The Feature Space in Parallel Grammar Writing

Tracy Holloway King, Martin Forst, Jonas Kuhn, and Miriam Butt
Palo Alto Research Center, IMS Stuttgart, University of Texas at Austin, and Universität Konstanz

Abstract. This paper discusses the methodology and tools applied in the Parallel Grammar project (ParGram) to support consistency and parallelism of linguistic representations across multilingual Lexical Functional Grammar (LFG) grammars. A particular issue is that the grammars in the ParGram project are developed at different international sites. The approach that was established over several years relies on (i) a technical tool for checking adherence to the best-practice feature declaration for linguistic representations, (ii) the coordinated, systematic use of templates for expressing generalizations across lexicon entries and grammar rules, and (iii) a grammar code reviewing committee in which extensions to the existing representations are critically discussed.

Keywords: LFG, parallel grammars, feature space, templates, grammar engineering

1. Introduction

At the very minimum, multilingual grammar development requires agreement on a common set of representations and broad agreement on the analyses of linguistic phenomena. Some advantages of this are the following: it reduces the amount of work required for adding an additional component to the various grammars, such as semantic analysis; it makes it easier for grammar engineers to maintain several grammars at the same time and faster to add a grammar for a new language to the family of grammars. On the application side, a common set of representations facilitates porting a language-technological system that applies one of the grammars to the other languages from within the multilingual development effort; and it is indispensible for certain multilingual applications like machine translation.

Discussions about the proper analysis and representation of phenomena such as subject-verb agreement, case marking, relative clauses, or the treatment of adjectives or adverbials can be conducted at a very high linguistic level. Ultimately, though, differences in opinion are reified at a very low level of engineering. Since its inception in 1995, the Parallel Grammar (ParGram) project has therefore included a "feature committee," whose job is to find norms for the use and definition of a common multilingual feature space. Adherence to feature committee decisions is technically supported by a routine that checks the grammars for compatibility with a feature declaration. Parallelism

at the level of grammatical constraints or descriptions is facilitated by systematic use of means of abstraction in the grammar specification code.

The ParGram project is an international collaboration aimed at producing broad-coverage computational grammars for a variety of languages ((Butt et al., 1999; Butt et al., 2002); see (Riezler et al., 2002) on the coverage of the English grammar). The grammars (to date of English, French, German, Japanese,[1] Norwegian, and Urdu) are written in the framework of Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1982; Dalrymple, 2001), and they are constructed using a common engineering and high-speed processing platform for LFG grammars: XLE (Maxwell and Kaplan, 1993).

In keeping with standard LFG practice, these grammars assign two levels of syntactic representation to the sentences of a language: a surface phrase structure tree (called a *constituent structure* or c-structure) and an underlying matrix of features and values (the *functional structure* or f-structure). The c-structure records the order of words in a sentence and their hierarchical grouping into phrases. The f-structure encodes the grammatical functions, syntactic features, and predicate-argument (dependency) relations conveyed by the sentence. F-structures are meant to encode a language-independent level of syntactic analysis, allowing for crosslinguistic parallelism at this level of abstraction. For example, while the analysis of the English, French, and German versions of a sentence like (1) will necessarily differ at the c-structural level (different word orders, different numbers of auxiliaries), at the f-structure level all the grammars produce something like the dependency structure in Figure 1, in which the main predicate is the verb with one subject argument. The verb is modified by an adjunct at the top level of the f-structure, which also includes information about the clause type (declarative). The tense and aspect information is collected under the feature TNS-ASP (Butt et al., 1996). All this information will be common to the f-structures produced for English, French and German. However, the language-particular dependencies between tense inflection, auxiliaries, and verbs are not encoded at f-structure.

(1) a. Tomorrow the letter will have arrived.

b. Demain la lettre sera arrivée. (French)
tomorrow the letter will-be arrived

[1] A Korean grammar is currently being ported from the Japanese grammar to determine how quickly a deep grammar can be developed when bootstrapped from the grammar of a typologically similar language (Kim et al., 2003). Grammars for Arabic, Chinese, Malagasy, Turkish, and Welsh are just being started.

c. Morgen wird der Brief angekommen sein. (German)
tomorrow will the letter arrived be

```
"Morgen wird der Brief angekommen sein."

[PRED    'an#kommen<[41:Brief]>'                                          ]
[ADJUNCT { 0[PRED     'morgen' ] }                                        ]
[        { [ADV-TYPE temp     ] }                                        ]
[        [PRED  'Brief'                                                  ]
[        [NSEM  [COMMON count]                                           ]
[        [NTYPE [GRAIN common]                                           ]
[SUBJ    [SPEC  [DET [PRED     'die' ]                                   ]
[        [           [DET-TYPE def   ]                                   ]
[     41 [CASE nom, GEND masc, INFL strong-det, NUM sg, PERS 3           ]
[TNS-ASP [ASPECT [FUT +_, PERF +_]                                       ]
[        [MOOD indicative, TENSE present_]                               ]
[TOPIC   [0:morgen]                                                      ]
[  21 [CLAUSE-TYPE declarative, STMT-TYPE declarative, VTYPE main        ]
```

Figure 1. F-structure for (1c)

It is important to keep in mind that LFG f-structures are *syntactic* levels of representation, i.e., they are not intended as a level of semantic or knowledge representation. As such, different languages are expected to at times have different f-structure analyses for sentence with similar meanings (see (Butt et al., 2002) for some examples).[2] The idea behind parallel grammar writing based on f-structure analysis is to use similar analyses for all languages in which that analysis can be linguistically justified.

In this paper, we discuss some of the technical and organizational means that have been developed in the ParGram project to establish and enforce cross-grammar standards. Section 2 addresses the definition and comparison of allowable features and their proper values. Section 3 discusses the systematic use of templates in the grammar description language as a way of making generalizations explicit and transparent across grammars. Section 4 provides some discussion and a conclusion.

[2] That is, f-structures are meant to encode underlying structural similarity, not crosslinguistic synonymy, since synonymy is a concept rooted in semantics. Of course, the same underlying syntactic structure (f-structure in LFG) should generally correspond to synonymous sentences crosslinguistically; however, it cannot exclusively determine the set of all sentences that will be synonymous. In other words, we would not expect all synonymous sentences to have exactly the same syntax.

2. Defining the Feature Space

In this section, we discuss the feature space in the LFG ParGram grammars and how it is defined and regulated.

2.1. The Status of Feature Appropriateness Conditions

One way of ensuring parallelism at f-structure is to define a crosslinguistically relevant feature space in advance. The idea of theory-driven type/sort definitions for the feature structure representations as used in Head-driven Phrase Structure Grammar (HPSG; (Pollard and Sag, 1994)), for example, would appear to be ideal for such a purpose. The space of valid representations is restricted by the use of typed values for each feature, where the type of a feature-structure object defines exactly what embedded features are *appropriate*. Very elaborate type hierarchies have been proposed (see (Copestake, 2002) for implementation and documentation of typed feature structure grammars).

