Computer Science Journal of Moldova, vol.**2**, no.1(4), 1994

MT and Unification*

D.Estival

1 Introduction

This paper presents one of the projects pursued at ISSCO, the development of a prototype for a machine translation system and describes ELU, the environment built for this project and in which the prototype was developed. This project has been very much of a cooperative effort and the list of those who made substantial contributions to ELU or its intellectual ancestor, UD, is too long to allow for detailed individual acknowledgement here. Successive stages of its development can be traced through the papers by Johnson and Rosner (1989), Warwick, Russell, Boschetti and Bouillon (1989), Russell, Warwick and Carroll (1990), Estival, Ballim, Russell and Warwick (1990), Estival (1990b), Russell, Ballim, Estival and Warwick (in prep.), which are referenced in the bibliography.

The experimental work of building a machine translation prototype is embedded in more general projects such as designing an evaluation methodology for machine translation systems, experimenting with different theories of translation or testing the adequacy of linguistic theories. ELU (Environnement Linguistique d'Unification) is a linguistic development environment based on unification. A linguistic development environment is a software system which serves as a guide or framework for linguists building natural language processing systems. The aim of this paper is to explain the base concepts of this environment and show their relations to the goal of machine translation (MT).

In computational linguistics, the traditional division between the "developers" and the "users" of a computational tool is mirrored by

*Appeared in *Communication with men and machines,* A Festschrifft for Dalle Molle, M. King ed. pp. 86-112 (1991). Reprinted with permission.

the two issues which must be confronted for every problem, i.e. the computational and the linguistic aspects of the problem. For MT, the two issues can be characterized as follows:

- the *computational* issue: the design and implementation of a computational tool for translation.
- the *linguistic* issue : the description of the linguistic problem of translation.

ELU was designed and built as a computational tool general enough for the development of computational linguistic applications in general, but more specifically for MT, and its design already presupposes a certain conception of the interaction between the computational implementation and the linguistic description.

2 MT as Knowledge Description

Research in MT may have several purposes, but any MT system is by definition required to accept as input a text in one language and to produce as output a text in another language. A possible approach to MT is to view it as modelling the knowledge of a good translator performing the translation. Regardless of the psychological implications of such an approach, this knowledge can be characterized as involving the following two components which must be represented in the knowledge base of any MT system:

- separate knowledge of source and target languages (linguistic ability);
- translation specific knowledge (translation skill).

The division between the two kinds of knowledge implies that the translation skill of a translator is separate from that translator's linguistic ability in the different languages involved. The requirement that the knowledge of the source and target languages be separate implies that linguistic ability in one language is not affected by linguistic ability in another language. This is a more controversial assumption psychologically but a reasonable one in the absence of a good theory of language interference. It also is a crucial requirement in the design of a flexible MT system, allowing it to be easily extendable to different language pairs.

Therefore, an important task in building an MT system is to write grammars for each of the languages independently of each other, and a fundamental step in the design of the system is the definition of the formalism for the grammars. The term **grammar** is taken in the larger sense and means the description of linguistic knowledge, or more formally, the characteristic function of the (infinite) set of the well-formed expressions of the language.

A linguistic description contains different types of information: lexical, morphological, syntactic and semantic. An important point is that all these partial descriptions are reversible, that is they can be used for synthesis as well as for analysis.

The translation process can be represented with the classic U-shaped schema of Figure 1, indicating the role played by the grammars. The arrows leading to the **transfer** component show the use of the grammar for **analysis** while the arrows leading away from the **transfer** component show the use of the grammar for **generation** (i.e. synthesis).

As for translation itself, we see in Figure 1 that it is viewed as a mapping between representations in a language description into representations in another language description. This means that we have made a choice and decided not to pursue the other approach to translation in MT, i.e. "interlingua", but follow the "transfer" approach. This is not the place for a discussion of the merits of the two approaches.

For each language, the grammars produce representations which abstract away from the particular form of the language at different levels of abstraction, and one must therefore decide at which level the mapping for transfer (both lexical and structural) should be made. The approach adopted is that it should be at the highest level of abstraction, under the hypothesis that this will be the most uniform across languages. This hypothesis is independent of any semantic theory but

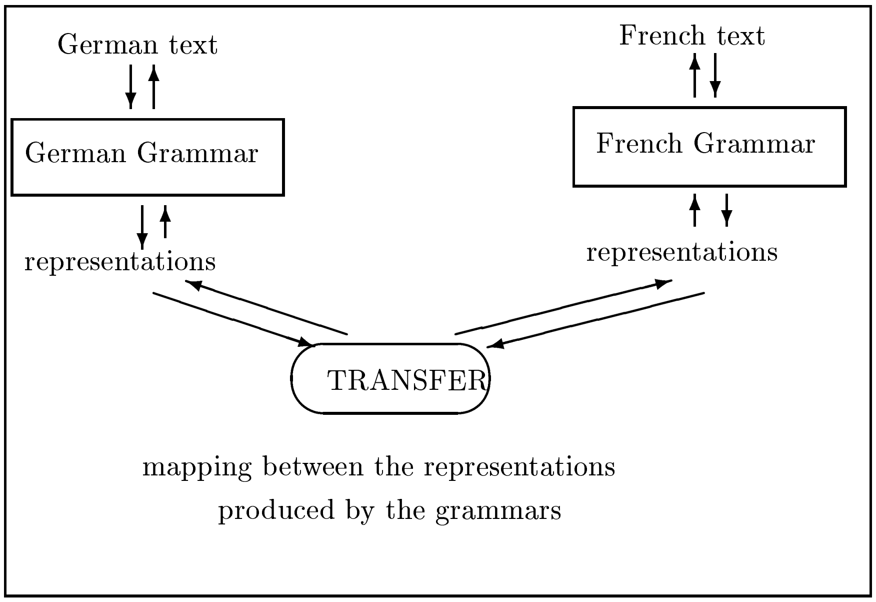

Figure 1:The Translation Process

depending on the linguistic theory chosen the level of abstraction will be the level of semantic representation, of logical form, of functional structure, etc.

3 ELU (Environnement Linguistique d'Unification)

ELU has its first origins in the system developed at ISSCO by Johnson and Rosner (called UD for "Unification Device"), presented in Johnson and Rosner 1989, which can be described as an enhanced PATR-II style environment for linguistic development. The original version has been revised and extended and now contains a generator (described in

Warwick et al. 1989 and Russell et al. 1990) and a transfer component (described in Estival et al. 1990). Even in those components which do not exist in PATR, the syntax of ELU still follows rather closely the PATR formalism which has become a standard for unification-based systems (Shieber 1986).

An environment is both a computational tool and a formalism for representing linguistic knowledge, and these two aspects, as a computational work environment and as a metalanguage for stating linguistic descriptions, correspond closely to the two issues of MT mentioned above. In ELU, both of them are determined by the fact that the formalism adopted is based on unification, and that the syntax of ELU has a declarative semantics.

