Subgrammars, Rule Classes and Control in the Rosetta Translation System *

Lisette Appelo Carel Fellinger Jan Landsbergen

Philips Research Laboratories
P.O. Box 80 000, 5600 JA Eindhoven, The Netherlands

Abstract

The paper discusses a recent extension of the linguistic framework of the Rosetta system. The original framework is elegant and has proved its value in practice, but it also has a number of deficiencies, of which the most salient is the impossibility to assign an explicit structure to the grammars. This may cause problems, especially in a situation where large grammars have to be written by a group of people. The newly developed framework enables us to divide a grammar into subgrammars in a linguistically motivated way and to control explicitly the application of rules in a subgrammar. On the other hand it enables us to divide the set of grammar rules into rule classes in such a way that we get hold of the more difficult translation relations. The use of both these divisions naturally leads to a highly modular structure of the system, which helps in controlling its complexity. We will show that these divisions also give insight into a class of difficult translation problems in which there is a mismatch of categories.

1 The Rosetta Framework

In this section we will give an outline of the approach to machine translation pursued in the Rosetta project, which takes place at Philips Research Laboratories. The linguistic framework of Rosetta can be characterized by a number of principles. These are 'working principles', intended to be helpful for systematic research on translation and for the actual construction of translation systems.

The principles are discussed here to the extent in which they are relevant to this paper.

- **Principle of Explicit Grammars:** There is an explicit grammar for both the source and the target language.

  In most translation systems the target language is defined indirectly by means of contrastive transfer rules that specify the differences with the source language. We think it important to have an independent criterion for correctness of the target text.

- **Compositionality Principle:** The meaning of an expression is a function of the meaning of its parts and the way in which they are syntactically combined.

  This principle was adopted from Montague Grammar (cf. Thomason, 1974). Obviously, this principle will lead to an organisation of the syntax that is strongly influenced by semantic considerations. But as it is an important criterion of a correct translation that it is meaning-preserving, this seems to be a useful guideline in machine translation.

  The compositional grammars of Rosetta, called M-grammars, consist of three components: a syntactic, a semantic and a morphological component.

  The **syntactic component** defines surface trees of sentences. The surface trees used in Rosetta, called S-trees, are ordered trees of which the nodes are labelled with syntactic categories and attribute-value pairs that bear other morpho-syntactic information. The branches are labelled with syntactic relations. S-trees are used as intermediate representations as well.

  The syntactic component defines the set of correct S-trees by specifying:

  1. a set of **basic expressions.**
  2. a set of compositional **syntactic rules.**

  These rules make it possible to derive new S-trees and ultimately surface trees of sentences from the basic expressions. The rules have 'transformational power', they may perform various operations on S-trees. The process of deriving a surface

*This paper is the merger of two complementary papers on the Rosetta translation system that were submitted to the European ACL Conference 1987, i.e. 'Subgrammars and Rule Classes in the Rosetta Translation System' by Appelo and Fellinger and 'Controlled M-Grammars in the Rosetta System' by Landsbergen.

This research was partially sponsered by Nehem (Nederlandse Herstructureringsmaatschappij).

tree starting from basic expressions by applying syntactic rules recursively, in a 'bottom-up' way, can be represented in a **syntactic derivation tree** with the basic expressions at the terminals and the names of the applied rules at the non-terminals. With each node of the derivation tree an intermediate resulting S-tree can be associated, i.e. the S-tree that is the result of the application of the rule of that node on the resulting S-trees of its daughters (see figure 1).

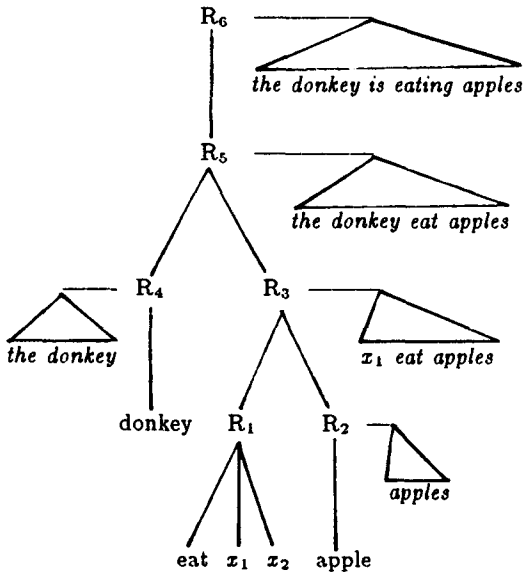

Figure 1: syntactic derivation tree, the derived S-trees are paraphrased by strings

The leaves of a complete surface tree correspond to the words of the sentence, but they have the form of categories and attribute-value pairs. The **morphological component** relates these leaves to actual symbol strings. In this paper we will ignore this morphological component and the S-trees will be 'paraphrased' by strings most of the time to enhance the readability of these trees.

The M-grammars have a **semantic component** that specifies

1. the meaning of the basic expressions (**basic meanings**).
2. the meaning of the rules (**rule meanings**).

In Montague Grammar these meanings are expressed in intensional logic. In the Rosetta system the meanings of rules and basic expressions are not elaborated on in a logical language, but they are represented by means of unique names. The consequence is that a meaning of a sentence can be represented as a so-called **semantic derivation tree**: a tree with the same geometry as the syntactic derivation tree but labelled with names of rule meanings and basic meanings instead of syntactic rules and basic expressions. In figure 2 an example of a semantic derivation tree is given, corresponding to the syntactic derivation tree of figure 1.

As basic expressions may have various meanings, there is in general a set of semantic derivation trees corresponding to a syntactic derivation tree. There is in general a set of syntactic derivation trees corresponding to each semantic derivation tree, because a basic meaning may correspond to various basic expressions and a meaning rule may correspond to various syntactic rules.

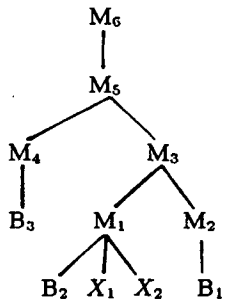

Figure 2: semantic derivation tree corresponding to the syntactic derivation tree of figure 1

- **One Grammar Principle:** The analysis and generation components for one language are based on the same grammar.

  In other terms, we require the compositional grammar defined above to be 'reversible'. The analysis component maps sentences onto derivation trees, the generation component maps derivation trees onto sentences.

  Because of this principle M-grammars have to obey certain conditions. The most important condition is that for each generative syntactic rule there must be a reverse analytical rule. For a more extensive discussion of these conditions we refer to Landsbergen (1984). Thanks to these conditions analysis algorithms can be defined which yield for any input sentence the set of syntactic derivation trees of that sentence (see section 6 for the formal definitions).

  In addition to theoretical motives, there are economic motives for adopting the One Grammar Principle. If we plan to make translation systems that translate both from and into a particular language, it is efficient if these systems can be based on one grammar.

  Because of this principle it suffices most of the time to discuss the grammars from a compositional, generative point of view only.

- **Isomorphy Principle:** Two sentences are translations of each other if their meanings are derived from the same basic meanings in the same way, i.e. if they have the same semantic derivation tree.

  So this principle says that the information that has to be conveyed during translation is not only the meaning, but also the way in which the meaning is derived.