The grounding of the type hierarchy in the theoretical framework would seem to provide ideal support for a multilingual grammar writing effort (Bender et al., 2002). However, for use in broad-coverage grammars, the central role played by feature appropriateness conditions in HPSG theory can also pose problems. As the discussions in the HPSG community show, even quite fundamental configurations in the feature geometry have been subject to dispute and revision over the years. An example for such a discussion is the question whether or not the ARG-ST feature should be embedded under HEAD so it is percolated from a word to its phrasal projections; compare e.g., (Przepiórkowski, 2001). The consideration of additional languages or phenomena may lead to insights that justify, or even require, potentially fundamental revisions to the feature space. This can pose considerable problems for a continuing grammar development effort. Moreover, it is not entirely clear whether there is (or should be) a crosslinguistically uniform type hierarchy, although this issue is being tested and explored in the Matrix project (Bender et al., 2002; Flickinger and Bender, 2003).

The methodology for ensuring parallelism and consistency in the ParGram project relies on checking feature appropriateness in a similar way to HPSG (section 2.3); however, the conditions are established in a less theory-driven manner. From its inception, LFG linguistic theory has chosen not to enforce feature appropriateness conditions *per se*. Instead, these conditions are viewed as a means of expressing generalisations, and as an engineering-level construct they are used for the purpose of consistency-checking. This encourages an explorative comparison of different possible representations, something which is particularly

useful for phenomena not discussed in the theoretical literature. The resulting feature geometry is typically "leaner" than an HPSG feature geometry, which goes along with its lesser theoretical status and makes it less prone to major revision.

Related to this, a crucial technical difference between feature declarations in the ParGram project and the effect of feature appropriateness conditions in HPSG lies in their status with respect to the specification of the actual grammatical constraints: the LFG feature declarations act only as filters, which check whether the feature usage in the grammar conforms to conventions. Features that would be appropriate for a particular feature structure, but which are not explicitly mentioned in any of the constraints relevant for a given analysis will not show up in the f-structure representation. This is because of the LFG assumption that the f-structure for a sentence is defined as the minimal model which satisfies the grammatical constraints. In HPSG, on the other hand, the feature appropriateness conditions associated with types in an inheritence hierarchy interact with the HPSG assumption that the described objects are totally well-typed, sort resolved feature structures. This means that if, according to the type hierarchy, all subtypes of some type t for which a feature F is defined also bear feature G, feature G will be introduced "automatically" if some grammatical constraint mentions feature F on a structure of type t. This can, of course, be exploited to state linguistic generalizations in a very compact way, but in the context of explorative grammar writing it can occasionally lead to unintended effects and may cause the grammar writer to spend time defining workarounds in the type hierarchy. This is avoided in the approach described here. In the remainder of this section, we describe two tools used to ensure parallelism in the feature space: feature declarations and the feature table.

2.2. Feature Declarations

While c-structure analyses are subject to language particular variation by definition, the idea behind the f-structures is that they reflect a more language-independent analysis. In order to maintain parallel grammar development, it is therefore vital to have identical features playing identical roles in all of the grammars.

For some features, the discussions and decisions to be made in a multilingual context are relatively straightforward. For example, the fact that there should be a CASE feature universally is generally undisputed, as is the idea that most languages code tense/mood/aspect and definiteness in some way. However, the precise values of such features continue to be disputed in linguistic theory and are thus also subject to

debate within ParGram. For example, much discussion went into the definition of the tense/aspect features: deciding which attributes and values to use meant engaging in a deep discussion of the underlying theoretical treatment of tense/aspect.

Case provides another type of example. While there is general agreement that languages encode core values of NOM(inative), DAT(ive), and ACC(usative) in some form or another, one could question whether a language like English does indeed need to encode case (e.g., (Hudson, 1995)). At present, the English ParGram grammar encodes case, but this is mainly to ensure representational parallelism with the other languages in the project.

At an even more basic level, issues arise as to whether the names of features should be spelled the English way or, for example, the German way (e.g., ACC(usative) vs. AKK(usativ)). While these latter sorts of questions seem relatively low-level, the core of language individual analyses does depend on the precise declaration and space of the features. When viewed from this perspective, questions at this level do not seem so unimportant any more.

2.2.1. *The Syntax of Feature Declarations*

In feature declarations, two basic types of feature values are distinguished: atomic (or constant) features values and complex values, which are sub-f-structures.[3] By convention, atomic feature values are always lower-cased, whereas feature names are always upper-cased. This has the effect of rendering the representations and the grammar code much more transparent than is the case with the "classic" LFG convention of using upper-case for all features and values. For atomic features, the feature declaration specifies that the value of a feature FEAT (denoted by FEAT:→; the right-arrow introduces the definition of the *possible values* of the feature to its left) has to be a member of a set of atomic values:

(2) FEAT:→ ∈ { $\texttt{value}_1$ $\texttt{value}_2$ ... $\texttt{value}_n$}.

[3] A further type of feature values are closed-set values, as argued for in (Dalrymple and Kaplan, 2000). An indeterminate pronoun may, for example, have the closed set { `nom, acc` } as the value of its CASE feature. (Under a closed-set value analysis, constraints on case may check for membership of `nom` (or `acc`) in the set, so the indeterminate pronoun is indeed compatible with both cases.) The feature declaration for a closed-set-typed case feature involves using the set subsumption operator ≪:
CASE: → ≪ { `nom gen dat acc` }.
Set valued features are used to encode PSEM in English (section 2.2.2).

For complex features, the declaration specifies the features that the embedded feature structure may contain, using the subsumption operator $\ll$:

(3) FEATA:$\rightarrow \ll$ [FEATB$_1$ ... FEATB$_n$].

Consider the TNS-ASP feature and the features it calls in the feature declaration in (4). TNS-ASP itself takes an embedded feature structure as its value, with four possible features. (Note that the complex feature structure may consist of a subset of these features.) The feature MOOD has an atomic value, likewise the remaining three features.

(4) TNS-ASP: $\rightarrow \ll$ [MOOD PERF PROG TENSE].
MOOD: $\rightarrow \in$ { `imperative indicative subjunctive` }.
PERF:$\rightarrow \in$ { +_ −_ }.
PROG: $\rightarrow \in$ { +_ −_ }.
TENSE: $\rightarrow \in$ { `fut null past pres` }.

Even from a cursory examination of the features in (4), it is easy to tell that tense/mood/aspect information is grouped together under the TNS-ASP feature (instead of having the individual features directly in the verb's f-structure). In addition, we can see that the grammars do not make use of composite tense/aspect values such as past perfect, pluperfect, or future perfect. Early on in the ParGram project, this type of composite feature value for tense/aspect was attempted for the German, English, and French grammars. However, not only did the cost of determining the right value turn out to be too great (since these tenses are often encoded as periphrastic constructions, a complex system of interdependencies needs to be checked), but it was difficult to establish a coherent crosslinguistic feature space. It was therefore decided to fall back on a more atomic encoding of tense/aspect, in which atomic features are simply registered under the TNS-ASP feature. That is, the English auxiliary *will* contributes the feature FUT, the English auxiliary *have* contributes the PERF +_,[4] etc. The more intricate problem of a precise and crosslinguistically valid semantic analysis of tense/aspect is left to a separate semantic component, which can base its analysis on the information collected under the TNS-ASP feature.

