A relational approach to translation

Rémi Zajac

Project POLYGLOSS*

University of Stuttgart
IMS-CL /IfI-AIS, KeplerstraBe 17
7000 Stuttgart 1, West-Germany
zajac@is.informatik.uni-stuttgart.dbp.de

Abstract. In this paper, we show how the notion of "co-description" in LFG used for specifying a translation correspondence in a declarative way [Kaplan et al. 89] can be reformulated in a relational framework gaining modularity and reversibility of grammars. We illustrate this approach with a transfer example taken from [Kaplan et al. 89] using a logic formalism which has feature terms as basic data structures. This formalism is based on term rewriting and it integrates a powerful typing mechanism. In this formalism, the integration of a reversible grammar for a "source" language, a reversible contrastive grammar and a reversible grammar for a "target" language produces a reversible machine translation system.

* Research reported in this paper is partly supported by the German Ministry of Research and Technology (BMFT, Bundesminister für Forschung und Technologie) under grant n° 08 B31163. The views and conclusions contained herein are those of the author and should not be interpreted as representing official policies.

0. Introduction

Unification-based grammar formalisms are now widely used for parsing, and to a smaller extent, for generation. However, few proposals have been made for transfer (e.g. [Kay 84, Isabelle & Macklovitch 86, Kudo & Nomura 86, Netter & Wedekind 86, Kaplan et al. 89, Rupp 89, Zajac 89]). One of the problems is that unification grammars are usually tied to a context-free component describing the constituent structure of one language and that transfer is usually not specified at the constituent level. But then, there is no clear general mechanism for relating two feature terms representing a higher level of linguistic description.

We describe a relational approach to transfer using a logic formalism called TFS [Emele & Zajac 90b] whose basic data structures are feature terms. It is based on term rewriting and it integrates a powerful typing mechanism following ideas from [Aït-Kaci 86]. This approach is relational in the sense that the formalism allows to specify and compute *relations* between classes of linguistic objects defined as feature types. As feature types define constraints on admissible sets of feature terms, the interpreter computes the solution of a feature type system by solving these constraints. Thus, grammars written in this formalism are *inherently* reversible and can be used to compute one or the other element of a relation using the same interpreter (see e.g. [Emele & Zajac 90a]). This computation depends only on the form of the input and on an abstract ordering between feature type definitions (a subsumption ordering). Moreover, the formalism features a multiple inheritance mechanism that allows to modularize linguistic descriptions and to integrate them during computation.

We illustrate this approach by making a comparison with [Kaplan et al. 89] who use the mechanism of "co-descriptions" in LFG to specify translation correspondences. We first recall briefly the notion of co-description and introduce the syntax and an informal semantics of TFS. We then take one example presented in [Kaplan et al. 89] and show how the intuition underlying the notion of co-description can be formulated in TFS gaining modularity in linguistic descriptions and reversibility of grammars.

As shown in [Emele 90, Emele & Zajac 90a], the TFS formalism can be used to describe grammars which are reversible. We show that it is possible, using an appropriate representation, to integrate in a very simple way modular descriptions of:

(1) a grammar describing a relation between a set of strings and a set of linguistic structures for one language,

(2) another grammar of the same type for a second language and

(3) a transfer grammar describing a relation between the two sets of linguistic structures

into one single grammar defining a translation relation between the two sets of strings of the two languages as a product of more elementary relations.

1. LFG co-descriptions

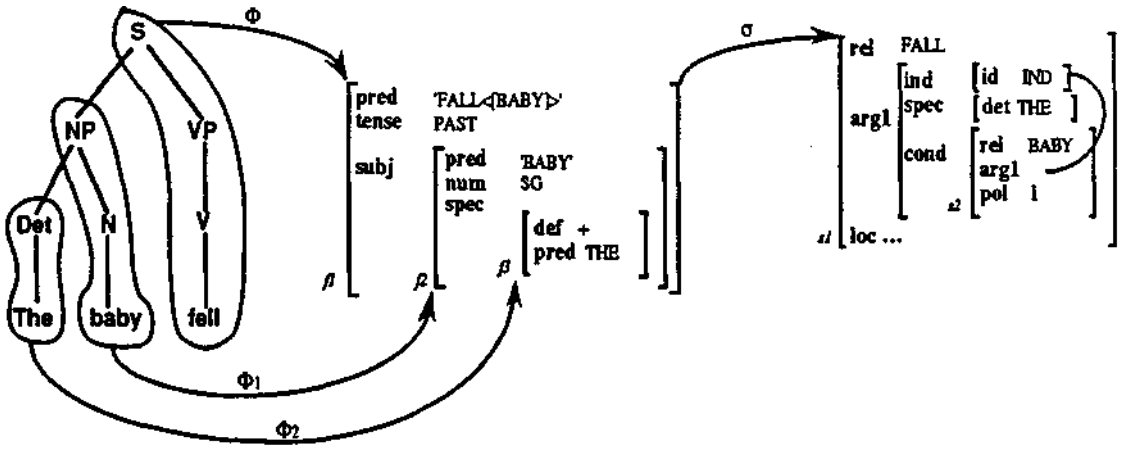
LFG [Kaplan & Bresnan 82] distinguishes two levels of linguistic description:

