On the Portability of Complex Constraint-Based Grammars

C.J. Rupp* and Rod Johnson

Formerly of IDSIA, Corso Elvezia 36, CH900 Lugano

1 Introduction

Recent years have seen the appearance of a number of grammar formalisms[1] sharing a strong family resemblance, which we have characterised elsewhere [Rupp *et al.*, 1994] as the property of being *constraint-based*. As well as having in common many formal properties, these formalisms also support, often by explicit design, descriptions from a similarly convergent range of linguistic theories, which we might reasonably label "HPSG-like".

Given the considerable common ground between such formalisms, it is reasonable to begin to ask questions about their intertranslatability, or, in programming language terms, the relative ease with which it is possible to "port" a grammar from one such formalism to another. Such questions are clearly of interest for the enterprise of recovering as much as possible of the existing stock of already encoded linguistic knowledge, perhaps for reuse in a more modern theoretical framework. They are also of relevance for any attempts to build in portability from the start in ongoing new grammar writing.

At present, the criteria for determining whether a particular translation is successful are extremely fuzzy. Apart from anything else, they will presumably depend to some extent on external goals, such as, for example, whether the results will be used in a practical, running system or just in a laboratory experiment to show the feasibility of a particular theoretical approach. In our work, we have assumed that, if the translation is intended as more than a sterile exercise, then the information in the source description must be worth conserving and hence worth translating. Moreover, we suppose that the resulting target grammar will need to be maintained and extended, and hence should be well-understood and well-behaved. Given these assumptions, we can begin to impose some conditions on what constitutes a "good" translation; in effect, in a translation from grammar A to grammar B:

- B and A should have the same input-output behaviour.
- B should conserve as much as possible of the conceptual shape of A.
- B should have comparable or better run-time performance with respect to A.

The first condition is a consequence, if somewhat oversimplified, of the assumptions we made above, that the main purpose of the exercise is to preserve useful information.

The second condition has to do with the relative expressivity of the two formalisms involved. In effect, how much of the conceptual and organisational structure of a linguistic description can pass over unchanged, and to what extent do conceptual changes that may have to be made obscure our subsequent understanding of the description as a whole?

The question of performance is not limited to the relative execution speed of source and target grammars, though its importance for subsequent maintenance and development cannot be overstated. How do we approach the case, for example, where the source grammar runs normally in its native environment but the translated form fails to terminate unless the description is completely restructured? And what if the two systems use conflicting criteria for the apportionment of procedural control between the linguist and the implementation?

Over the past year, we have been engaged on a number of experiments designed to investigate these portability issues, and in particular to bring out the implications behind the two related sets of questions about expressivity and performance. In some ways, our work is similar in spirit to the reusability experiments reported in [Arnold *et al.*, 1993], though these appear to have been limited to translation to a single, rather general formalism, and to have been concerned almost entirely with questions of relative expressivity.

The remainder of this paper discusses our own experiments and comments on some of our more important findings so far.

*Currently affiliated to the Institute for Computational Linguistics, University of Stuttgart, Azenbergstr.12, 70174 Stuttgart, Germany, cj@ims.uni-stuttgart.de

[1] In the interests of brevity, we shall often use the term *grammar* to refer to the collection of formal devices which comprise all aspects of a linguistic description, encompassing both grammatical and lexical information. This is purely a notational convenience and in no way implies a commitment to the primacy of syntax.

