INTERFACE. Journal of Applied Linguistics 3.1 (1988), 5-32

EUROTRA: AN OVERVIEW

Anthony RAW, Bart VANDECAPELLE & Frank VAN EYNDE
Eurotra - Leuven; Katholieke Universiteit Leuven

ABSTRACT

Eurotra is a project for machine translation which is sponsored and organised by the Commission of the European Communities. It has two aims: the development of a system for machine translation for the nine official EC languages and the spreading of expertise, research and education in the fields of machine (aided) translation, natural language processing and computational linguistics in the twelve EC Member States. This article first explains why the EC set up this project and situates the Eurotra effort in a world-wide context. It briefly discusses the internal organisation and the planning of the whole project and then concentrates on the two main parts of the Eurotra system: the formalism and mechanisms which underly the software and the linguistic theory. In an annex we also provide a concrete example: the translation of the sentence "The commission has sent the proposal to the Council" into its Dutch equivalent "De commissie heeft het voorstel naar de Raad gestuurd".

1. THE NEED FOR MT IN THE EC

From its inception, the European Community has adhered to the principle of equal treatment of all Community languages. Initially there were four such languages: Dutch, French, German and Italian. Due to the successive enlargements of the Community their number has grown to nine: the four original ones plus Danish, English, Greek, Portuguese, and Spanish. As a consequence, the number of language pairs has increased from twelve (4×3) to seventy two (9×8).

The translation needs within the EC institutions are hence large and growing, and in order to keep up with the demand the EC has created the largest translation and interpretation services in the world. Unfortunately, even these can only provide a limited service and due to a lack of resources many documents are not translated into all EC languages. This often leads to delays in the enforcement of political measures since the latter become only effective after they have been translated and published in all languages in the Official Journal.

In short there is a mismatch between the demand for translation and the resources – both human and financial – which one can afford to spend on translation.

An easy way to resolve this problem would be to reduce the number of official EC languages (to Dutch and Danish, for instance), but this is unacceptable from a political point of view. An alternative solution would be to enhance the productivity of the translation services by providing them with more powerful tools, such as electronic dictionaries, terminological data bases, text formatting devices and, ideally, machine translation systems. From a technical point of view there have been doubts about the feasibility of this alternative – especially about the feasibility of machine translation – but since various research and development projects in the USA, Canada and Western Europe had already come up with results in the domain of machine aided translation, it was decided to explore this possibility.

The first step was the acquisition in 1976 of the MT system Systran. Systran had been developed in La Jolla (California) for the language pairs Russian → English and English → French, and the initial aim of the EC was to extend the number of language pairs and to enlarge the dictionaries, so that it would become a useful tool for its translation services. In practice these extensions turned out to be far more problematic than foreseen, and this, together with the awareness that in the EC itself some research centers were developing MT systems of a more advanced design (especially Grenoble and Saarbrücken), led the Commission in 1978 to the conclusion that the time was ripe for launching a European R&D project in MT. The project was given the name Eurotra and for its preparation a group was formed of representatives of some thirty European universities and research centers; the Eurotra Coordination Group.

It took this group and the Commission services four years to get the approval of the European Parliament for an R&D programme for the creation of a prototype translation system for all Community languages. It was further stipulated that the system should be of advanced design, that it should be geared to the translation of official EC documents regarding information technology, and that the dictionaries should contain approximately 20.000 entries (for comparison, the Dutch Van Dale dictionary contains ± 200.000 entries).

A second but equally important aim of the Eurotra programme is the encouragement of research and education in the areas of computational linguistics, natural language processing and machine translation in the EC countries.

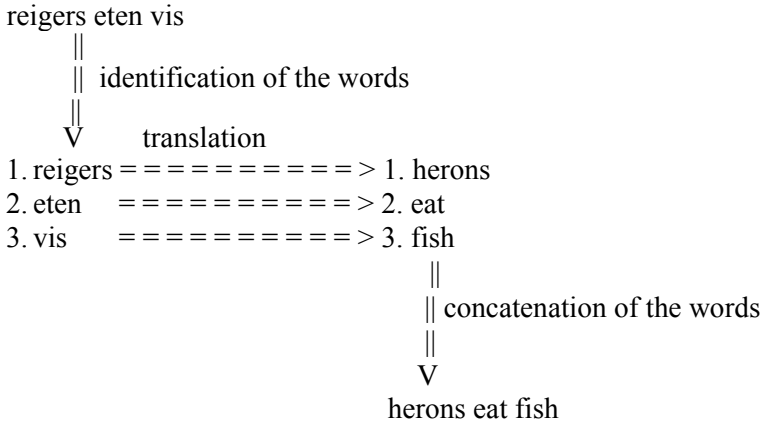
2. THE PLACE OF EUROTRA WITHIN MT

2.1. Design of an MT system

The first attempts on this planet to build a system for machine translation are from the fifties. The design of the early systems was very simple: they identified the words in the source language text and mapped them one-by-one onto their target language equivalents. The Dutch sentence

(1) reigers eten vis

for instance, would be mapped onto its English equivalent in the following way:



It goes without saying that this approach is very limited. To illustrate the insufficiency of this word-for-word translation model just consider the French equivalent of (1), which is not something like "hérons mangent poisson" but rather

(2) les hérons mangent du poisson

Somewhere in the translation process the words "les" and "du" have to be added, but it is not clear where or how or why.

Another example concerns word order. In English, the word order in subordinate clauses is basically the same as in main clauses, but in Dutch there is a difference: in main clauses as in (1) the verb is in second position (after the subject or another constituent) whereas in subordinate clauses it is in final position. As a consequence, the word order has to be changed when one translates Dutch subordinate clauses into English ones. An example:

(3) het schijnt dat reigers vis eten

(4) it seems that herons eat fish

* it seems that herons fish eat

Of course it is easy to state a rule that two words have to be permuted, but if one wants this rule to apply only when it is necessary and to come up with the right results in all cases, then one has to rely on information which the words themselves do not contain: e.g. the information that a given clause is a main clause or a subordinate clause and the information that "eten" in (3) is a verb and not a noun.

In order to overcome problems like these the MT researchers started to integrate morphological and syntactic modules in their systems. Instead of simply identifying the words of the source language text and concatenating their target language equivalents (with possible insertion, deletion or permutation) they developed a substantially different design, which they called the *transfer* model. The structure of a transfer system appears as follows :

source language sentence
||
|| morpho-syntactic analysis
V
morpho-syntactic representation of the source language sentence
||
|| transfer (lexical and structural)
V
morpho-syntactic representation of the corresponding target language sentence
||
|| synthesis
V
the target language sentence

The source language sentence is analysed morphologically and syntactically and the result of this analysis is stored in a morpho-syntactic representation. The latter is then mapped onto a similar representation of the target language sentence in transfer and finally there is the synthesis step which maps the target language representation onto a target language sentence (this last step is sometimes called "generation").

The unit of translation in this approach is no longer the word but the analysed sentence, i.e. the sentence plus its morpho-syntactic structure. Different transfer systems take different views on

— the division of labour between analysis/transfer/synthesis
— the nature of the morpho-syntactic representations (dependency graphs, augmented phrase structures, lexical-functional structures, etc.)
— the internal structure of the analysis and synthesis components (directly from sentence to structure or via intermediate representations)

but the basic design of all transfer systems is the same. Their main merit is that the addition of the morpho-syntactic information to the sentences makes it possible to formulate general rules for the manipulation of structures in transfer (cf. changes of word order).

A third type of MT systems are the so-called *interlingual* systems. They do not assign morpho-syntactic structures to sentences but rather universal (semantic) representations which do not have to undergo any lexical or structural transfer. As a consequence, these systems only contain an analysis and a synthesis module. The transfer module is superfluous because of the highly abstract nature of the representations.