3.1 Unification-based Formalism

I briefly review the general idea of a unification-based approach to grammar. Unification is a formal operation which operates on certain kinds of objects, and all the linguistic unification-based formalisms rely on two basic assumptions:

- the informational elements are feature structures (FSs)
- the operation combining this information is the unification of FSs.

A feature structure (FS) is an organized representation of the information needed by the system manipulating it. All the knowledge embodied by the system is contained in FSs. An FS is a set of features, where each feature consists of an attribute-value pair (an attribute-value pair can be viewed as a path in a graph and be written <attribute value>). The value of an attribute can be an atom or an FS, as seen in (1). This is an example of a complex FS, where the attribute "agreement" has a complex value which is itself an FS consisting of a set of three features.

$$(1) \quad \left[\begin{array}{llll} cat & n & & \\ head & sem & pred & Pierre \\ agreement & gender & masc & \\ & number & sg & \\ & person & 3 & \end{array}\right]$$

The unification of two FSs combines the information contained in them. The operation of unification is based on the relation of subsumption. This relation holds between two FSs F_i and F_j, i.e. F_i subsumes F_j, iff all the attribute-value pairs of F_i form a subset of the attribute-value pairs of F_j. Unification between two FSs succeeds iff the common attributes of these two FSs either end in the same values or can be unified (note that the definition is recursive because the value of a common attribute may itself be an FS). The result of unification is a third FS which contains the attribute-value pairs which are found in either one or both of the original FSs.

The two FSs of (2) can serve as an illustration of unification as information combining operation. If the FSs in (2.a) and (2.b) are unified, they form the FS given in (1).

$$(2) \quad \text{a.} \quad \left[\begin{array}{llll} cat & n & & \\ head & sem & pred & Pierre \\ agreement & gender & masc & \\ & number & sg & \\ & person & & \end{array}\right]$$

$$\text{b.} \quad \left[\begin{array}{lll} cat & & \\ head & sem & pred \\ agreement & gender & \\ & number & sg \\ & person & 3 \end{array}\right]$$

Neither (2.a) nor (2.b) subsumes the other, but each subsumes (1), the FS resulting from their unification.

Unification can be required between FSs by using constraint equations, in the rules or in the lexical entries. For instance, (3) is an example of a constraint equation: it indicates that the FSs which are the value of the "agreement" attributes of "XP" and "head" have to unify.

(3) <XP agreement> = <head agreement>

Solving a constraint equation means checking that the path description given by that equation unifies with the FS under consideration. Constraint equations are the only way not only of specifying information, but of specifying how to use that information. To get a result, all the applicable constraints must have been solved, and there is no ordering among the constraints. This is what makes the system declarative.

3.2 ELU: The Computational Tool

Since one of the goals of the project was to provide linguists (seen as users in the process of developing an MT system) with a flexible tool in which they could experiment with several analyses and state them easily, some of the characteristics of ELU as a computational tool for linguistic development are worth pointing out, especially those which are due to the formalism chosen. Unification is a clear and well-defined operation, and a system based on unification allows the representation of linguistic knowledge independently of any particular machine or application:

- *machine independence*: ELU is a LISP program (Common LISP); it was developed on SUNs but does not require a specific interface.
- *general purpose*: ELU is a computational tool suitable for a large range of linguistic applications, such as:
  - the description of a particular language at a particular level: lexical, morphological, syntactic, or semantic.
  - a specific task such as *analysis* or *synthesis* of particular languages, or *transfer* between FSs. Transfer between FSs has application beyond MT, see Russell et al. (in prep.).

  - or *translation*, i.e. an application which comprises all these more specific tasks.

- *user-friendliness*: ELU offers a clear and flexible syntax for writing linguistic descriptions, incremental compilation and debugging facilities for testing them.

- *modularity*: A consequence of the formalism, this property is related to user-friendliness. Our own experience with the development of ELU is an illustration of the benefits of this aspect of the system. Although at first efforts had concentrated mainly on the morphological and syntactic components and on the lexicon/morphology/syntax interfaces for the descriptions used in analysis, as soon as the generator was added, those descriptions were formally usable for generation as well (but see Russell et al. 1990 and Estival 1990a for some linguistic issues). Once the transfer component was added, these monolingual descriptions were also usable without modification, the linguistic work consisting in adding the transfer rules.

3.3 ELU: The Formalism

ELU is designed to offer the same formalism in all of its components, be it for synthesis, analysis or transfer. To be usable in such a broad range of tasks, a formalism must have a clear transparent syntax and a declarative semantics. The basic properties of ELU as a formalism for linguistic description are that it is *unification-based* and that it is *declarative*. These two properties are related to each other, since a formalism which only allows the operation of unification is necessarily declarative: the only statements which are expressible are constraint equations. These two properties immediately give the system certain advantages.

3.3.1 Unification-based

Among the advantages deriving from being based on unification, we first note that unification has become a central concept for a number of computational tools: PATR-II, FUG, GDE, (cf. Shieber 1986, Kay 1983, Carroll et al. 1988); and for a variety of linguistic theories: GPSG, LFG, HPSG (cf. Sag et al. 1986, Johnson 1988).

More concretely, besides declarativeness, a unification-based formalism can be characterized by the following properties:

- *expressivity*: as a metalanguage, the formalism can be used to express different types of analysis, and analyses at different levels of abstraction.

- *uniformity*: across these different analyses and across the various grammatical components.

- *theoretical neutrality*: the system does not impose any particular linguistic theory. Nevertheless, it is particularly well-suited for the implementation of the fundamental properties of modern linguistic theories, (i.e. declarativeness and lexicalism); ELU therefore allows the linguist to simulate various syntactic theories, such as LFG, GPSG, HPSG, or GB.

3.3.2 Declarativeness

Declarativeness is the fundamental property from which all the other properties mentioned follow and can best be explained by way of a negative definition. A declarative program, description, or rule is not procedural, it does not state how to do something, but under what conditions something is done or is true. The interpretation of declarative statements does not depend on the processes running them; in other words, declarative statements have a meaning independently of their application. In addition, the ordering among the statements is irrelevant, since what matters for the result is that all the applicable statements can be shown to hold at the same time.

The property of declarativeness means that a linguistic description is a set of independent statements (constraint equations) about the well-formed expressions of the language. Each statement or rule is local and bidirectional.

Locality of rules is a necessary property for any system which has any claim to generality. As in other formalized fields using rules and representations, it is one of the essential principles in linguistics that a rule must only refer to the smallest possible context in which it applies. FSs can be very complex structures, but the rules working on them do not have to be as complex, because those structures may be decomposed into smaller parts. Locality of the rules allows the description as a whole to be:

- *flexible*: permitting changes during development.
- *incrementable*: permitting additions during development or even in application. (The modularity of the system as a whole parallels the incrementability of the descriptions.)