  This implies that we have to attune the grammars of the system in the following way:

  1. each basic expression in one grammar corresponds to at least one basic expression in the other grammar with the same meaning (i.e. corresponding to the same basic meaning).
  2. each syntactic rule of one grammar corresponds to at least one rule in the other grammar with the same meaning (i.e. corresponding to the same rule meaning).

  So, two sentences are translations of each other if they have corresponding, **isomorphic** syntactic derivation trees, i.e. trees with the same geometry and corresponding basic expressions and corresponding rules at the leaves and at the nodes respectively (see figure 3).

  Following this principle there are corresponding sets of rules, related to the same meaning rule, and corresponding sets of basic expressions, related to the same basic meaning. We call the grammars isomorphic if these corresponding sets of rules obey certain applicability conditions.

  The Isomorphy Principle is the most characteristic principle of the Rosetta system, as it expresses our compositional theory of translation.

  In this approach complex structural transfer rules are avoided, as rules and basic expressions of the source language are related locally to rules and basic expressions of the target language, although, of course, the individual grammars may be complicated because of the attuning.

- **Principle of Interlinguality:** There is an intermediate language into which analysis components of various languages translate and from which the generation components of these languages are able to translate. If we combine this principle with the Isomorphy Principle, the main consequence is that the semantic derivation trees constitute the intermediate language and that the attuning of the grammars is done for possibly more than two grammars.

  It should be stressed that the isomorphy and not the interlinguality is the primary characteristic of the Rosetta framework.

For a more extensive discussion of these principles and more interesting examples we refer to Appelo and Landsbergen (1986). Leermakers and Rous (1986) give an introduction to the Rosetta method along different lines.

The global design of the Rosetta system, which follows from these principles is sketched in figure 4. For each M-grammar the following system components are defined:

- an analytical and a generative morphological component, **A-MORPH** and **G-MORPH**. They account for the relation between strings and lexical S-trees (i.e. S-trees corresponding to words).
- an analytical and a generative syntactic component, **M-PARSER** and **M-GENERATOR**. They account for the relation between surface trees and syntactic derivation trees. These system components follow directly from the syntactic component of an M-grammar. Their formal definition is given in subsection 6.1.
- an analytical and a generative semantic component, **A-TRANSFER** and **G-TRANSFER**. They account for the relation between syntactic and semantic derivation trees.

M-PARSER is preceded by a component called **S-PARSER** (for surface parser) which maps a sequence of lexical S-trees (which is the output of A-MORPH) onto a set of surface trees of which the lexical S-trees are the leaves. This set should contain the correct surface trees, but may contain also incorrect ones. The generative counterpart, **LEAVES**, is trivial; it maps the surface tree onto the sequence of its leaves.

2 Problems with the Rosetta framework

The framework outlined above has been worked out in a way that is simple and mathematically elegant, as the formal definitions in subsection 6.1 will illustrate. This formalism has also proved its value in practice: the implemented systems Rosetta1 and Rosetta2 have been written in this framework. In the sequel we will refer to it as the Rosetta2 framework. However, it also has a number of deficiencies, which may cause problems, especially in a situation where large grammars have to be written by a group of people. Three kinds of problems can be distinguished.

1. **Lack of structure in M-grammars**
   Grammars for natural languages are very large and inherently complex. In an M-grammar the syntactic component specifies a set of rules without any internal structure. Although the mathematical elegance of free production systems is appealing, they are less suited for large grammars. As the number of rules grows, it becomes more and more desirable that the syntax be subdivided into parts with well-defined tasks and well-defined interfaces with other parts.

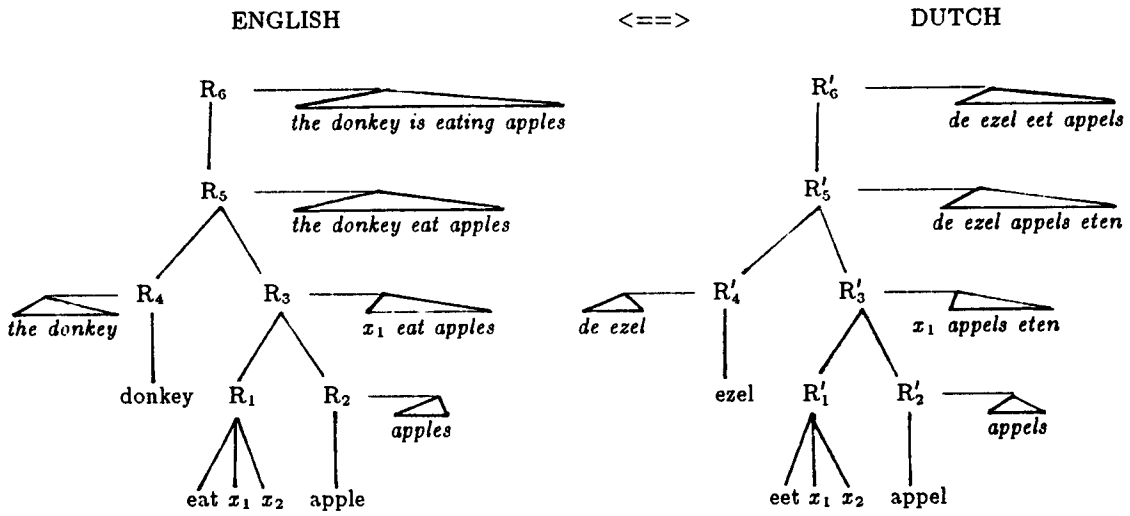

Figure 3: isomorphic syntactic derivation trees for the sentence *The donkey is eating apples* and its translation in Dutch *De ezel eet appels*

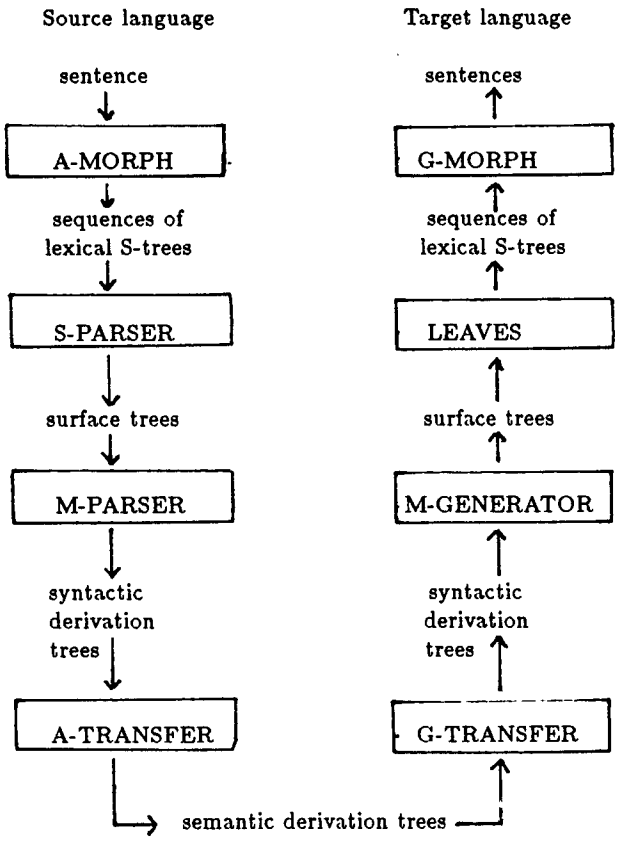

Figure 4: Global design of the Rosetta system

This holds in particular if the grammars are developed by a group of people. It is necessary to have an explicit division of tasks and to coordinate the work of the individuals in a flexible way so that the system will be easy to modify, maintain and extend.