Eurotra has chosen the transfer approach and adheres to the following general principles:

— analysis and synthesis are strictly monolingual, i.e. there is one analysis module and one synthesis module per language (and not per language pair)
— transfer is bilingual, i.e. there is one transfer module per language pair, and it is kept as simple as possible : the aim is to limit the transfer operations to the replacement of lexical units (lexical transfer)
— the representations which function as output of analysis and as input of synthesis are alled interface structures; they take the form of depen-

dency structures which are enriched with various types of morpho-syntactic and semantic information

— the mapping of sentences onto interface structures (and vice versa) is not one-shot, but is performed via a number of intermediate representations.

All of these properties are further explained in section 5.

2.2. Implementation of an MT system

Apart from the basic design of the translation process there is another important parameter with respect to which MT systems can be classified. This concerns the computational aspects, more specifically the formal tools which are used for performing the linguistic and translational operations.

The first MT systems made use of the then existing programming languages like Fortran, Cobol, Algol or even Assembler. The problem with these languages, however, is that they are geared to mathematical applications and that their format is not particularly adequate for the formulation of linguistic rules and translational operations.

More adequate are some of the programming languages which were developed in the early seventies, like Pascal and Prolog, and which are -- together with Lisp – by far the most popular languages in natural language processing at the moment.

Even more popular, however, is the definition of special purpose tools for linguistic applications. Instead of making use of any particular general purpose programming language for writing the grammar rules, the dictionaries and the transformations, one develops formalisms which are especially made for doing dictionary coding or grammar writing or any other kind of translational operation. These formalisms are not identical to any existing programming language and the linguists who use them do not write their grammars and dictionaries in Prolog or Pascal or Lisp, but in terms of the formalism.

Eurotra has chosen for the latter option and the formal tools and the formalism itself will be described in section 4.

3. ORGANISATION AND PLANNING

One of the characteristic features of the Eurotra project is its highly decentralised organisation. Apart from the general management, administration and coordination, which rest with the Commission of the European Communities – more specifically with the Directorate-General for Telecommunications, Information Industries and Innovation – all tasks are performed by the Eurotra research units in the twelve member states.

There are two reasons for this *decentralisation:* primo, the lack of a special EC research centre for natural language processing and, secundo, the explicit wish to spread the expertise over the different member states.

The consequence of this policy is that also tasks of central importance, such as the definition of the linguistic theory and the development of the core formalism, are carried out by teams whose members are spread all over

Europe. At the moment there are three such teams: the linguistic research group, the framework group and the dictionary task force. They provide the tools, the legislation and the guidelines for the nine language groups.

The main task of a language group is to develop modules for the analysis and generation of its own language plus the modules for transfer from the eight other languages. The following is a list of the nine EC languages with a reference to the corresponding research unit(s):

Dansk
- Københavns Universitet (Danmark)

Nederlands
- Rijksuniversiteit Utrecht (Nederland)
- Katholieke Universiteit Leuven (Belgie)

English
- UMIST (Manchester, United Kingdom)
- University of Essex (Colchester, United Kingdom)

Français
- Université de Nancy II (France)
- GETA (Grenoble, France)
- Université de Paris VII (France)
- Université de Liège (Belgique)

Deutsch
- Institut für Angewandte Informationsforschung (Saarbrücken, BRD)

Greek
- Eurotra Greece (Athens, Greece)
- University of Rethymnon (Creta, Greece)

Italiano
- Gruppo Dima (Torino, Italia)
- Universita di Pisa (Italia)

Português
- Universidade de Lisboa (Portugal)

Español
- Eurotra España (Barcelona, España)
- Universidad Autonoma de Madrid (España)

Ireland and Luxembourg, the two remaining member states, have been assigned tasks of a general nature. Ireland monitors the work on terminology and Luxembourg functions as a documentation centre and a software clearing house.

All management decisions are taken by the Liaison Group, which consists of representatives of the different research units (one voting member per member state) and the head of the project. This group meets once per month in Luxembourg. It is mainly responsible for management, coordination and planning.

On a very general level the planning distinguishes *three phases :* the first phase was used for setting up the research units in the various member states and for making them operational. In the second phase the implementation work started and by June 1988 when this phase finished all language groups had developed analysis and synthesis modules for their languages containing ± 2.500 lexical entries and covering a restricted set of syntactic con-

structions. Next to these monolingual modules the language groups also developed modules for transfer into their own language.

The third phase which started in July 1988 and which will finish in July 1990 will be used for extending the syntactic and the lexical coverage of the system: from 2.500 to 20.000 entries and from a restricted set of syntactic constructions to a much larger set.

By mid 1990 the project will have produced a prototype system which will be further developed on an industrial basis. The existing research units in the member states will then go on with linguistic research and development for other EC projects.

4. THE EUROTRA FRAMEWORK: FORMALISM AND MECHANICS

The Eurotra framework is a model of translation which provides a linguist with the *concepts* and *tools* with which he is able to describe (for all official languages of the EC):

— the analysis of a language from surface text to some abstract representation termed an interface structure
— the transfer of information between the interface structures of different languages
— the synthesis of a language from an interface structure to surface text.

That is, the 'analysis/transfer/synthesis' model of translation.

The concepts which the framework provides comprise the linguistic theory of Eurotra which is the topic of the next section. The tools provided by the framework to express this linguistic theory will be discussed in this section.

The two most interesting 'meta-components' of this set of tools are :

— a formalism in which a linguist is able to describe a language
— a mechanism for applying these rules to text and to abstract representations of text.

We call the first of these the *user-language* (where the 'user' is a linguist attempting to describe a language), and the second the *virtual machine.*

Naturally there are other components in the set of tools, both internal and external to the Eurotra framework, which, although not topics of this article, should be mentioned. Internal to the framework is a compiler to translate rules written in the user-language into programs in the language of the virtual machine. External to the framework there is a compiler to translate virtual machine programs into the language of a machine, i.e. machine code. Lastly, of course, there is the machine itself.

The description that a linguist makes of a language is actually a series of descriptions of different abstract levels of representation. It is therefore not only necessary to describe an abstract representation but to also define the relationship between two adjacent representations. We call these abstract levels *representation levels.* The descriptions that define representation levels are called generators. The relations between adjacent representation

levels are defined in terms of the relations between the generators of those levels and are called *translators.* The remainder of this section will examine these three concepts in some detail with special regard to the formalism and mechanics of the framework.

4.1. Representation Levels

A main principle of the Eurotra framework is that both the analysis of text to an interface structure and the synthesis of text from an interface structure are performed as a series of steps between intermediate levels of representation. For example, in analysis (where *i* designates intermediate representation levels and *n* designates a particular level, the interface structure) :

$$Text \rightarrow i_1 \rightarrow i_2 \rightarrow ... \rightarrow i_n$$

The purpose of this stratification is that, because we want the gaps between interface structures of different languages to be as small as possible (the notion of *simple transfer),* the gaps between text and interface structures become quite large. Thus it becomes an extremely complex task to relate text to interface structure in a single operation. By decomposing analysis and synthesis into a series of primitive translations between intermediate levels of representation the task becomes much more manageable. (Note that from now on, unless explicitly stated otherwise, when we refer to *translation* we mean translation between levels of representation and not direct text-to-text translation).

This model can be extended further because the transfer component between the interface structures of different languages can also be considered as a translation between two representation levels. Thus, the whole translation process can be illustrated (where *s* designates the source language, *t* the target language, and -> designates the translation between levels) as :

$$Text^s \rightarrow i_1^s \rightarrow i_2^s \rightarrow ... \rightarrow i_n^s \rightarrow i_n^t \rightarrow ... \rightarrow i_2^t \rightarrow i_1^t \rightarrow Text^t$$

In section 5 we will see what linguistic information is actually represented by these intermediate levels. In section 4.2 we will examine how generators define these representation levels and in section 4.3 how translators define the relations between adjacent levels.