Bidirectionality of rules is an essential property of the declarative formalism, and it allows the construction of:

- *reversible grammars*: for each language, only one grammar is needed for both parsing and generation.
- *bidirectional transfer*: between two languages, only one set of rules is needed to express the translation relation in both directions.

Grammar reversibility is a working hypothesis which is worth pushing as far as possible, because it imposes certains constraints for the linguistic analyses (Dymetman et Isabelle 1988, Russell et al. 1990) and permits a better testing of the covering of the grammar. If it is only used for analysis a grammar might overgenerate (and accept ungrammatical input), while if it is only used for generation, the grammar might undergenerate (select one paraphrase for generation and not be able to accept other grammatical variants). A reversible grammar has to strike a balance between the two. It is also more efficient for development and maintenance, since changes need not be repeated. Like grammar reversibility, transfer reversibility is also a hypothesis that we are pursuing (Estival et al. 1990). The consequences of transfer reversibility are harder to foresee, because the properties of a transfer module are much less well understood. The strong claim it makes about the nature of translation may prove not to be tenable, but this claim must be investigated, and this investigation is possible precisely because the formalism allows the transfer rules to be bidirectional.

3.4 Extensions to the formalism

The unification formalism is very powerful but its simplicity is also extremely constraining and some extensions are needed to increase its expressivity. Most systems based on unification have found it useful and convenient to add some features to the basic formalism. The problem all have had to face is to ensure that these extensions do not detract from the advantages of the formalism.

The extensions of the formalism which are specific to ELU and distinguish it from other unification-based formalisms, such as PATR-II or FUG, are the following:

- ELU provides pre-defined expressions for *negation* and for *disjunction*
- ELU accepts *terms* (complex objects) and *lists* as attribute values.
- ELU accepts *variables* over attribute-value paths.
- ELU permits defining new data types with *typed expressions*.
- ELU allows abstracting over sets of constraints with *parameterized templates*.
- ELU allows direct manipulation of lists defined as feature values.

These extensions to the general unification formalism make the metalanguage of ELU more expressive, and permit a more direct expression of linguistic generalizations.

Parameterized templates are a way of abstracting over sets of constraints. Like PATR templates, they are a shorthand for sets of constraint equations and thus permit the statement of lexical, morphological, syntactic and semantic generalizations in a concise way. But parameterized templates are a much more powerful mechanism than simple templates, because they can take arguments. Moreover, in ELU a parameterized template admits not only recursive but also multiple definitions. The recursive and multiple definitions of parameterized templates introduce a certain amount of non-determinism which may cause some difficulties in generation, but this question falls outside the scope of this paper and will not be addressed here (see Russell et al. 1990, Estival 1990a and 1990b).

The simple template of (4.a) defines a three place agreement relation on *, A, B , which is expressed by the two constraint equations. The variable "*" in an equation always refers to the FS described by the rule to which the equation is attached. A call like (4.b) then triggers the set of equations to apply.

(4) a.
```
# Define Agree(A,B)
  <* agreement> = <A agreement>
  <A agreement> = <B agreement>
```

b.
```
!Agree(A,B)
```

An example of the use of multiple definitions is shown in (5), in which the template "X1" allows the complement of an N to be an NP, a PP or an SBAR, or the complement of a V to be an NP, a PP, an SBAR or a VP.

(5)
```
X1(Compl, H)
    <H cat> = n
    <Compl cat> = np/pp/sbar
    <H compl> = Compl

X1(Compl, H)
    <H cat> = v
    <Compl cat> = np/pp/sbar/vp
    <H compl> = Compl
```

List manipulation is facilitated by two Prolog-like operators "++" and "−−". The equation in (6) is an example of list manipulation on lists defined as feature values. It is an illustration of the standard way of dealing with the subcategorization requirements of lexical items. The equation permits unification only if the NP mentioned in the right-hand side of the equation can itself be unified with an element of the list which is the value of the attribute "subcat".

(6) <* subcat> = <VP subcat> −− NP

This equation takes a member off the list of subcategorized items attached to the VP structure, and makes the reduced list the value of the FS which is the "subcat" attribute of the FS described by the rule to which the equation is attached, i.e. "*". The set of equations in (7.a) and (7.b) give examples of the possible result of unifying (6) with other equations imposing different constraints on the value of the subcat list.

(7) a.
```
<VP subcat> = [NP]
<* subcat> = []
```

b.
```
<VP subcat> = [NP,PP]
<* subcat> = [PP]
```

None of the extensions listed above destroys declarativeness, i.e. makes the system procedural. In particular, negation is only allowed for atomic values, and so the question of ordering of statements does not arise. Disjunction is a very difficult problem in all the proposed extensions to the pure unification mechanism (cf. Ramsay 1990). After experimenting with general disjunction, i.e. allowing it for every kind of terms, it was deemed safer to keep it restricted to atomic values. The only problem with lists and typed expressions is that they may be tricky for a beginner to manipulate.

4 The components of an MT system

The architecture of the system shown in Figure 1 is the classic one, with the three components of a transfer system: **analysis** of the source language, **transfer** between representations for the source and target languages, and **generation** of the target language. The bidirectionality of rules given by the declarativeness of the formalism makes it possible to have reversible grammars, or linguistic descriptions, and to clearly separate them from the processes of analysis and generation. Similarly, the declarativeness of the formalism not only allows the transfer module to be clearly separated from the analysis and generation modules, but it makes it possible to have bidirectional transfer.

4.1 The Linguistic Descriptions

The linguistic description of a language is divided into rules describing lexicon, morphology, syntax and semantics. The declarativeness and expressiveness of the formalism allow a great latitude and permit a healthy eclecticism in the choice of linguistic orientations. Nevertheless, lexicalism has remained a fundamental principle in our approach.

In a unification formalism, the lexicon is treated as a dynamic object, containing information about the morphological and syntactic processes that a given lexical item might undergo as well as static syntactic and semantic information. This information can be stated in a concise way via the parameterized templates expressing powerful lexical generalizations.

The morphological component of ELU is a finite state automaton based on sublexicons of stems, affixes, and continuation classes defining their combinations. A new version of the morphological component will use a default inheritance approach.

A set of annotated context-free rules forms the skeleton of the linguistic description. The context-free part of the rules defines the concatenation of constituents; like production systems, the context-free rewrite rules can be interpreted in either direction. The constraint equations which constitute the annotations on the context-free backbone of the grammar rules determine the combination of syntactic and semantic information on these constituents. These constraints have the same syntax and are of the same kind as the lexical constraints expressing lexical redundancy rules in the lexicon, or any other constraint equation.

