore, parasitic gap phenomena in English as in *That was the rebel leader who rivals of _ shot _* suggest that at least as far as the NFP is concerned it is problematic to asssume that elements in the difference list representations of sets are unique. This assumption is crucial to modeling set union in terms of list concatenation.

Formally, HPSG principles can either be given the status of proper types or that of typed feature structure templates acting as constraints on other feature structures. In ALEP the first option is not available to us since apart from subtype or supertype information the type system specification does not allow the specification of a type other than in terms of its root attributes and the type of their corresponding values and more importantly it does not support multiple inheritance required to inherit principles to other types. In order to recapture some of the loss of modularity in compiling out HPSG principles over sets of ALEP rules we thus tried to pursue the second option using m4 macros to directly state principles. m4 is a standard UNIX facility which allows for parameterised and non-parameterised macros, conditional expansions and numeric operations. Macros are expanded externally to ALEP and not during compilation time. Each HPSG principle can be represented as a feature structure template which in turn can be specified in terms of a macro definition, or so it seems. The problem here, however, is that since HPSG principles mutually constrain signs, the conjunction of such principles (at least in simple cases) corresponds to the unification (or merging) of their feature structure template representations (if the conjunction is satisfiable). What standard macro facilities achieve is effectively a simple lexical expansion of strings and it is impossible to get the merging effect of unification of template feature structures out of a modular macro specification of such templates. Basically, three options are available to us:

(i) To get the overlapping effect of unification we integrate different principles into one macro.

(ii) We define extended types with special attributes for each of the relevant HPSG principles which are expanded by modular macro definitions of the principles and get the unification effect from ALEP at compile time through proper coindexation.

```
phrase{phrase => @S{PHRASE},
       hfp    => @S{HEAD_FEATURE_PRINC},
       sp     => @S{SEMANTICS_PRINC},
       qip    => @S{QUANTIF_INHERIT_PRINC},
       valp   => @S{VALENCY_PRINC}}
```

(iii) We use a more powerful 'macro' processor like e.g. Prolog which provides the unification effect and define a map into ALEP.

In the case of (i) the modularity of HPSG with separately stated, but interacting principles is lost. (ii) has the disadvantage that the ALEP specifications grow in size while in the case of (iii) we are not con-

sidering the expressivity of the target formalism itself.

2.3 Automatic Migration from ETS B-rules

In this section we draw some general conclusions following from our experience of attempting *automatic* migration from an *untyped rule-based* formalism. Specifically, the source for this experiment was the structure-building rules of some relatively large ETS grammars. The ETS formalism is "badly behaved" in that it contains a rich array of devices additional to the structure-building or B-rules, many of which are non-monotonic, and which apply at run-time (they are mainly output filters and various types of feature percolation rules). We have written an automatic compiler in Prolog which calculates a very simple type system and automatically migrates the structure rules and lexical descriptions. With respect to the source formalism in question, the following points are to be noted:

- The run-time non-monotonic devices found in ETS are extremely problematic to take into account in automatic direct migration. We doubt whether it would be possible to write an intelligent compiler which directly encoded the effect of these devices in the resultant ALEP rule set. If they are ignored in the migration process, then of course the source and target descriptions are not I/O equivalent.
- The B-rules themselves allow optionality, Kleene closure, positive Kleene closure and disjunction over (sequences of) daughters to any degree of embedding within each other. In ALEP such rules have to be compiled out into a normal form which allows only for optionality over single daughters and no disjunctions of daughters. The size of the resulting rule set is such that it cannot be reasonably maintained. The size also means that it is impossible for a linguist to manually "correct" an overgenerating grammar resulting from the omission of filters and feature rules above.
- In some cases, it became apparent during the migration process that the intended semantics of the (very complex) phrase structure rules was unclear (e.g. regarding the scope of variables in Kleene starred constituents).

One conclusion is that one of the crucial ingredients is the quality and detail of the *documentation* of grammars. With good documentation it is often possible to get around the effects of unclear rule semantics, because the rule writers intention can be understood. The lack of such documentation is serious, since it means the migrator has to try to intuit the intended behaviour by attempting to run the source grammars in the source formalism.