- a constituent structure (c-structure) that describes the linear and hierarchical arrangement of phrases and subphrases (S, NP, VP,...) in a sentence as a tree.
- a functional structure (f-structure) that describes grammatical functions (subject, object,...) in a sentence as a feature structure.

For example, the rule (1) has two equations (so-called "functional annotations"), one associated with NP and the other with VP. The equation associated with NP, (↑SUBJ) = ↓, says that the f-structure corresponding to the NP node denoted by I is equated to the value of the feature SUBJ of the f-structure corresponding to the dominant node S, the l.h.s. of the CF rule, denoted by ↑. The second equation ↑=↓ equates the f-structure corresponding to the dominant node S to the structure corresponding to the VP node.

$$\begin{array}{lcll} S & \rightarrow & NP & VP \\ & & (\uparrow SUBJ) = \downarrow. & \uparrow = \downarrow \end{array} \qquad (1)$$

As shown in Fig.l (after [Kaplan et al. 89]), these equations define a one-way correspondence Φ between a set of c-structures (the tree on the left of Fig.l) and a set of f-structures (the attribute-value matrix in the middle), by combining recursively correspondences (Φ_1 and Φ_2) between partial c-structures and partial f-structures.



(Figure 1)

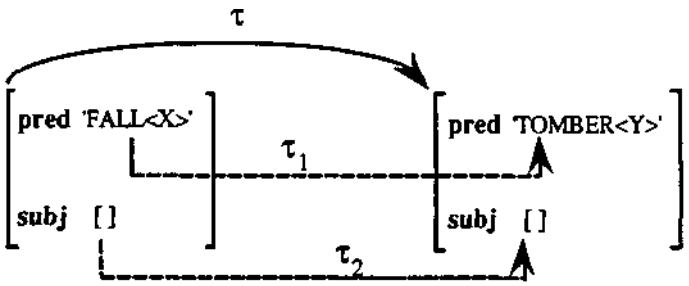
As discussed for example in [Kaplan 87, Halvorsen & Kaplan 88], the system of correspondences can be extended to describe a more abstract level of linguistic description, for example a semantic structure encoding predicate-arguments relations. This can be done by using another correspondence σ from f-structures to s-structures, and one would add semantic *co-descriptions* to CF rules and lexical entries using the function name σ, as shown in (2).

fall : V, (↑PRED) = 'fall<(↑SUBJ)>' (2)
(σ↑PRED) = fall
(σ ↑ARG1) = σ(↑SUBJ).

This framework can be generalized to describe other levels of linguistic representation (e.g. anaphoric and discourse structures). This idea is further extended in [Kaplan et al. 89] to define a correspondence τ between f-structures of a source language and f-structures of a target language by attaching τ co-descriptions to rules and lexical entries. The lexical entry for *fall* with the French equivalent would then be written as in (3), where the attribute FN is used to designate the function-name in the semantic form (i.e. 'fall<(TSUBJ)>'). This would define a partial correspondence τ as shown in (Fig.2).

fall : V, (↑PRED) = 'fall<(↑SUBJ)>' (3)
(τ ↑PRED FN) = tomber
(τ ↑SUBJ) = i(↑SUBJ).

tomber: V, (↑PRED) = 'tomber<(↑SUBJ)>'.



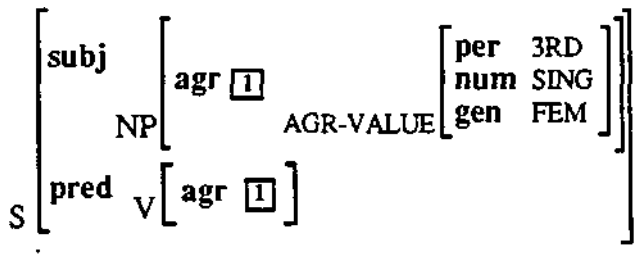
(Figure 2)

Co-descriptions describe one-way correspondences from a source f-structure to a target f-structure: technically Φ and τ are functions [Kaplan 87]. As it would be desirable to use the same grammar for both parsing and generation, ways of inverting such functions have been investigated (e.g. [Wedekind 86, Momma & Dorre 87]). From a linguistic point of view, it would in fact be most convenient to think of these specifications as purely declarative non-directed equations, i.e. as specifications of two-way correspondences or more precisely as *relations.* And this is in fact the intuition which lies behind "co-descriptions".