The syntactic approach in the grammars written in ELU is theoretically undogmatic, i.e. it borrows from several frameworks the best each has to offer; nevertheless, given the formalism in which we are working, we are naturally closer to surface-oriented theories which are described in unification and feature structure terms (e.g. HPSG and LFG, cf. Torris 1988 for a discussion of those issues).

4.2 The Analysis component

The parsing part of the grammar is the set of context-free rewrite rules annotated with constraints. The current ELU parser was written at ISSCO by R. Johnson. Reflecting the division of the grammar into the context-free backbone and the set of constraints, this two-pass parser at first builds a syntactic tree and then checks the constraints.

The first pass is based on the Earley algorithm, and constructs a phrase structure tree. The extension to the Earley algorithm lies in the prediction step. Instead of relying simply on the syntactic categories of the mother and daughter constituents in the grammar rules to restrict the search space, it can be driven by pre-compiled restrictors. Roughly speaking, restrictors cut down the amount of useless work done by the parser. A restrictor is a specification of a value which can be computed from an FS. Any attribute can act as a restrictor, and the attributes to be taken as restrictors are specified by the grammar writer. If no restrictors are specified, the parser behaves as a simple CKY parser. If only the syntactic category is specified as a restrictor, then the parser is equivalent to the unextended Earley algorithm. The values of the restrictors are checked before attempting to use a grammar rule, and a rule is discarded from the set of rules to be tried if the values of the restrictors in the rule do not match the values of the restrictors in the input. See Shieber 1985 for the use of restrictors in parsing, and the

ELU User Manual (Estival 1990b) for a more detailed description of their use in ELU.

The second pass is a recursive depth-first descent through the phrase structure tree created by the first pass, solving the constraint equations from the rules, i.e. doing the unifications required by the constraints annotated on the rules. During the second pass, the whole grammar (i.e. all the components of the linguistic description) is involved, since the solution to the constraint equations annotated on the context-free rules may involve constraints from other parts of the grammar.

The parser is non-deterministic and returns all the valid parses. If a sentence receives several valid parses, these may be ordered according to a ranking pre-defined by the user.

4.3 The Generation component

While parsing techniques have been studied intensively and there are a number of well-known algorithms in unification-based frameworks, generation is a more recent subject of investigation which is still undergoing development. It is therefore worth spending more time describing the generation component.

4.3.1 The Generation algorithm

The generation algorithm of ELU is based on the algorithm described in Shieber et al. 1989; it was developed at ISSCO by J. Carroll.

Generation is head-driven: each rule has a "semantic head" and the head daughter of a rule is generated before its siblings. The semantic head of a grammar rule is specified by the grammar writer. Thus, the context-free part of the grammar rule in (8) is the traditional rewrite rule S $->$ NP VP, with the additional information that the **vp** (marked with an "H") is to be taken as the head daughter.

(8)
```
s -> np Hvp
 <Hvp cat> = vp
 <Hvp head form> = finite
 <Hvp subcat> = [Subj]
 <np> = Subj
 <s subcat> = []
```

The generator distinguishes two types of rules in the grammar, "chaining" and "non-chaining" rules, and following from this distinction, it employs both bottom-up and top-down processing. The distinction can be explained as follows. All rules have a semantic head, and in the general case, the semantic head of a rule is conceptually similar to the mother node for that rule, e.g. the head of a VP is a verb or the head of an NP is a noun. However, not only is the semantic head not required to unify with the mother node, no part of it is required to match any part of the mother node. Now, in a chaining rule, some subpart of the FS for the head daughter, defined as its "semantics", must unify with a subpart of the FS for the mother of the rule. Any rule in which this is not the case is a non-chaining rule.

The partition of the set of grammar rules into chaining and non-chaining rules is pre-compiled from the specification of what counts as the "semantics" of an FS. The grammar writer specifies it with a statement like the one shown in (9), which is encoded as a set of restrictor values during compiling.

(9)
```
# Sempaths
 <* head sem>
```

With (9), the chain rules of the grammar are all the rules in which the value for the attribute <* head sem> on the head daughter subsumes the value for the attribute <* head sem> on the mother.

Chaining rules are used bottom-up from the "pivot", which is defined as the lowest point in the path through the head daughters of chaining rules at which the semantics of the FS remains unchanged. Non-chaining rules are used top-down from the pivot.

4.3.2 Generation with a unification grammar

Some problems may be encountered by the writer of a unification grammar suitable for generation. Some of these are due to the simple fact that the generation process itself imposes a mode of thinking which is strange to a linguist accustomed to writing grammars for analysis, others stem from the nature of the linguistic theories which the unification formalism makes so attractive and for which it seems designed, while some are directly attributable to the formalism itself.

Coherence and Completeness The generation algorithm guarantees neither the "completeness" nor the "coherence" of the resulting FS. The coherence and completeness of FSs is a pervasive problem in all the unification-based formalisms (cf. Kaplan and Bresnan 1982). In ELU, we use the usual approach of typing. The responsibility for preventing the generation of FSs unifying with the input but being incomplete (i.e. ensuring completeness) rests with the grammar writer: any structure which needs to be generated in its entirety should not be represented as an unconstrained FS, but must be specified as another data type, i.e. a list, a tree, or a user-defined type expression.

E.g., from an FS such as (10.a), both sentences (10.b) and (10.c) would be generated (if the grammar allows the verb *to eat* to be both transitive and intransitive, of course), because their corresponding FS representations both unify with (10.a). The FS corresponding to (10.b) is (10.a) itself, while the FS corresponding to (10.c) is (10.d), which subsumes (10.a).

(10) a. $\begin{bmatrix} \mathit{head} & \mathit{sem} & \mathit{pred} & \mathit{eat} \\ & & \mathit{arg}1 & \mathit{John} \\ & & \mathit{arg}2 & \mathit{bananas} \end{bmatrix}$

b. John eats bananas.

c. John eats.

d. $\begin{bmatrix} \mathit{head} & \mathit{sem} & \mathit{pred} & \mathit{eat} \\ & & \mathit{arg}1 & \mathit{John} \end{bmatrix}$

A better representation for the arguments of a predicate would use a list for the arguments, as is done in (11.a) from the set of constraints in (11.b).

(11) a.

$$\left[\begin{array}{llll} \mathit{head} & \mathit{sem} & \mathit{pred} & \mathit{eat} \\ & & \mathit{args} & \left[\begin{array}{l} \mathit{John} \\ \mathit{bananas} \end{array}\right] \end{array}\right]$$

b.

```
<* head sem pred>  =  eat
<* head sem args>  =  [Arg1, Arg2]
Arg1  =  John
Arg2  =  bananas
```

Alternatively one could define a structure type, for instance **Trans** with the type definition given in (12.a), and use it as a constraint in a type expression as done in the set of constraints in (12.b).