	Type Hierarchy	Explicit Parser	Dedicated Morphology	Contol Determined	Lazy Evaluation	Host Language
UD	no	yes	yes	globally	yes	Common Lisp
TFS	yes	no	no	globally	yes	Common Lisp
CUF	yes	no	no	locally	yes	Prolog
ALE	yes	yes	yes	locally	no	Prolog

Table 1: A checklist of the significant properties of the sample implementations

2 Formalisms

In our experiments to explore the portability of complex constraint-based grammars we have considered a sample of four implemented formalisms:

- UD (Unification Device) [Johnson and Rosner, 1989, Rupp *et al.*, 1992][2].
- TFS (Typed Feature Structures) [Emele and Zajac, 1990].
- CUF (Comprehensive Unification Formalism) [Dörre and Eisele, 1991, Dörre and Dorna, 1993]
- ALE (Attribute Logic Engine) [Carpenter, 1992]

The original reason for selecting this sample was practical: the availability of these systems in the public domain at the appropriate time[3]; but on further reflection this sample turns out to be quite representative of the major differences which may occur in formalisms of this type (cf the very coarse-grained classification in Table 1)[4]. The consequences of these distinctions are explored in more detail below.

The nature of experiments in portability requires not only the selection of source and target formalisms, but also of example descriptions to be translated. In this respect we opted for taking grammars "from the wild", i.e. native code from one of the sample formalisms that was not designed with any prior consideration of its potential portability. To be more precise, we have worked with a small, but formally representative HPSG grammar, originally provided as sample data with the TFS system, and a somewhat larger and quite intricate UD grammar of French, which touches on such thorny issues as clitic placement and object agreement. The initial experiments were in translating the TFS grammar into UD, and then subsequently into the other two formalisms. Our attempts to translate the UD French grammar into ALE were not quite as successful, as a substantive alteration to the structure of the syntactic analysis proved necessary. The situation with CUF is more promising, even though the definition of an explicit parsing strategy within the formalism was required. These two issues are discussed further in Section 4.

[2]For the purposes of this paper we see no significant differences between UD and its derivative ELU, see e.g. [Estival, 1990].

[3]We did toy with the idea of entitling this paper: "Off the CUF remarks on how much ALE UD need to make sense of a TFS grammar", but thought better of it.

[4]See also [Rupp, 1992, Johnson and Rupp, 1993]

3 Expressivity

The underlying assumption that is crucial to the nature of this work is that these formalisms have highly comparable expressivity, i.e. they share more than separates them. This is central to the success of the enterprise since preservation of concepts defined by the linguist is an essential part of grammar translation. Consequently, we are particularly concerned here with the main constructs of a linguistic description: types, relations and lists. We also consider, though to a lesser extent, purely notational devices like macros, which can be useful in organising the conceptual structure of a description. Of lesser importance in the present context is the treatment of logical structure, in particular disjunction; in any case, this topic has received a good deal of attention elsewhere (cf [Trost, 1993]).

3.1 Types

The role of feature structure types in constraint-based linguistics has gained increasing importance as a result of the increasing popularity, some might say dominance, of HPSG [Pollard and Sag, 1987, Pollard and Sag, forthcoming]. In HPSG the type system, or *type signature*, plays a significant role in defining the class of legal linguistic objects. In fact in the current version of the theory only objects whose typing information is fully resolved are considered to be adequate models of naturally occurring linguistic constructs. Each of the formalisms we consider permits the definition of feature structure types, but the form and expressivity of these type definitions differ quite considerably, as does the significance of type definitions in the description as a whole. The extreme cases are TFS, in which the type system is virtually all there is, and UD, where type definitions simply constrain the attributes which can occur on a feature structure.

At this point we should note that a type system in the "true" or HPSG sense, requires a notion of type inheritance which can be further subdivided into three concepts:

- subtype/supertype relations
- feature appropriateness conditions
- closure conditions

Type definitions which form a type system usually encode immediate subtypes and feature appropriateness conditions, which specify, at least, the attributes which

```
head = subst | funct.

subst = noun | verb | adj | prep.

subst[PRD:boolean].

noun[CASE:case].

verb[VFORM:vform,
     AUX:  boolean,
     INV:  boolean].
```

Figure 1: A fragmentary type system rooted in head and written in TFS

are licensed by the type and the types of their values, as in Figure 1. Closure is usually a derived notion, in that only attributes licensed by the type or one of its supertypes may occur, an unlicensed attribute incurring either further subtyping or inconsistency. UD type definitions cannot of themselves be used to define a hierarchical type system. They give an entirely flat system with the most absolute closure and the most minimal appropriateness conditions. The type definitions of the other formalisms, TFS, CUF and ALE, differ mainly in the expressivity of their appropriateness conditions, in order of decreasing expressivity, cf [Manandhar, 1993] for a more detailed comparison of these type systems.

