A Syntax and Semantics for Feature-Structure Transfer

Dominique Estival, Afzal Ballim, Graham Russell, Susan Warwick

ISSCO, 54 rte des Acacias, CH-1227 Geneva
estival@divsun.unige.ch

1. Introduction

The field of Machine Translation can benefit from the techniques of computational linguistics in the search for a formal basis for discussing MT problems and their solutions. However, there has been a regrettable lack of formal description in the MT literature; too often, both the problems and their solutions lie embedded inside programs in procedural statements. In particular, in contrast with well-known methods for analysis and generation, transfer in MT is still treated as a black box. No formal description of a transfer component with a clean formalism has ever been given and as far as we know does not exist When they are described at all, the transfer components mentioned in the literature appear to be procedural, thus forcing grammar writers to deal with unforeseeable interactions between their statements.

In formal computational linguistics, the unification paradigm has proved to be a fruitful one, and with the development of systems such as PATR-II (Shieber 1988), it has become a standard for a growing sector of the research community. The unification formalism, as part of the general trend in computer science towards "logic programming", provides the means of writing purely declarative statements whose interpretation does not depend on the processes running them. This point is crucial for a clear conceptual separation between statements about the problems and the way they are solved. The work on which we report here is an extension of the unification formalism to provide a solid basis for transfer in unification-based systems.[1]

The transfer component we describe here is an existing program which is integrated into ELU, an enhanced PATR-II style unification grammar linguistic environment based on the UD system described in Johnson and Rosner 1989. ELU already provided a general purpose tool for writing grammars which, because of the declarative semantics of the formalism, can be used for both analysis and generation (Russell et al. 1990). With the addition of a transfer component, ELU now constitutes an environment with a clean formalism in which linguists working specifically on the problems of machine translation can express and investigate theories of translation.

2. Transfer on feature structures

In ELU, the formalism for transfer is similar to the formalism for parsing and generating, thereby allowing the linguist to focus on translation problems while working in an already familiar language. Specifically, we regard transfer as a relation between feature structures (FSs), and we require that the transfer component of an MT system be embodied in statements with the same declarative semantics as the other components of the system.[2]

The main motivation for performing transfer on FSs is that these are the representations which systems in the unification paradigm parse into and generate from, and that this paradigm provides a transparent formalism with a declarative semantics. FSs also represent the abstract level of representation at which it is appropriate to perform transfer. Although the mapping between FSs cannot be performed by the operation of unification

[1] There has been some recent work on extending the unification formalism to deal with translation problems (e.g. Kaplan et al. 1989, van Noord et al. 1989); however, no formal description of the language for those extensions has been given.

[2] In this paper, we focus on the use of mapping between FSs as the transfer component in MT systems. However, a declarative formalism for expressing relations between FSs has applications which are not restricted to MT; for instance, transfer of feature structures could be used for the reduction of logically equivalent expressions to canonical form in database query systems. These ideas are further developed in Russell et al. (in preparation).

alone, our proposal for an extension of the unification formalism preserves for transfer rules the properties which we consider desirable and even necessary, of **bidirectionality** and **locality**.

Rule bidirectionality is an essential property of the declarative formalism, which allows the construction of reversible grammars: for each language, only one grammar is needed for both parsing and generation. Similarly, for transfer, it is possible to have only one set of rules to express the relation in both directions between two natural languages. Like grammar bidirectionality, bidirectionality of transfer is a working hypothesis, and the strong claim it makes about the nature of translation may prove not to be tenable. However this claim must be investigated, and this investigation is possible precisely because the formalism allows the transfer rules to be bidirectional. We come back to the issue of directionality in the discussion of the formalism, but first note that it is not necessary to assume the bidirectionality hypothesis in an actual system built with ELU: two separate sets of transfer rules can be written for any language pair, each used for transfer in a particular direction

Needless to say, locality of the 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.

In defining a formalism for mapping between FSs, some of the specific problems to be considered are the mapping between non-isomorphic FSs, and solving constraint equations on local graphs independently of phrase structure rules. We address these points by describing the formalism for transfer and presenting the control strategy employed by the transfer system. This in turn leads us to discuss the notion of specificity of a rule.

3. Description of the Formalism

ELU is designed to offer the same formalism in all of its components, be it for synthesis, analysis or transfer. We give a brief general description of the formalism for monolingual grammars and lexicons to set the context for the bilingual transfer component.