First, however, let us consider the notion that each representation level is a formal language. Such a language is a set of *objects* which are either
— primitive objects
— structures built from primitive objects.

In the Eurotra framework, primitive objects are termed *feature bundles* which consist of sets of features, each feature in turn being built out of an *attribute* and a value where the value of the attribute may be a variable.

For example,

lex is an attribute

eat is a value

lex=eat is a feature

{lex=eat, lu=eat, cat=v, nb=X, tense=pres} is a feature bundle

The set of legal primitive objects which may exist within a level of representation is defined by the *feature theory* of that level. A feature theory of a representation level consists of:

— a set of *feature declarations* which define which attribute=value pairs constitute legal features

— a set of co-occurrence *restrictions* which define which sets of features constitute well-formed feature bundles.

Structured objects are trees of feature bundles and can be represented, for example as:

{cat=s}
    {cat=np,nb=plur}
        {lex=herons,lu=heron,cat=n,nb=plur}
    {cat=vp,nb=plur}
        {lex=eat,lu=eat,cat=v,nb=plur,tense=pres}
        {cat=np,nb=X
            {lex=fish,lu=fish,cat=n,nb=X}

The reason that trees are such good mediums for representing linguistically relevant information is that they describe two important relations between the nodes of a tree :

— the *dominance* (mother/daughter) relation, e.g. that the *{cat=s}* node dominates the *{cat=np,nb=plur}* node and the *{cat= vp,nb=plur}* node

— the *precedence* (sister/sister) relation, e.g. that the *{cat=np,nb=plur}* node precedes the *{cat=vp,nb=plur}* node.

The set of well-formed structured objects which may exist within a level of representation is defined by a set of rules. These rules, together with the feature theory to define the set of legal primitive objects of a level, comprise the generator for that level.

Before examining generators in more detail, we must first make one more distinction with regard to objects: the distinction between consolidated objects and unconsolidated objects. A *consolidated object* is one which has been proven to be well-formed for a given representation level. That is, its features are proven as being legal with regard to the feature theory of the generator of that level, and its structure is proven as being legal with regard to the rules of the generator. An *unconsolidated object,* on the other hand, is more of a hypothesis - its features and/or its structure have not yet been proven as being well formed with regard to the feature theory and/or the rules of the generator for a given level.

4.2. Generators

4.2.1. *Consolidation*

The input to a generator is a series of unconsolidated objects supplied by a translator. The objects represent an entire tree structure and are fed to the generator bottom up. That is, first leaf nodes (feature bundles dominating an empty sub-tree) are provided, then the sub-trees which have those leaves as daughters, then sub-trees which have the previous sub-tree heads as daughters and so on until the root of the entire tree has been supplied.

For each object that the generator receives, it attempts to *consolidate* both the contents of the feature bundle nodes within the object and the dominance and precedence relations between nodes. This consolidation is performed by checking and possibly modifying feature bundle nodes and the relations between them with regard to the rules and feature theory of the generator.

The mechanism for applying generator rules to objects is performed by controlled unification which 'matches' objects and rules, the control being provided by a variant of the Earley parsing algorithm – this will be exemplified in section 4.2.3. Because rules may contain variables as well as values an important consequence of rule/object matching by unification is that these variables may become instantiated. This is the mechanism by which feature agreement and percolation in generator rules is propagated.

The output from a generator is one or more representations of the entire input tree structure with all nodes and relations between nodes consolidated. If the generator produces no output then consolidation has failed.

4.2.2. *Generator Rules*

There are three basic rule types in a generator – structure building rules, feature rules and filter rules – and all of them have the same basic shape:

fbd_0 [arg_1, arg_2 ..., arg_n]

where the head of the rule is a feature bundle description (these are the descriptions in rules of feature bundles in objects) and each argument in the body of the rule is either a feature bundle description or itself a head with its own arguments (recursive) depending upon the specific rule-type. The head of the rule has immediate dominance over the sequence of arguments in the body of the rule which are ordered according to the precedence relation.

Various operators are allowed over the arguments in the body of generator rules to allow concise rule writing. These operators include:

— the iteration marker (Kleene star *), i.e. **arg* states 0,1 or more occurrences of *arg*
— the optionality marker (^), i.e. *âarg* states 0 or 1 occurrences of *arg*
— the alternation marker (;), i.e. (*arg*$_1$; *arg*$_2$) states either *arg*$_1$ or *arg*$_2$
— the insertion marker (!), i.e. !*arg* states insert *arg* if it is not already present.

Structure building rules are the main rule type in a generator, the first to apply to input objects, and the controllers of the process of consolidation. They also provide the main description of all well-formed objects for their levels of representation.

Structure building rules consolidate the structure of objects in slightly different ways depending upon the objects to which they are applied:

— if the head and the body of a rule unify directly with the mother and daughters of an object then all relations are consolidated immediately (straightforward unification)
— if the body of a rule unifies with a sequence of nodes in an object then the head of the rule is inserted vertically as a new mother (bottom-up parsing)
— if the head of a rule unifies with a node in an object which has no daughters then the body of the rule is inserted vertically as a new sequence of daughters (downward expansion)
— if a feature bundle description in the body of a rule is marked for insertion and it does not appear as a node in an object then it will be inserted horizontally (insertion)

Feature rules cannot modify the structure of objects but may alter the information contained within feature bundle nodes to consolidate the nodes themselves. They are useful for stating generalisations regarding feature percolation and agreement that structure building rules cannot make.

The arguments in the head and body of a feature rule contain two parts: a context part and an action part. Feature rules are applied to objects to objects if the structure of a rule matches the structure of an object and the context part is satisfied. If both conditions are met the action part is performed. The action is to either add values (instantiate variables), change values, delete values, or any combination of these. A special type of feature rule, the *lexical feature rule,* is used to add dictionary information to the feature bundle nodes which are the leaves of input objects.

Filter rules can modify neither the structure of objects nor the contents of nodes within objects. They are used to check the well-formedness of objects and any object that is deemed ill-formed is deleted. The main purpose of filter rules is to filter out any exceptional objects created by possible over-generalisations in structure building rules and in feature rules. *Strict filter rules,* like feature rules, contain context and action parts. If the structure of a rule matches the structure of an object and the context part is satisfied then the rule is applicable. Subsequently the action part must also be satisfied for the object to survive – otherwise it is considered ill-formed and is deleted. *Killer filter rules* contain context parts only – the action is always deletion of the object. If the structure of a rule matches the structure of an object and the context part is satisfied then the object is considered ill-formed and is deleted.

4.2.3. *Example of Generator Rule Application*

As a very simple example of the application of generator rules to objects consider the following unconsolidated object:

{*cat=s*} < {*lex=herons*},{*lex=eat*},{*lex=fish*} >

which is the hypothesis that the node {*cat=s*} dominates the nodes between angle brackets and that these nodes have the correct precedence relation. That is, 'Is the string ,"herons eat fish" a sentence and, if so, what are its characteristics ?'. The following 'trace' shows how this object is consolidated via parsing by unification. (Feature declarations and co-occurrence restrictions are omitted for the sake of brevity).

The generator receives this object bottom-up, so it first tries to consolidate the leaves of the object. This takes the form of applying lexical feature rules to each node (i.e. 'dictionary lookup') and may well result in a series of consolidated nodes of the form :

{*lex=herons,lu=heron,cat=n,nb=plur*}
{*lex=eat,lu=eat,cat=v,nb=X,tense=pres*}
{*lex=fish,lu=fish,cat=v,nb=X,tense=pres*}
{*lex=fish,lu=fish,cat=n,nb=X*}

Note that some of the values for *nb* are uninstantiated variables as their values are ambiguous. Also note that the consolidation of "fish" has resulted in two possibilities.