Evidently, one of the most basic hurdles to translating any of the other formalisms into UD is the reconstruction of the type system. This was the problem posed in our initial experiment of porting an HPSG grammar encoded in TFS into UD. Our solution to this problem, cf Figure 2, consists of separating out the hierarchies of sub- and supertype dependencies from those of feature appropriateness, so that each node in the type hierarchy is represented by two unary abstraction definitions in the UD encoding. UD types[5] are only utilised on the terminal nodes of the type hierarchy to ensure ultimate closure. In principle the use of any pseudo-type definition will work its way down the dependency hierarchy to the terminal node and then back up the appropriateness hierarchy to gain more information. While this sounds dreadfully inefficient the lazy evaluation strategy adopted in UD in fact avoids most of the computational overhead.

3.2 Relations

The other main constructs used for expressing linguistic concepts are relations — or more specifically definite relations since most of these formalisms are in fact instantiations of the Höhfeld and Smolka notion of a Constraint Logic Programming language [Höhfeld and Smolka, 1988]. While the same essential notion occurs in all these formalisms the terminology is quite

[5]Type assignments in UD have the form: `Variable == type.`

```
head(X): !subst(X)

head(X): !funct(X)

subst(X): !noun(X)

subst(X): !verb(X)

subst(X): !adj(X)

subst(X): !prep(X)

Subst(X): <X prd> = yes/no

noun(X) X == noun
        !Subst(X)
        !case(<X case>)

verb(X) X == verb
        !Subst(X)
        <X aux> = yes/no
        <X inv> = yes/no
        !vform(<X vform>)
```

Figure 2: The head system rewritten in UD

diverse, including, for instance, relational abstractions (UD) and parametric sorts (CUF). In fact in TFS relational constructs actually take the form of types with features expressing their argument structure, although a relational notation is provided to sweeten the syntax slightly. Since definite relations occur in each of the formalisms, their translation does not pose any immediate problems, and many of their usages are the same, e.g. accounting for relational dependencies and principles in HPSG-style grammars, cf Figure 3. Difficulties do however occur where the usage of relational constructs is restricted. ALE imposes the restriction that true definite relations may only be used in the phrasal domain, attached to phrase structure rules. On first impression, this could pose a serious problem for translations from other formalisms where relations may be used freely in the lexicon. Our experience has shown that many such lexical relations can in fact be encoded using ALE macros, as in Figure 4, which may be parameterised, but require a deterministic expansion. Where operations involving recursive or disjunctive relations are required there is still the option of encoding the construct as a lexical rule, though with the risk of losing some of the conceptual structure.

```
hfp(synsem: loc: cat: head: Head) :=
          synsem: loc: cat: head: Head.
```

Figure 3: A CUF encoding of a Head Feature Principle as a unary parametric sort

```
np(Case) macro
        @nominal(Case),
        @saturated,
        @lex(false).
```

Figure 4: An ALE macro definition

3.3 Lists

The last class of constructs that we consider in detail are lists, or sequences. Our objective here is slightly different than in the last two cases, since all the formalisms support lists and most even supply the same, Prolog-style, notation. There is however a more subtle difference between UD and the more strongly typed formalisms, since in all the other formalisms the list notation is purely syntactic and masks a typed feature structure that is either atomic or has two attributes. In UD where lists are "real" objects, the unifier is more explicitly polymorphic, but also admits the provision of built-in functions over sequence data-types, whose computational behaviour is more predictable than that of defined constructs like relations. UD provides both append and member (or perhaps better "extract") over lists and -- since strings are also a full data type -- concatenation over strings. The effects on performance of hard-coding frequently used constructs can be quite dramatic. We do not pursue this question here since the associated design issues are comparable with those associated with the decision to incorporate dedicated modules which are discussed in the next section.