In computer science it is common practice to divide a large task into subtasks with well-defined interfaces. This is known as the **modular approach**. This approach has gained recognition in the field of natural language processing too (cf. Isabelle and Macklovitch, 1986 and Vauquois and Boitet, 1985). The question is how such a modular approach can be applied in a compositional grammar, in an insightful and linguistically motivated way.

2. **Lack of control on rule applications**
In many cases the grammar writer has a certain ordering of the rules in mind, e.g. he may want to express that the rules for inserting determiners during NP-formation should be applied after the rules for inserting adjectives. In the M-grammar formalism explicit ordering is impossible, but the rules can be ordered implicitly by characterizing the S-trees in a specific way, e.g. by splitting up a syntactic category into several categories, and by giving the rules applicability conditions which guarantee that the aspired ordering is achieved. For example, if one wishes to order two rules that both operate on an NP, this can be achieved by creating categories NP1, NP2 and NP3 and to let the first rule transform an NP1 into an NP2 and the second rule an NP2 into an NP3. This approach was followed in Rosetta2. One of its

disadvantages is that it leads to a proliferation of rather unnatural categories.

It is hard to find an elegant and transparent way of specifying rule order in a compositional grammar; the situation is more complicated than in transformational systems like ROBRA (Vauquois and Boitet, 1985), because rules may have more than one argument.

In addition to linear ordering one may want to add other means of controlling the application of rules, e.g. one may want to make a distinction between obligatory, optional and recursive rules. In M-grammars all rules are optional and potentially recursive. It is not clear how to add obligatory rules to such a free production system; in fact it is hard to understand what that would mean. There is also a problem with the reversibility of obligatory rules: a rule that is obligatory during generation is not necessarily obligatory during analysis.

3. **Lack of structure in the translation relation**
As we have explained in section 1, the translation relation between languages is defined by attuning the grammars to each other. In this way complex structural transfer (as discussed in Nagao and Tsujii, 1986) can be avoided, but in some cases the dependency between the grammars may complicate individual grammars. **Category mismatch** is one of these translation problems, e.g. the *graag/like* case, where a Dutch adverb corresponds to an English verb. In cases like this there is a mismatch of syntactic categories coupled with different behaviour with respect to, e.g., tense: a verb has tense, whereas an adverb has not.

   In Landsbergen (1984) a solution of the *graag/like* problem by means of isomorphic grammars was discussed, for small example grammars. For larger grammars a more systematic and structured treatment of these translation problems is needed, but this is not supported by the Rosetta2 formalism.

   Another problem is caused by the fact that in the isomorphic grammar framework each syntactic rule of one grammar must correspond to at least one rule of another grammar. For rules that contribute to the meaning this is exactly what we want, because what has to be conveyed during translation is not only the meaning, but also the way in which the meaning is derived. However, there is a problem with rules that are only relevant to the form of the sentence and that carry no translation-relevant information, especially if they are language-specific. A purely syntactic transformation as Verb-Second in an SOV language like Dutch does not correspond in a natural way to a syntax rule of English. In Rosetta2 this problem could be solved in one of the following two ways: by adding a corresponding rule to the English syntax that did nothing more than change the syntactic category or by merging the Dutch transformation rule with a meaningful rule. These solutions are not very elegant and complicate the grammars unnecessarily. It would be better if the correspondence between rules as required by the Isomorphy Principle must hold for meaningful rules only. The translation relation would then be defined in terms of a reduced derivation tree, which is labelled with meaningful rules. The generation component (M-GENERATOR) will operate on such a reduced tree and will have to decide what syntactic transformations are applicable at what point of the derivation. This requires some way of controlling the applicability of the transformation rules.

In the next sections we will describe the modular approach chosen for the development of Rosetta3, which may help to solve the above-mentioned problems. We will discuss a syntax oriented division into subgrammars in section 3 and a translation oriented division into rule classes in section 4. In section 5 we will argue that a combination of the two divisions is needed. In section 6 the newly introduced notions will get a formal treatment. It will turn out that the way in which subgrammars are defined enables us to define the control of rule applications in a transparent way.

The proposed modifications are completely in accordance with the basic principles mentioned in section 1.

3 Subgrammars, a Syntax Oriented Division

From the computer language Modula2 (cf. Wirth, 1985) we learned the essentials of the modular approach:

1. divide the total into smaller parts (**modules**) with a well-defined task,
2. define explicitly what is used from other parts (**import**) and what may be used by other parts (**export**),
3. separate the definition from the implementation.

The explicit definition of import and export and the strict separation of implementation and definition makes it possible to prove the correctness of a module in terms of its imports, without having to look at the implementation of the imported modules. This tackles the above-mentioned complexity problem and the coordination problem caused by the lack of structure in the M-grammars nicely. In our view, applying the modular approach to grammars comes down to the following requirements:

1. dividing the grammar into **subgrammars** with a well-defined linguistic task,

2. defining explicitly what is visible to other subgrammars (**export**) and what is used from other subgrammars (**import**),
3. ensuring that the actual implementation (i.e. the rules) is of local significance only.

Dividing grammars into subgrammars with a linguistic task has been done before, e.g. in the GETA-systems (cf. Vauquois and Boitet, 1985). However, to our knowledge, they do not meet requirement 2 and 3

The actual subdivision chosen for the development of Rosetta3 was inspired by the notion **projection** from the $\overline{X}$-theory of Transformational Generative Grammar (cf. e.g. Chomsky, 1970): every major category X is said to have a maximal projection X^{max}, e.g. NOUN has the maximal projection NP. Such projections provide a syntactic division of the constituents of language and appear to be a useful choice for modular units in a natural language system.

Applying this idea to the compositional grammars of Rosetta implies that basic expressions have a major category X and that there are syntactic rules that will ultimately compose S-trees of category X^{max}. For each maximal projection a subgrammar can now be defined that expresses how X^{max} can be derived from X and other imported categories. We will call a possible derivation process of the projection from X to X^{max} a **projection path** (see figure 5. The most important major categories (and their projections) in use in the Rosetta systems are: NOUN (NP), VERB (VP), ADJ (ADJP), ADV (ADVP) and PREP (PP).

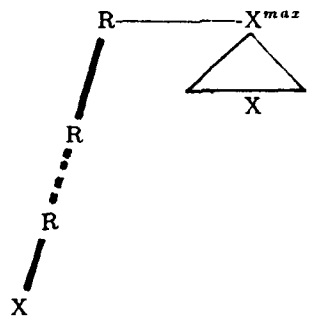

Figure 5: A projection path from X to X^{max}

$\overline{X}$-theory also states that all projections have a similar syntactic structure (i.e. phrase marker), which is represented in the schema of figure 6, but this aspect is less relevant for the Rosetta grammars. For us, it is of more interest whether they are the result of similar derivations. We will come back to this point in section 5.