[4] The use of the underscore in the value '+_' indicates its status as an *instantiated symbol*. This means that it is resource-sensitive in that it can be introduced only once through a *defining* feature equation. In other places, only *constraining* equations can be used to check for the feature value. The most well-known instantiated symbols are the semantic forms used as values of the PRED feature. Here, instantiation excludes a duplication of identical material in various clause positions. See also (Dalrymple, 2001, 107).

When the grammar is loaded, the feature declaration is checked against the compiled grammar rules. If any undefined features are found or if any undefined values of declared features are found, XLE returns an error message indicating the offending feature and where it appeared. The grammar cannot be loaded until the feature declaration violation is fixed.[5] For example, if the feature declaration allows for ADV-DEGREE `positive` and `comparative` but the grammar also has ADV-DEGREE `superlative`, as might occur when the grammar writer is adding superlative adverbs to the grammar, then XLE provides a message as in (5).

(5) `Feature declaration violation near line 43, column 23`
`in file /pg/eng/standard/english-templates.lfg:`
ADV-DEGREE `cannot be equal to superlative`

Thus, XLE forces compliance to the feature declaration by not loading a grammar using features that conflict with or are missing from those in the declaration.

2.2.2. *Multiple Feature Declarations*
XLE allows the grammar to call multiple feature declarations. These declarations are given a priority order and can modify entries posited in previous feature declarations. The ability to include multiple feature declarations can be used in a multilingual context to create a common feature declaration which can then be specialized for particular languages. A "universal" or "common" feature declaration can be created for all languages. This common feature declaration is then invoked by all of the grammars. If a specific language needs additional features or feature values which are not in the common feature table, then these can be specified in the language particular feature table. This language-specific table is then given higher priority than the common one, as in (6a), and can then be further specified for special versions of that language's grammar, as in (6b).

(6) a. FEATURES: COMMON STANDARD-FRENCH.

b. FEATURES: COMMON STANDARD-ENGLISH EUREKA-ENGLISH.

[5] In order to check for feature declaration violations in the lexicon, it is necessary to load the generator for the grammar since only in generation is the entire lexicon indexed. As with feature violations in the grammar, XLE will indicate the feature in question and where it was found; once the violations are fixed, the generator can be loaded.

With this ordering, the STANDARD-FRENCH feature declaration takes priority over the COMMON one. Special edit operators are used to manipulate the features (see (Kaplan and Newman, 1997) for how edit operators can be used with multiple lexicons in a similar fashion).

There are three edit operators (+ & !). In addition, a feature can appear without an operator. We demonstrate these operators with an expository example first and then discuss examples from the ParGram grammars.

Consider a COMMON feature declaration as in (7).

(7) COMMON FEATURES
CASE: → ∈ { `nom acc dat gen` }.
NUM: → ∈ { `sg pl` }.
GEND: → ∈ { `masc fem neut` }.

If we want to add a new feature that is not mentioned in COMMON, then it appears without an operator. This is shown in (8) for PERS and results in the effective feature declaration in (9).

(8) LANGUAGE1 FEATURES
PERS: → ∈ { `1 2 3` }.

(9) EFFECTIVE-LANGUAGE1 FEATURES
CASE: → ∈ { `nom acc dat gen` }.
NUM: → ∈ { `sg pl` }.
GEND: → ∈ { `masc fem neut` }.
PERS: → ∈ { `1 2 3` }.

If we want to add a new *value* to an existing feature, then the + operator is used. This is shown in (10) to add the value `erg`(ative) to the feature CASE (as needs to be done for the Urdu grammar). Note that it is possible to add multiple values to a feature in this way. (10) results in the effective feature declaration in (11).

(10) LANGUAGE1 FEATURES
+CASE: → ∈ { `erg` }.

(11) EFFECTIVE-LANGUAGE1 FEATURES
CASE: → ∈ { `nom acc dat gen erg` }.
NUM: → ∈ { `sg pl` }.
GEND: → ∈ { `masc fem neut` }.

If we want to restrict the set of values of a feature with respect to the common set of values, the & operator is used. The & operator denotes

intersection and the list of values appearing in the language-specific feature declaration shows the ones that are to be *kept* from the original list. So, if we want to remove the value `neut` from the feature GEND, then we would list only the values `masc fem`. This is shown in (12) and results in the effective feature declaration in (13).

(12) LANGUAGE1 FEATURES
&NUM: → ∈ { `masc fem` }.

(13) EFFECTIVE-LANGUAGE1 FEATURES
CASE: → ∈ { `nom acc dat gen` }.
NUM: → ∈ { `sg pl` }.
GEND: → ∈ { `masc fem` }.

Finally, it is possible to completely replace a feature with a new set of values or even a new type of values. As discussed below, other than in a deletion context, this is strongly discouraged in a parallel grammar setting and is used only as a development tool. If we wanted to replace the feature GEND with one that took complex instead of atomic features, we could do this as in (14) (we also add the new MASC and FEM features for completeness; since these are new features, they do not need an edit operator). This results in the effective feature declaration in (15).

(14) LANGUAGE1 FEATURES
!GEND: → ≪ [MASC FEM].
MASC: → ∈ { + − }.
FEM: → ∈ { + − }.

(15) EFFECTIVE-LANGUAGE1 FEATURES
CASE: → ∈ { `nom acc dat gen` }.
NUM: → ∈ { `sg pl` }.
GEND: → ≪ [MASC FEM].
MASC: → ∈ { + − }.
FEM: → ∈ { + − }.

One interesting side effect of the ! operator is that it can be used to effectively remove a feature from the COMMON set by setting the value of the new feature to the empty set. For example, to remove NUM altogether (some languages indeed have no number specification), the LANGUAGE1 feature declaration would be as in (16), with the effective feature declaration in (17).

(16) LANGUAGE1 FEATURES
!NUM: { }.

(17) EFFECTIVE-LANGUAGE1 FEATURES
CASE: → ∈ { nom acc dat gen }.
GEND: → ∈ { masc fem neut }.

Thus, if we look at the COMMON feature declaration repeated in (18a) and the LANGUAGE1 feature declaration in (18b), the effective feature declaration for LANGUAGE1 is as in (18c).