Although the feature bundles themselves can now be considered consolidated, the relations between each of the nodes and their mother are still weak. This structural consolidation of dominance and precedence is the next step. Assume a structure building rule of the form:

{*cat=np,nb=X*}
[{*cat=n,nb=X*}]

which states that the feature bundle description {*cat=np,nb=X*} immediately dominates the single feature bundle description {*cat=n,nb=X*}. The consolidated leaves are processed from left to right, and this rule unifies with the first of them resulting in the structure:

{*cat=np,nb =plur*}
{*lex=herons,lu=heron,cat=n,nb=plur*}

Note that the variable for *nb* has become instantiated by unification and has percolated up to the mother node. Note also that the features not mentioned in the rule (i.e. *lex* and *lu)* are kept – features are not deleted from objects unless deletion is explicitly stated in rules.

Now assume a structure building rule of the form:

{*cat=vp,nb=X*}
[{*cat=v,nb=X*},
{*cat=np*}

The second of the consolidated leaves unifies with the first argument in the body of the rule resulting in the structure:

{*cat=vp,nb=X*}
{*lex=eat,lu=eat,cat=v,nb=X,tense=pres*}
{*cat=np*}

This time the value for *nb* is not instantiated as it is still ambiguous. We are now looking for an object to unify with the second argument of this rule (i.e. {*cat=np*}). The first of our rules unifies with only one of our readings for "fish", resulting in the structure :

{*cat=np,nb=X*}
    {*lex=fish,lu=fish,cat=n,nb=X*}

which then completes the former rule resulting in the structure :

{*cat=vp,nb=Xl*}
    {*lex=eat, lu=eat,cat=v,nb=Xl*}
    {*cat=np,nb=X2*}
        {*lex=fish,lu=fish,cat=n,nb=X2*}

Note that no relationship is stated between the values of *nb* for some parts of the rule, between, for example, {*cat=vp,nb=Xl*} and {*cat=np,nb=X2*}.

Finally, assume a structure building rule of the form :

{*cat=s*}
    [{*cat=np,nb=X*},
     {*cat=vp,nb=X*}
    ]

This rule unifies with the two structures we have created so far resulting in the final structure:

{*cat=s*}
    {*cat=np,nb=plur*}
        {*lex=herons,lu=heron,cat=n,nb=plur*}
    {*cat=vp,nb=plur*}
        {*lex=eat,lu=eat,cat=v,nb=plur,tense=pres*}
        {*cat=np,nb=X*}
            {*lex=fish,lu=fish,cat=n,nb=X*}

Note that, wherever possible, percolation of values has occurred and that the number agreement restriction specified in the body of the {*cat=s*} rule has been satisfied.

This resulting structure is a fully consolidated object and the output from our simple generator. All feature bundle nodes have been proven as being well-formed and the structure of the input object has been modified to produce consolidated dominance and precedence relations. That is, we have proved the initial hypothesis that {*cat=s*} dominates the string of leaves and have produced a fully consolidated representation of the result.

Note that the example and the rules are oversimplified. To describe even a small fragment of English requires a much larger set of more complex rules. The example should suffice, however, to give some insight into the workings of unification and bottom-up parsing.

4.3. Translators

4.3.1. *Translation*

Translators are simple devices performing the minimum amount of tasks and leaving the bulk of the work to the generators. They are "one-shot" devices in that the output of a source generator becomes the input to a target generator after some modification, but without creating any intermediate representations within the translator.

The input to a translator is a single representation which is a fully consolidated object created by its source generator. The translator processes the representation top-down from the root node to the leaves creating a series of unconsolidated sub-objects which are immediately passed on to its target generator. Thus a basic principle of the concept of translators is compositionality – the translation of an object is equivalent to the translation of its parts.

A translator is defined by three components: a feature theory, a default translation mechanism, and a set of user-defined translator rules.

The feature theory of a translator defines the set of basic units of data over which the translator can operate. It is simply defined as the intersection of the feature theories of its source and target generators. That is, the feature declarations and co-occurrence restrictions that exist in both levels.

Built in to the system is a mechanism for the default translation of objects from source to target generators. The mechanism will fire unless overridden by explicit user-defined translator rules. The structure of objects is translated by copying consolidated dominance and precedence relations between feature bundle nodes at the source level into unconsolidated relations at the target level. That is, the relations are maintained but 'weakened' so that they are subject to possible modification by the target generator. Similarly, the features contained in the nodes of objects are translated by copying consolidated feature bundle nodes at the source level into unconsolidated nodes at the target level (provided that those features belong to the feature theory of the translator).

4.3.2. *Translator Rules*

A user is able to define two types of translator rules: structure translator rules and feature translator rules.

Structure translator rules, if applicable, override structure translation by the default mechanism. Structure translator rules are the main rule type in a translator, the first to apply to input objects and, together with the default mechanism, the controllers of the process of translation. The rules define the translation of consolidated dominance and precedence relations between feature bundle nodes of objects from the source level into unconsolidated relations at the target level.

The rules contain three elements: a left hand side (lhs), the translation operator ($\Rightarrow$), and a right hand side (rhs). The basic shape of the lhs of a structure translator rule is of the form:

$0:$fbd_0$ [$1:$arg_1$, $2: arg_2 ..., $n: arg_n]

where *$i* are indices, the head of the rule is a feature bundle description and each argument in the body of the rule is either a feature bundle description or itself a head with its own arguments (recursive).

The rhs of a structure translator rules is of the form :

*$*0 < *$*1, *$*2,. . ., *$*n >

where *$i* are the lhs indices specifying the new soft dominance and precedence relations between nodes. The lhs is a pattern to be matched against a source level object and the rhs specifies what unconsolidated object should be created on the basis of a translation of the lhs.

Structure translator rules perform several operations by altering the position of indices on the rhs of rules. These operations may have the effects of:

— modifying dominance relations between nodes
— modifying precedence relations between nodes
— removing the precedence relation between nodes (i.e. introducing an unordered set)
— deleting nodes

Some semi-formal examples of the above :

$0:$fbd_0$ [$1:$fbd_1$, $2:$fbd_2$ [$3: fbd_3]] ⇒ $0 < $1,$2, $3 >
$0:$fbd_0$ [$1:$fbd_1$, $2:$fbd_2$, $3: fbd_3] ⇒ $0 < $2, $1,$3 >
$0:$fbd_0$ [$1:$fbd_1$, $2:$fbd_2$,$3: fbd_3] ⇒ $0 < $1, ($2, $3) >
$0:$fbd_0$ [$1:$fbd_1$ ~:fbd_2, $3: fbd_3] ⇒ $0 < $1, $3) >

Note the use of the set operator (parentheses) in the rhs of the third example and the deletion marker (~) in the lhs of the fourth example.

Feature translator rules, if applicable, override feature translation by the default mechanism. The rules define the translation of consolidated features contained in feature bundle nodes from the source level into unconsolidated features at the target level. The rules have the same basic form as structure translator rules. That is, a lhs, the ⇒ operator, and a rhs:

fbd_0 [arg_1 arg_2 .... arg_n] ⇒ fbd_0

The body of the lhs of a rule is usually empty. The rhs of the rule *(fbd_0)* is a translation of the feature bundle node matched by the head of the lhs of the rule (fbd_0). If structure is stated in the body of the lhs of the rule then the structure serves only as a context for the application of the rule and none of its arguments are actually affected. That is, the lhs of the rule is a pattern to be matched against a source level object and the rhs specifies what unconsolidated object should be created on the basis of a translation of the head of the lhs.