A sentence is usually seen as a subject-predicate relation, i.e. a combination of an NP and a VP. But other

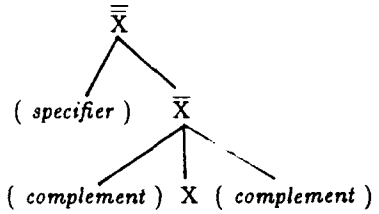

Figure 6: The projection of X to X^{max}

XP (i.e. X^{max}) categories than VP, together with an NP, can express a subject-predicate relationship as well (cf. Stowell, 1981). Such subject-predicate relations are called small clauses. For example, the NP *him* and the ADJP *funny* in *I think* [*him funny*], or the two NP's *him* and *a fool* in *I consider* [*him a fool*] form a small clause. In Rosetta such tenseless clauses are called XPPROP in which X stands for the X of the predicate. For example, in [*him funny*] we have ADJPPROP (with X = ADJ) and in [*him a fool*] we have NPPROP (with X = NOUN). A tensed XPPROP is called a CLAUSE in Rosetta. For example, in the sentences *I think that he is sleeping* and *I think that he is funny* we have the CLAUSEs [*that he is sleeping*] and [*that he is funny*] respectively.

This means that, starting from a basic expression of category X, in principle three S-trees with a different top category X^{top} can be derived: XP, XPPROP and CLAUSE. Figure 7 shows some of the resulting derivation trees and S-trees of the examples given above.

Defining subgrammars in accordance with these 'projection paths' provides a natural way of expressing the application order of the rules within a subgrammar: the order is defined with respect to the projection path only. A side effect of this explicit ordering of rule application is that it enables us to use a more efficient parse algorithm (M-PARSER).

A subgrammar can now be characterized as follows:

1. **export** S-tree of category X^{top} (XP, XPPROP or CLAUSE)
2. **import:**
   - S-tree with a special category, the X-category, also called the **head category**.
   - S-trees with categories that are exported by other subgrammars and that can be taken as an argument by rules with more than one argument.
3. **rules:** a set of rules that take care of the projection from X to X^{top}. Every rule has one argument, which is called the **head argument**, i.e. the S-tree with the head category or one of the intermediate results during its projection.

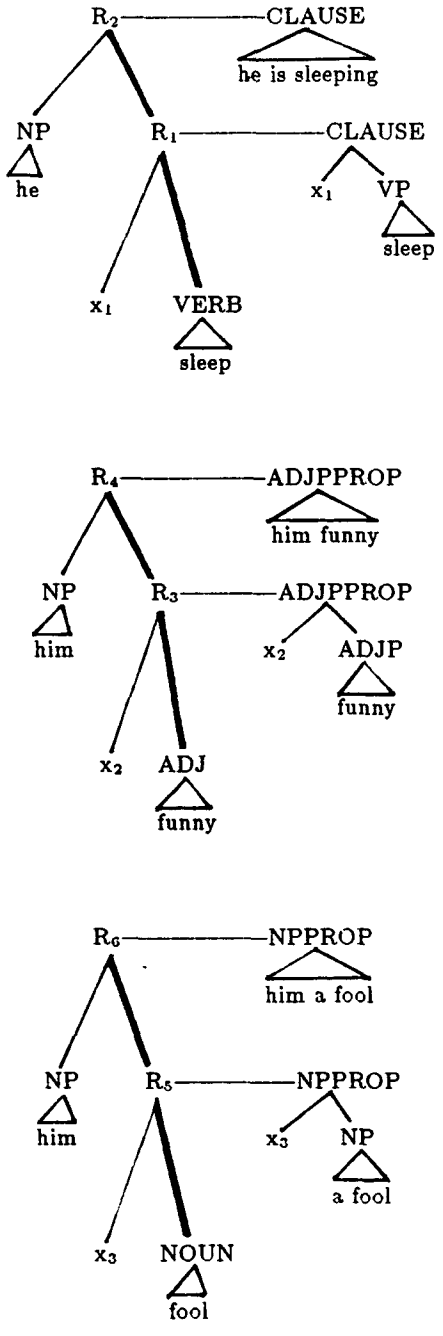

Figure 7: The derivation trees with the resulting S-trees of the projection of VERB to CLAUSE, ADJ to ADJPPROP and NOUN to NPPROP.

4. **control expression:** a definition of the possible application sequences of the rules, ordered with respect to their head arguments.

Neither the rules nor the intermediate results are known to other subgrammars. They can be considered local to the subgrammar. So 1 and 2 define the relation with other subgrammars, whereas 3 and 4 are only of local significance, thus meeting our requirements for the modular approach.

An example of a subgrammar is the NP-subgrammar with a NOUN as head and exporting an NP. Other categories that are imported by this subgrammar are DETP, ADJPPROP, etc. the set of rules contains modification rules and determiner rules. the control expression indicates that the modification rules can be applied recursively and that they precede the determiner rules.

Obviously, there will now be subgrammars that contain the same rules, e.g. the subgrammars for NOUN to NP and PRONOUN to NP. For efficiency reasons, it is allowed to merge such subgrammars by defining a set of heads as import and a set of top categories as export.

For an elaboration of the notion control expression and a formalisation of subgrammars we refer to section 6.

The advantages of this division into subgrammars are 1) that the structure of the grammar has become more transparent, 2) that we now have units with well-defined interfaces which enables us to divide the work over several people, 3) that we can work at and test smaller parts of a grammar.

4 Rule Classes, a Translation Oriented Division

In the Rosetta framework as sketched in section 1, the translation relation is defined at the level of rules and basic expressions. If there is a rule or basic expression in one grammar, there must be at least one rule or basic expression in the other grammar with the same meaning (the Isomorphy Principle). It is hard to get hold of the translation relation as a whole in terms of these primitives alone. What we need is some structure of a higher order.

1. We distinguish purely syntactic rules called **transformations** and **meaningful rules.**

   Some rules in the Rosetta grammars do not carry 'meaning', but serve only a syntactic, transformational purpose. In the Rosetta2 framework these meaningless rules, which are often of a highly language-specific character, sometimes required rules in other languages that were of no use there.

This point was already mentioned in section 2. Such rules are now no longer considered to be part of the translation relation that is expressed by the isomorphy relation between the grammars. Therefore, they can be added freely to a grammar. In this way a better distinction can be made between purely syntactic (and hence language-specific) knowledge and **translation relevant** knowledge. The translation relation now can be freed from improper elements, which is highly desirable.

In section 2 it was noticed that the introduction of transformation rules requires some way of controlling their applicability. The control expressions introduced in section 3 and formalised in section 6 provide for this.

2. The set of rules of the grammars are divided into groups called **rule classes**, each of which handles some linguistic phenomenon. These rule classes are subdivided into transformation classes and meaningful rule classes. A meaningful rule class handles a linguistic phenomenon of which the semantics should be preserved during translation.

   Such *translation relevant* linguistic phenomena are, e.g., valency/semantic relations, scope, time, negation and voice. The translation relation can be further structured by these meaningful rule classes. Only rules of different languages that belong to the same meaningful rule class may correspond to each other or, to put it in other words, rules that do not belong to the same meaningful rule class can never be translations of each other (see figure 8). Within a meaningful rule class there can, of course, be some 'semantic differentiation', which should be retained under translation. For example, in the time rule class more than one time reference can be distinguished, each with a distinct meaning. [1]

   There can also be 'corresponding' transformation classes in the grammars for different languages - e.g. agreement rules -, but they do not play a role in the translation relation.