2. TFS relations

We introduce briefly the syntax and an informal semantics of the TFS formalism.

The basic data structure of the formalism is a typed feature term: ordinary feature terms can be extended to typed feature terms by associating a type symbol to a node in the directed graph that represents the feature term. For some type symbols, a type definition is supplied (by the grammar writer) which can be a disjunction of typed feature terms or a conjunction of typed feature terms (or any expression built up from disjunctions and conjunctions). The disjunction operator is noted "|", and the conjunction operator is noted "&". An inheritance hierarchy of feature terms can be derived from a set of feature type definitions.



(Figure 3)

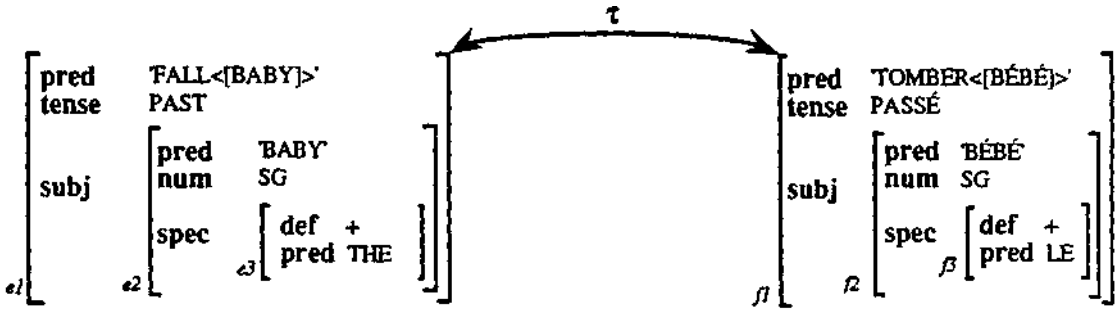
We write feature names in lowercase letters, type symbols in uppercase letters, and we use symbols starting with "#", called tags, for denoting shared values. For example, the feature term (Fig.3) is written in linear form as

```
S[subj:NP[agr:#1=AGR-VALUE[per:3RD, num:SING, gen:FEM]],          (4)
  pred:V[agr: #1]]
```

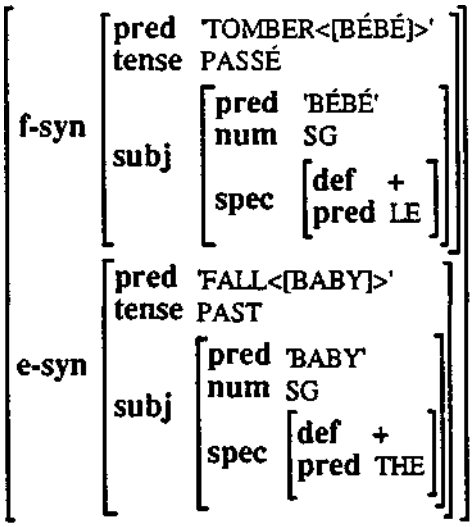
which is read "the feature term (4) is of type S and has two features, `subj` and `pred`. The feature `subj` has as value a feature term of type NP ...".

The evaluation of a feature term amounts to type checking and type inference: the interpreter checks whether a given feature term is consistent with respect to a set of feature type definitions. A typed feature term is consistent if it unifies with the definition of its type, and if each of its subterms is also consistent. Conjunctions and disjunctions are evaluated in the usual way. The application of a feature type definition may recursively introduce subterms that will also need to be evaluated. The semantics of the formalism which is implemented in the interpreter guarantees that the solution computed by the interpreter is the least fixed point of the set of definitions augmented by the initial term [Emele & Zajac 90b].

We assume in the following that we have a grammar written in TFS which defines a set of well-formed lists of words, a set of well-formed syntactic structures, and a (binary) relation G between those two sets, as described for example in [Emele & Zajac 90a]. For the purpose of the exposition, let us also assume that the set of well-formed syntactic structures is defined as f-structures in LFG. We are then interested in defining a relation between a set of source f-structures and a set of target f-structures. We will take examples from French and English: if necessary, we shall distinguish between features or types belonging to the grammar of these languages by using a prefix ('f for French, 'e' for English).



