[*Proceedings of a Workshop on Machine Translation*, July 1990, UMIST]

The EUROTRA Machine Translation System

Anand SYEA
Centre for Computational Linguistics
UMIST
Manchester M60 1QD, UK
E-mail: anand@uk.ac.umist.ccl

July, 1990

1. The Eurotra Machine Translation System

The Eurotra MT project was launched in order to meet the translational needs within the EEC institutions. Currently, it involves nine languages (Danish, Dutch, English, French, German, Greek, Italian, Portuguese and Spanish) and seventy-two language pairs. The system is being developed with a view to translating the official EEC documents within the domain of information technology.

2. The place of Eurotra within Machine Translation

2.1 Design

Like other MT systems, Eurotra accepts the principle that translation is a three-step process: analysis, transfer and synthesis. It is designed to be transfer-based, rather than an interlingual system, because there are few ideas about what interlingual representations should look like. Accordingly, it insists on separate transfer components existing alongside the monolingual components (ie analysis and synthesis).

As a transfer-based system, Eurotra adheres to the following general principles:

- transfer should be as simple as possible
- analysis and synthesis should be strictly monolingual
- the output representations of analysis and transfer which serve as input to transfer and synthesis respectively are called interface structures (or ISs)
- the mapping of texts onto ISs (and ISs onto texts) involves a series of translations and several levels of representations. (This is known as the stratification principle)

2.2 Implementation

Eurotra uses a special purpose language written in Prolog for implementation. This special purpose language is thought to be much more user-friendly for the linguists and lexicographers who have to formalise and code their grammars and dictionaries.

3. Formalism and Mechanism

The Eurotra framework is a model of translation which provides the linguists with concepts (ie the different linguistic specifications - see below) and tools amongst which are:

- a user language (a formalism in which to describe a language
- the virtual machine (a mechanism for applying grammar rules

The linguist makes a series of descriptions of a language, of different abstraction, to correspond to the different levels of representation which separate texts from their ISs. A description for a particular level is called a generator and the relations between generators (and therefore levels of representation) are called translators.

3.1 Representation levels

In attempting to keep transfer as simple as possible, the gap between text and IS is quite large. Relating text and IS is therefore a complex task which can be executed only by having several intermediate levels of representation. Each level is a formal language which is a set of objects, either simple or structured. Simple objects are feature bundles (ie sets of attribute-value pairs) and these are legal for some level of representation only if they are defined by the feature theory of that level. Structured objects are trees of feature bundles which describe dominance (mother-daughter) relation and precedence (sister-sister) relation. Only those structured objects which are defined by the generator (grammar) rules may exist at a particular level of representation.

3.2 Generators

3.2.1 Consolidation

A generator consists of the grammar rules of a level. The input to a generator is a tree structure produced by a translator. This tree structure is called an unconsolidated object and can be seen as an hypothesis of what the representation will be like within the given level. The generator attempts to consolidate the input object, by proving that the feature bundle nodes and the dominance and precedence relations between nodes conform to the feature theory and the grammar rules for the given level. This proving mechanism takes the object bottom-up, checking the attribute-value pairs and applying generator rules to the object by controlled unification. Control is provided by a variant of the Earley parsing algorithm (see section 3.2.3 for example). The matching of objects and rules can modify the feature bundles and their relations. Rules may contain variables as well as constant values, and these variables may become instantiated by unification. In this way agreement and percolation of features can be dealt with in generator rules. When all nodes and their relations have been consolidated, a consolidated object is produced by the generator. The output may be more than one object, if the input tree matches rules in several ways or if it matches several alternative rules. If the object cannot be proved, consolidation fails and there is no output.

3.2.2 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:

$$fbd\ [\ arg_1\ ,\ arg_2\ \ldots\ ,\ arg_n\ ]$$

where the head of the rule is a feature bundle description (these are the descriptions in rules of the 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.

Structure building rules are the main rule type in a generator, the first type 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.