3.1. Unification Formalism

ELU is a member of the PATR-II family of unification-based systems (Shieber 1986), and as such its syntax is very similar to that of PATR. It also contains some extensions to the PATR formalism which make it easier for a grammar writer to express linguistic generalizations. These extensions, however, do not introduce procedurality into the system and do not make its semantics less declarative (Johnson and Rosner 1989).[3]

An FS is an unordered set of attribute-value pairs, where an attribute (also called a "feature") is a path description, and where the value of an attribute is an atom, a list, a tree, or another FS.

A rule is a set of path descriptions (or "constraint equations") holding over FSs. For a given path description, there is only one path which can fit that description in a particular FS.

FSs can be related to each other by the relation of subsumption, which holds between two FSs Fi and Fj if all of the attribute-value pairs of Fi form a subset of the attribute-value pairs of the other FS, Fj.

The operation of unification of two FSs is based on the relation of subsumption, and is the basic mechanism for solving the constraint equations in rules. Given two FSs Fi and Fj, unification succeeds if and only if 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 Fi and Fj.

Solving a constraint equation from a rule means checking that the path description given by that equation unifies with (a subpart of) the FS under consideration.

[3] These extensions include: (i) the use of lists and trees as terms of the language; (ii) the implementation of disjunction and negation; (iii) the use of variables to range over every kind of term. Example (2) shows both the use of lists and of variables, and example (10) that of negation.

It should be emphasized at this point that since ELU is only a formalism which provides a way of thinking about the general problem of the relations between FSs as linguistic representations, what we say about transfer between FSs is also valid for systems which are implemented in other unification based formalisms.

3.2. A Simple Example

Before describing the transfer component of ELU in detail, we first give a simple example to show that translation can be seen as a declarative, bidirectional relation between two FSs.[4] The sentences given in (La) and (l.b) are translations of each other in French and in German.

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

b. Maria aime Paul.
Maria loves Paul

(2.a) and (2.b) are simplified representations of the FSs for (1.a) and (1.b) respectively. Each contains the path descriptions for the lexical head of the main predicate of the sentence, and for the list of its arguments (each of them being in turn further specified by other path descriptions).

```
(2) a.   ┌head sem  ┌pred  lieben                        ┐ ┐
         │          │args [<1> head sem pred Maria,      │ │
         │          │        <2> head sem pred Paul]     │ │
         └          └                                    ┘ ┘

    b.   ┌head sem  ┌pred  aimer                         ┐ ┐
         │          │args [<1> head sem pred Maria,      │ │
         │          │        <2> head sem pred Paul ]    │ │
         └          └                                    ┘ ┘
```

These two FSs can be mapped into each other by a rule like the transfer rule shown in (3). Such a rule contains two patterns, consisting in sets of path descriptions under L1 and L2, each of which must subsume either the source or the target FS representing the sentence being transferred. The path descriptions in L1 and L2 may contain variables, referring to parts of the source and target FSs.[5] The rule also contains a third set of equations, under X, which give the transfer correspondences between the variables mentioned in the L1 and L2 parts of the rule.

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

As it is stated in (3), the rule is bidirectional, i.e. either L1 or L2 can be the source language. When translating from German into French, L1 is the source, and when translating from French into German, L1 is the target.

[4] Note that transfer is a relation (not a function), which defines a many-to-many (not a one-to-one) mapping between FSs.

5 Note that these are not variables in the source and target FSs, but only in the path descriptions for the constraints in the transfer rule.

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.

3.3. **Formalism for Transfer**

The transfer part of an MT system built with ELU consists of a set of bidirectional rules used to perform transfer between two ELU language 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 rules have the format of the simple rule shown in (3). Each transfer rule is bidirectional and the set of bidirectional rules is itself bidirectional. For a rule, bidirectionality means that it can be interpreted for either direction of transfer. For a set of transfer rules, bidirectionality means the following: 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.[6]

Like the other rules in this formalism (e.g. in the lexical entries, or the grammars used for the analysis and generation of one language), transfer rules contain sets of constraints. The three sets of constraints in a transfer rule (L1, L2, and X) correspond to the three components of a transfer-based 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 the L1 and L2 patterns describe parts of the source and target FSs. However, since transfer rules are bidirectional, L1 does not mean source language, nor L2 target language. It is only when, as described below, the transfer rules are compiled into two unidirectional sets of rules, that 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.[7]

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 by the analysis 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). Note that 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.

The 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 correspondences between objects (the FSs) through the explicit correspondences between variables in the rule.

Because FSs are recursive (by definition, since the value of an attribute may be another FS) and since transfer rules are applied recursively, the FS being considered by a particular rule may itself be a subpart of a

[6] The result of transfer is a set of FSs because transfer is a relation (cf. fn.4). This corresponds to the notion of "possible" translations as opposed to "best" translation, where the former but not the latter is obtained by a reversible function (Landsbergen 1987).