(18) a. COMMON FEATURES
CASE: → ∈ { nom acc dat gen }.
NUM: → ∈ { sg pl }.
GEND: → ∈ { masc fem neut }.

b. LANGUAGE1 FEATURES
PERS: → ∈ { 1 2 3 }.
+CASE: → ∈ { erg }.
&GEND: → ∈ { masc fem }.
!NUM: → { }.

c. EFFECTIVE-LANGUAGE1 FEATURES
CASE: → ∈ { nom acc dat gen erg }.
GEND: → ∈ { masc fem }.
PERS: → ∈ { 1 2 3 }.

The above example was expository in nature, but used a set of features and changes that could be possible in a parallel grammar setting. We now discuss some actual uses of the edit operators in the ParGram project grammars.

Consider the English grammar. English has an impoverished set of case features. However, the set of possible values for CASE in the common feature declaration is large, as seen in (19a), since these values are needed for languages like Urdu and are typologically quite common. As such, the & operator is used to restrict this set, as in (19b). A similar situation is found for mood, as seen in (20).

(19) a. COMMON FEATURES:
CASE: → ∈ { acc dat erg gen inst loc nom obl }.

b. STANDARD-ENGLISH FEATURES:
&CASE: → ∈ { obl gen nom }.

(20) a. COMMON FEATURES:
MOOD: → ∈ { imperative indicative subjunctive
successive }.

b. STANDARD-ENGLISH FEATURES:
&MOOD: → ∈ { `imperative indicative subjunctive` }.

Some complex features are not found in English and hence must be removed from the set of common features. This is the case for GEND which encodes grammaticalized gender, as in (21).[6]

(21) a. COMMON FEATURES:
GEND: → ∈ { `fem masc neut` }.

b. STANDARD-ENGLISH FEATURES:
!GEND: → ∈ { }.

In general, the strategy within ParGram has been to identify the features used by indvidual grammars and discuss their functionality and what they are intended to denote. If the feature committee finds that the features introduced by the individual grammar are there for sound linguistic and/or implementational reasons, then these features are ratified and may enter the common feature space. In practice, therefore, values should rarely be added to the standard versions of the languages because these values should have occurred in the common feature declaration.

As a temporary measure, however, individual grammars will add values until the addition of the value is approved by the feature committee (adding values is more common in specialized grammars; see (30) and (31) below). An example of such an addition is a value for the TIME feature of `day-part`, as shown in (22), to encode time expressions such as *evening* and *morning* for a grammar which has to pay particular attention to date-time expressions (for example, this could be useful for scenarios in which appointments are to be arranged; this was the domain of the machine translation project Verbmobil (Wahlster, 2000)).

(22) a. COMMON FEATURES:
TIME: → ∈ { `date day hour minute month season second week year` }.

b. STANDARD-ENGLISH FEATURES:
+TIME: → ∈ { `day-part` }.

Features are very rarely redefined, since the redefinition of features is generally an indication that something is wrong with the analysis in

[6] English does have values for GEND-SEM which reflects semantic gender and occurs with pronouns and proper names.

that it is not shared accross the grammars. However, it can be useful as a development tool: for example, in the current English grammar, the feature PSEM is replaced, not because its values are different, but because PSEM is a set-valued feature in English instead of having atomic values like in the other languages (compare footnote 3). The English grammar was chosen to experiment with having PSEM be a set value feature because many prepositions are ambiguous as to their class (e.g., both directional and locative).

(23) a. COMMON FEATURES:
PSEM: $\rightarrow \in \{$ `ag ben compar dir inst loc manner num part poss purp temp` $\}$.

b. STANDARD-ENGLISH FEATURES:
!PSEM: $\rightarrow \ll \{$ `compar dir loc manner num part purp temp` $\}$.

Finally, a number of specialized features appear in the English grammar. Many of these are FORM features and hence their values are language-specific. Examples from the English feature declaration are shown in (24) for PRT-FORM (particles in particle verbs) and COMP-FORM (complementizers for subcategorized subordinate clauses).

(24) a. STANDARD-ENGLISH FEATURES:
PRT-FORM: $\rightarrow \in \{$ `back_ by_ down_ in_ off_ on_ out_ over_ up_ aside_ away_ about_ around_ open_ round_ together_ through_ along_` $\}$.
COMP-FORM: $\rightarrow \in \{$ `for if null that whether` $\}$.

The organization of the feature space in this way thus has the major advantage that high-level similarity across languages can be registered in a space of common features. Individual languages can then choose to manipulate details of that feature space for linguistically justified reasons and can further define language particular information, such as the precise form of the complementizer, in a way that will not to interfere or damage the generally established feature space. We see this flexible and yet transparent mode of organization as a major advantage over the rather rigid architecture prescribed by inheritance hierarchies.

A look at German, a language whose feature space is by no means identical to that of English, serves to further illustrate the definitional ease of the underlying formalism. Although German differs from English, in the German grammar, the common feature declaration is modified in a very similar way to the methodology employed by the English grammar. The set of possible values of CASE, for example, is also restricted by means of the & operator, as is shown in (25):

(25) STANDARD-GERMAN FEATURES:
&CASE: → ∈ { `acc dat gen nom` }.

Features that are not grammaticalized in German, such as PROG (progressive) in (26), are removed from the feature declaration by overwriting the common declaration by the empty set.

(26) STANDARD-GERMAN FEATURES:
!PROG: → ∈ { }.

Just as is the case for the English grammar, the German grammar is also subject to continual upgrades and revisions. One of the current phenomena under revision is the treatment of articles (SPEC) in German. The standard within ParGram allows for only a complex DET feature within specifiers; however, it may also make sense to allow for a complex interrogative (INT) feature. This is currently being experimented with in the German grammar and the INT value is therefore added in as a temporary measure (subject to subsequent approval by the feature committee). Thus, the organization of the feature space allows for a flexible experimentation with new analyses: there are no immediate deep architectural consequences if changes in the feature space are made, as would be the case with an inheritance hierarchy.

(27) STANDARD-GERMAN FEATURES:
+SPEC: → ≪ { INT }.

Finally, the German grammar, just like the English grammar, uses a small number of specialized features such as PRECOORD-FORM, which is exemplified in (28). Since it is a FORM feature, its values are language-specific.[7]

(28) STANDARD-GERMAN FEATURES:
PRECOORD-FORM: → ∈ { `entweder nicht ` nur sowohl weder` }.

The XLE system also allows for grammar adaptation whereby a standard grammar developed for a language can be modified to a specific corpus (Kaplan et al., 2002). Among the possible adaptations is the capability to allow controlled changes of the feature declaration. In particular, the specialized grammar can use the standard features as a default, while overriding only those features which have changed in the

[7] The back quote in `nicht` nur` is the escape character used to indicate a space in a multiword expression.

specialized version. This way, any changes made to the feature declaration of the standard grammar are automatically incorporated into the specialized grammar. This is done in the grammar configuration file where the feature declaration is stated, as in (29), which gives priority to the EUREKA feature declaration while using the STANDARD one as a default. As before, the COMMON feature declaration has the lowest priority.

(29) FEATURES: COMMON STANDARD-ENGLISH EUREKA-ENGLISH.