(12) a.

```
Trans  =  (pred, arg1, arg2)
```

b.

```
<* head sem>  ==  Trans
<* head sem pred>  =  eat
<* head sem arg1>  =  John
<* head sem arg2>  =  bananas
```

The grammar writer and the generator share the responsibility for preventing additions to the input FS (i.e. preserving coherence): the grammar writer must also select appropriate data types, and the generator "freezes" uninstantiated variables that occur in the input.

Unification, Lexicalism and Generation The unification formalism makes it very natural and easy to encode linguistic information in the lexicon, and thus to implement lexicalist linguistic theories, such as LFG, GPSG, or HPSG. These theories have proved extremely successful for parsing, and they are attractive for grammar writing because of the economy and the expressive power they allow the grammar writer. However, because a great amount of syntactically constraining information is found in the lexicon, it is not available during top-down processing. Lexicalism thus deprives the generator of information which, strictly speaking, is syntactic until the lexical item has been found. Subcategorization, for instance, is a lexical property of heads in lexicalist theories, and thus the syntactic properties of a verb which depend on its subcategorization are only available when the lexical head has been proposed for bottom-up generation.

In order to reach the lexical level as soon as possible (otherwise, too many lexical heads will be proposed), most rules of the grammar should be chaining rules, which can be used bottom-up. The grammar writer must ensure that the rules which should be chaining rules satisfy the condition that the "semantics" of the head daughter subsumes the "semantics" of the mother.

The grammar writer must also ensure that the rule does not modify the attributes (of either the mother or the head daughter) which are defined as restrictors and which serve to limit the search space while going bottom-up through the semantic heads of the chaining rules. This use of restrictors in generation was first proposed in van Noord 1988.

Since by definition, the semantics of the mother and head daughter of a chaining rule are identical, care must be exercised in determining what attributes count as "syntactic" or "semantic". Restrictors are also used heavily in the selection of lexical items, so the attributes that are chosen as restrictors must also be good discriminants between FSs. The choice of appropriate data types and restrictors is therefore crucial to ensure that the grammar is not only efficient but usable in generation.

4.4 The Transfer component

As was shown in Figure 1, translation is accomplished by performing transfer between linguistic descriptions. In the MT literature, transfer is rarely described precisely, and there have not been many attempts to formalize it. However, see the Third International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages (Austin TX, June 90), for reports on some recent work along lines very similar to ours. Part of the reason is that in order to solve any problem, one needs a clear conceptual separation between the different parts of the problem and the way these may be solved, but that the problems of translation have often been dealt with in terms of the procedures devised to solve them. As a result, both the problems and their solutions are intertwined in program statements which are very difficult to disentangle, and transfer is often a "black box" in MT systems.

The unification paradigm provides a transparent formalism with a declarative semantics. The declarative semantics on which we have been insisting is what allows a separation of the problems and their solutions. When there are only constraint equations which must all be solved independently, one cannot describe how to solve the problem (as with a procedure), but one must decompose the problem in simple statements about its subparts and the links between those subparts.

Unsurprisingly, therefore, the transfer component of ELU is an extension of the unification formalism. That is, the linguistic descriptions on which transfer is performed are FSs, and the relation on which transfer is based is that of subsumption.

4.4.1 Transfer on feature structures

An MT system built with ELU contains a set of bidirectional rules to perform transfer between two linguistic descriptions. The transfer rules describe mappings between FSs which are the output of analysis with one language description and FSs which are the input to generation with the other language description. Transfer can thus be viewed as a declarative, bidirectional relation between two FSs. For a simple example, consider the sentences given in (13.a) and (13.b). which are translations of each other in French and German, and the structures shown (14.a) and (14.b), which give simplified FS representations for (13.a) and (13.b) respectively.

(13) a. Maria liebt Paul.
Maria loves Paul

b. Maria aime Paul.
Maria loves Paul

(14) a.
$$\left[\begin{array}{llll} head & sem & pred & lieben \\ & & args & \left[\begin{array}{lllll} <1> & head & sem & pred & Maria \\ <2> & head & sem & pred & Paul \end{array}\right] \end{array}\right]$$

b.
$$\left[\begin{array}{llll} head & sem & pred & aimer \\ & & args & \left[\begin{array}{lllll} <1> & head & sem & pred & Maria \\ <2> & head & sem & pred & Paul \end{array}\right] \end{array}\right]$$

Each FS contains the path descriptions for the lexical head of the main predicate of the sentence it represents, and for the list of its arguments (each of them being in turn further specified by other path descriptions). These two FSs can be mapped into each other by a rule like the *lieben-aimer* transfer rule given in (15). Like any other rule in the unification formalism, such a rule contains sets of constraints: two sets of path descriptions (named L1 and L2), each of which must subsume either the input or the output FS representing the sentence being transferred; the third set (named X) consists of equations which give the transfer correspondences between variables mentioned in L1 and L2.

(15)
```
:T: lieben-aimer
:L1: <* sem pred> = lieben
     <* sem args> = [Xg,Yg]
:L2: <* sem pred> = aimer
     <* sem args> = [Xf,Yf]
:X: Xg <=> Xf
    Yg <=> Yf
```

The rule, as stated in (15), is bidirectional, i.e. either L1 or L2 can be the source language. Here, when translating from German into French, L1 is the source, and when translating from French into German, L1 is the target. Intuitively what the rule means can be informally stated as follows:

- the path descriptions in the L1 pattern require that the head of the German FS be the lexical item *lieben*, which has two arguments;
- the path descriptions in the L2 pattern require that the head of the French FS be the lexical item *aimer*, which also has two arguments;

- the transfer correspondences given under X state (recursively) that the transfer relation holds between the two respective heads of the FSs described by L1 and L2 if and only if the transfer relation holds between the respective arguments of these two heads.

4.4.2 Formalism for Transfer

Each transfer rule is bidirectional. The set of bidirectional rules is bidirectional. As with grammars, bidirectionality is made possible but not guaranteed by the formalism. The grammar writer is responsible for bidirectionality of a given set of rules (e.g. when dealing with reentrancy, see below). As a rule, bidirectionality can be interpreted for either direction of transfer. For a set of transfer rules, bidirectionality means, that when transfer is performed in one direction from a source FS and then performed in the reverse direction with any member of the resulting set of target FSs as the source, the original source FS is guaranteed to be in the set of target FSs resulting from the second application of transfer. The result of transfer is a set of FSs because transfer is a relation. This corresponds to the "possible" translation as opposed to "best" translation, where the former (not the latter) is obtained by a reversible function (cf. Landsbergen 1987).