[7] All the paths descriptions mentioned in 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 macros which are deterministic, i.e. equivalent to PATR templates, are allowed in the transfer rules.

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 transfer 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. We will take up the more complex issues of variable binding with reentrancy later, but first, it is time to show an example of a small transfer section as it would appear in an ELU user program.

For simplicity of exposition, the only rules shown for nouns in example (4) are quite trivial, as 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.

Note that not all the paths contained in the source FS produced by analysis need be transferred. Which paths 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).

```
(4) # 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
```

Note that 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.

3.4. Lexical 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. Although lexical transfer is often, as here, contrasted with the more complex structural transfer, it can itself be more or less complex.

Simple lexical transfer is the direct correspondence between two words. It 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 (4) 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 rules can be specified by default with a special rule format used as shorthand for the minimum path specification involving only the distinguished variable "*". This is convenient for writing simple lexical rules; for instance, the atomic rule given in (5.a) is equivalent to the full transfer rule in (5.b), with the three sets of constraints (L1, L2 and X).

```
(5) a. :TA: Hund chien

    b. :T:   Hund-chien
       :L1:  <*> = Hund
       :L2:  <*> = chien
       :X:   -
```

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 rule *sempred* defined in (6.a), the atomic rule *Hund chien* of (5.a) is now equivalent to the transfer rule with full path specifications shown in (6.b).

```
(6) a. :T:   sempred
       :L1: <* sem pred> = X
       :L2: <* sem pred> = Y
       :X: X <=> Y

    b. :T:   Hund-chien
       :L1:  <* sem pred> = Hund
       :L2:  <* sem pred> = chien
       :X:   -
```

It is worth noting that atomic rules can be used for any atomic value. For instance, the last rule of example (4) 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.

3.5. A more complex example: Structural Transfer

The two sentences given in (7.a) and (7.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.

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

b. Maria aime nager.
Maria likes to swim

The German verbal modifier *gern* in (7.a) must be translated as the French main verb *aimer* in (7.b), while the main verb of the German sentence translates as the infinitive complement in the French sentence. (8.a) and (8.b) are the simplified representations of the two FSs corresponding to (7.a) and (7.b) respectively.

```
(8) a.   ┌head sem ┌pred   schwimmen                    ┐ ┐
         │         │ args   [<1> head sem pred Maria]   │ │
         │         │ mod   gern                         │ │
         └         └                                    ┘ ┘


         ┌head sem ┌pred aimer                                  ┐  ┐
         │         │ args  [<1> head sem pred Maria,            │  │
         │         │          <2> head sem ┌pred nager ┐        │  │
         └         └                       └ args  [#1]┘  ]     ┘  ┘
```

We can perform transfer between these two FS with the small transfer section given in (4), in particular with the *gern-aimer* rule, repeated as (9).

```
(9)   :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
```

Recall that in a rule, the root of the current FS is represented by the variable "*", and that the FS induced by the path descriptions for the source language must **subsume** the source FS. Informally, what the *gern-aimer* rule means is as follows:

• in the German FS, *gern* is the modifier of the predicate of the root;
• in the French FS, *aimer* is the predicate of the root; it has two arguments which enter in a control relation;
• the transfer relation holds between the two FSs if (recursively) the transfer relation holds between the German main verb and the second argument of the French verb *aimer,* between the subject of the German main verb and the subject of the French main verb, and between whatever other arguments the German main verb and the French infinitive complement may require.

Going into the details of each constraint:
• the equations under L1 require that:

a) the atom *gern* appear as the value of the path <* head sem mod>, representing the modifier of the current root of the German FS,
b) the root also have (at least) one argument (*Ag*).
• the equations under L2 require that:
a) the atom *aimer* appear as the value of the path <* head sem pred>, representing the predicate of the current root of the French FS,
b) the root have two arguments (*Af* and *Vf),*
c) its first argument *(Af)* be also the first argument of its second argument (*Vf*).[8]
• The correspondences under X require that transfer relations hold:
a) between the variable bound to the FS for the predicate of the root of the German FS *(Rg, i*.e. the verb *schwimmen)* and the variable bound to the FS for the predicate of the second argument of the root of the French FS *(Rf,* i.e. the verb *nager),*
b) between the variable bound to the FS for the first argument of the root of the German FS *(Ag,* i.e. *Maria),* and the variable bound to the FS for the first argument of the root of the French FS *(Af,* i.e. *Maria),*
c) between the variable bound to the FS for the tail of the argument list of the root of the German FS *(Tg),* and the variable bound to the FS for the tail of the argument list of the second argument of the root of the French FS *(Tf)*-- which in this example are both bound to the empty list

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 control strategy embodied by the program which applies the rules for transfer between FSs is one of recursive application of those rules to parts of the source FS. The choice of which rules to apply at a given point is determined by:

a) a partial ordering of the transfer rules based on a relation called **specificity,** and
b) checking for 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.