5 Combining Subgrammars and Rule Classes

Having introduced some order into the syntactic rules of the grammar and into the translation relation, we see that these divisions of rules are along 'vertical' and 'horizontal' lines respectively (see figure 9). The projections of basic categories in one grammar, leading to the division of the grammar into subgrammars, are

[1] For each distinct time reference meaning a separate rule can be defined, but it is also possible to introduce abstract basic expressions ranging over the possible time references and have one rule that has such an abstract basic expression as argument.

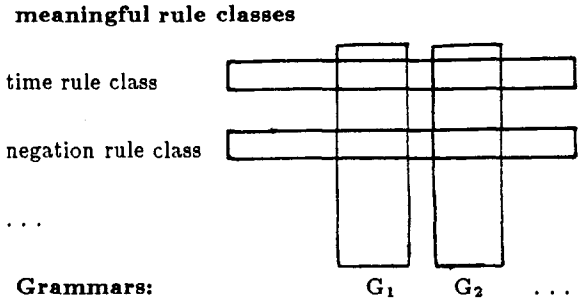

Figure 8: meaningful rule classes bring order in the translation relation between the grammars of the languages involved

along vertical lines. The relations between the grammars, leading to the division of all the rules of the grammars into (meaningful) rule classes, are along horizontal lines.

These two ways of dividing grammars have several consequences.

On the one hand, subgrammars help to structure the grammar in a more modular way; they also give some insight into the translation relation, but only in the more 'trivial' cases, where the corresponding basic expressions have the same syntactic category, subgrammar G_i of grammar G corresponds solely to subgrammar G_i' of grammar G'. In category mismatch cases the corresponding basic expressions fall into different subgrammars (e.g. the *graag/like* case of section 2).

On the other hand meaningful rule classes group together semantically related rules, which gives insight in what has to be preserved during translation, but they are not the right unit to make a modular structure. This makes it hard to define an adequate interface (import/export) between rule classes, because e.g. the rule that negates a sentence is determined more by the rules that form a sentence than by the other negation rules (e.g. in an adjective phrase) with which it forms the negation rule class.

However, both subgrammars and rule classes allow for a division of the labour over people. That this is the case with subgrammars is trivial, as subgrammars form a modular structure. The reason that rule classes are also useful units to divide the work is that knowledge of a specific linguistic topic is needed for every rule class, knowledge that can typically be handled by one person.

In order to have the benefits of both we combined subgrammars and rule classes in the following way:

1. the rules of subgrammars are divided into **rule subclasses**, which are subsets of rule classes
2. the application sequences of rules are defined in terms of rule subclasses instead of rules.

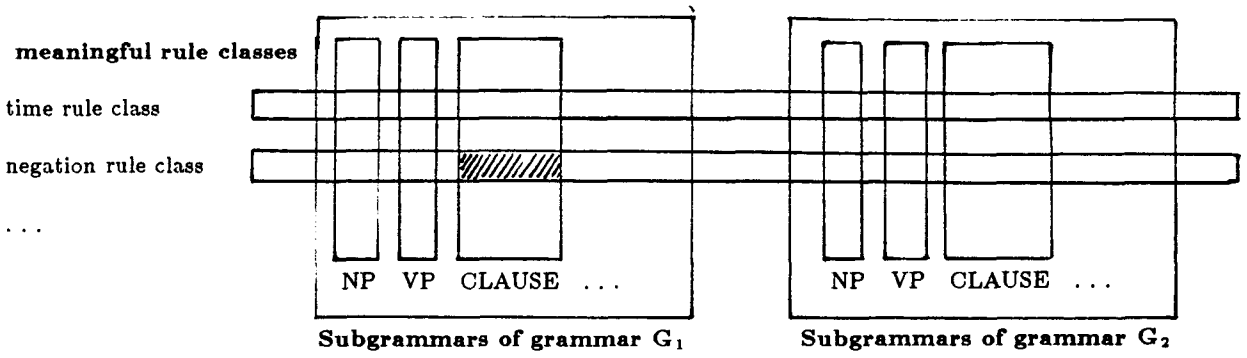

Figure 9: horizontal and vertical division within grammars. The shaded part denotes the subclass of the negation rules for the CLAUSE subgrammar of G_1.

The combination results in a modular structure of each grammar and helps to reduce the complexity of the translation relation. It also helps to solve the class of category mismatch problems elegantly.

Isomorphic subgrammars

As was already mentioned in section 3, $\bar{X}$-theory states that the projections of all major categories have a similar structure. The division of the grammars into subgrammars was based on the notion major category and the sorts of projections that we recognize (XP, XPPROP and CLAUSE). The fact that in $\bar{X}$-theory the phrase markers of the resulting constituents are similar, suggests that it is possible to assign similar derivations to them in a compositional grammar. This similarity is also suggested by the fact that most rule classes handle phenomena that play a role in every subgrammar. For example, in all subgrammars rules for valency/semantic relations and negation are found. They may differ, of course, in their transformations.

The fact that we consider the Dutch NPs *de ezel die appels eet* and *de appels etende ezel* to be paraphrases which are both translations of the English NP *the donkey that is eating apples*[2] suggests that a tensed relative clause should be composed similar to a tenseless 'adjectival' relative clause, or in other terms: that their derivation trees should be **isomorphic** with respect to their meaningful rules. The same can be said for the adjectival phrase *smart* and the relative clause *that is smart* in *the [smart] girl* and *the girl [that is smart]* respectively.

To make it possible that such phrases and clauses are translations of each other, the subgrammars involved are attuned as far as possible, resulting in 'isomorphic' subgrammars within one grammar.

We will discuss two cases:

1. Same head category, but different top category
2. Different head category

Same head category, but different top category

In the example of *the smart girl / the girl that is smart* the subgrammars for the projection of ADJ to ADJPPROP and ADJ to CLAUSE are involved. They differ in that a transformation exists for the insertion of the auxiliary verb *be* in the clause case. For isomorphy reasons, in both cases a rule for time reference is needed: in the clause case it spells out the tense for the verb *be*; in the adjectival case it seems to be superfluous, but with model-theoretical semantics in mind it can be argued to be needed, if we assume a model with a time component (see figure 10).

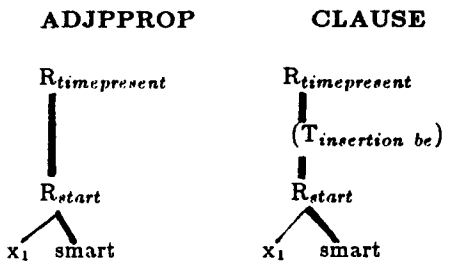

Figure 10: Derivation trees resulting from the subgrammars ADJ to ADJPROP and ADJ to CLAUSE

This kind of paraphrasing can be helpful if the literal translation is not allowed in the other language as is the case with *de appels etende ezel*, which cannot be translated into **the eating apples donkey* or *I expect him to leave* which cannot be translated into **ik verwacht hem te vertrekken*, but has to be translated into *ik verwacht dat hij vertrekt* (*I-expect-that-he-leaves*).

[2] *the eating apples donkey* is ungrammatical.