(Figure 4)



(Figure 5)

A member of the translation relation TAU holding between a French f-structure and an English f-structure as in Fig.4 will be encoded as a feature term with two features f-syn and e-syn as in Fig.5, and written as follows:

```
[f-syn:     [pred:    TOMBER,                                  (5)
             tense:     PASSE,
             subj:      [pred:    BEBE,
                         num:      SG,
                         spec:     [def:     +,
                                    pred:   LE]]],
 e-syn:     [pred:    FALL,
             tense:    PAST,
             subj:     [pred:    BABY,
                        num:     SG,
                        spec:    [def:      +,
                                  pred:     THE]]]]
```

The translation relation TAU can then be decomposed in a modular way into different sub-relations, including:

- a bilingual relational lexicon TAU-LEX
- a structural contrastive grammar TAU-STR
- a tense-mood-aspect contrastive grammar TAU-TMA
-

TAU is then defined as the product of these relations:

```
TAU = TAU-LEX & TAU-STR & TAU-TMA ...
```

Each of these modules can interact with each other: for example, the lexical relation can impose constraints on the syntactic structure or on tense values, constraints stated as partial descriptions (subterms of feature terms) which are combined (i.e. unified) during evaluation. The evaluation mechanism allows to achieve both modularity of description and integration of distributed constraints during evaluation.

3. An example

[Kaplan et al. 89] give an example where a time adjunct has to be translated by a verb whose main semantic content is aspectual information:

The baby just fell ↔ Le bébé vient de tomber

They propose an LFG lexical entry for *just,* where *just* is analyzed as a predicate which takes as an argument the VP it modifies :

just: ADV, (↑PRED) = 'just<(↑ARG)>' (6)
(τ ↑PRED FN) = venir
(τ ↑XCOMP) = τ(↑ARG).

The corresponding TFS definition (7) has to specify the full constraints for *venir,* which in other contexts has the meaning "to come" (`VENIR-1`): in this context, *venir* is a subject-control verb (`VENIR-2`). This definition also has to say that the `arg` of *just* corresponds to the `xcomp` of *venir :* this is done by introducing a condition part under the feature `cond` specifying that the relation `TAU` also holds between these two.

```
TAU-LEX-JUST =                                                    (7)
     [e-syn: [pred: JUST,
              arg:  #e-arg],
      f-syn: [pred: VENIR-2,
              xcomp: #f-xc=[compl: DE,
                            tense: INF],
     cond:   TAU[e-syn: #e-arg, f-syn: #f-xc]] .
```

Additional lexical definitions for this example are :

```
TAU-LEX-FALL =                                                    (8)
     [e-syn: [pred: FALL,
              subj: #e-subj],
      f-syn: [pred: TOMBER,
              subj: #f-subj],
      cond:   TAU[e-syn: #e-subj, f-syn: #f-subj]].

TAU-LEX-BABY =                                                    (9)
     [e-syn: [pred: BABY,
      f-syn: [pred: BEBE]].
```

The relation TAU is decomposed into a product of several modules which are applied simultaneously, and one of them is the lexical relation TAU-LEX, defined as a disjunction of bilingual lexical entries:

```
TAU-LEX = TAU-LEX-JUST | TAU-LEX-FALL | TAU-LEX-BABY | ...
```

```
    TAU[e-syn: [pred: JUST,                                       (10)
                arg:  [pred:  FALL,
                       tense: PAST,
                       subj:  [pred: BABY,
                               num:  SG,
                               spec: [def:+,
                                      pred: THE]]]]]
```

Assume that we give the structure (10) as input to the interpreter. The interpreter checks if one of the disjuncts of TAU-LEX can be unified with the term and succeeds with the definition of TAU-LEX-JUST introducing by unification the other part of the relation f-syn (the predicate and the partial xcomp structure), and a condition which also has to be checked, namely that the relation TAU holds between the e-syn: arg and the f-syn: xcomp (11).

```
TAU[e-syn: [pred: JUST,                                                    (11)
           arg:  #e-arg=[pred:  FALL,
                        tense: PAST,
                        subj:  #e-subj=[pred: BABY,
                                        num:  SG,
                                        spec: [def:+,
                                               pred: THE]]]],
    f-syn: [pred: VENIR-2,
            xcomp: #f-xcomp=[compl: DE,
                             tense: INF]],
    cond:  TAU[e-syn: #e-arg,
               f-syn: #f-xcomp]]
```