Similarly, so long as the intended interpretation is clear, it may be possible to deal with non-monotonic devices. This is most obvious where the non-monotonic effects do not persist to run-time (but see also our discussion of the LFG migration below). For example the ALVEY grammar [Carroll 1991] has them, but since there is an object grammar stage in which all this is compiled out, the non-montonic devices can be avoided by taking the object grammar as the input to migration. The issue is then whether it is possible to automatically 'recompact' the target grammar in some linguistically useful way, or whether all extension and maintenance should be done in the source formalism.

Note further that even if the grammars resulting from a migration are not linguistically useful (for example, because the grammar is not maintainable or extensible), they may serve some purpose in testing the capacity of the target formalism to operate (efficiently) with very large rule sets (for example, in our experimentation, a rule set of some 1,500 rules derived by automatic migration caused ALEP to fail to compute the link relation).

ETS lexical descriptions are more successfully migratable because their semantics is clear. Simple parameterised macros have been used in a semi-automatic migration process.

2.4 Automatic LFG importation into ALEP

LFG is an untyped constraint-based linguistic formalism with rich expressive devices built around a CFG backbone. The formalism has been implemented in various systems, including XEROX PARC's *Grammar Writer's Workbench*, and the CHARON system developed as part of the accompanying research for EUROTRA-D carried out at the University of Stuttgart. Our automatic migration experiment started from grammars written for the latter system. We have written a Prolog program that translates automatically from an LFG notation that is very close to the original specification in [Bresnan 1982] into ALEP. For reasons explained further below, the program cannot succeed in all cases. It is, however, capable of detecting those cases reliably, and generates warnings where the fully automatic translation fails. [6] Examples for typical rules from the source grammar are shown in figure 3.[7]

The translation of the rule format illustrated in figure 3 into a PROLOG readable form is performed by a subcomponent of the CHARON system. The automatic translation procedure makes use of the output of this precompilation step.

The rule format supports optionality of constituents, nested optionalities and Kleene starred rule parts, which have to be expanded in the ALEP translation. ALEP only supports optionality of single daughters in the RHS of rules. In our case, this part of the expansion is done by the preprocessor. The

[6] The program was developed by Dieter Kohl at IMS.

[7] The caret sign and the lowercase v are ASCII representations of the metavariables ↑ and ↓, respectively.

```
VP'' -> VP'      {/ (^ VCOMP) = v
                  /  ^ = v /}
        [V       ^ = v].


C1   -> C        ^ = v
        VP2       {/ ^ = v
                     {/(^ VTYPE) = v2
                      /(^ VTYPE) = v1 /}
                    /(^ FCOMP) = v
                     {/ (^ VTYPE) = vfin
                      / (^ VTYPE) = inf /}
                   /}.
```

Figure 3: Sample grammar rules from the source description

result of compiling out Kleene starred rules and optionalities is that the object grammar quickly reaches a size that can no longer be reasonably maintained and the target description contains elements (in this case auxiliary categories) which are not part of the linguistic intuition of the grammar writer.

The second characteristic feature of rules like the ones shown in figure 3 is the massive use of complex disjunctions over feature structures (indicated by the {\ and \} pairs). Although the ALEP formalism supports disjunctions over complex feature structures, due to problems in the implementation available at the time of the experiment, they had to be multiplied out into a possibly large number of separate rules.

The next example (figure 4) shows a typical lexical entry from the source grammar.

```
bietet: V, (^ OBJ AGR CAS) = acc
           (^ PRED) = "bieten <(^ SUBJ)(^ OBJ)>"
           (^ SUBJ AGR NUM) = sg
           (^ SUBJ AGR CAS = nom
           (^ TENSE) = present
           (^ INF) = -
           (^ FORM) =c an                    <---
           (^ VERBTYPE) = particle.
```

Figure 4: Sample lexicon entry from the source description

The basic part of the annotations of the LFG rules and lexicon, i.e. the defining equations, are mapped easily into ALEP. The work here is divided between the CHARON preprocessor which converts feature descriptions (the equations) into feature terms, and the output routine which maps feature terms into ALEP rules and lexicon entries.