The three sets of constraints in a transfer rule correspond to the three components of a translation system: a set of path descriptions for one language (L1), a set of path descriptions for the other language (L2), and a set of transfer correspondences (X) holding between the bindings of the variables in L1 and L2. The constraint equations in L1 and L2 patterns describe parts of the source and target FSs. When the transfer rules are compiled, L1 becomes the source language in one direction and the target language in the other. Each of the L1 and L2 sets of path descriptions in a rule describes one single FS. All the path descriptions from one set of constraints must therefore be rooted in (i.e. there must be a path leading from) the same variable, which is the root for that rule. For this reason, only deterministic parameterized templates are allowed in the rules.

The FS induced by the set of path descriptions in the source language side of a rule is required to **subsume** the source FS, which was produced analysing of the source text. The relation is that of **subsumption** so as to ensure coherence, i.e. that no extra material is introduced.

The FS induced by the set of path descriptions in the target language side of a rule must **unify** with the target FS, which will be the input to generation. On this side we have to use the operation of **unification**, in order to obtain a target FS which characterizes the set of FSs related to the source FS by the transfer relation.

Transfer correspondences introduce the recursion which allows transfer to traverse the source FS. A transfer correspondence is a relation which states that the binding of a variable in L1 is related through the transfer rules to the binding of a variable in L2. A transfer rule thus states FSs correspondences through the explicit correspondences between variables in the rule.

FSs are recursive by definitio. The attribute value may be another FS. Transfer rules are also applied recursively. Therefore, an FS may itself be a subpart of a larger FS. As is standard in the unification formalism, the distinguished variable "*" in a path description stands for the FS being described (Kaplan et al. 1989). In a rule, the variable "*" occurs in the path descriptions in L1 and L2 where it refers to the root of the FS being transferred. Thus, transfer rules refer to local roots in the same way that grammar rules and lexical descriptions do.

In (16), we have an example of a small transfer section as it would appear in an ELU user program. Not all the paths contained in the source FS produced by analysis need be transferred. The set of paths which require transferring is left to the grammar writer who specifies it in a separate statement at the top of the transfer section, under PATHS1 or PATHS2.

In the *general-pred-args* rule, the variables *Lg* and *Lf* may stand for lists of arguments, the transfer relation being defined on lists, trees and other ELU terms, as well as on simple attribute-value pairs. For simplicity, the only rules shown for nouns in example (16) are quite trivial (they map proper names to themselves), but they allow us to present a simple formalism for lexical transfer, before describing the more complex case of structural transfer.

(16)
```
# Transfer german french
:PATH1: <* cat>
        <* head>
:PATH2: <* cat>
        <* head>

:T: cat
:L1: <* cat> = Cg
:L2: <* cat> = Cf
:X: Cg <=> Cf

:T: sempred
:L1: <* head sem pred> = Rg
:L2: <* head sem pred> = Rf
:X: Rg <=> Rf

:T: general-pred-args
:L1: <* head sem pred> = Rg
     <* head sem args> = Lg
:L2: <* head sem pred> = Rf
     <* head sem args> = Lf
:X: Rg <=> Rf
    Lg <=> Lf

:T: gern-aimer
:L1: <* head sem pred> = Rg
     <* head sem args> = [Ag|Tg]
     <* head sem mod> = gern
:L2: <* head sem pred> = aimer
     <* head sem args> = [Af,Vf]
     <Vf head sem pred> = Rf
     <Vf head sem args> = [Af|Tf]
:X: Rg <=> Rf
    Ag <=> Af
    Tg <=> Tf

:TA: s s
:TA: Paul Paul
:TA: Maria Maria
:TA: lieben aimer
:TA: schwimmen nager
```

4.4.3 Lexical Transfer and Structural Transfer

From a linguistic point of view, lexical transfer is the statement of correspondences between lexical items, as in the individual entries in a bilingual dictionary. Lexical transfer is often contrasted with the more complex structural transfer but can itself be more or less complex.

Simple lexical transfer is the direct correspondence between two words, and is most easily stated with an atomic transfer rule which performs transfer between two atomic FSs without any other constraints. These atomic transfer rules are equivalent to simple lexical entries in a bilingual dictionary. The last four rules given in (16) are examples of atomic transfer rules performing simple lexical transfer.

The rule *gern-aimer* on the other hand is not an atomic transfer rule: besides the correspondence between the two lexical items *gern* and *aimer*, it also contains equations which constrain the transfer of other elements in the FSs.

Atomic lexical transfer can be further constrained by other transfer rules defining other attribute-value pairs which are required to be transferred for lexical items. For instance, with the general rules *sem-pred* and *general-pred-args* in (16), the atomic rule *schwimmen nager* is equivalent to the transfer rule with full path specifications shown in (17).

(17)
```
:T:schwimmen-nager
:L1: <* sem pred> = schwimmen
     <* head sem args> = Lg
:L2: <* sem pred> = nager
     <* head sem args> = Lf
:X: Lg <=> Lf
```

It is important to note that atomic rules can be used for any atomic value. For instance, the last rule of example (16) is an atomic transfer rule for the constant *s*. Combined with the transfer rule *cat*, it allows the transfer of the syntactic category for sentences. Any syntactic or semantic value which is needed for translation can be similarly transferred.

Structural Transfer The two sentences given in (18.a) and (18.b) present an example of a notorious class of translation problems, where a head-modifier relation in one language is not preserved in the other.

(18) a. Maria schwimmt gern.
Maria swims with pleasure

b. Maria aime nager.
Maria likes to swim

The German verbal modifier *gern* in (18.a) must be translated as the French main verb *aimer* in (18.b), while the main verb of the German sentence translates as the infinitive complement in the French sentence. (19.a) and (19.b) are the simplified representations of the two FSs corresponding to (18.a) and (18.b) respectively.

(19) a.
$$\left[\begin{array}{llllllll} head & sem & pred & schwimmen & & & & \\ & & args & [\ <1> & head & sem & pred & Maria\ ] \\ & & mod & gern & & & & \end{array}\right]$$

b.
$$\left[\begin{array}{llllllll} head & sem & pred & aimer & & & & \\ & & & [\ <1> & head & sem & pred & Maria \\ & & args & \ \ <2> & head & sem & pred & nager\ ] \\ & & & & & & args & [\#1]\ ] \end{array}\right]$$

We can perform transfer between these two FS with the small transfer section given in (16), in particular with the *gern-aimer* rule.

4.4.4 The Transfer Relation

The transfer relation between two FSs is defined both by the set of transfer rules between the two languages for which these FSs are valid representations and by the way these rules are applied to perform transfer. The rules themselves are declarative, so each of them can be interpreted independently of the process running them, but the transfer component presupposes a particular control strategy, specifically one of recursive application of the rules for transfer to parts of the source FS. The choice of which rules to apply at a given point during transfer is determined by:

a) the relation of **subsumption** between (subparts of) the source FS and the FS induced by the set of path descriptions for the source language in the applicable rules, and

b) a partial ordering of the transfer rules based on the relation of **specificity**

While (a) is part of the formalism on which ELU is based, since unification is based on subsumption, (b) is actually a matter of theoretical choice, not required by the formalism.