Note that while (b) is part of the formalism defined quite generally for the approach to natural language processing advocated in this framework, (a) is actually a matter of theoretical choice, which is not required by the formalism. We first go over the more general aspect, dealing with the recursive application of the transfer rules, before we explain what we mean by specificity of rules and motivate our use of an ordering relation between transfer rules.

4.1. Recursive application of the roles

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 bold 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* (with root "*"), we try in parallel all the applicable rules.[9] 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

[8] This requirement follows from the variable *Af,* which appears in two different constraints and expresses a reentrancy.
[9] If the set of rules is ordered according to the relation of **specificity** (see below), 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.

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 shown in (10) going from L1 to L2 (and similarly for the opposite direction).

```
(10) :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. ELU allows FSs to be cyclic, but transfer of cyclic FSs can only be guaranteed if the cyclicity is specifically treated by a transfer rule.

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 Fi 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.

4.2. 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. We first give a definition of the relation in (11).

(11) 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).

The relation is calculated at compile time for every pair of rules.[10] Since its definition mentions the source

[10] Therefore, the ordering relation between two rules cannot be altered by the grammar writer. At first sight, it might appear necessary to be able to specify precedence relations between rules in some cases, e.g. phrasal expressions which are more specific than individual words, but there are a number reasons for not allowing precedence relations to be specified by the user. On the one hand, it would require checking at compile time for coherence and compatibility, and this checking would be computationally very expensive, indeed in some cases intractable. On the other hand, it is better to keep the notion of specificity as it applies to rules formally defined. As an area of research in itself, it should lead to interesting generalizations and allow the formalization of our intuitions about rule specificity.

language, 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.

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 '≤') of **specificity** ("is more specific than") as in (12).[11]

(12) A ≤ B iff
(i) B = A, or
(ii) F(B) ≠ F(A) & F(B) subsumes F(A), or
(iii) F(B) = F(A) & V(B) <V(A)

Two abstract elements, TOP and BOTTOM, are added to the poset of transfer rules. These two elements are defined as follows: TOP is the overspecified rule, from which transfer may be viewed as starting; BOTTOM is the underspecified rule, which causes failure of transfer if it is reached.[12]

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 dealing with blocking phenomena. Blocking phenomena occur in cases where there is a specific translation which one might wish to block the more general interpretation (e.g. *Schimmel - cheval blanc,* see van Noord et al. 1989). 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 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. During transfer, the only condition that needs to be added to the recursive application of rules is the following:

(13) When a rule R*i* succeeds, no rule R*j* less specific than R*i* (i.e. where R*j* ≤ R*i*) is tried.

In this fashion, the more general rules are automatically blocked and cannot apply when a more specific rule succeeds. Recall that the most general rules of the transfer system are the atomic transfer rules and any transfer rule which, while not being an atomic rule, is more general than any other rule involving either its L1 or L2 root[13]

It is worth noting that all the rules from the set of applicable rules at a given point are applied in parallel For example, the two rules given in (14) might embody the *essen/fressen (eat* [of humans]/[of animals]) alternation (the value of the attribute <* sem class>, where the symbol "¬" stands for negation, will distinguish the

[11] Note that in (i), '=' is the symbol for equality between rules, and not identity. I.e. "B = A" can be rewritten as "F(B) = F(A) & V(B) = V(A)". Two rules may be different but equivalent, thus having the same ordering.

[12] It is worth pointing out mat the addition of the TOP and BOTTOM elements to the poset does not necessarily produce a lattice. This is because we are not guaranteed that for any given pair of rules written by the grammar writer, there will be a rule corresponding to the greatest lower bound of the two, nor to their least upper bound.

[13] More formally, these rules are the atomic elements of the poset, that is they "cover" the rule FAIL (i.e. the underspecified element BOTTOM, here the underspecified rule signifying failure of transfer). An example is the rule *gern-aimer,* which is a minimal rule of the transfer component in the German-French direction, because no other rule involving *gern* in that direction is less specific.

two cases).