4 Performance

The second class of issues which affect the porting of a grammar from one formalism to another is connected with the relative performance of the two instantiations. We consider two aspects of this topic, the provision of explicit modules for processing in a particular domain, such as syntactic or morphological analysers, and the complex and thorny issue of control information, or who gets control of control. First, though, it is worth emphasising why we consider performance to be a significant issue at all. We are not -- yet, anyway -- particularly concerned with the real time performance of "end-user" applications. We view all of the systems that implement these formalisms as development environments, even if they were originally developed as "academic" prototypes, in several cases with a view to demonstrating a particular theoretical perspective. Accordingly, we feel that it is more appropriate to evaluate their performance with respect to the development loop associated with grammar writing. More concretely, if either the analysis or compilation times exceed certain acceptable bounds (determined by pragmatic, external considerations like the attention span of a grammar developer or lexicographer), then the grammar under development should be regarded as being, in a purely practical sense, no longer extensible. These may be rather harsh criteria, but we believe they reflect a more realistic sense of what these systems are good for[6].

4.1 Dedicated Modules

A further explicit distinction arises between those formalisms which include explicit modules for treating either phrasal or morphological structure (UD, ALE), and those which only provide a theorem prover over linguistic constraints (TFS, CUF). In general, we expect that, other things being equal, a formalism whose implementation contains dedicated processors for phrase structure parsing and/or string processing will have better run-time performance than one which does not, and this is indeed borne out empirically in the behaviour of the systems we considered.

The presence or absence of an explicit parser also has obvious consequences for porting experiments. If there is a parser in the target system and not in the source system then some phrase structure component must be supplied. This may just be a vacuous structure or it may be derived from existing components of the source description. Hence we have produced three instantiations of the UD translation of the TFS-HPSG grammar: one involving a vacuous phrase structure description, one in which grammar rules are derived from the phrase structure definitions of the TFS encoding and one in which full strings are associated with a lexicon of garbage tokens to avoid invoking either of UD's dedicated modules for morphology and syntax.

Portability in the other direction poses considerably greater problems, since not only must the phrase structure description be encoded, but some parsing strategy must also be defined. In translating the UD grammar into CUF we encoded a *head corner parser* (cf e.g. [van Noord, 1994]) directly in the CUF formalism. In order to obtain adequate results with this strategy it was necessary to make use of all the facilities offered for determining both global and local process control. This sheds a certain amount of doubt on the possibility of replicating the CUF results within TFS, where explicit local control statements are not permitted. We address the more general problems with the incorporation of control information in the next section.

While the question of translating more or less explicit phrase structure information is already a difficult one, the issue of porting morphological information is quite chaotic. There is even less agreement on the information structure of morphological regularities than there is on syntactic patterning, and this fact is reflected in the fact that two of the systems we have been working with do not offer any apparatus at all for dealing with sub-word-level phenomena. Moreover, the two formalisms in our sample which do admit explicit morphological descriptions differ so greatly in

[6] That is apart from acquiring publications or qualifications

the form that these components take that they are not directly comparable even with each other[7].

4.2 Control Information

The final issue that we turn to is one which is in effect most revealing about how system developers view their users. In terms of our sample formalisms, we once again can distinguish a two-way split, which actually cuts across all of the groupings that we have observed above. The crude characterisation of this distinction is that some formalisms permit the grammar writer to influence the local processing strategy, either in the good, old-fashioned Prolog manner of ordering clauses, as in ALE, or by providing additional control information, such as delay statements in CUF. The other two systems eschew this kind of local tweaking of the processing strategy and rely on a global specification of processing behaviour. Of course, this apparent dichotomy is to some extent illusory. Those systems which retain global control usually permit the user to modify certain parameters of this behaviour, and those that permit local control information must also assume a global control strategy which may be less forgiving than that in an apparently more totalitarian system. We have two observations in respect of the control strategies adopted by these systems.