Recursive application of the rules The choice of which subparts of the source FS to use as input to transfer at a given point is determined by the correspondences stated over variables in the X part of the rules. Each correspondence requires that the transfer relation hold between the binding of a variable in the source FS and the binding of a variable in the target FS. The object in the source FS which unifies with the variable in the source part of the rule becomes the current input to transfer, and the root of that FS must unify with the root of the path descriptions (represented by the variable "*") in the transfer rules to be applied recursively.

For a given source FS F_i, we try in parallel all the applicable rules. If the set of rules is ordered according to the relation of **specificity** (see Estival et al. 1990), then the applicable rules are the most specific rules w.r.t the source FS at that point, and the starting point is the TOP element in the poset of transfer rules.

Trying to apply a rule R_i to F_i is defined as matching the FS F(R_i) induced by the path descriptions of R_i for the source language against F_i. If F(R_i) **subsumes** F_i, the rule may be tried, and the variables in F(R_i) are bound to objects in F_i.

A transfer rule R_i succeeds w.r.t. a source FS F_i and a target FS F_j if all the path descriptions it contains for the source language, including the ones entailed by the transfer correspondences between variables, **subsume** the source FS, and if all the path descriptions it contains for the target language, including the ones entailed by the transfer correspondences between variables, **unify** with the target FS.

For every rule R_i which succeeds, the FS induced by the path descriptions for the source language, i.e. F(R_i), is unified with the FS F_k resulting from the unification of all the F(R_j) of the rules R_j which have successfully applied. F_k is used for checking completeness of transfer.

Since transfer is performed by actually creating a target FS F_j, some of the problems presented by transfer are akin to those found in generation, namely termination of the process and ensuring completeness and coherence of the target FS F_j.

Termination is guaranteed if there is no rule of the form (20)– shown only going from L1 to L2– where an FS ("*") is rewritten as a whole.

(20)
```
:T: infinite-recursion
:L1: <*> = X
:L2: .....
:X: X <=> Y
```

Any other rule requires that transfer be applied recursively to a subpart of an FS. FSs being finite and there being a finite number of rules, termination for acyclic FSs is guaranteed. While ELU allows any FS to be cyclic, transfer of a cyclic FS can only be guaranteed if its cyclicity is specifically treated by a transfer rule; moreover, generation from a cyclic FS cannot be guaranteed.

Completeness of the resulting target FS w.r.t the source FS and the set of transfer rules is ensured by matching the source FS F_i with F_k (the unification of the FSs induced by the source patterns of all the rules that have successfully applied). Transfer is complete if every subpart of the source FS F_i which requires transferring subsumes a subpart of F_k.

On the other hand, coherence is ensured by the rules themselves: nothing is introduced in the target FS which is not mentioned in the target part of the transfer rules which have applied. Therefore, every subpart of the target FS must be subsumed by the FS induced by the path descriptions in the target part of those transfer rules.

Reentrancy An important point about the control of transfer concerns reentrancy. Reentrancy is a powerful mechanism commonly employed in unification-based grammars: it occurs when two attributes in an FS share the same value. Typically, reentrancies are used to express linguistic phenomena such as agreement or control, that is in cases where it is not enough to state that the values are similar, but where we want those attributes to bind the same object. The treatment of reentrancy in the mapping between FSs requires special care. The approach taken here is to ignore reentrancies which occur in the source FS unless they are explicitly mentioned in transfer rules.

When a reentrant feature is not explicitly mentioned as being reentrant in a transfer rule, the effect is equivalent to unfolding the source FS, with the consequence that the target FS is also unfolded, i.e. the feature is not reentrant in the target FS. When the FS is unfolded, the two instances of the feature receive the same translation because the same set of transfer rules will apply to them, but they do not necessarily bind the same variable. The reentrancy then becomes a question to be solved during the generation of the target language. If the target language grammar can create a reentrancy from the variables present in the target FS used as input to generation, i.e. can determine that two variables actually bind the same FS, then this reentrancy doesn't have to be mentioned in the transfer component. However, this an unlikely situation, even if a large amount of semantic information is included in the descriptions.

If a transfer rule mentions a reentrant feature in both L1 and L2, as in the example of (21), where two attributes share a variable, then this reentrancy is automatically preserved in the target FS. This example is not meant to provide an analysis of German and French modal verbs, but simply to illustrate the point that the same analysis can be given in the two languages, and preserved through transfer.

(21)
```
:T: wollen-vouloir
:L1: <* head sem pred> = wollen
     <* head sem args> = [Ag,Vg]
     <Vf head sem pred> = Rg
     <Vf head sem args> = [Ag|Tg]
:L2: <* head sem pred> = vouloir
     <* head sem args> = [Af,Vf]
     <Vf head sem pred> = Rf
     <Vf head sem args> = [Af|Tf]
:X:  Rg <=> Rf
     Ag <=> Af
     Tg <=> Tf
```

In the *gern-aimer* rule given in (16), there is a reentrant feature on only one side of the rule, namely in L2. This reentrancy will be created during transfer between German and French, but transfer from French into German will ignore it.

Thus, in effect, the user can "make or break" reentrancy. This allows us to deal with cases where the phenomena of control and anaphora are different across languages, and we consider that it is neither necessary nor desirable to automatically keep reentrancies in translation. This approach contrasts with that of the MiMo2 system (van Noord et al. 1990), where reentrancies in the source FS are always preserved in the target FS. It would be possible to do so in ELU: the approach would consist in refolding the FS at the end of transfer, by adding a separate section of rules which would apply on the target FSs.

Rule Specificity Specificity is a relation between transfer rules defining a partial order in a set of such rules. This ordering permits a particular control strategy for the application of the rules which itself has interesting consequences for the linguistic aspects of transfer.

The relation of specificity is only defined between unidirectional rules: for a given pair of transfer rules, which rule is more specific depends on the source/target choice, i.e. on the direction in which transfer is to be done. When compiling a set of bidirectional transfer rules, two partially ordered sets (posets) are built, each corresponding to transfer in one of the two directions. The **specificity** relation between two transfer rules A and B is defined by:

a) the relation of **subsumption** between the two FSs produced by their respective sets of constraint equations for the source language, which we write F(A) and F(B).

b) the number of variables mentioned in their respective sets of correspondences, which we write V(A) and V(B).

For any pair of rules, either one of the rules is more specific than the other, or they are not comparable. Given any two rules A and B for transfer from L1 to L2, we then define the partial ordering relation (i.e. a transitive, reflexive and anti-symmetric relation, denoted by '$\preceq$') of **specificity** ("is more specific than") as in (22).