The specialized feature table then only needs to contain newly introduced features, e.g., FIELD in (30a) and EUREKA-TYPE (30b), or features whose values have changed, e.g., NSEM and NUMBER-TYPE, in (31).

(30) a. FIELD: → ∈ { `cause item problem solution` }.

b. EUREKA-TYPE: → ∈ { `adjust-num cause-num dc-num fault-num part-name part-num pl-num qual-num repair-num tag-num xerox-acron` }

(31) +NSEM: → ≪ [EUREKA-TYPE].
+NUMBER-TYPE: → ∈ { `hex` }.

Thus, the same tools that can be used to specialize a common set of features to a specific language can be used to create a grammar for a specialized corpus from a core or standard grammar.[8] These tools allow the grammar writer to maintain the core grammar and yet create a specialized version with the minimal number of changes. As such, any improvements to the core grammar will be automatically incorporated into the specialized one. In addition, the grammar writer can create multiple specialized grammars without worrying about cross-specialization interference in which a change intended for one corpus adversely affects the coverage of a different one.

2.3. The Feature Committee and the Feature Table

Within ParGram, the definition of the multilingually relevant feature space is established via twice-yearly meetings among the grammar writers. Here, new developments in the individual grammars are checked and discussed. New features introduced for the grammar of a particular language are skeptically reviewed and are only sanctioned if they can

[8] See (Kaplan et al., 2002) for ways in which the rules themselves can be shared and modified for specialized grammars.

Table I. Sample FEAT-TABLE

Feature	ENGLISH	GERMAN	JAPANESE
CASE	acc gen nom	acc dat gen nom	acc dat gen nom
INF-FORM	bare to	zu	
TNS-ASP	MOOD PERF	FUT MOOD PASS-SEM	MOOD PROG
	PROG TENSE	PERF TENSE	TENSE
VTYPE	copular main modal	copular main modal	copular main

be shown to have a universal application or if it becomes clear that the individual grammar could not have done without this feature. Every effort is made to keep the feature-space as small as possible and to assimilate new analyses within the existing feature-space. To facilitate this, the feature declarations are combined into a feature table (FEAT-TABLE) which has a column for each language and the features as rows; the cells are the values of the features for a particular language. A sample feature table is shown in Table I.[9]

The FEAT-TABLE can be used at a fairly high level to determine which grammars cover which constructions and what types of analyses are applied. The FEAT-TABLE can also be used to determine trivial differences across the grammars, like the spelling of CASE values mentioned above (e.g., ACC vs. AKK).

One typical situation in which the FEAT-TABLE proved useful is exemplified by the treatment of noun types. The precise classification of different types of nouns cannot be imported from theoretical linguistics directly into grammar development (unlike, for example, analyses of case or tense/aspect). Rather, as the grammars grew and had to parse different kinds of nouns (e.g., as encountered in the Wall Street Journal), individual grammar writers introduced more and more features to constrain the number and types of different analyses. These were all duly recorded in the FEAT-TABLE, enabling the grammar writers to easily compare their analyses and naming conventions and then agree on a standardized approach across the grammars. In particular, a very flat structure had arisen whereby most of the features were immediate attributes of the noun's predicate. Most of these were then rearranged under an NTYPE feature and further subdivided into syntactic (NSYN) and semantic (NSEM) features which in turn could have complex values.

[9] A Perl script is used to create the feature table automatically from the feature declarations of the individual grammars.

The syntactic values include information as to whether the noun is, for example, `common`, `pronoun`, or `proper`; the semantic values include information as to what type of proper noun it is (e.g. `location`, `name`). This more hierarchical structure allows the grammar writer to envision more easily the possible types of nouns in the grammar and to make reference to noun classes with fewer disjunctions.

In summary, once a language particular feature F (or similarly, a language particular value for a common feature) has been proposed, the feature committee can then examine the nonconforming specifications and decide whether (i) the feature F is actually something that is crosslinguistically relevant, but has not come up yet as a part of the analyses in any of the other grammars. In this case, F should be added to the common feature specification unless the feature is typologically very rare. On the other hand, (ii) F may be the result of an aberrant analysis on the part of the grammar writer and therefore should be disallowed. Finally, (iii) F may reflect a language particular or very rare characteristic and should therefore be included as part of the language-specific feature declaration. In particular, it is worth examining in detail anywhere that a + or ! operator was used to add a feature value (+) or to completely rewrite a feature (!). Features that appear in a particular language without an operator are interesting in that they may indicate features necessary for some phenomenon not yet covered in the other grammars. As such, these features may need to be added to the common feature declaration. Thus, the ability to invoke multiple feature declarations in an override fashion allows for more direct multilingual grammar development and highlights differences among the feature spaces of the languages.

3. Capturing Generalizations through Templates

The discussion in the previous section focused on the use of feature *representations* according to an agreed-upon definition of the feature space. This itself does not necessarily have implications for the generality in which *descriptions of* or *constraints on these representations* are expressed in the grammars. It would be possible (although difficult) to write a grammar that does not capture any linguistic generalizations, but conforms to the feature declaration in each of its *ad hoc* rule statements. This section addresses what devices the LFG ParGram project employs to achieve and exploit parallelism across grammars at the level of descriptions.

3.1. Benefits of sharing constraints across grammars

Much of the grammar development work goes into the identification of suitable generalizations, since they are crucial for making the grammar readable, extensible, and maintainable. Although parallelism across multilingual grammars at the description level is not as indispensable as the representation-level parallelism (which is sufficient for applications like machine translation), it is possible to avoid a significant amount of duplicate work by using a common specification of language-independent principles across grammars. Ultimately, a systematically organized grammatical description is also the best way to guarantee consistent representations.

An obvious example for a language-independent subsystem required for each of the grammars is the organization of the lexicon, particularly with regard to subcategorization. For example, verbs can universally be classified in terms of notions such as intransitive, transitive, or ditransitive. While languages may differ in terms of case marking or other requirements, there is a level at which generalizations in terms of subcategorization classes are very useful. When a new grammar is added to the project, the grammar writer can immediately expect to distinguish between intransitive, transitive, and ditransitive verbs. Given a universally defined lexical rule which generates passives from actives, the grammar writer may also expect to find passives in the language. The same holds for templates which define subject-verb agreement.

As was the case for the discussion of the feature space definition in section 2, one possible view to take with respect to the organization of the lexicon is that the linguistic theory should enforce a highly systematic structure. In the LFG ParGram project, it is an option to express high-level linguistic generalizations explicitly and share them across the grammars of multiple languages, but the formalism and grammar-writing conventions do not enforce such a policy. In the remainder of this section we use the organization of subcategorization in the lexicon as an example to argue that the flexible depth of systematicity is justified and advantageous for broad-coverage grammar development; in the subsequent sections we will show how cross-linguistic generalizations can be expressed and enforced across grammars.