The first of these is that some form of lazy evaluation, such as that assumed as a global strategy in both UD and TFS, can become a requirement of a target system when the source system permits lazy evaluation. More explicitly a description may rely on a particular evaluation strategy that cannot be emulated in the target system. This situation actually occurred in the porting of the UD French grammar to ALE. The lack of a lazy evaluation strategy in ALE required a change in the analysis of verbal structure[8], so the ALE description is actually different from the original UD one. In a very real sense the port failed, in that, even though in terms of the declarative formalism a compatible description was definable, it turned out that this was not runnable. The class of portable descriptions between ALE and any of the other formalisms is therefore further constrained by the ALE's underlying evaluation strategy.

The second point we would like to make harks back, in many ways, to the warnings inherent in Kaplan's "procedural seduction". Kaplan [Kaplan, 1987] reports experiences with the use of ATN parsers which ended with both grammar writers and system developers attempting to improve the performance of the same parser and effectively getting in each other's way. More generally, every time we think we may be making a smart move by some kind of local fix to the control strategy we also make it more difficult for a really smart optimising controller to do its job properly. Of course we have progressed considerably in the declarativity and monotonicity of our formalisms which we now tend to view as specialised logics, but where we have not learnt so much is in our view of the kind of people who are going to use the implemented system and what they are capable of. Where local control information is specified in the ordering of statements in definitions, we are effectively requiring that the grammar writer be an accomplished logic programmer. Where local control information is added to supplement an existing grammar description the implicit assumption is even more demanding: that there are individuals capable of appending local control information to descriptions that other people have written — or worse still translated — and of getting it right. Both of these approaches ultimately assume that it is not only possible but relatively easy to retain a detailed picture of the behaviour of a complex constraint solver.

When translating to a formalism which permits local control from one which does not, the issue may come down simply to a question of relative speed of computation, which is important enough in itself in practical situations, as we have already pointed out. In cases where the target formalism, like ALE, *requires* local control information in order to guarantee termination, much more is at stake.

5 Conclusion

We readily admit that the experiments reported here are still quite unscientific — or, we would prefer to think, prescientific — and we are still feeling our way towards a more rigorous approach to the question of comparability of implemented formalisms, even though the task is noticeably simplified by recent convergence of goals and methods in constraint-based computational linguistics.

Nonetheless, our experience already suggests, in keeping with [Arnold *et al.*, 1993], that from the point of view of relative expressivity it is possible to move grammars from one formalism to another, and even perhaps to conceive of new grammars which are designed from the start to be portable across a range of related formalisms.

As regards the set of issues which we have classed together under the heading of performance, on the other hand, there are still many open questions which need to be addressed before porting grammars to serious, extensible and maintainable applications can become a realistic enterprise.

Acknowledgements

The research reported in this paper was funded by the Swiss National Fund for Scientific Research, under project No. 12-32604.91 *Situations and Discourse*.

[7] In the case of ALE it would probably be incorrect to speak of a morphological analyser since lexical forms are expanded at compile time.

[8] At the corresponding point in the CUF translation lazy evaluation had to be explicitly enforced by the use of a `delay` statement

This work would not have been possible without the cooperation of the developers of the various systems. We would like to thank Martin Emele, Jochen Dörre, Michael Dorna, Bob Carpenter and Gerald Penn for making their systems available and for their patience in answering questions, even when these were either trivial or relatively bizarre. Any misrepresentation of their work that occurs here is entirely the fault of the authors. We would also like to thank Mike Calcagno for assisting us in some of this work and carrying out the translation of the TFS-HPSG grammar into ALE. No thanks are due to Angelo Dalle Molle and his foundation whose antics have made completion of the work reported here more difficult than it need have been.