(22) For two transfer rules A and B, A $\preceq$ B iff

i) B = A, or

ii) F(B) $\neq$ F(A) & F(B) subsumes F(A), or

iii) F(B) = F(A) & V(B) $\prec$ V(A)

The partial ordering of the transfer rules defines a set of applicable rules w.r.t. the previous point during transfer, i.e. a subset of the transfer rules which contains the most specific rules whose source language descriptions subsume the source FS at that point. A control strategy which incorporates this partial ordering of rules therefore finds all the rules that can apply to the object for transfer such that for each successful rule there is no more specific rule that can succeed. The only condition that needs to be added to the recursive application of rules is (23):

(23) If a rule R_i succeeds, no rule R_j s.t. $R_j \preceq R_i$, is tried.

The motivation for defining the relation of specificity between transfer rules and for using the ordering it defines as part of the control strategy is that it provides an elegant and consistent way of blocking a general interpretation when there is a specific translation which is more appropriate. For instance, *cheval - Pferd* is the general rule, but *cheval blanc - Schimmel* is more specific and should block it. If the partial ordering of the transfer rules according to the relation of specificity is integrated into the control of transfer, then the grammar writer does not have to worry about blocking rules: the more general rules are automatically blocked and cannot apply when a more specific rule succeeds. The most general rules of the transfer system are the atomic transfer rules and any transfer rule which is more general than any other rule involving either its L1 or L2 root.

However, it is worth noting that in this respect, the system behaves differently when performing analysis/generation and when performing transfer. The strategies used for analysis and for generation do not prevent the application of a general rule when a more specific one succeeds. If a grammar rule can apply, then it does and all the valid FSs which can be built in accordance with the applicable rules are returned. The control strategy of transfer based on the specificity relation defined here therefore amounts to a strong claim about translation namely that, unlike in the grammars, in the transfer component, more specific rules override less specific ones. This may prove to be too strong and may have to be relaxed.

Nonetheless, even if this claim was too strong, and it turned out that we must allow the application of the more general as well as of the more specific rules, there are a number of advantages in ordering the transfer rules according to the relation of specificity. ELU is a development tool and this ordering, which formally defines the start and the end of the transfer process, helps the user in keeping track of failures during testing. Secondly, the partial ordering of the rules defines an ordering of the results, which could itself be a research tool: if we had to allow the more general rule (e.g. *cheval blanc - weißes Pferd*) to apply, we could order its result w.r.t. the more specific rule (e.g. *cheval blanc - Schimmel*) which also succeeds. This ordering of the results might then reveal generalizations to be made, and give insights for further refinements of the rules.

5 Conclusion

Unification gives a clean declarative formalism with well-understood properties. In particular the essential properties of bidirectionality and locality of the rules present obvious advantages in natural language processing. For MT, reversible grammars represent a considerable saving of time and increase of efficiency for development and maintenance, even if mastering the generation process still presents some difficulty.

In developing ELU to build a prototype for a machine translation system, we have made precise the concept of transfer as a mapping between FSs and shown that the unification paradigm can be extended to transfer without losing the essential properties of bidirectionality and locality in the rules. With its unification-based transfer component, ELU provides a formal basis for experimentation with different theories of translation.

The properties of a reversible transfer component are now better understood, but the full range of its implications and consequences are still being investigated. For instance, the definition of bidirectionality we gave for transfer is necessarily different from the definition of bidirectionality for grammars. Another area of research is the ordering of the transfer rules as part of the control strategy adopted here.

References

[1] Carroll J., B.Boguraev, C.Grover and T.Briscoe. A Development Environment for Large Natural Language Grammars. Technical Report 127, Computer Laboratory, University of Cambridge. 1988

[2] Dymetman M. and P.Isabelle. Reversible Logic Grammars for Machine Translation. Proceedings of the Second International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages. Carnegie-Mellon University, Pittsburgh. 1988

[3] Estival D. Generating French with a Reversible Unification Grammar. Proceedings of the 13th International Conference on Computational Linguistics (COLING), Helsinki, (1990a), pp.106-112.

[4] Estival D. ELU User Manual, Technical Report, ISSCO. (1990b)

[5] Estival D., A.Ballim, G.Russell and S.Warwick. A Syntax and Semantics for Feature Structure Transfer. Proceedings of the Third International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, University of Texas at Austin, 1990b, pp.131-143.

[6] Johnson M. Attribute-Value Logic and the Theory of Grammar. CSLI Lecture Notes, 16. CSLI, Stanford California. 1988

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

[8] Kaplan R. and J.Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In The Mental Representation of Grammatical Relations. // J.Bresnan, ed. Cambridge: MIT Press. 1982, pp.173-281.

[9] Kaplan R., K.Netter, J.Wedekind, A.Zaenen. Translation by Structural Correspondences. Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics, Manchester. 1989.

[10] Kay M. Unification Grammar. Technical Report, Xerox Palo Alto Research Center. 1983.

[11] Landsbergen J. Montague Grammar and Machine Translation. In Linguistic Theory and Computer Applications, // P.Whitelock, M.M.Wood, H.L.Somers, R.Johnson and P.Bennett, eds. 1987, pp.113-139. London: Academic Press Limited.

[12] Ramsay A. Disjunction without Tears. Computational Linguistics, 16.3, 1990, pp.171-174.

[13] Russell G., A.Ballim, D.Estival and S.Warwick (in prep.). A Language for the Statement of Binary Relations over Feature Structures. Internal report, ISSCO.

[14] Russell G., S.Warwick and J.Carroll. Asymmetry in Parsing and Generating with Unification Grammars. Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics, Pittsburgh. 1990.

[15] Sag I., R.Kaplan, L.Karttunen, M.Kay, C.Pollard, S.Shieber and A.Zaenen. Unification and Grammatical Theory. Proceedings of the West Coast Conference on Formal Linguistics, 1986, vol.5, U. of Washington. Stanford Linguistics Association.

[16] Shieber S. Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics. Chicago, 1985, pp.145-152.

[17] Shieber S. An Introduction to Unification-based Approaches to Grammar. CSLI Lecture Notes, 4. CSLI, Stanford California. 1986.

[18] Shieber S., van Noord G., R.C.Moore, and F.C.N.Pereira. A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms. Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, Vancouver. 1989.

[19] Torris T. Word Order and Phrase Structure. ms. ISSCO. 1988.

[20] van Noord G. BUG: A Directed Bottom Up Generator for Unification Based Formalisms. Dept. of Linguistics, Trans 10, Utrecht University. 1988.

[21] van Noord G., J.Dorrepaal, P. van der Eijk, M.Florenza & L. des Tombe. The MiMo2 Research System. Proceedings of The Third

International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, University of Texas at Austin, 1990, pp.213-233.

[22] Warwick S., G.Russell, R.Boschetti and P.Bouillon. Un riche environnement pour l'analyse et la génération du français. ms. ISSCO. 1989.

Dominique Estival
ISSCO, Université de Genève