In LFG, path specifications in equations can be variables, as in the `(^(v PCASE))` case, where the attribute under which the f-structure associated with `v` is determined by the value of a feature *inside* `v`. ALEP does not support variable path expressions, therefore we have to enumerate all possible paths in a large disjunction which adds another factor to the multiplicative expansion of the rule set. Similar facts hold for the implementation of functional uncertainty, where we have to deal with regular expressions over paths.[8]

LFG permits "special" types of equation besides the standard defining ones. Constraining ($=_c$ type) equations in our source grammar typically occur in lexical entries as the one shown in figure 4, where a given form of e.g. a verb has to be distinguished, because it is only used in particular contexts. The equation is then typically a specification of a special subclass of a more general class of verbs (here a verb which can occur with a separable prefix). Where this is the case, in the migrated description the relevant distinction can be made in the type system, ensuring that non-membership in the particular subtype is explicitly stated for all (relevant) members of the supertype.

Another, potentially very powerful expressive device in the LFG formalism is the use of existential and negative existential constraints (in the CHARON notation expressed as `!(^ INF)` and `~(^ INF)`, respectively). Current implementations of LFG delay the evaluation of such constraints, because in general, they can only be tested at the end of the processing of a whole utterance. It turns out, however, that quite often existential and negative existential constraints can be disposed of, if a full type system is available. Careful examination of the source grammars reveals that the prevalent use of such constraints is exactly to model what feature appropriateness conditions in a type system do: they restrict the application of particular rule types to feature structures where a given set of features is either present or absent. To model this by using the type system instead, we introduce subtypes of the structure where the path has to or must not exist.

If the source grammar only uses negative existential constraints for atomic valued features, we could easily formulate a proper type system, and do away with '~', and '!' in a rather straightforward manner. Typical uses of e.g. negative existential constraints are shown in the rule and lexical entry in figure 5.

LFG uses set values for collecting e.g. adjuncts which do not have any other distinguishing function on the f-structure level. ALEP does not support the direct expression of sets as values. Given the facts of German word order, generation would seem to require sets of ADJUNCTS as values, rather than lists. Here we do in fact loose some expressivity if we try to model adjuncts in ALEP using lists, because the canonical set operations are not available.

Finally, we have to be able to express the (non-monotonic) global *completeness* and *coherence* con-

[8] In a recent experiment, the implementors of the CHARON system added support for functional uncertainty modelled via an interpretation of paths as sequences and general operations on these sequences.

```
C  -> V  ^ = v
        {/(^ VTYPE) = v2
         /(^ VTYPE) = v1 /}
         ~(^ INF).

kennen: V, (^ PRED) = "kennen<(^ SUBJ)(^ OBJ)>"
           (^ OBJ AGR CAS) = acc
           {/ (^ SUBJ AGR NUM) = pl
              (^ SUBJ AGR CAS) = nom
              (^ TENSE) = present
              ~ (^ INF)
            / (^ INF PERF) = -
              (^ UNACC) = -  /}.
```

Figure 5: Examples for negative existential constraints in the rules and the lexicon

straints which help to control subcategorisation. Of these two, the coherence condition can be easily controlled by defining types with the appropriate number of features, one for each of the subcategorised functions. The introduction of additional syntactic functions which are not subcategorised for is then prevented by the type system. The completeness condition, however, which is supposed to guarantee that all syntactic functions in the subcategorisation frame are filled, can not be handled that easily. The main problem here is, that while we are able to require that a certain feature be present in a feature structure, we cannot express restrictions on the degree of instantiation of the value of that feature.

There is, of course, another option: If we model subcategorisation more explicitly, introducing 'subcat lists' as data structures in much the same way as HPSG does, we can add the requirement that PS rules consume elements of the subcat list. Besides the question whether such a modelling is still compatible with the spirit of LFG theory as it stands, the proposed solution does not solve the problem for a German LFG grammar: in order to model the variability of German word order, we have to be able to pick arbitrary elements from the subcat list, rather than relying on a fixed order in which elements are picked. Since list operations (or functional constraints in general) are not available in ALEP, this can currently not be modelled perspiciously.