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 condition part and an action part. Feature rules are applied to objects if the structure of a rule matches the structure of an object and the condition 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 well-formedness, 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 feature rules. There are two types of filter rules: strict and killer. Strict filter rules, like feature rules, contain condition and action parts. If the structure of a rule matches the structure of an object and the condition 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 condition parts only - the action is always deletion of the object. If the structure of a rule matches the structure of an object and the condition part is satisfied then the object is considered ill-formed and is deleted.

3.2.3 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 angled 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 (ie 'dictionary look-up') 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 <u>nb</u> are uninstantiated variables as their values are ambiguous. Also note that the consolidation of 'fish' has resulted in two possibilities.

Although some of the feature bundles themselves can still 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 <u>nb</u> has become instantiated by unification and has percolated plur to the mother mode.

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 (ie {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 so far created, 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.

The 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.

3.3 Translators

3.3.1 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 without creating any intermediate representations within the translator.

The input to a generator 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, decomposing the input object into a number of unconsolidated sub-objects which are immediately passed on to the target generator. The basic principle of the concept of translators is thus that of compositionality - the translation of an object is a function of 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. For default translation, this is defined as the intersection of the feature theories of the source and target generators, ie the feature declarations and co-occurrence restrictions that exist at both levels. For user-defined translator rules it is defined as the union of the feature theories of the source and target generators.

Built in to the system is a mechanism for the default translation of objects from source to target generators. The mechanism is intended to simplify the job of rule-writing by linguists by providing a default translation for default cases. 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 modifications 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 are part of the feature declarations of the target level).

3.2.2 Translator Rules

A user can 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. 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, for example:

```
     A:fbd [B:argl, C:arg2, N:argn]
     => A <B, C, ... N>
```

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 (ie introducing an unordered set)
- deleting nodes
- inserting nodes

Feature translator rules, if applicable, override feature translation by 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:

$$\text{fbd } [\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_n] \Rightarrow \text{fbd}$$

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.

3.3.3 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 3.2.3 might be translated to the next level of representation. Assume the structure translator rule:

```
     A:{cat=s}
          [B:{cat=np}
           ~:{cat=vp}
          [C:{cat=v}
           D:{cat=np}] ]
     => A < C:{frame=subj_obj}, B, D >
```

which will match with our consolidated object from the previous example 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=X}
          {lu=fish, cat=n, nb=X}
```

3.4 Software implementation

3.4.1 Implementation of the Virtual Machine

The generator and translator, which form the 'core' of the system, are written in Prolog which offers several advantages:

- the virtual machine is unification-based and Prolog's built-in mechanism of unification offers an ideal environment
- the virtual machine is non-deterministic (ie it computes all possible alternatives) and Prolog's backtracking mechanism allows this sort of non-determinism
- the ease of program development within Prolog means modifications to the Eurotra framework can be quickly implemented

3.4.2 Software Environment

A number of tools, written in Prolog, are available to the linguists to write correct generator and translator rules. These are:

- a debugging system to trace application rules and their effects
- a pretty-printer to display objects in various formats
- a command interpreter to manipulate objects

The 'user language' (ie rules written by linguists) is translated **into** Prolog clauses prior to translation by a compiler written in Y**acc.**

The Eurotra Translation System also contains an interface to the Unify relational database system where a large number of dictionary items are stored. Also, the interface between user and ETS is in the form of a menu-driven interface which enables the **user** to access components of the system and external tools such as editors and the operating system UNIX.

4. Linguistic Specifications

This section looks at the linguistic contents of the representation levels. As was mentioned earlier, Eurotra embodies the principle of stratification, ie texts are related to their interface structures (which turn out to be the most abstract level of representation) via a series of intermediate levels of representations. The current assumption is that there are three levels of representation between a text and its IS:

```
(source  language)  Text =>  EMS  =>  ECS  =>  ERS  =>  IS
(target language)  Text  <=  EMS  <=  ECS  <=  ERS  <=  IS
```

Note that representation languages are linguistic in nature as the translation relation between them is a relation between linguistic objects.

4.1 The Frontend

The frontend consists of three levels: ETS (Eurotra Text Structure), ENT (Eurotra Normalised Text) and EMS (Eurotra Morphological Structure). The first two map texts onto a uniform machine readable representation. The third, which is more linguistic, builds morpho-syntactic representations by morphological rules. Cases like inflection would be handled here.

4.2 ECS (Eurotra Configurational Structure)

An ECS representation is basically a labelled bracketing which maintains the linear order of the string and indicates its surface constituency as well as expresses generalisations about surface word order. Crucially, it does not have empty categories nor coindexing and has no notion of head/governor.

An ECS grammar uses both lexical and phrasal categories which can be the traditional categories (eg v, vp, n, np, p, pp, etc) or categories like co-ordinator, quantifier, complementiser, etc.

An example of a grammar rule (in this case a rule for building NPs in Dutch - which incidentally also deals with agreement outside NP) at ECS is:

```
{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}]
```

4.3 The Deeper Levels

ERS and IS are both lowered-dependency grammars. They differ from ECS as follows:

- the leading notion is syntactic dependency
- the order of constituents is canonicalised
- certain nodes which exist at ECS are featurised, eg articles, auxiliaries, verb particles and strongly bound prepositions

Dependents at these levels are either complements (those which must fill in a slot in the frame of their governor) or modifiers (those which do not fill in a slot).