This last condition is evaluated in the same manner, and for the lexical part, the definition of TAU-LEX-FALL is applied successfully, introducing TOMBER as the head of the f-syn:xcomp and an empty attribute for the subject, and again a condition cond: cond that Specifies that the f-syn: xcomp: subj corresponds to the e-syn: arg: subj in (12).

```
TAU[e-syn:   [pred:   JUST,                                                (12)
              arg:     #e-arg=[pred:     FALL,
                       tense:   PAST,
                       subj:    #e-subj=[pred:   BABY,
                                        num:     SG,
                                        spec:    [def:   +,
                                                  pred:   THE]]]],
    f-syn:  [pred:     VENIR-2,
    xcomp:  #f-xcomp=[pred:   TOMBER,
                      compl:   DE,
                      tense:   INF,
                      subj:   #f-subj]],
    cond:   TAU-LEX-FALL[e-syn:   #e-arg,
                         f-syn:   #f-xcomp,
                         cond:   TAU[e-syn:   #e-subj,
                                     f-syn:   #f-subj]]
```

This last condition is also evaluated in the same manner, and for the lexical part, the definition of TAU-LEX-BABY is applied, introducing the subject of the infinitive clause. The final result of the evaluation is then (13), and it would be the same if one started evaluating with the f-syn part only (the conditions in (13) are left out).

```
[e-syn: [pred: JUST,                                                    (13)
         arg:  [pred:  FALL,
                tense: PAST,
                subj:  [pred: BABY,
                num:  SG,
                spec: [def:+,
                       pred: THE]]]],
 f-syn: [pred:  VENIR-2,
         tense: PRESENT,
         xcomp: [pred: TOMBER,
                 compl: DE,
                 tense: INF,
                 subj: [pred: BEBE,
                        num:  SG,
                        spec: [def: +]]]]]
```

The syntactic module TAU-STR is responsible for the translation of definiteness and number, and the tense module (TAU-TMA) for the translation of the tense of the clause (which interact here with the lexico-syntactic structure). All these modules are in fact applied simultaneously as TAU is defined as the product TAU-LEX & TAU-STR & TAU-TMA: the constraint defined by TAU is the conjunction of the constraints defined by TAU-LEX, TAU-STR and TAU-TMA.

Because the definitions for the translation correspondence given above are purely contrastive, the syntactic structure produced for French is underspecified: in order to generate the complete syntactic structure, constraints belonging to the French grammar should be applied to introduce for example the subject of VENIR-2 (a subject-control verb), to specify the gender of BEBE, and to introduce the definite article: these constraints are not (and should not be) part of the translation correspondence proper.

4. Discussion

After a brief remark about a suitable linguistic level for transfer, we discuss the problem of modularity vs. integration of descriptive levels from a descriptive and a computational point of view: modularity of descriptions (e.g. separate grammars for analysis, transfer and generation) usually implies separate successive corresponding computational processes; on the other hand, integrated processing usually implies to have one integrated description of the computation, like "co-descriptions" for integrating parsing and transfer. Thus, modularity of descriptions seems incompatible with integrated computation. Finally, we discuss the problem of reversibility, which is basically a functional concept, i.e. how to compute the inverse of a function (which is in the general case not a function).

The adequate linguistic level for transfer

As we have seen in the previous example with the translation *of just,* the syntactic level of representation is not always suitable for expressing translation correspondences as simple as possible: in fact, one can argue that the representation chosen for *just* is more semantically than syntactically motivated. This raises the question of the suitable level of linguistic representation for expressing simple translation correspondences. This kind of example is not an isolated phenomenon, and for unrelated families of languages (e.g. English/Japanese), the problem becomes more severe. However, if one chooses a more abstract level of representation (e.g. a semantic level describing predicate-arguments relations), the problem is simplified and expressing correspondences between two languages belonging to unrelated families is not out of hand [Nagao & Tsujii 86, Zajac 89].

Modular descriptions vs. integrated descriptions

Using the co-description mechanism, pieces of information belonging to different strata of linguistic description are mixed together in a single place: c-structure, f-structure, and target f-structure. However, even if source and target grammars are independently motivated, it seems difficult to characterize the co-description approach as transfer based (as pointed out by [Kaplan et al. 89]), since the grammar of the source language and the contrastive descriptions are tightly integrated. In fact, there is only one grammar which merges what is traditionally divided into source language grammar and contrastive transfer grammar [Yngve 57], and there is only one lexicon, both for analysis and transfer. Such an integration makes

any change or adaptation of the grammar difficult. Furthermore, for changing the target language or adding another one, the only feasible solution would be to remove transfer information for the first language and extract the source grammar, and add co-descriptions for the second language. Thus, using the co-description approach, the modularity that could theoretically be achieved is lost in the actual implementation.