In summary, then, the philosophy of the grammar can be maintained, and a type system can be provided. To a certain extent, it can express LFG's non-monotonic devices such as existential, negative existential and constraining equations and the global wellformedness constraints of completeness and coherence. The target grammar is less compact, because generalisations are lost, through the multiplicatory effect of spelling out optionalities, Kleene stars and variables over attribute names.

2.5 Technical description of the automatic conversion procedure

The automatic conversion has to accomplish three basic tasks:

- A conversion of the grammar rules into ALEP format
- A conversion of lexical entries into the ALEP lexicon format
- The extraction of a certain amount of type information from the LFG grammar to be used in the ALEP descriptions.[9]

We will not go into details of the CHARON precompilation, since the techniques employed are standard (expansion of optionality and Kleene star constituents, as well as compilation of feature descriptions into feature terms). As regards the extraction of type information from the untyped LFG description, more explanation is needed, however.

In the current incarnation of the conversion routine, the following strategies are used:

- each attribute is assigned (at least) one type name
- atomic-valued features and PREDs are used during compilation to compute value ranges for their corresponding types
- features with complex values have their possible values (and the attributes therein) collected during compilation, and the compiler then determines the corresponding types at the end of the compilation.
- the output routines take care of the fact that types that represent atomic values or terms are spelt out correctly (i.e. that they do not show up as type definitions, but are inserted directly)
- if we encounter more than one type name for the value of a given attribute, further processing is necessary, because reentrancies are involved or we have an interaction with the c-structure skeleton which has to be handled separately.

In all those cases, where the compilation cannot produce satisfactory results, the intermediate structures are printed out instead, together with a comment saying which steps failed indicating where further hand-tuning is required.
In particular,

- sets are encoded as open ended lists, thus not solving the free order problem mentioned above
- the uniqueness condition is marked through the use of a term for the value of PRED
- for compilation steps which modify the original structure of the grammar (e.g. turning inequations in finite domains into disjunctions, mapping constraining equations onto defining ones, if the automatic inference of the proper subtypes

[9] We also have to provide the ALEP runtime system with information about headness in grammar rules, which is crucial for the proper operation of at least one of the parser modules provided with the system.

is not yet possible, etc.) a warning is issued in the resulting ALEP code in the form of a comment

- headness information is selected according to the following heuristics:
  - derivation to ϵ have no head information associated (naturally)
  - unary-branching nodes have a trivial head
  - for non-unary-branching rules
    * those categories that can rewrite to ϵ are eliminated from the list of head candidates (if *all* daughter nodes are eliminated this way, the first daughter is selected as the head, and a comment appears with the rule)
    * if pure preterminal nodes are among the remaining ones, the first one is selected as the head
    * otherwise, all left-recursive nodes are eliminated (with a similar strategy for taking the remaining leftmost node, if all nodes would be eliminated)
    * among the remaining nodes, again the leftmost node is selected as the head
    * if everything is left-recursive, the leftmost node is selected, and a comment is generated accordingly in the output.

Compiling out the rule given in figure 3 yields (among others) the ALEP structure in figure 6, the result of the compilation of the lexical entry from figure 5 is shown in figure 7 (again, only one of the disjuncts is shown).

```
vp2_vp_v =
        ld: { spec => ger_Specifier_t: { },
              syn => vp2_Syntax_t: { },
              fs => @V_FS vp_Cat_t:
                    { vcomp => Vp_1_FS},
              pho => phones: { string => Vp_Str,
                               rest => Rest } }
         ->
        [ld: { syn => vp_Syntax_t: { },
               fs => Vp_1_FS,
               pho => phones: { string => Vp_Str,
                                rest => V_Str } }
         ld: { syn => v_Syntax_t: { },
               fs => V_FS,
               pho => phones: { string => V_Str,
                                rest => Rest } }]
        head 2.
```

Figure 6: Compiled rule from figure 1

3 Conclusion

Our experiments have demonstrated that migrations of various sorts can be performed with a reasonable degree of success.

```
kennen ~
 ld: {spec => ger_Specifier_t: {},
      pho => phones:
             {string => [kennen | R],
              rest => R},
      syn => v_Syntax_t: { },
      subcat => [ld:
                 {syn => dp_Syntax_t: {},
                  fs => Subj},
                 ld:
                 { syn => dp_Syntax_t: {},
                   fs => Obj}],
      fs => cp1_Cat_t:
            { pred => pred_FS_t:
                      {semname => kennen,
                       semargs => subj_obj},
              subj => @Subj dp_Cat_t:
                      {pred => _},
              obj => @Obj dp_Cat_t:
                     {pred => _,
                      agr => agr_FS_t:
                             { cas => acc}},
              inf => inf_FS_t_kv: {perf => -},
              unacc => -}}.
```

Figure 7: Compiled lexical entry from figure 3

As regards the general questions about migration posed at the beginning, we can formulate some (partial) answers.