Both these levels are constrained by the principles of Completeness (all the complements of a governor must be present) and Coherence (no more complements are present than the frame of the governor admits).

4.3.1 ERS (Eurotra Relational Structure)

The leading notion here is surface syntactic dependency. ECS objects are translated into representations labelled in terms of syntactic functions: subject, object, etc. Phenomena such as subject-verb agreement, control, raising and long distance dependencies are handled at ERS.

4.3.2 IS (Interface Structure)

The assumption with regard to IS is that it should be as neutral as possible with regard to the different languages. It is basically a 'linguistic' representation so far since 'real world knowledge' is almost non-existent. Unlike ERS, IS represents deep syntactic dependency and representations are expressed in terms of deep syntactic functions namely argl, arg2, etc, augmented with semantic features (eg human, animate, abstract, etc) and semantic analyses of such phenomena as Tense, Mood, determination, negation and quantification.

Beside dependent relations, two non-dependent relations also exist at IS namely, transconstructional (constituents modifying the sentence as a whole) and conjuncts. These relations imply the existence at IS of non-lexical governors (in the case of transconstructional) and governor-less constructions (in the case of conjuncts).

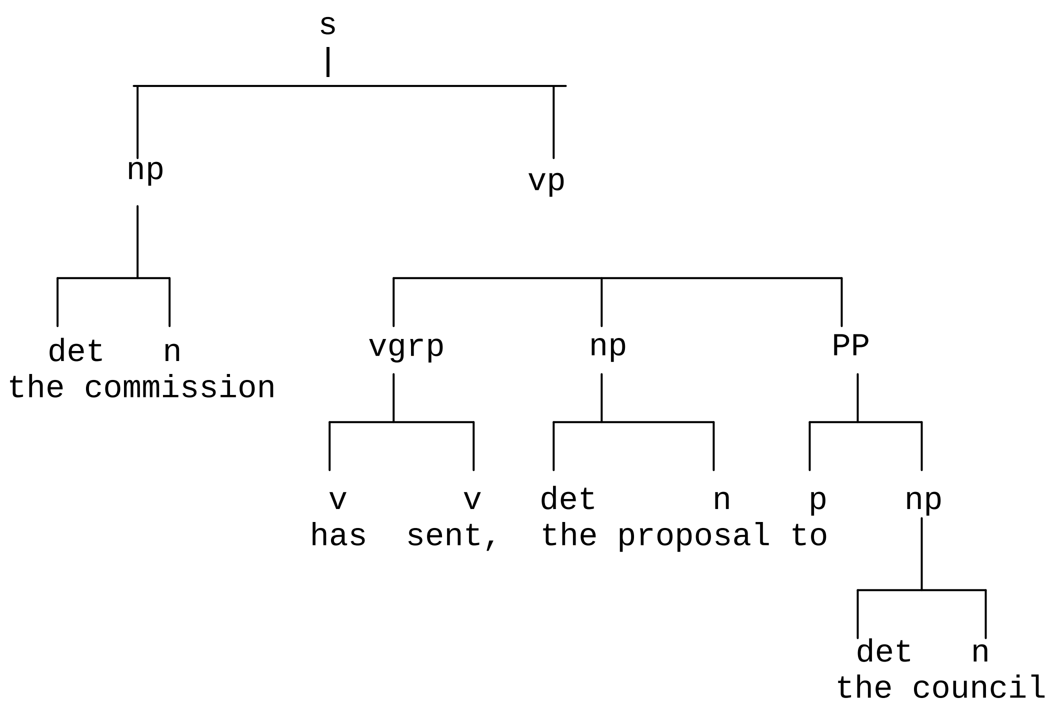
4.4 Example of ECS, ERS and IS representation

A sentence like:

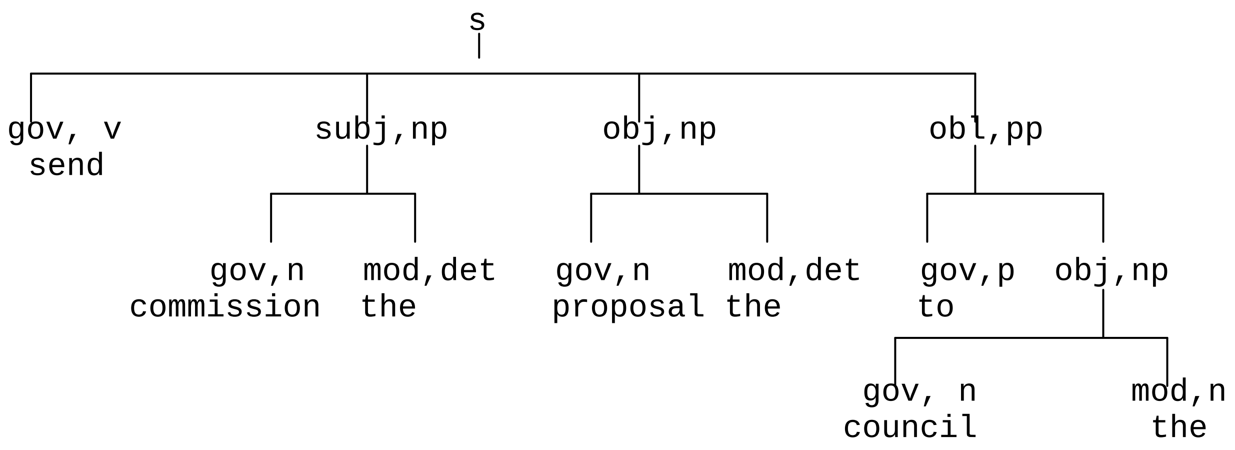
The commission has sent the proposal to the council

will have the following ECS, ERS and IS representations in analysis:

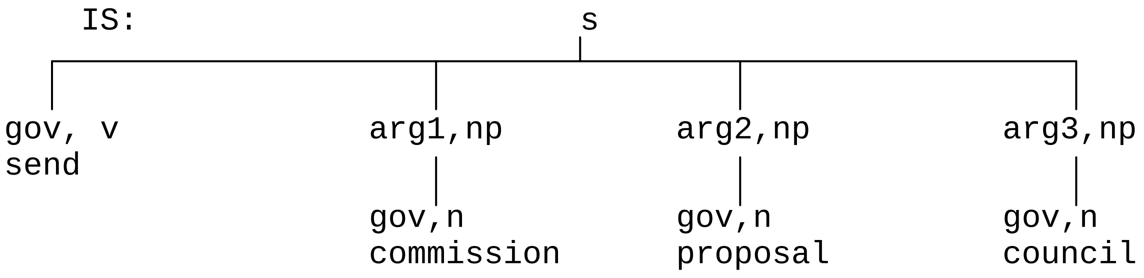
ECS:



ERS:



IS:



s
gov, v
send
arg1,np
gov,n
commission
arg2,np
gov,n
proposal
arg3,np
gov,n
council