The type of operations that feature translator rules perform are changing the value of features, introducing new features and deleting features, all in the context of the pattern specified by the lhs of the rule. Feature translator rules are also able to state generalisations regarding feature translation that structure translator rules are unable to make.

4.3.3. *Example of Translator Rule Application*

As a simple example of the translation process, we can examine how the object created by the simple generator in section 4.2.3. might be translated to the next level of representation. Assume the structure translator rule:

*$*0:{*cat=s*}
[*$*1:{*cat=np*}
~:{*cat=vp*}
[*$*2:{*cat=v*},
$3:{cat=np}
]
]
⇒ *$*0 < *$*2:{*frame=subj_obj*}, *$*1, *$*3 >

which will match with our consolidated object and create a rhs with altered dominance and precedence relations (the {*cat=vp*} node is deleted and the {*cat=v*} node is 'raised', moved, and given some extra information regarding the *frame* that it expects). Features will be translated by the default mechanism, with the result that the unconsolidated object created by the translator for input to the target generator will look something like:

{*cat=s*}
{*lu=eat,cat=v,nb=plur,tense=pres,frame=subj_obj*}
{*cat=np,nb =plur*}
{*lu=heron,cat=n,nb=plur*}
{*cat=np,nb=nb=X*}
{*lu=fish,cat=n,nb=X*}

Note that some information (in this case the feature *lex)* has been deleted by an explicit feature translator rule (not shown) as this information is not considered to be relevant for higher levels of representation.

The target generator will then have the task of consolidating this object by unification with a new set of generator rules which define the linguistic theory of the next level of representation (the introduction of the *frame* feature suggests that the next level will be concerned with syntactic dependency, i.e. finding the *subject* and *object* of the governing verb).

4.4. Software Implementation

The Eurotra Translation System (ETS) is still very much in a prototypical stage and will remain so until "industrialisation" of the software during the third phase of the project. Thus the project has seen, and will continue to see, quite a number of different software implementations over the years.

The following is a description of the current software, which includes an implementation of the virtual machine as described in the previous sections and of a supporting software environment.

4.4.1. *Implementation of the Virtual Machine*

The generator and translator components, i.e. the "core" of the system, are written in the programming language Prolog. Despite the relative inefficiency of the language in terms of time and space, it does offer several advantages.

— the virtual machine is a unification-based machine and Prolog's built in mechanism of unification offers an ideal environment
— the virtual machine is a non-deterministic machine (it always computes all possible alternatives) and Prolog's built in backtracking mechanism is a good means of achieving this sort of non-determinism
— due to the ease of program development (rapid prototyping) within Prolog, modifications to the Eurotra framework can be quickly implemented.

It is envisaged that the core of the prototype will continue to be written in Prolog for the foreseeable future, although there may well be a transition from the currently used C-Prolog to a faster compiled Prolog known as Yap whose developers are working in close collaboration with the software team in Eurotra.

4.4.2. *Software Environment*

Surrounding the core, but still written in Prolog, are a number of tools to aid linguists in writing correct generator and translator rules. These include:

— a debugging mechanism to trace the application of rules and their effects
— a pretty-printer to display objects in various formats
— a command interpreter to manipulate objects.

Rules are written in a formalism (i.e. the user language) which is not the language of the virtual machine (i.e. Prolog). These rules are translated from the formalism into Prolog clauses by a rule compiler written in Yacc prior to translation for improved time and space efficiency during translation.

ETS also contains an interface to the Unify relational database system where a large number of dictionary items for each representation level of a language can be entered, stored and updated. A direct interface between the Prolog core and Unify is currently under development.

Finally, the top-level interface between the user and ETS is in the form of a menu-driven interface which gives the user access to all components of the system plus access to external tools such as editors and the operating system UNIX† on top of which the entire system resides.

† UNIX is a trademark of Bell Laboratories.

5. THE EUROTRA FRAMEWORK : LINGUISTIC THEORY

In the preceding section the definitions and the formal properties of generators and translators were introduced. This section focuses on the linguistic contents of the representation levels. There are two key-ideas to be discussed here before going into detail with regard to each representation level separately:

Firstly, the representation languages must be *linguistic* in nature as the translation relation is fundamentally a relation between linguistic objects. Consequently, the intermediate levels cannot be completely neutral with regard to different natural languages, in the way a real 'interlingua' would be.

Secondly, there is the observation that in the adopted transfer-model the gap between text and interface structure is quite large. So, it seems impossible to relate text to interface structure (and vice versa) in one go. For this reason the *stratification* idea is introduced: the analysis and synthesis steps are decomposed into a sequence of primitive mappings between a number of intermediate levels of representation. It is also possible that this strategy can play a role in making the system robust. Robustness means that if at some point in the translation process the system fails (due to ungrammatical input, or simply because of a real system deficiency), it can still produce some (probably incorrect) translation on the basis of objects created on lower levels.

The current 'standard' hypothesis is that there are four intermediate representation levels (apart from text itself) for each language:

$Text^s \rightarrow Base^s \rightarrow ECS^s \rightarrow ERS^S \rightarrow IS^S \rightarrow IS^t \rightarrow ERS^t \rightarrow ECS^t \rightarrow Base^t \rightarrow Text^t$

5.1. Base level

The base level consists of three sublevels: ETS (Eurotra text structure), ENT (Eurotra normalised text) and EMS (Eurotra morphological structure). The first two are very low level and consequently not very linguistic in nature. They map text onto a uniform machine readable representation where characters are normalised, etc. This comes down to some variant of the ASCII character set. These levels make it possible for the linguist to abstract away from typographical and typesetting information in the actual text, so that he can assume a standard text format as input for analysis.

The idea of the EMS-generator is that it builds representations of the morpho-syntactic structure of word-forms by means of general morphological rules. Inflexion is already settled now by means of elevation (deletion of nodes in a tree representation and re-coding of its information via some feature on another node) of inflexion endings. The example below shows how the lexical form (i.e; actual word appearing in the input text) of a word is mapped onto its corresponding lexical unit (stem-form), and what kind of features are added to the lexical unit:

{*lex=plays*} → {*lu=play, cat=n,nb=plur*}
{*lex=plays*} → { }

It is clear from the example that one lexical form can get more than one morphological interpretation.

Derivation and compounding are still focuses of actual research.

Note that these levels have not been implemented yet and that work in practice translates text immediately to ECS. This is the reason why in the example (cf annex) at ECS there is still a distinction between the attributes *lex* (lexical form) and *lu* (lexical unit). The lexical form should disappear at ECS as soon as EMS is fully implemented.

5.2. ECS (Eurotra configurational structure)

ECS is a level of *phrase structure* closely related to the level of c-structure in Lexical-Functional Grammar. An ECS-representation is a labelled bracketing which maintains the linear order of the string and indicates its superficial hierarchical structure. ECS therefore does not contain empty elements nor coindexing (unlike Generalised Phrase Structure Grammar or s-structures in Government and Binding) and it has no notion of head or governor (unlike Dependency Grammar).

One of the important aspects of writing an ECS grammar is determining the correct set of categories for a specific language. In addition to the traditional categories (noun, adverb, verb, etc.) other lexical categories can be included: coordinator, quantifier, complementiser, etc. Also phrasal categories (e.g. noun phrase, prepositional phrase, adverbial phrase) are needed as the function of ECS is to group sequences of words together into phrases which will form constituents at the next level.

Note that the ECS-theory does not put any severe restrictions on the category set to be used ; much scope for language-specific divergences is left here, e.g. the notion of VP (verbal phrase) seems more suitable for certain languages whilst other can do perfectly well without it. This is entirely appropriate since ECS is presumably the level where Eurotra languages differ most.