- Successful migration obviously involves more than just I/O equivalence of source and target descriptions. One also looks for similar degrees of 'descriptive adequacy' (i.e. compactness, perspicuity, maintainability etc.). Clearly reusability implies usability. However, this is not an absolute property, and a small loss of such properties can be acceptable. It is clear, however, that the loss of maintainability that we have experienced in some of the migration activities above is unacceptable.
- How conceptually close must source and target be for migration to be successful? We have seen that in principle it is possible to migrate resources across certain formal/ideological divides — for example, from HPSG, which has no rules, but uses types extensively, to ALEP, which has a weaker type system, and is CF-PSG rule based; and from LFG (which does not use typed feature structures) to ALEP. The migration of HPSG specifications into the rule based ALEP entails a considerable degree of fragmentation and particularisation of the linguistic information encoded in the original specification. To a certain extent this can be recaptured if the target formalism provides an integrated template facility which is not restricted to simple lexical expansion. We have also suggested that good documentation can alleviate the effects of distance

between formalisms.

- With respect to the migration of descriptions using richer expressive devices, it is clear that it is sometimes possible to dispense with the richer devices, and that some descriptions couched in richer formalims do not use them in any crucial way. The HPSG conversion experiment, however, has clearly shown that for set valued features, and operations on sets, a naive encoding is simply unacceptable.
- We have seen that the effect of non-monotonic devices in a source formalism can be serious, especially when it is combined with unclear rule semantics (c.f. the ETS conversion experiment). However, the existence of an 'object' formalism where the non-monotonic devices are compiled out (like in the case of the ALVEY grammars) is an asset, and again, good documentation helps. Particularly in the case of the LFG conversion experiment it became clear that often there is a crucial difference between the availability of certain non-monotonic devices and their actual use. E.g. it was found that existential constraints are often used to express subtype information. If the type system is rich enough, this information can be modelled in the type system specification in the target formalism.
- As expected, we have found macros and preprocessors a useful tool, especially in the semi-automatic migration of lexical resources. In order to approximate a principles based style of linguistic description like in HPSG the target formalism should be extended with an integrated template facility which determines satisfiability of templates (principles) in terms of unification.

References

[Alshawi *et al.* 1991] Hiyan Alshawi, Arnold D J, Backofen R, Carter D M, Lindop J, Netter K, Pulman S G, Tsujii J & Uszkoreit H, (1991), *Eurotra ET6/1: Rule Formalism and Virtual Machine Design Study (Final Report)*, CEC 1991.

[Bresnan 1982] Joan Bresnan (ed.), (1982). *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts, 1982

[Carroll 1991] J. Carroll, E. Briscoe & C. Grover (1991). *A Development Environment for Large Natural Language Grammars*, distributed with the Third Release.

[Meier 1992] Meier, J. (1992). "Eine Grammatik des Deutschen im Formalismus der Lexikalisch Funktionalen Grammatik unter Berücksichtigung funktionaler Kategorien". Report, Universität Stuttgart.

[Mellish 1988] Chris Mellish (1988) "Implementing Systemic Classification by Unification", *Computational Linguistics*, 14, pp 40–51.

[Pollard & Sag 1992] Carl Pollard & Ivan Sag, (1992). *Head Driven Phrase Structure Grammar*, Chicago University Press, forthcoming.

[Shieber 1988] Stuart M. Shieber (1988), "Separating Linguistic Analyses from Linguistic Theories", in U. Reyle and C. Rohrer *Natural Language Parsing and Linguistics Theories*, D. Reidel Publishing Co. Dordrecht, pp 33–68.