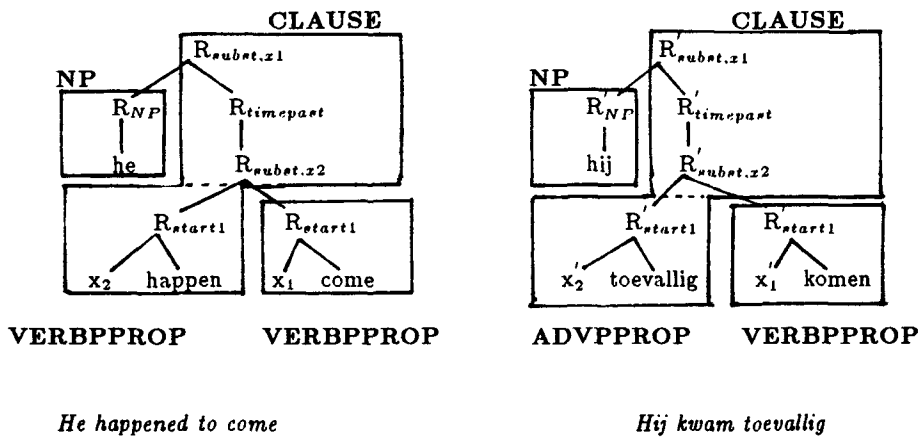

Figure 11: Syntactic derivation trees with the relevant subgrammars

Different head category

If the approach of attuning subgrammars as far as possible is extended to subgrammars with different head categories, then it can help to solve the problems with the above-mentioned class of category mismatch cases.

For example, in *he happened to come* the raising verb *happen* occurs and in *hij kwam toevallig* the sentential adverb *toevallig*. As these two sentences are considered translations of each other, the subgrammars for VERB and ADV should be attuned to each other. This seems to be impossible because it is quite natural that the complement of *happen*, i.e [*he come*] is inserted into the clause of *happen*, whereas *toevallig* (the basic expression that corresponds to *happen*) is inserted into the clause corresponding to the complement clause of *happen*, i.e [*hij komen*].

Semantically, in both cases, the clause is the argument of *happen* and *toevallig*, but from a syntactic viewpoint adverbs are inserted into their arguments. We can solve this problem by allowing in these cases a 'switch of subgrammar'. This is possible if the subgrammars are split into two parts in such a way that the place of this subdivision coincides with the 'switch point'. There is another argument for making this subdivision: the first part of the control expression of the subgrammars for X to XPPROP and X to CLAUSE is the same. The succeeding part is in the CLAUSE case very similar for all X.

Figure 11 sketches how the examples *He happened to come* and *Hij kwam toevallig* now can be derived isomorphically.

We noticed that this kind of mismatch of syntactic category appears most frequently with modal verbs and adverbs, auxiliaries and semi-auxiliaries, at least in the languages Rosetta deals with (Dutch, English and Spanish). In translation systems dealing with Japanese and English these phenomena occur more frequently. (cf. Nagao and Tsujii, 1986).

Partial isomorphy of subgrammars

Isomorphy between grammars of different languages must be defined in terms of isomorphy between the subgrammars of these languages. It should be noted that it is not always possible to make a subgrammar of one language completely isomorphic to one subgrammar of the other language. However, it is possible to make subgrammars partially isomorphic and sets of subgrammars completely isomorphic, both within one language and between different languages. For example, within one language the subgrammars for ADJ to CLAUSE and ADJ to ADJPPROP need not be completely isomorphic, neither do the ones for ADV to CLAUSE and VERB to CLAUSE. But together the subgrammars for ADJ to CLAUSE and ADJ to ADJPPROP for Dutch can be completely isomorphic to the corresponding subgrammars for English.

6 Formal aspects

In this section we will discuss the main consequences for the Rosetta formalism of the ideas put forward in sections 3, 4 and 5. These consequences relate in particular to the definition of M-PARSER and M-GENERATOR. We will first give - in section 6.1 - the original definitions for the free, i.e. 'uncontrolled' M-grammars of Rosetta2. In 6.2 we will give the revised definitions for controlled M-grammars, currently used for the development of Rosetta3.

6.1 Free M-grammars

The syntactic component of an M-grammar defines a set of objects called S-trees (surface trees).

An **S-tree** is
- a node N,
- or an expression of the form

$$N[r_1/t_1,\ldots, r_n/t_n] \qquad (n>0)$$

where N is a node, the r_i's are syntactic relations and the t_i's are S-trees.

(we will often use this kind of recursive definition: the second - recursive - part of the definition indicates that S-trees may have arbitrary, but finite, depth; the first part shows how the recursion terminates: the leaves of the trees are always (terminal) nodes)

A node N is defined as a syntactic category followed by a tuple of attribute-value pairs $(a_i:v_i)$.

$$N = C\{a_1:v_1,\ldots, a_k:v_k\} \qquad (k\geq 0)$$

For each syntactic category the corresponding attributes are defined, for each attribute the set of possible values is defined. So, given a set of syntactic relations and a set of syntactic categories with the corresponding attributes and values, the set of possible S-trees is defined. This set is called T: the domain of S-trees.

So the general form of an S-tree t is

$$t = C\{a_1:v_1,\ldots, a_k:v_k\}\ [r_1/t_1,\ldots, r_n/t_n]$$

C is called the syntactic category of t.

The syntactic component of an M-grammar defines

- the domain T by enumerating the syntactic relations, the categories and the corresponding attributes and values.
- the set T_M of well-formed S-trees, a subset of T. T_M consists of the surface trees of sentences that are well-formed according to the grammar.

T_M is defined by specifying:

1. a set B of basic S-trees,
2. a set of syntactic rules, called M-rules,
3. a special category: SENTENCE.

ad 1. The set of basic S-trees is a subset of T (the basic lexicon). A basic S-tree b has a unique name, to be denoted as $\underline{b}$.

ad 2. An M-rule R_i defines a compositional function F_i from tuples of S-trees to finite sets of S-trees. So application of R_i to a tuple $t_1,\ldots,t_n$ yields a set $F_i(t_1,\ldots,t_n)$. The set is empty if the rule is not applicable.

Each M-rule is reversible, i.e. it also defines an analytical function F_i', the reverse of F_i.

$t \in F_i(t_1,\ldots,t_n) \iff (t_1,\ldots,t_n) \in F_i'(t)$

S-trees are constructed by applying M-rules recursively, starting from basic expressions, The set T_M is the set of S-trees that can be derived in this way and that have the category SENTENCE.

The derivation process can be displayed in a syntactic derivation tree.

A **derivation tree** is
- the name $\underline{b}$ of a basic expression,
- or an expression of the form

$R_i<d_1,\ldots,d_n>$, $\qquad (n>0)$,

where R_i is a rule name and $d_1,\ldots,d_n$ are derivation trees.

On the basis of the syntactic component of an M-grammar the functions M-GENERATOR and M-PARSER can be defined. M-GENERATOR is applied to a derivation tree and yields a set of S-trees; M-PARSER is applied to an S-tree and yields a set of derivation trees.