What kind of phenomena are treated at ECS? Although most syntactic generalisations and constraints are stated at ERS, some order-based facts such as verb-second in German and Dutch, can be stated at ECS. Furthermore there are certain phenomena (such as some types of agreement) which can already be dealt with. This makes the level rather different from the purely superficial c-structure of LFG.

By way of an example the structure building rule which also takes care of agreement inside an NP for Dutch is shown :

```
{cat=np,nb=N,gender=G,ncase=Y,ntype=T}
[    ^ {cat=detp,nb=N,gender=G,msdefs=D},
     * {cat=ap,nb=N,gender=G,msdefs=D},
       {cat=n,nb=N,gender=G,ncase=Y,ntype=T},
     * {cat=pp}
]
```

This rule says that an NP obligatory consists of a noun (*n*) which can be preceded (optionality marker) by a determiner phrase (*detp*) and any number (Kleene star marker) of adjectival phrases (*ap*), and which can be followed by any number of prepositional phrases (*pp*). The features stated

on the daughter nodes require certain attributes to have the same value (variable sharing). If these conditions are met, a linguistic object at ECS can be created.

The rule will accept NP's such as :

de rode auto

een huis in de stad

but not:

* het man

as the gender information of the determiner and the noun is contradictory.

The rules and structures in section 4.2.3. also belong to ECS.

5.3. ERS (Eurotra relational structure)

ERS is clearly influenced by the f-structure of LFG. The most striking correspondence between them is the notion of syntactic dependency. This notion means that there is a governing lexical item (called *gov* in Eurotra) which governs its dependents. There are two classes of dependents :

— those which fill a slot in the frame of their governor (valency bound), called *complements* (subject, object, indirect object, etc.).

John sleeps

subj gov

— those which do not fill a slot because they are not required by a certain governor, called *modifiers.*

Bill hit Tom in the kitchen

modifier

ERS representations are flat trees (restricted bar-level) as the maximum category level that can be associated with a major lexical category is legislated to be one (bar-level = 1). Moreover, some elements with a minor lexical category at ECS are re-expressed by means of features on nodes in the representation ('elevation'). The fact that Eurotra prefers flat trees on ERS should be seen in the light of the ultimate IS-structure. Flat structures are easier to handle from a point of view of transfer as they reduce the need for structural transfer, which is one of Eurotra's main strategies.

It is required that ERS representation are *complete* and *coherent:* they are complete if and only if all the dependents of some governor are represented as part of the governor's construction; they are coherent if and only if the representation contains no more governable functions than required by the frame of the governor.

Word order in general is a phenomenon treated between the levels ECS and ERS. Treatment of word order means that the syntactic function of a phrase occurring at some position in the sentence is determined. Word order is neutralised at ERS as the position of an item is irrelevant for translation because every language has its own rules for where certain syntactic functions have to occur in the clause (for an example an ERS object see chapter 4.3.3.

A typical example of a syntactic phenomenon treated at ERS is subject-verb agreement. As subjects and verbs can appear in plenty of surface con-

figurations it is impossible to treat their agreement at ECS in a simple and unified way. At ERS, however, where syntactic relations are explicit it will be easy to capture the generalisation. The following strict filter rule takes care of subject-verb agreement:

```
{/cat=s}
[       {nb=N/sf=gov},
        {nb=N/sf=subj,cat=np},
     *{}
]
```

The slash ('/') separates the context and action part of the rule. Everything on the right hand side of the slash defines the context, the left hand side is called the action part, which can be empty for some of the nodes, as is here the case for the mother node.

It is clear that the passive construction is not 'undone' (reduced to its active equivalent) at ERS because agreement in passives is with the surface subject:

John was hit by his teachers
sg sg plur

Other phenomena which deserve special treatment at ERS are control, raising and long distance dependencies. The problem with these constructions is that from a surface point of view they violate the completeness and coherence principles. Thus, moved and deleted phrases have to be represented in their deep, original position in such a way that if they are governed by some element they also appear in the local tree of their governor. Eurotra has been experimenting with a special tool for unbounded dependencies (the so-called *recursion marker)* which will become part of the legislation quite soon.

5.4. IS (Interface structure)

The IS level is the most abstract level in Eurotra. It serves as input to the transfer components. Because of the requirement of 'simple transfer' the IS level should be as neutral as possible with regard to the different languages.

For the moment, IS is a rather straightforwardly 'linguistic' representation, with little or no provision for the expression, or use of 'real world knowledge'. This is the result of the characteristically linguistic view of translation taken in Eurotra. This is not to deny that real world knowledge is often crucial in selecting the correct translation.

IS is a *deep syntactic* level augmented by semantic features (e.g. animateness, abstractness, collective, etc.). These features can play a role in the disambiguation process of structures created by surface syntactic levels. For example the following Dutch sentence can be interpreted in two ways at ERS depending on surface function assignment:

Water drinkt de man
subj obj
obj subj

At IS, however, a selectional restriction based on the fact that only animates can drink, is able to rule out the first possibility.

The deep syntactic basis has been or will be enriched with some model-theoretic approaches to restricted linguistic areas like tense and aspect, determination, quantification, negation and mood;

The IS-representation is based on a *lowered-govemor deep dependency* grammar together with a frame theory. Just as at ERS we recognise two different types of dependency relations with regard to the governor:

— Arguments : called *arg*l, *arg*2*, arg*3 and *arg*4
— Modifiers.

Two further non-dependency relations being:

— Transconstructionals: constituents modifying the construction as a whole rather than the gov
— Conjuncts : gov-less construction (cf. infra).

These relations are universal in two senses:

— they constitute the smallest adequate set of dependency types with regard to the Eurotra languages
— they constitute the structural interface to all Eurotra languages.

The principles of completeness and coherence also applies at IS. Another principle which is also borrowed from ERS is that one bar level is the maximal projection for a phrase. Note that there may be an exception to the unique governor constraint with regard to coordination structures; the present legislation deletes the coordinators between ERS and IS, so that the remaining structure is 'gov-less' which is linguistically the most natural representation. This is an example of a very controlled exception.

The surface ordering of the string is replaced by a language-universal canonical order, viz. governors, arguments, modifiers.

In practice, this means, for example, that the difference between active and passive structures is not represented by different sets of relations (as it is at ERS), but by means of some feature. Other nodes which disappear at IS are: articles, demonstratives, auxiliaries, subordinate conjunctions, verb particles, argument bound prepositions, coordinators, surface place holders, etc. but their linguistic information is preserved in the representation by means of features on other, usually governor, nodes. At the end of this paper (cf. annex) an machine-made example shows the deletion and re-coding of articles and argument bound prepositions from ERS to IS.

Just as an example, the following structure building rule creates an IS object which consists of a verbal governor whose frame needs an *arg*l (deep subject) and an *arg*2 (deep object). The rule also applies to passive sentences; in that case the first slot is filled by the '*by*-phrase' (which is the deep subject), and the second by the surface subject (which is the deep object). The rule further allows for an indefinite number of modifiers. Note that the semi-colon in the rule is the alternation marker:

{*cat=s*}
[{*role=gov,cat=v,is_frame=argl_arg*2},
({*role=arg*l,*sf=subj*}; {*role=arg*l,*sf=by* }),
({*role=arg*2,*sf=obj*}*; {role=arg*2,*sf=subj*}),
* {*role-mod,sf= mod*}
]

The rule accepts both the following sentences :
the boy hit the girl yesterday
the girl was hit by the boy yesterday.

6. FUTURE PERSPECTIVES

Eurotra is the largest MT project that the world has seen so far: it concerns 9 languages, 72 language pairs, twenty research units, more than 150 linguists, computer scientists and translators, and an overall budget of 20,5 million ECU.