The TFS formalism encourages a modular approach to programming by providing means of specifying interface structures (in the sense used in object-oriented programming, i.e. specifications of what is the output of a module and the input of another - see for example [Vauquois 84]), and relations between these different structures. Assume that one would like to define

- a relation $\mathbf{G_e}$ between a set of lists of words W_e and a set of syntactic structures S_e for English : $\mathbf{G_e}$ is included in $W_e \times S_e$
- a relation $\mathbf{G_f}$ between the set S_f and a set of lists of words W_f for French : $\mathbf{G_f}$ is included in $S_f \times W_f$
- and a relation $\mathbf{T_{ef}}$ between the set of syntactic structures S_e and the set of target syntactic structures S_f: $\mathbf{T_{ef}}$ is included in $S_e \times S_f$
- and then a translation relation in $W_e \times S_e \times S_f \times W_f$ defined as the product of the three relations $\mathbf{G_e}$, $\mathbf{T_{ef}}$, and $\mathbf{G_f}$.

The sets S_e and S_f representing the valid structures of source and target language are defined in TFS as feature types. Once well-formedness conditions on these structures are defined, the relations $\mathbf{G_e}$, $\mathbf{T_{ef}}$, and $\mathbf{G_f}$ can be defined *independently of each other*. An element of the product of these relations is encoded in a single feature term with four features : `e-string` for the English string, `e-syn` for the English structure, `f-syn` for the French structure and `f-string` for the French string, corresponding to the four arguments of the relation in $W_e \times S_e \times S_f \times W_f$ respectively. If ENG defines the relation between the set of English strings and the set of English structures, FR the relation between the set of French strings and the set of French structures, and TAU the relation between the sets of English and French structures (that includes the lexicon, syntactic structures, semantic structures, etc.), we write ENG-FR = ENG & TAU & FR to define the translation system between English and French. A feature term like (14) is a member of this relation.

It is clear that for adding another language one would have to define the valid linguistic structures for that language and develop a grammar for it, and contrastive grammars (i.e. other TAU relations) independently from the rest. The same modular approach can also be used to develop a monolingual grammar, specifying well-formedness conditions for different levels of linguistic description (e.g. configurational, syntactic, semantic, etc.) and the relations between these levels.

```
ENG-FR[e-string: <the baby just fell>,                                    (14)
        e-syn: [pred: JUST,
                phon: <just>
                arg:  [pred:  FALL,
                       tense: PAST,
                       phon: <fell>,
                       subj:  [pred: BABY,
                               phon: <baby>
                               num:  SG,
                               spec: [def:+,
                                      pred: THE,
                                      phon: <the>]]]],
         f-syn: [pred:  VENIR-2,
                 tense: PRESENT,
                 phon: <vient>
                 subj: #f-subj=[pred: BEBE,
                                phon: <bébé>
                                num:  SG,
                                gen: MASC
                                spec: [pred: LE,
                                       def: +,
                                       phon: <le>]],
                 xcomp: [pred: TOMBER,
                         phon: <tomber>
                         compl: DE,
                        tense: INF,
                        subj: #f-subj]],
         f-string: <le bébé vient de tomber>]
```

Separate processing vs. integrated processing

The conventional view of transfer-based machine translation, as proposed by [Yngve 57], is to have three different processes communicating only through the data structure for translating from English to French:

$$W_e \rightarrow S_e \quad || \quad S_e \rightarrow S_f \quad || \quad S_f \rightarrow W_f$$

Using LFG co-descriptions which describe one-way correspondences from the source f-structure to the target f-structure (Φ and τ are functions [Kaplan 87]), the first two steps are integrated:

$$W_e \rightarrow S_e \rightarrow S_f \quad || \quad S_f \rightarrow W_f$$

As pointed out by [Kaplan et al. 89], one big advantage of the co-description mechanism is that all interactions between different strata (c-structure, f-structure and target f-structures), even those not described explicitly, come into play during parsing, where all equations are solved simultaneously. This allows one to reject a potential solution as soon as there is an incoherence stemming from any of these levels, for example an incoherence stemming from the transfer will be already detected during parsing. This advantage follows directly from the co-description approach, where pieces of information belonging to different strata of linguistic description are mixed together in a single place.

Using TFS, it is possible to integrate all components during processing gaining the computational advantages of the "co-descriptive" mechanism (all constraints stemming from
all levels are solved simultaneously) without loosing the benefits of modularity. When a relation is defined as a product of relations (like ENG-FR = ENG & TAU & FR in $W_e \times S_e \times S_f \times W_f$), all elementary relations are combined during computation, and applied *simultaneously onto the same structure.* This follows from the general evaluation strategy used by the interpreter of the formalism which uses an ordering of rewriting depending only on the form of the input and an abstract ordering on feature type definitions (a subsumption ordering). For example, starting from a string and computing the corresponding elements of ENG-FR, the interpreter will apply constraints from ENG, TAU and FR simultaneously, building up the whole structure (like e.g. (14)) by unifying subterms of the initial structure with the feature type definitions of ENG, TAU and FR.