In HPSG, the feature structure entities representing (certain aspects of) words and phrases are organized in an inheritance hierarchy of types/sorts. This makes it possible for general properties holding for a large class of items to be specified once at a high level of abstraction and for the specific instantiations to inherit these properties. (Davis, 2001) discusses in detail how linguistic generalizations over classes of lexical items can be expressed in an inheritance hierarchy; specifically,

he aims to derive the grounding of the subcategorization behavior of lexical items in their lexical semantics. This can be achieved to a large degree by assuming a hierarchical organization of lexico-semantic relations and enforcing that the specialization hierarchy of these relations is reflected homomorphically in the organization of the verbs and prepositions realizing them. It has to be admitted however (Davis, 2001, sec. 5.3) that there cannot exist a perfect homomorphism, since there are near-synonymous verbs with distinct subcategorization: *await* vs. *wait* (*for*), *seek* vs. *look* (*for*), etc. To account for such facts, two separate hierarchies for syntactic subcategorization and for semantic relations have to be assumed that are interrelated in a fairly intricate way using the "junk slot" technique.

In the context of broad-coverage grammars, a general derivation of syntactic subcategorization behavior from lexico-semantic properties is impractical because of the lack of sufficiently large lexical resources that could provide the necessary detail. On the other hand, it is relatively straightforward to obtain near-exhaustive lists of the surface-level subcategorization behavior of lexical items, using corpus-linguistic techniques (see e.g., (Eckle and Heid, 1996; Kuhn et al., 1998)). So, it is possible to provide the subcategorization information that is crucial for broad-coverage parsing. In a framework that is flexible with respect to the expression of higher-level generalizations, nothing enforces an additional cross-classification of lexical items according to their (unknown) semantic class. So an *ad hoc* solution for this problem can be avoided. Generalizations for which sufficient information is available can nevertheless be expressed.

3.2. The Syntax of Templates

Templates are used in the XLE implementation of LFG to allow for the grouping of equations in lexicons and in rules. From a linguistic perspective, these groupings allow the grammar writer to capture linguistic generalizations. From an engineering perspective, they allow for much improved grammar maintenance.

The template definitions appear in their own section of the grammar. Their basic format is:

```
template_name(parameters) = equations.
```

Templates are called via:

```
@(template_name parameters)
```

The template will then expand to the relevant equations with the values provided by the parameters in the call.

The template can have any number of parameters, including none. These values will be passed in when the template is called. The equations consist of any equations that would be valid in a lexical entry or rule. An example of a template with no parameters is shown in (32a) and a call to the template is shown in (32b). This template simply assigns the value + to the feature PASSIVE.

(32) a. PASSIVE =
(↑ PASSIVE)=+

b. @(PASSIVE).

A simple example of a template with a parameter is shown in (33a) with a sample call in (33b); the expansion of the call in (33b) is shown in (33c). This template assigns the value passed in by the parameter to the feature TNS-ASP TENSE and assigns the value `indicative` to the feature MOOD.

(33) a. TENSE-IND(_TNS) =
(↑ TNS-ASP TENSE) = _TNS
(↑ TNS-ASP MOOD) = `indicative`.

b. @(TENSE-IND `past`)

c. (↑ TNS-ASP TENSE) = `past`
(↑ TNS-ASP MOOD) = `indicative`

Templates can call other templates and can pass the parameter values that they receive on to these other templates. For example, in the English grammar the TENSE-IND template that was shown in (33a) in fact calls two other templates, as shown in (34a) with the other templates in (34b) and (34c) (the templates in (34b) and (34c) are templates shared across the grammars; see section 3.3). Note that this does not affect the call shown in (33b).

(34) a. TENSE-IND(_TNS) =
@(TENSE _TNS)
@(MOOD `indicative`).

b. TENSE(_TNS) =
(↑ TNS-ASP TENSE) = _TNS.

c. MOOD(_MD) =
(↑ TNS-ASP MOOD) = _MD.

In addition to passing values of features as parameters of the templates, it is also possible to pass in path names as parameters. This is extremely useful for templates which are called in the rules. An example of this is the template that creates the features associated with null pronouns (e.g., in pro-drop and in infinitives with arbitrary pronominal subjects). This template is shown in (35). A sample call within a verb phrase (VPinf) and the resulting expansion are in (36).[10]

(35) NULL-PRONOUN(_PATH) =
@(PRED_DESIG _PATH pro)
@(PRON-TYPE_DESIG _PATH `null`)
@(NSYN_DESIG _PATH `pronoun`).

(36) a. VPinf: @(NULL-PRONOUN (↑ SUBJ))

b. VPinf: (↑ SUBJ PRED)='pro'
(↑ SUBJ PRON-TYPE) = `null`
(↑ SUBJ NTYPE NSYN) = `pronoun`

Just as the grammars share a common feature declaration, they also share a common set of templates. Each grammar then expands on these templates as needed to capture language particular generalizations and patterns. The types of templates that occur in the common template space are discussed in the next section.

3.3. Shared Templates

As mentioned before, the LFG grammar development philosophy allows the grammar writer of an individual language a large degree of freedom as to whether or not a highly structured subsystem of recursive template calls is used in a specific situation. This has the advantage that less studied linguistic phenomena can be added to the grammar without a major modeling effort. Also, ill-understood exceptions to a given generalization can be included in a grammar without having to change the system of representations itself, something which would be required in a strictly inheritance-based framework. At the same time, this large degree of liberty requires a certain discipline in the context of a multi-site multi-language grammar development project. The code reviewing instance of the feature committee has proven to be one helpful way of controlling the freedom offered by the LFG formalism.

There are approximately 270 templates that are currently shared by the ParGram grammars. Most of these are concerned with assigning

[10] The intermediate XYZ_DESIG templates called in (35) will be discussed below.

values to features and maintaining the correct feature geometry for features like TNS-ASP and NTYPE which have complex features as their values. There are also features defined for common annotations in the rules. Finally, there are some templates that define several notational patterns that are useful to have in template form. We discuss these types more in this section.

First consider the feature templates. For each feature in the feature declaration, there are a set of templates that can be called to assign it. The basic version takes two parameters: a path and a value. The path is the functional path to the feature and the value is the value assigned to the feature. Most of the time, the path will be ↑. As such, an additional template is defined which already includes the path in its call. This is the version that is used most frequently by the grammars. However, the other version is available for rules where other paths are necessary. The two templates for CLAUSE-TYPE are shown in (37).

(37) a. CLAUSE-TYPE(_VAL) =
@(CLAUSE-TYPE_DESIG ↑ _VAL).

b. CLAUSE-TYPE_DESIG(_PATH _VAL) =
(_PATH CLAUSE-TYPE) = _VAL.

For features which always appear as part of a complex feature, the common templates include the information about the complex feature. This makes it easier for the grammar writer to assign the correct feature geometry. This is shown in (38) for TENSE which always appears under TNS-ASP. Note that TNS-ASP_DESIG is called by all of the templates for the features that appear under TNS-ASP, e.g., MOOD, PERF.