References

[Arnold *et al.*, 1993] Doug Arnold, Toni Badia, Josef van Genabith, Stella Markantonatou, Stefan Momma, Louisa Sadler, and Paul Schmidt. Experiments in reusability of grammatical resources. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 12 -20, Utrecht, 1993.

[Carpenter, 1992] B. Carpenter. *The Attribute Logic Engine User's Guide.* Laboratory for Computational Linguistics, Philosophy Department, Carnegie Mellon University, Pittsburgh PA 15213, December 1992.

[Dörre and Dorna, 1993] J. Dörre and M. Dorna. CUF - a formalism for linguistic knowledge representation. In J. Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description I*, pages 1–22. ILLC/Department of Philosophy, University of Amsterdam, 1993. DYANA-2 Deliverable R1.2.A.

[Dörre and Eisele, 1991] J. Dörre and A. Eisele. A comprehensive unification-based grammar formalism. DYANA deliverable R3.1.B, Centre for Cognitive Science, University of Edinburgh, Scotland, January 1991.

[Emele and Zajac, 1990] M. Emele and R. Zajac. Typed unification grammars. In *Proceedings of the 13th International Conference on Computational Linguistics, COLING 90*, pages 293–298, Helsinki, 1990.

[Estival, 1990] D. Estival. Generating french with a reversible unification grammar. In *Proceedings of the 13th International Conference on Computational Linguistics, COLING 90*, volume 2, pages 106–111, 1990.

[Höhfeld and Smolka, 1988] M. Höhfeld and G. Smolka. Definite relations over constraint languages. LILOG-Report 53, IBM Deutschland GmbH, Stuttgart, 1988.

[Johnson and Rosner, 1989] R. Johnson and M. Rosner. A rich environment for experimentation with unification grammars. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, pages 182–189, Manchester, 1989.

[Johnson and Rupp, 1993] R. Johnson and C. J. Rupp. Evaluating complex constraints in linguistic formalisms. In H. Trost, editor, *Feature Formalisms and Linguistic Ambiguity.* Ellis Horwoood, Chichester, 1993.

[Kaplan, 1987] R. M. Kaplan. Three seductions of computational psycholinguistics. In P. Whitelock, M. M. Wood, H. L. Somers, R. Johnson, and P. Bennett, editors, *Linguistic Theory and Computer Applications*, pages 149–188. Academic Press, London, 1987.

[Manandhar, 1993] Suresh Manandhar. CUF in context. In J. Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description I*, pages 43–53. ILLC/Department of Philosophy, University of Amsterdam, 1993. DYANA-2 Deliverable R1.2.A.

[Pollard and Sag, 1987] C. Pollard and I. A. Sag. *Information-Based Syntax and Semantics: Volume 1 Fundamentals.* Number 13 in CSLI Lecture Notes. CSLI, Stanford University, 1987.

[Pollard and Sag, forthcoming] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar.* CSLI and University of Chicago Press, Stanford and Chicago, forthcoming.

[Rupp *et al.*, 1992] C. J. Rupp, R. Johnson, and M. Rosner. Situation schemata and linguistic representation. In M. Rosner and R. Johnson, editors, *Computational Linguistics and Formal Semantics*, pages 191–221. Cambridge University Press, Cambridge, 1992.

[Rupp *et al.*, 1994] C. J. Rupp, R. Johnson, and M. Rosner. Overview. In C. J. Rupp, M. Rosner, and R. Johnson, editors, *Constraints, Language, and Computation*, pages xi–xxiii. Academic Press, London, 1994.

[Rupp, 1992] C. J. Rupp. Abstraction mechanisms in constraint-based linguistic formalisms. Working Paper 6, IDSIA, 1992.

[Trost, 1993] H. Trost. *Feature Formalisms and Linguistic Ambiguity.* Ellis Horwoood, Chichester, 1993.

[van Noord, 1994] G. van Noord. Head corner parsing. In C. J. Rupp, M. Rosner, and R. Johnson, editors, *Constraints, Language, and Computation*, pages 315–338. Academic Press, London, 1994.