Reversibility

Inverting a grammar is a nontrivial task. All the approaches that try to use the same grammar both for parsing and generation have two different compilers, one for generating code for a parser and the other for generating code for a generator. In the LFG framework, as in [Momma & Dörre 87] for example, there are LFG grammars that cannot be automatically

inverted, and additional constraints usually have to be hard-coded (in PROLOG) such that the generation process does not overgenerate [Kohl et al. 89]. Even in the most declarative approaches currently used, the linguist has to include special control annotations as in [Dymetman & Isabelle 88] or to specify a "leading attribute" (e.g. *string* for analysis and *semantic_head* for generation) as in [Shieber et al. 89], where these annotations and attributes are used as directives for compiling an analysis or a generation program.

The problem of inverting a whole machine translation system seems even more complex. [Dymetman & Isabelle 88] do not seem to address this problem. It is not possible to invert an LFG grammar that uses co-descriptions for specifying transfer, at least in the current state of the implementations; even if it were possible, it would require to add source co-descriptions to the target grammar and to the dictionary. The only work known to the author is [van Noord 89] who proposes a reversible architecture for MT based on ideas from [Shieber et al. 89] extended to transfer, where the transfer part is coded in the grammar in a way similar to LFG "co-descriptions". One could imagine that the reversible machine translation system would then be the chaining of three different programs, the parser, the transfer grammar and the generator, that would be produced by a compiler using different "leading attributes" to compile the correct PROLOG code for each of the programs.

Using TFS relations, there is no inverse to compute: the concept of inverting a one-way correspondence (i.e. computing the inverse of a function) is *not relevant* in a truly relational framework. Actually, the user does not have access to any control mechanism: there is nothing like control annotations or "leading attributes"; thus the linguist is freed from taking care of the control. The interpreter applies the *same evaluation mechanism using the same set of definitions whatever the input is,* and the actual order of computation of the relation depends only on the form of the input and on a subsumption ordering between feature type definitions: the compiled form of a grammar is exactly the same for any kind of input (and this also simplifies a lot of the system developer's task since there is only one compiler for the system!).

In fact, parsing and generation are special cases of the evaluation of an input structure, where the input structure given to the interpreter is either a string (for parsing) or a linguistic structure (for generation). The same applies for "inverting" a machine translation system: the input can be a string of one or the other of the languages. It is in fact possible to specify any of the elements of a relation and compute all the others which are compatible. For example,

starting from an element of S_f it is possible to compute the whole product thus generating the strings for both W_e and W_f. the formalism allows one to start evaluation from a partially specified input structure, thus generating all possible members of the relation which are compatible with the input. It can also be run for checking if two given strings are in the translation relation, for example.

It is clear that the behavior of the system is considerably different from traditional procedural systems, and even from PROLOG based systems, where the order of application of clauses is specified by the user: in TFS, the user does not have any control on the order of evaluation, and this order will depend on the input and on a subsumption ordering between type definitions. However, given the simple and intuitive semantics of the formalism [Emele & Zajac 90b], the behavior is still very simple to follow, as it is apparent when the grammar writer uses the tracing facilities to debug a grammar. We argue that the computational behavior of a TFS grammar is more understandable than the same grammar written in a procedural language, where unexpected behavior often results from the nonmonotonic kind of processing, essentially due to complex deletions, the transformation of structures and the ordering of those steps. The only source of bugs that is left is incoherent definitions, and this source exists in any kind of system.

As already noted by [Dymetman & Isabelle 88], to be able to reverse grammars is very useful for debugging, as some omissions which can go unnoticed for one way (usually analysis) become apparent running the grammar the other way (overgeneration). We hope that this feature will also prove useful for contrastive grammars.

Conclusion

We argue that using a general purpose constraint-based formalism that integrates concepts from logic programming, object-oriented programming, knowledge representation and functional unification grammars, it is possible to build a whole machine translation system that has interesting properties:

(1) The machine translation system is modular: the feature type system allows one to define relations between independently specified structures and also to define relations as a product of more elementary relations.

(2) The processing is integrated: all constraints from all sources are integrated during evaluation and apply simultaneously to determine a complete description of a solution.

(3) The translation system is defined as a relation between sets of strings (and sets of linguistic structures): one can compute any element of a relation starting from any other element. Thus "reversibility" is an inherent feature of the system.

Such an approach has also practical advantages: the linguist has to know only one formalism and one computer environment to write any of the modules; the programmers have to develop a system for only one formalism: that means only one compiler, one tracing facility, etc.

The TFS system has been implemented in Common-Lisp by Martin Emele and the author and has been tested on Symbolics, TI Explorer, VAX and Allegro Common-Lisp. Sample grammars are documented in [Emele 88, Zajac 89, Emele & Zajac 90a].