(38) a. TENSE(_VAL) =
@(TNS-ASP TENSE _VAL).

b. TNS-ASP(_ATTR _VAL) =
@(TNS-ASP_DESIG ↑ _ATTR _VAL).

c. TNS-ASP_DESIG(_PATH _ATTR _VAL) =
(_PATH TNS-ASP _ATTR) = _VAL.

Rule annotations assigning grammatical functions are also invoked as templates. These help in maintenance (e.g., all the grammars should spell OBJ-TH, the grammatical function for secondary, thematic objects similarly). In addition, they make the grammar code cleaner to read since there is a simple template call instead of a potentially complex equation. An example for the annotation of a subject is shown in (39). It would be called in a rule as in (40a) if the annotation was intended

to be as in (40b), the most common annotation for subjects in the grammars.

(39) a. SUBJ =
@(SUBJ_DESIG ↑ ↓).

b. SUBJ_DESIG(_PATH1 _PATH2) =
@(GF_DESIG _PATH1 SUBJ _PATH2).

c. GF_DESIG(_PATH1 _GF _PATH2) =
(_PATH1 _GF) = _PATH2.

(40) a. S → NP: @(SUBJ); VP.

b. S → NP: (↑ SUBJ)=↓; VP.

Related to this, there are a few templates that define common relations among grammatical functions, such as the subject control template shown in (41).

(41) SUBJ-SUBJ-CONTROL =
(↑ SUBJ)=(↑ XCOMP SUBJ).

Finally, there are a handful of notational templates. These are used to assign default values, expand as *if* and *if and only if* operators, etc. The *default* and *if and only if* templates are shown in (42a) and (42b). (The { | } indicates disjunction,[11] the ∼= stands for ≠. Negation (∼) can also be applied to an entire equation or set of equations like _P and _Q in (42b).)

(42) a. DEFAULT(_FEAT _VAL) =
{ _FEAT
_FEAT ∼= _VAL
| _FEAT = _VAL }.

b. IFF(_P _Q) =
{_P _Q
| ∼_P ∼_Q}.

[11] The first disjunct in (42a) makes use of a so-called existential constraint: if _FEAT is for example instantiated to (↑ SUBJ NUM), the use of just the feature path (without a value assignment) means that in the respective f-structure under (↑ SUBJ), there has to exist a NUM feature with some value, introduced from some other source. The full default template can be paraphrased as follows: "either _FEAT has been introduced independently with some value distinct from _VAL, or else its value is hereby defined as _VAL."

The parameters for these templates can be extremely complex. To assign a default value of singular to the subject's number feature, DEFAULT can be called as in (43).

(43) @(DEFAULT (↑ SUBJ NUM) `sg`).

In addition, these templates can have template calls as their parameters. These will be expanded out appropriately when the grammar is compiled. For example, the grammar writer can use IFF to assign an OT mark (Frank et al., 2001) to disprefer specifiers on proper nouns as in (44). (44) states that if the f-structure corresponding to the template call has a SPECifier and the NTYPE of a proper noun, then the OT mark `SpecProper` is invoked. (Note that square brackets [ ] are used to group together sets of equations so that they can be interpreted correctly as the first parameter of the template call.)

(44) @(IFF [(↑ SPEC) (↑ NTYPE NSYM)=C `proper`]
@(OT-MARK `SpecProper`))

Thus, there are a number of templates that are shared among the ParGram grammars. In a way similar to the common feature declaration, these are used to help maintain parallelism among the grammars. In addition, the use of templates helps with grammar maintenance in that the templates capture generalizations and encode them in one place in the grammar, namely in the template definition. This is particularly notable in the lexicon where the same set of equations is called repeatedly, e.g., all transitive verbs call the same sets of equations. In order to make a change to a class of lexical entries, it is only necessary to change the internal expansion of the relevant template; no change at all needs to be made in the lexicon itself.

3.4. Template Hierarchies

LFG does not assume a sort hierarchy as part of the linguistic theory. However, means of abstraction like templates in the grammar specification make it possible to organize the lexicon in the same general way, following a specialization hierarchy where appropriate: more specific templates can call more general templates as part of their definition. The verbal subcategorization frame templates within the ParGram grammars provide a good example.

The central contribution of a verb subcategorization frame is a specification of the verbal stem and the number and type of arguments this verb subcategorizes for. The information about the stem is passed in as a variable, as shown in (45a). However, it is usually the case that other

restrictions also apply to verbs with that basic subcategorization frame. These restrictions can be invoked by calling other templates, as in (45b) which invokes templates that require a finite COMP and that block particles and sentential subjects. In turn, this template may be called by other templates that further restrict it, as in (45c) which requires a *that* complement and (45d) which requires a *whether* complement.

(45) a. V-SUBJ-COMP_PRED(_STEM) =
(↑PRED)='_STEM<(↑SUBJ)(↑COMP)>'.

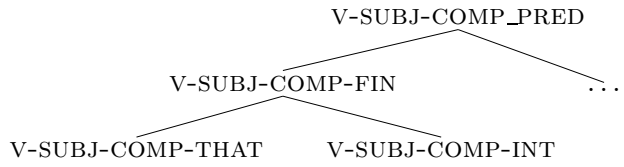
b. V-SUBJ-COMP-FIN(_STEM) =
@(V-SUBJ-COMP_PRED _STEM)
@COMP-FIN @NO-PRT @NO-CL-SUBJ.

c. V-SUBJ-COMP-THAT(_STEM) =
@(V-SUBJ-COMP-FIN _STEM)
@(COMP-FORM that).

d. V-SUBJ-COMP-INT(_STEM) =
@(V-SUBJ-COMP-FIN _STEM)
@(COMP-FORM whether).

The example in (45) results in a partial template hierarchy as in (46).

(46)



Similar dependencies among the templates can be found throughout the template system, e.g., the nominal templates also form a loose hierarchy. Thus, although theoretical LFG does not have a type hierarchy, the XLE implementation of LFG allows for templates to be used to capture many of the generalizations found in type hierarchies, but in a more flexible manner. This is because no type hierarchy is *defined* a priori, but it rather *emerges* as part of the structure of the grammar. This means that changes to the "hierarchy" involve simple and perhaps even incidental changes at the level of template calls—they do not have possible major architectural consequences.[12]

[12] It should be pointed out that there are, of course, ways of sidestepping a given type hierarchy in HPSG, if grammar engineering makes it necessary; but it seems generally easier to deal with "non-canonical phenomena" in a leaner and less theory-loaded hierarchy of grammatical descriptions.