M-GENERATOR(d) = $_{def}$

$$\{\, t \mid \exists\, b \in B\colon d=\underline{b} \text{ and } t=b \,\}$$

$$\cup\ \{\, t \mid \exists\, t_1,\ldots,t_n,\ d_1,\ldots,d_n, R_i :\ d = R_i<d_1,\ldots,d_n> \text{ and } t_1 \in \text{M-GENERATOR}(d_1) \text{ and } \ldots \text{ and } t_n \in \text{M-GENERATOR}(d_n) \text{ and } t \in F_i(t_1,\ldots,t_n) \,\}$$

(In this definition d, d_1, d_n are derivation trees, t, t_1, t_n are S-trees, B is the set of basic S-trees, b is a basic S-tree, $\underline{b}$ is the name of a basic expression, F_i is the compositional function defined by rule R_i)

M-PARSER(t) $=_{def}$

$$\{\, d \mid \exists\, b \in B\colon t=b \text{ and } d=\underline{b} \,\}$$

$$\cup\ \{\, d \mid \exists\, t_1,\ldots,t_n, d_1,\ldots,d_n, R_i :\ (t_1,\ldots,t_n) \in F_i'(t),\ d_1 \in \text{M-PARSER}(t_1) \text{ and } \ldots \text{ and } d_n \in \text{M-PARSER}(t_n) \text{ and } d = R_i<d_1,\ldots,d_n> \,\}$$

(F_i' is the analytical function defined by rule R_i)

Given the reversibility of the M-rules, it is easy to prove that

$t \in \text{M-GENERATOR}(d) \iff d \in \text{M-PARSER}(t)$

Note that M-PARSER and M-GENERATOR can both be used to define the set T_M of well-formed S-trees. T_M can be defined as the set of S-trees (of category SENTENCE) that can be derived by applying M-GENERATOR to all possible derivation trees. T_M can also be defined as the set of S-trees (of category SENTENCE) for which M-PARSER yields at least one derivation tree. Because these definitions are equivalent, the One Grammar Principle is obeyed.

An M-grammar has to obey the **measure condition**:

First, a measure on S-trees, i.e. a function from S-trees to positive integers, must be defined.

The condition says: if t is the result of applying a compositional rule to S-trees $t_1,\ldots,t_n$ then t is bigger according to this measure than each of the arguments $t_1,\ldots,t_n$. So application of an analytical rule yields smaller S-trees, which guarantees that the recursion in M-PARSER is finite.

The algorithms for the components M-PARSER and M-GENERATOR follow directly from the set-theoretic definitions.

6.2 Controlled M-grammars

The syntactic component of a **controlled M-grammar** defines the domain T of S-trees in the same way as for free M-grammars.

The set T_M of well-formed S-trees is defined by specifying:

1. a set B of basic S-trees,
2. a set of subgrammars,
3. a special category: SENTENCE.

A **subgrammar** G_i consists of:

- a set $EXPORTCATS_i$ of syntactic categories (the categories of S-trees that can be exported).
- a set $HEADCATS_i$ of syntactic categories (the categories of S-trees that are allowed as the head),
- a set $IMPORTCATS_i$ of syntactic categories (the categories of other S-trees that may be imported),
- a set of M-rules, subdivided into a set $MF\text{-}RULES_i$ of meaningful rules and a set $TR\text{-}RULES_i$ of transformation rules. For each meaningful M-rule one of the arguments has to be defined as the 'head' argument (transformations have only one argument). For notational convenience we will assume here that the head is always the first argument.
- a control expression ce_i, which indicates what sequences of rule applications are possible, from imported head to exported result. (The ordering of the rules concerns the head arguments)

The **control expression** indicates what the rule subclasses are, how they are ordered, what rules they consist of, and whether they are recursive, optional or obligatory. A control expression ce has the following form:

$ce = A_0 \,.\, A_1 \,.\, \ldots \,.\, A_n$,

where each A_i is a rule subclass, either a meaningful rule class or a transformation class.

A rule class A_i may be

- obligatory: written as $(R_1 \mid \ldots \mid R_k)$, where the R_i are either meaningful rules or transformations.
- recursive: written as $\{ R_1 \mid \ldots \mid R_k \}$,
- optional: written as $[R_1 \mid \ldots \mid R_k]$.

An example:

$$(R_1) \,.\, [R_2 \mid R_3] \,.\, \{ R_4 \mid R_5 \} \,.\, (R_6 \mid R_7)$$

This control expression defines all sequences beginning with R_1, then R_2 or R_3 or neither, then an arbitrarily long sequence (possibly empty) of R_4 or R_5, then either R_6 or R_7.

Actually a control expression is a restricted kind of regular expression over the alphabet of rule names. (It is not restricted in its constructions but in the possible combinations of these constructions.) Each regular expression denotes a set of instances: sequences of rule names. Each such rule sequence is a possible **projection path** in the subgrammar (cf. section 3). Note that the rules in a sequence need not be applicable, this depends on the applicability conditions of the rules themselves.

It is required that each instance of the regular expression contains at least one meaningful rule.

The definition of a derivation tree has to be adjusted as follows.

A **derivation tree** is:

- the name b of a basic expression,
- or an expression of the form

$(G_i,R_j)\langle d_1,\ldots,d_n\rangle \qquad (n>0)$,

where G_i is (the name of) a subgrammar, R_j is (the name of) a meaningful M-rule, and $d_1,\ldots,d_n$ are derivation trees.

There are two differences with the old definition. The first is that the non-terminal nodes contain the subgrammar name, next to the rule name. The second is that the derivation tree is no longer a complete trace of rule applications, because the transformations are not indicated explicitly.

In the revised definition of M-GENERATOR and M-PARSER we will use a kind of incomplete derivation tree, defined as follows.

An **open derivation tree** is:

- the 'empty derivation tree', D_E.

- or an expression of the form
$(G_i,R_j)<D_1,\ldots,D_n>$,

where G_i is the name of a subgrammar, R_j is the name of a meaningful M-rule, D_1 is an open derivation tree, $D_2,\ldots,D_n$ are derivation trees.

So an open derivation tree is like an ordinary derivation tree, but with an empty derivation tree as leftmost leaf.

Where this is useful we will refer to an ordinary derivation tree as a **closed** derivation tree.

Be given open derivation trees D_1 and D_2.

We define $D_1[D_2]$ as the open derivation tree that results if D_2 is substituted for D_E in D_1.

If D_2 is a closed derivation tree, the result of the substitution $D_1[D_2]$ is a closed derivation tree.

We will now present the revised definitions of M-PARSER and M-GENERATOR, for controlled M-grammars. The definitions are not only valid for the restricted control expressions, but in fact for any regular expression. Here the set-theoretical definitions are given, the algorithms can be derived from them directly. The set T_M of well-formed S-trees can be defined in terms of these functions, in the same way as in subsection 6.1.

Revised definition of M-PARSER

First we will give an informal description of M-PARSER, in an 'operational' way.

M-PARSER operates on an S-tree t. If t is a basic expression b, M-PARSER(t) yields the derivation tree $\underline{b}$. For a non-basic t M-PARSER(t) tries to apply subgrammar parsers, by calling SG-PARSER(G_i,t) for all subgrammars G_i with the appropriate export categories (note that for the analytical functions the export categories indicate what can be 'imported'). SG-PARSER(G_i, t) tries to apply the rules of control expression ce_i to t (i.e. the analytical versions of the rules, starting at the right of the control expression).

A successful application of SG-PARSER yields a pair (D, u), where D is an open derivation tree and u is the resulting 'head' S-tree.