The prototype system which will be developed by 1990 is based on the most recent developments in the fields of computational linguistics (cf. the unification based formalism, the treatment of long distance dependencies, the inclusion of formal semantics, the attempts to incorporate insights from theories of discourse representation).

If this project fails, the consequences for the whole field of natural language processing and machine translation in Western Europe will be devastating, for it will become very difficult then to find financial support for similar efforts.

However, if it succeeds, the end of the century will show a boom of research and development activities in what is already called the field of Language Industry and Technology.

REFERENCES

ARNOLD D. (1986). *Eurotra: a European Perspective on MT.* In : Proceedings of the IEEE, vol. 74, no. 7.

CLOCKSINN W.F. and MELLISH C.S. (1987). *Programming in Prolog*. Third Edition, Springer-Verlag.

DAMAS L. (forthcoming). *YAP User's Manual.*

EARLEY J. (1970). *An Efficient Context Free Parsing Algorithm.* In: Communications of the ACM, vol. 13, no. 2.

EUROTRA-Reference Manual, version 5.0 (1988). Internal document, Luxembourg.

JOHNSON R., KING M. and des TOMBE L. (1985). *EUROTRA: a multilingual system under development.* In : Computational Linguistics 11, 2-3.

JOHNSON S.C. (1975). *Yacc:* Yet Another Compiler-Compiler. In: Computing Science Technical Report no. 32. AT&T Bell Laboratories.

KING M. and PERSCHKE S. (1987). *Eurotra*. In : Machine Translation today, M. King (ed.), Edinburgh University Press, Edinburgh.

MULTILINGUA 5-3 (1986). Mouton De Gruyter, Amsterdam.

PEREIRA F. (1984). *C-Prolog User's Manual version 1.5.* Ed CAAD, Edinburgh.

SLOCUM J. (1985). *A survey of machine translation: its history, current status and future prospects*. In: *Computational Linguistics 11-1.*

UNIFY Relational Data Base Management System: Reference Manual Release 3.2.

VAN EYNDE F. (1986). *Automatische vertaling: een overzicht van veertigjaar onderzoek en een blik in de toekomst.* In : Linguistica Antverpiensia XX, Antwerpen.

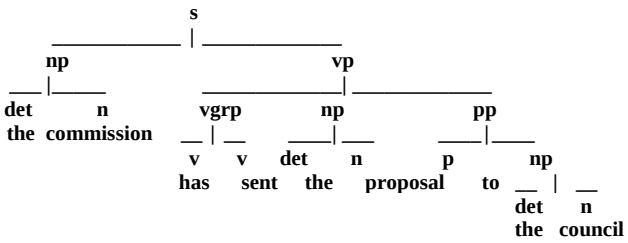
Annex: An Example

In this last section we present the machine-made translation from English to Dutch for the following sentence:

The commission has sent the proposal to the Council.

This example is meant to clarify the Eurotra translation system and to give the reader a more concrete idea of what the different levels of representation are like. For ease of presentation we have left out wrong representations on lower levels which eventually get filtered out at higher ones. Generator rules and translator rules are not shown as these modules are quite big (say 100 pages for analysis and synthesis together not including the lexicon). We give both a tree representation and an exhaustive labelled bracketing, accompanied by some very short remarks. Note that we go from text immediately to ECS, thus skipping the base levels.

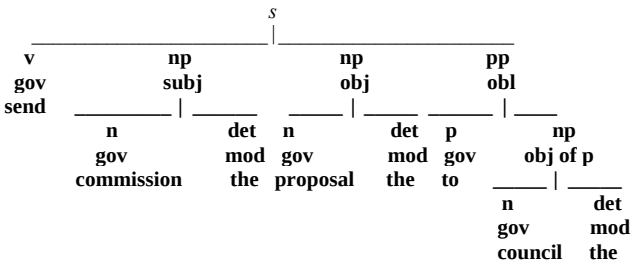
ECS:



(cat=s)
(cat =np ncase =nongen)
(cat =det, lu =the, lex=the msdefs =msdef)
(cat=n,lu =commission, lex=commission,nb =sing ncase =nongen, cattype =common)
(cat=vp)
(cat=vgrp)
(cat=v,lu=have, lex=has ,vform=fiv, finform=tsg ,cattype =aux)
(cat=v,lu=send, lex=sent, vform=past_part,cattype=main)
(cat =np, case =nongen}
(cat=det, lu=the, lex=the, msdefs=msdef)
(cat=n, lu =proposal, lex =proposal , nb =sing, ncase =nongen, cattype =common)
(cat=pp)
(cat=p, lu=to, lex=to]
(cat=np, ncase =nongen)
(cat=det ,lu=the, lex=the, msdefs=msdef)
*(cat=n, lu =*council, l*ex =council, nb =sing, ncase =nongen, cattype=proper)*

Deletion of auxiliary and vp-node; assignment of surface syntactic functions (subject, object, ..) and normalisation of word order.

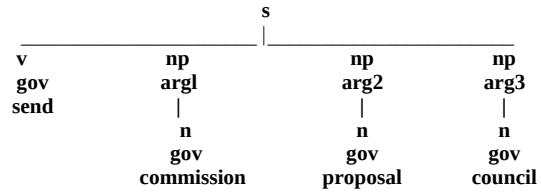
ERS:



(cat=s)
(sf=gov, cat=v, lu=send, msaspect=perf, mstns=pres, ers_frame=subj_obj_obl, diathesis =act, nb=sing)
(sf =subj,cat =np, nb=sing, ncase =nongen, cattype=common)
(sf =gov ,cat =n, lu =commission, nb =sing, ncase =nongen, ers_frame =none ,cattype =common)
(sf =mod, cat =det, lu =the, npdiacr =nonpro, msdefs =msdef)
(sf =obj, cat=np, nb=sing, ncase=nongen, cattype=common)
(sf =gov, cat =n, lu=proposal, nb=sing, nmorphol=deverbal, ncase =nongen ,ers_frame=none, cattype=common)
(sf =mod, cat =det, lu =the, npdiacr=nonpro, msdefs =msdef)
(sf =obl, c*at =pp, smod=yes, pform=to, pdist =nonadjl)*
(sf =gov, cat =p, lu=to)
(sf=objofp, cat=np, nb=sing, ncase=nongen, cattype=proper)
(sf =gov ,*cat =n, lu =council, nb =sing, ncase =nongen, ers_frame =none, cattype=proper)*
(sf =mod ,cat =det, lu=the, npdiacr=nonpro, msdefs =msdef)

Deletion of determiners and valency bound prepositions; assignment or roles (gov, argl, ..).

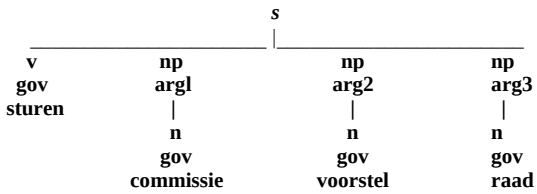
IS:



(cat=s,s_type =main)
(role =gov, sf=gov, cat=v, lu =send, msaspect=perf, mstns=pres, pform of arg 3=to, *is_frame=arg l_arg 2_arg 3, diathesis=act , nb=sing)*
(role =arg 1, s*f =subj, cat =np, nb =*sing, *ncase =nongen, cattype =common, msdefs =msdef)*
(role = gov, sf*=gov, cat =*n, l*u=commission, nb=sing, ncase=nongen, is_frame =none, cattype =common)*
(role=arg 2,sf=obj ,cat=np ,nb=sing, ncase=nongen, cattype=common, msdefs=msdef)
(rote=gov, sf=gov,cat=n, lu=proposal, nb=sing, nmorphol=deverbal, ncase=nongen, is_frame=none, cattype =common)
(role=arg3, sf=objofp ,cat=np, nb =sing, ncase =nongen, cattype =proper, msdefs =msdef)
(role =gov, sf =gov, cat =n, lu =council, nb =sing, ncase =nongen, is_frame =none, cattype =proper)

Simple transfer. Only the lexical items in the tree are translated. The interaction of dictionary entries and feature translation rules takes care of correct feature-assignment.