Acknowledgments. The TFS system implemented at the IMS is based on the experience gained in a previous implementation of a similar system carried out by Martin Emele and the author at ATR, Kyoto, as a part of a visiting research program [Emele & Zajac 89]. We would like to thank Dr. Akira Kurematsu, president of ATR Interpreting Telephony Research Laboratories, for making our stay possible, and Mr. Teruaki Aizawa, head of the Natural Language Understanding Department, for his constant support. This paper has benefited from many comments from our colleagues at the IMS, University of Stuttgart.

References

Hassan Aït-Kaci. 1984. *A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures.* Ph.D. Dissertation, University of Pennsylvania.

Hassan Aït-Kaci. 1986. An Algebraic Semantics Approach to the Effective Resolution of Type Equations. *Theoretical Computer Science* 45, 293-351.

Marc Dymetman and Pierre Isabelle. 1988. Reversible logic grammars for machine translation. *2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages.*

Martin Emele. 1988. A typed feature structure based approach to generation. *Proceedings of the WGNLC of the IECE,* Oita University, Japan.

Martin Emele and Rémi Zajac. 1989. *Multiple Inheritance in RETIF*. ATR Technical Report TR-I-0114, ATR Interpreting Telephony Laboratories, Kyoto.

Martin Emele and Rémi Zajac. 1990a. Typed Unification Grammars. *23th International Conference on Computational Linguistics, COLING-90,* Helsinki.

Martin Emele and Rémi Zajac. 1990b. A Fixed-Point Semantics for Feature Type Systems. *2nd International Workshop on Conditional and Typed Rewriting Systems, CTRS-90,* Montreal.

Per-Kristian Halvorsen and Ronald Kaplan. 1988. Projections and semantic description. *International Conference on Fifth Generation Computer Systems,* Tokyo.

Pierre Isabelle and Elliot Macklovitch. 1986. Transfer and MT Modularity. *11th International Conference on Computational Linguistics, COLJNG-86,* Bonn.

Ronald M. Kaplan. 1987. Three seductions of computational psycholinguistics. In Pete Whitelock et al. (eds.). *Linguistic theory and computer applications.* Academic Press, London.

Ronald M. Kaplan and Joan Bresnan. 1982. Lexical Functional Grammar: a formal system for grammatical representation. In Joan Bresnan (ed.), *The mental representation of grammatical relations.* MIT Press.

Ronald M. Kaplan, Klaus Netter, Jürgen Wedekind and Annie Zaenen. 1989. Translations by structural correspondences. *4th European ACL Conference,* Manchester.

Martin Kay. 1984. Functional Unification Grammars: a formalism for machine translation. *10th International Conference on Computational Linguistics, COLING-84,* Stanford.

Dieter Kohl, Agnes Plainfossé, Mike Reape and Claire Gardent. 1989. *Text generation from semantic representation.* ESPRIT Project 393 ACORD, ACORD Deliverable T2.10.

Stephan Momma and Jochen Dörre. 1987. Generation from f-structures. In Ewan Klein and Johan van Benthem (eds.), *Categories, Polymorphism and Unification,* Centre for Cognitive Science, University of Edinburgh.

Ikuo Kudo and Hirosato Nomura. 1986. Lexical-Functional Transfer: A Transfer Framework in a Machine Translation System based on LFG. *11th International Conference on Computational Linguistics, COLING-86,* Bonn.

Makoto Nagao and Jun-ichi Tsujii. 1986. The transfer phase of the MU machine translation project. *11th International Conference on Computational Linguistics, COLING-86,* Bonn.

Klaus Netter and Jürgen Wedekind. 1986. An LFG-based approach to machine translation. *Proc. of the IAI-MT86,* Saarbrücken.

Gertjan van Noord. 1989. *Reversible unification based machine translation.* Submitted to COLING-90.

C.J. Rupp. 1989. Situation semantics and machine translation. *4th European ACL Conference,* Manchester.

Stuart S. Shieber, Gertjan van Noord, Robert Moore and Fernando C.N. Pereira. 1989. A semantic-head-driven generation algorithm for unification-based formalisms. *27th Annual meeting of the ACL,* Vancouver.

Bernard Vauquois. 1984. *The organization of an automated translation system for multilingual translation at GETA.* IBM Europe Institute on Natural Language Processing, Davos.

Victor Yngve. 1957. A framework for syntactic translation. *Mechanical Translation,* 4/3.

Rémi Zajac. 1989. A transfer model using a typed feature structure rewriting system with inheritance. *27th Annual meeting of the ACL,* Vancouver.