```
(14) :T:   essen-manger
     :L1:  <* sem pred> = essen
           <* sem args> = [Xg | Tg]
           <Xg sem class> = human
     :L2:  <* sem pred> = manger
           <* sem args> = [Xf | Tf]
     :X:   Xg <=> Xf
           Tg <=>Tf

     :T:   fressen-manger
     :L1:  <* sem pred> = fressen
           <* sem args> = [Xg | Tg]
           <Xg sem class> = ~human
     :L2:  <* sem pred> = manger
           <* sem args> = [Xf | Tf]
     :X:   Xg <=> Xf
           Tg <=> Tf
```

These two rules cannot be ordered with respect to each other, since neither is more specific than the other.[14] If one of them is tried at a given point during transfer, so is the other.

4.3. Reentrancy

The last point to be mentioned about the control of transfer concerns reentrant FSs. Reentrancy 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 problem then becomes a question for 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 mat 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 (15), where two attributes share a variable, then this reentrancy is automatically preserved in the target FS.[15]

[14] In this case, the ordering is the same in both the German-French and the French-German direction. From L1 to L2, the two rules are not comparable because neither subsumes the other, from L2 to L1, they are equal by subsumption, but as they have the same number of variables, they are still not comparable, and cannot be ordered w.r.t. each other.

[15] 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.

```
(15) :T:   wollen-vouloir
     :Ll:  <* 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 (4), there is a reentrant feature on only one side of the rule, namely in *12.* 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.[16] This allows us to deal with cases where the phenomena of control and anaphora are different across languages, and we leave as an area for future research the question of whether it is actually necessary or desirable to automatically keep reentrancies in translation.

5. Conclusion

The transfer component we have described here is part of a general unification-based linguistic development environment It has been designed specifically for the development of MT applications, and provides a formal basis for experimentation with different theories of translation.

For instance, we can explore the implications of the hypothesis of bidirectionality for transfer rules. The definition of bidirectionality we gave for transfer is necessarily different from the definition of bidirectionality for grammars. However, as with grammars, bidirectionality is not guaranteed by the formalism, and the responsibility for ensuring that a given set of rules is in fact bidirectional ultimately rests with the grammar writer (for instance, w.r.t. reentrancy).

Another example is provided by a consequence of the control strategy adopted here. As mentioned earlier, the partial ordering of the transfer rules according to the relation of specificity and the use of this poset for the control of transfer give the grammar writer the freedom not to worry about blocking phenomena: the more general rules are automatically blocked and cannot apply when a more specific rule succeeds. 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. First the TOP and BOTTOM elements of the poset formally

[16] This approach contrasts with that of the MiMo2 system (van Noord et al. 1989), where reentrancies in the source FS are always preserved in the target FS. It would be possible in our approach to refold the FSs at the end of transfer, by adding a separate section of rules which would apply on the target FSs.

define the start and the end of the transfer process. This, in turn, allows the user to keep track of failures during development. Second, as a research tool, the partial ordering of the rules defines an ordering of the results: if we allow the more general rule (e.g. *cheval blanc - weißes Pferd)* to apply, we can order its result w.r.t. the more specific rule (e.g. *cheval blanc - Schimmel)* which also succeeds. This ordering of the results may then reveal generalizations to be made, and give insights for further refinements of the rules.

In any case, we have made precise the concept of transfer as a mapping between FSs, and shown that the unification paradigm can be extended to provide a dean declarative formalism for transfer without losing the essential properties of bidirectionality and locality in the rules.

REFERENCES

Dymetman, M. and P. Isabelle (1988). "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.

Isabelle, P., M. Dymetman and E. Macklovitch (1988). "CRITTER: a translation system for agricultural market reports". *Proceedings of COLING 1988,* Budapest, pp. 261-266.

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

Kaplan, R., K. Netter, J. Wedekind, A. Zaenen (1989). "Translation by Structural Correspondence". *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics,* Manchester.

Landsbergen, J. (1987). "Montague Grammar and Machine Translation". In *Linguistic Theory and Computer Applications,* edited by P. Whitelock, M.M. Wood, H.L. Somers, R. Johnson and P. Bennett. London: Academic Press Limited, pp. 113-139.

Russell, G., S. Warwick and J. Carroll (1990). "Asymmetry in Parsing and Generating with Unification Grammars", to appear in the *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics,* Pittsburgh.

Russell, G., A. Ballim, D. Estival and S. Warwick (in preparation). "A Language for the Statement of Binary Relations over Feature Structures". ISSCO.

Shieber, S. (1986). *An Introduction to Unification-based Approaches to Grammar.* Volume 4 of CSLI Lecture Notes. CSLI, Stanford California.

Shieber, S. (1988). "A Uniform Architecture for Parsing and Generation". *Proceedings of the 12th International Conference on Computational Linguistics,* Budapest, pp. 614-619.

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

van Noord, G., J. Dorrepaal, P. van der Eijk and L. des Tombe (1989). *The MiMo2 Research System,* ms. Utrecht University.