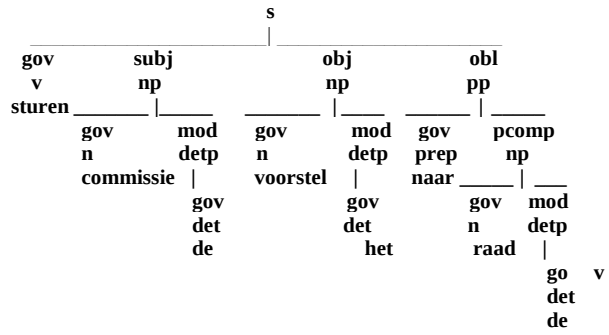
IS:



(cat=s, cs=no, s type=main, diath=act)
(role =gov, sf =gov, cat =v, lu=sturen, msaspect=perf, mstns =pres,cs=no, ers_frame =sobobl, is_framet=arg 1_arg2_arg 3, pform_of_arg3=naar,vtype=main, nb=sing)
(role =arg 1,*sf =subj,cat =np ,cs =*no, nt*ype =ordn, ncase =norm, elev =*yes, *gender =nneut, msdefs =msdef, nb=sing)*
*(role =gov, sf =*gov *,cat =n, lu =commissie ,cs =no, ntype =ordn, ncase =norm ,ers_frame =empty, is_frame =empty, gender =nneut, pers =t, nb =sing)*
*(role =*arg 2, sf=*obj ,cat =np, cs =no, ntype =ordn, ncase =norm, elev =yes, gender =neut, msdefs =msdef, nb=sing)*
(role =gov, sf =gov ,cat =n, lu =voorstel ,cs =no, ntype =ordn, ncase=norm, ers_frame =empty, is_frame =empty, gender=neut, pers=l ,nb=sing)
(role =arg 3, s*f =pcomp ,cat =np ,cs=no, ntype =ordn, ncase =norm, elev =yes gender =nneut, pform =naar, msdefs=msdef, nb=sing)*
*(role=gov, sf =gov ,cat =*n, l*u=raad ,cs=no, ntype =ordn, ncase =*norm *,ers_frame =empty, is_frame =empty, gender=nneut, pers=t, nb=sing)*

Insertion of determiners and valency bound prepositions; function assignment.

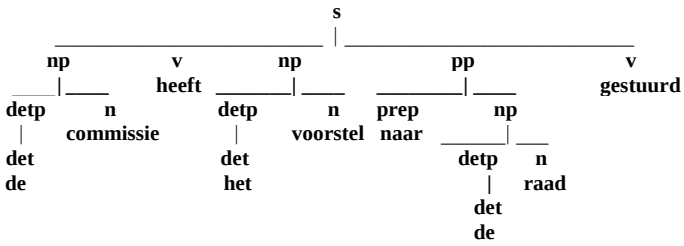
ERS:



(cat=s,cs =no, s_type =main, diath =act)
(sf =gov, cat =v, lu =sturen, msaspect =perf, mstns =pres, cs =no, ers_frame =sobobl, pform =naar, vtype =main, nb =sing)
(sf =subj ,cat =np ,cs =no, ntype =ordn, ncase =norm, gender =nneut, nb =sing)
(sf=gov, cat =n, lu =commissie ,cs =no, ntype =ordn, ncase =norm ,ers_frame =empty, gender =nneut, pers=t, nb=sing)
(sf =mod, cat =detp ,cs =no, dtype =art, elev =yes, gender=nneut, msdefs =msdef, nb =sing)
(sf=gov, cat =det, lu=de, cs=no ,dtype=art, elev=yes det=de, gender =nneut, msdefs=msdef, nb=sing)
*(sf =obj, cat =*np *,cs =no, ntype =ordn, ncase =norm, gender =neut, nb =sing)*
*(sf =gov ,cat =n, lu =voorstel ,cs=no, ntype =ordn, ncase =norm ,ers_frame =empty, gender =neut, pers =*t, *nb=sing)*
(sf =mod, cat =detp ,cs =no, dtype =art, elev =yes, gender =neut, msdefs=msdef, nb =sing)
(sf=gov ,cat=det ,lu=het ,cs=no, dtype=art, elev=yes, det=het, gender =neut, msdefs=msdef, nb=sing)
(sf=obl, cat =pp, cs =no, pform =naar)
(sf=gov ,cat=prep, lu=naar, cs=no ,ers_frame=np_compl)
(sf =pcomp, cat =np ,cs =no, ntype =ordn, ncase =norm, gender =nneut, nb =sing)
(sf =gov , cat =n, lu =raad, cs=no, ntype =ordn, ncase =norm ,ers_frame =empty, gender =nneut, pers-t, nb=sing)
*(sf =mod, cat =detp ,cs =no, dtype =art, elev =*yes, *gender =nneut, msdefs =msdef, nb =sing)*
*(sf =gov, cat =det, lu =de ,cs =*no, d*type =art, elev =yes, det =de, gender=nneut, msdefs=msdef, nb =sing)*

Insertion of auxiliary; note that the Dutch grammar does not use a vp-node. Surface word order is determined.

ECS:



(*cat = s ,cs =no, s_type=*main)
(cat =np, ntype =ordn, ncase =norm, gender =nneut, cs=no, nb =sing)
(cat=detp, dtype=art, msdefs =msdef, gender=nneut, cs=no, nb=sing)
(cat =det, lu =de, lex =de, dtype =art, msdefs =msdef, gender =nneut, cs =no, nb =sing)
(cat=n, lu=commissie, lex=commissie, ntype=ordn ,ncase=norm,gender =nneut ,cs=no, nb=sing)
*(cat =*v, l*u =hebben, lex =heeft, cs =no ,vtype =aux, finite =fin, vform =fin, nb =sing)*
(cat=np, ntype=ordn, ncase=norm, gender =neut ,cs=no, nb=sing)
(cat =detp, dtype =art, msdefs =msdef , gender =neut ,cs =no, nb =sing)
(cat=det, lex=het ,lu=het, dtype=art, msdefs=msdef, gender=neut ,cs=no, nb=sing)
(cat =n, lu =voorstel, lex =voorstel, ntype =ordn, ncase =norm, gender =nneut, cs =no, nb =sing)
(cat=pp,cs=no, preptype=no)
(cat =prep, lex =naar, lu =naar, cs=no)
(cat =np, ntype =ordn, ncase =norm, gender =nneut ,cs =no, nb =sing)
(cat =detp, dtype =art, msdefs =msdef, gender =nneut, cs=no, nb =sing)
(cat=det, lex=de, lu=de, dtype=art, msdefs=msdef, gender=nneut,cs=no, nb=sing)
(cat=n, lu=raad,lex=raad,ntype=ordn, ncase=norm, gender=nneut, cs=no, nb=sing)
(cat =v, lu =sturen, lex =gestuurd, cs =no ,vtype =main, finite =nonfin, vform =past_part)

From this ECS-representation the Dutch sentence is generated simply by concatenation of the lexical nodes (i.e. the values for lex):

De commissie heeft het voorstel naar de Raad gestuurd.

Address : Eurotra-Leuven, Maria Theresiastraat 21, B-3000 Leuven (Belgium)

Received: 8 October 1988