4. Conclusions

The ParGram project attempts to test LFG for its universality and coverage and to see how far parallelism can be maintained across languages; previous ParGram work (and much theoretical analysis) has largely confirmed the universality claims of LFG theory. The f-structures produced by the grammars for similar constructions in each language have the same major functions and features, with minor variations across languages (e.g., the f-structures for French nouns have a gender feature but not the English f-structures). This uniformity has the computational advantage that the grammars can be used in similar applications and that machine translation (Frank, 1999) can be simplified. It also has the advantage that when new grammars (such as Urdu or Japanese) are added to the project, they can be bootstrapped relatively efficiently using the existing declared feature space and the existing templates (see (Kim et al., 2003) on the more extreme case of porting an entire grammar from one language to another).

The ParGram methodology of establishing consistency and parallelism of representations relies on (i) a feature committee in which extensions to the existing representations are critically reviewed on linguistic and engineering grounds and (ii) technical tools for checking adherence to a feature declaration for linguistic representations and for encoding generalizations over descriptions. The resulting system of checking feature appropriateness resembles the theory-driven typed feature structure signatures of HPSG; however, the feature appropriateness conditions are not predominantly theory-driven, but are established by best practice. In particular, the very flexible architecture allows local changes to be implemented without compromising the overall architecture of the system. This allows for a transparent integration of on-going empirical and linguistic research into multilingual grammar development. In particular, it takes into account the specific needs of multi-site development, and seems to be more appropriate for broad-coverage grammars in which representations often have to be defined for phenomena not yet analyzed in the linguistic literature. A similar approach is taken in the systematic structuring of the grammatical descriptions: means of abstraction like templates can be used to implement a hierarchical organization of linguistic generalizations where appropriate. Such a structured grammatical subsystem can be used across grammars in a multilingual context.

Acknowledgements

This paper is a significantly revised and expanded version of our paper presented at ESSLLI 2003 in Vienna at the workshop on Ideas and Strategies in Multilingual Grammar Development. We would like to thank the participants for lively discussion and helpful suggestions and two anonymous reviewers for their extensive comments.

References

Bender, E., D. Flickinger, and S. Oepen: 2002, 'The Grammar Matrix: An Open-source Starter-kit for the Rapid Development of Cross-linguistically Consistent Broad-coverage Precision Grammars'. In: *Proceedings of the Workshop on Grammar Engineering and Evaluation*. pp. 8–14. COLING 2002 workshop.

Butt, M., H. Dyvik, T. H. King, H. Masuichi, and C. Rohrer: 2002, 'The Parallel Grammar Project'. In: *COLING 2002: Workshop on Grammar Engineering and Evaluation*. pp. 1–7.

Butt, M., T. H. King, M.-E. Niño, and F. Segond: 1999, *A Grammar Writer's Cookbook*. CSLI Publications.

Butt, M., M.-E. Niño, and F. Segond: 1996, 'Multilingual Processing of Auxiliaries in LFG'. In: D. Gibbon (ed.): *Natural Language Processing and Speech Technology: Results of the 3rd KONVENS Conference*. pp. 111–122.

Copestake, A.: 2002, *Implementing Typed Feature Structure Grammars*. Stanford, California: CSLI Publications.

Dalrymple, M.: 2001, *Lexical Functional Grammar*. New York: Academic Press. Syntax and Semantics, volume 34.

Dalrymple, M. and R. M. Kaplan: 2000, 'Feature Indeterminacy and Feature Resolution'. *Language* **76**(4), 759–798.

Davis, A. R.: 2001, *Linking by Types in the Hierarchical Lexicon*. Stanford, CA: CSLI Publications.

Eckle, J. and U. Heid: 1996, 'Extracting Raw Material for a German Subcategorization Lexicon from Newspaper Text'. In: *Proceedings of the 4th International Conference on Computational Lexicography, COMPLEX'96*. Budapest, Hungary.

Flickinger, D. and E. Bender: 2003, 'Compositional Semantics in a Multilingual Grammar Resource'. In: *ESSLLI 2003 Workshop on Ideas and Strategies for Multilingual Grammar Development*. This volume.

Frank, A.: 1999, 'From Parallel Grammar Development towards Machine Translation'. In: *Proceedings of MT Summit VII*. pp. 134–142.

Frank, A., T. H. King, J. Kuhn, and J. T. Maxwell: 2001, 'Optimality Theory Style Constraint Ranking in Large-scale LFG Grammars'. In: P. Sells (ed.): *Formal and Empirical Issues in Optimality Theoretic Syntax*. Stanford, California: CSLI Publications, pp. 367–397.

Hudson, R.: 1995, 'Does English really have case?'. *Journal of Linguistics* **31**, 375–392.

Kaplan, R. M. and J. Bresnan: 1982, 'Lexical-Functional Grammar: A Formal System for Grammatical Representation'. In: J. Bresnan (ed.): *The Mental Representation of Grammatical Relations*. The MIT Press, pp. 173–281.

Kaplan, R. M., T. H. King, and J. T. Maxwell III: 2002, 'The Parallel Grammar Project'. In: *COLING 2002: Workshop on Grammar Engineering and Evaluation*. pp. 29–35.

Kaplan, R. M. and P. Newman: 1997, 'Lexical Resource Reconciliation in the Xerox Linguistic Environment'. In: D. Estival, A. Lavelli, K. Netter, and F. Pianesi (eds.): *Computational Environments for Grammar Development and Linguistic Engineering*. pp. 54–61. Proceedings of a workshop sponsored by the Association for Computational Linguistics, Madrid, Spain, July 1997.

Kim, R., M. Dalrymple, R. M. Kaplan, T. H. King, H. Masuichi, and T. Ohkuma: 2003, 'Multilingual Grammar Development via Grammar Porting'. In: *ESSLLI 2003 Workshop on Ideas and Strategies for Multilingual Grammar Development*.

Kuhn, J., J. Eckle, and C. Rohrer: 1998, 'Lexicon Acquisition with and for Symbolic NLP-Systems – A Bootstrapping Approach'. In: *Proceedings of the First International Conference on Language Resources and Evaluation (LREC98)*. Granada, Spain, pp. 89–95.

Maxwell, III, J. T. and R. M. Kaplan: 1993, 'The Interface between Phrasal and Functional Constraints'. *Computational Linguistics* **19**, 571–589.

Pollard, C. and I. Sag: 1994, *Head-Driven Phrase Structure Grammar*, Studies in Contemporary Linguistics. University of Chicago Press.

Przepiórkowski, A.: 2001, 'ARG-ST on phrases: Evidence from Polish'. In: D. Flickinger and A. Kathol (eds.): *Proceedings of the 7th International Conference on Head-Driven Phrase Structure Grammar*. Stanford, California, pp. 267–284, CSLI Publications.

Riezler, S., T. H. King, R. M. Kaplan, R. Crouch, J. T. Maxwell III, and M. Johnson: 2002, 'Parsing the Wall Street Journal Using a Lexical-Functional Grammar and Discriminative Estimation Techniques'. In: *Proceedings of the ACL*.

Wahlster, W. (ed.): 2000, *Verbmobil: Foundations of Speech-to-Speech Translation*. Heidelberg, Germany: Springer.