To u M-PARSER is applied again. If successful, M-PARSER(u) yields a derivation tree d. Then D[d] is a derivation tree of t.

SG-PARSER is defined by means of a function CE-PARSER. CE-PARSER has 4 arguments (G_i, ce, D, t), where G_i is a subgrammar name, ce is a control expression, D is the open derivation tree resulting from previous applications of CE-PARSER, t is the S-tree that is yet to be parsed.

When CE-PARSER is called for the first time, D is the empty derivation tree and ce is the control expression ce_i of G_i.

CE-PARSER(G_i, ce, D, t) tries to apply the (analytical versions of the) rules of control expression ce to t, in right-to-left order. Successful application of a rule yields a tuple of S-trees $t_1,\ldots,t_n$. To t_1 the 'next' rule of the control expression is applied. To $t_2,\ldots,t_n$ the full M-PARSER is applied. During the recursive application of CE-PARSER D grows while ce shrinks.

Application of a meaningful rule R_j leads to substitution of a new node (G_i,R_j) in D. Application of a syntactic transformation does not change the derivation tree.

The result of applying CE-PARSER successfully to (G_i, ce_i, D, t) is a triple (D_2, u, A). All rules of one instance of ce_i have been applied then. D_2 has the form $D[D_1]$, where D_1 is the open derivation tree with the meaningful rules of this instance of ce_i at its projection path and u is the remaining S-tree to be parsed yet (the 'head'). A is a boolean, which tells whether a rule (or transformation) has been applied. This is needed to avoid vacuous recursion of CE-PARSER in case of control expressions of the form { ce }, where ce has empty instances, e.g. if ce has itself the form ce_1 . The boolean A would not be needed if the definitions would be tuned to the restricted form of control expressions as a sequence of rule classes.

The definitions:

M-PARSER(t) $=_{def}$
{d | $\exists$ b $\in$ B: d = $\underline{b}$ and t = b }
$\cup$ { d | $\exists$ G_i, d_1, D_2, u:
syncat(t) $\in$ EXPORTCATS$_i$ and
(D_2, u) $\in$ SG-PARSER(G_i,t) and
d_1 $\in$ M-PARSER(u)
and d = $D_2[d_1]$ }

(In this definition d, d_1 are closed derivation trees, D_2 is an open derivation tree, t, u are S-trees, syncat(t) is the syntactic category of t, b is a basic expression, $\underline{b}$ is the name of a basic expression, G_i is a subgrammar)

SG-PARSER(G_i,t) $=_{def}$
{ (D, u) |
(D, u, true) $\in$ CE-PARSER(G_i, ce_i, D_E, t)}

(ce_i is the control expression of ce_i)

CE-PARSER(G_i, ce, D, t) $=_{def}$

{ (D_2, u, A) | $\exists$ ce_1, ce_2, D_1, t_1:
ce = ce_1.ce_2 and "ce_2 is not a concatenation" and
(D_1, t_1, A_1) $\in$ CE-PARSER(G_i, ce_2, D, t) and
(D_2, u, A_2) $\in$ CE-PARSER(G_i, ce_1, D_1 ,t_1) and
A = A_1 or A_2 }

$\cup$ { (D_2, u, A) | $\exists$ ce_1, ce_2:
ce = ce_1|ce_2 and "ce_2 is not a disjunction" and
(D_2, u, A) $\in$ (CE-PARSER(G_i, ce_2, D, t)

$\cup$ CE-PARSER(G_i, ce_1, D, t)) }

$\cup$ { (D_2, u, A) | $\exists$ ce_1:
ce = $\lfloor ce_1 \rfloor$ and
((D_2 = D and u = t and A = false) or
(D_2, u, true) $\in$ CE-PARSER(G_i, ce_1, D, t) and
A = true) }

$\cup$ { (D_2, u, A) | $\exists$ ce_1:
ce = {ce_1} and
((D_2 = D and u = t and A = false) or
($\exists$ D_1, t_1, A_1:
(D_1, t_1, true) $\in$ CE-PARSER(G_i, ce_1, D, t) and
(D_2, u, A_1) $\in$ CE-PARSER(G_i, ce, D_1, t_1) and
A = true)) }

$\cup$ { (D_2, u, true) | $\exists$ k, n, R_k, d_2,...,d_n:
ce = R_k and
$R_k \in$ MF-RULES$_i$ and
D_2 = D[(G_i, R_k)<D_E, d_2,...,d_n>] and
(u, t_2,...,t_n) $\in$ F'_k(t) and
$d_2 \in$ M-PARSER(t_2) and
...
$d_n \in$ M-PARSER(t_n) }

$\cup$ { (D_2, u, true) | $\exists$ k, R_k:
$R_k \in$ TR-RULES$_i$ and
D_2 = D and ce = R_k and
u $\in$ F'_k(t) }

(ce, ce_1, ce_2 are control (sub)expressions, D, D_1, D_2 are open derivation trees, d_2,...,d_n are closed derivation trees, D_E is the empty derivation tree, t, t_1, t_2, t_n, u are S-trees, R_k is an M-rule, F'_k is the analytical function defined by rule R_k, A, A_1, A_2 are booleans)

An additional advantage of controlled M-grammars is that the measure condition (cf. 6.1) can be reformulated in a way that is much easier to obey that in the original framework.

The measure condition is reformulated as follows:

1. For the grammar as a whole a measure must be defined in such a way that application of a subgrammar in generation yields exported S-trees which are bigger than the imported S-trees. Consequently application of a subgrammar during analysis yields smaller S-trees. This measure is similar to the measure for rules we had in the free M-grammar formalism, but it is easier to define a measure for complete subgrammars than for rules. Possible measures are: the total number of nodes or the depth of an S-tree.

2. For each subexpression of the form { e } in a control expression a measure on S-trees must be defined, such that application of e during analysis yields output S-trees that are smaller than the argument S-trees according to this measure. This measure can be defined separately for each expression { e }.

Revised definition of M-GENERATOR

As the definitions relating to M-GENERATOR are completely symmetric to the definitions relating to M-PARSER, we will present them without further comments.

M-GENERATOR(d) $=_{def}$
{ t | $\exists$ b $\in$ B: d = $\underline{b}$ and t = b }
$\cup$ { t | $\exists$ G_i, d_1, D_2, u:
d = $D_2[d_1]$ and
u $\in$ M-GENERATOR(d_1) and
syncat(u) $\in$ HEADCATS$_i$ and
t $\in$ SG-GEN(G_i, D_2, u) }

(d, d_1 are closed derivation trees, D_2 is an open derivation tree, t, u are S-trees, b is a basic expression, $\underline{b}$ is the name of a basic expression, G_i is a subgrammar, syncat(u) is the syntactic category of u)

SG-GEN(G_i, D, u) $=_{def}$
{t | (t, D_E, true) $\in$ CE-GEN(G_i, ce_i, D, u)}

(ce_i is the control expression of G_i, D_E is the empty derivation tree)

CE-GEN(G_i, ce, D_2, u) $=_{def}$

{ (t, D, A) | $\exists$ ce_1, ce_2, D_1, t_1:
ce = ce_1.ce_2 and "ce_1 is not a concatenation" and
(t_1, D_1, A_1) $\in$ CE-GEN(G_i, ce_1, D_2, u) and
(t, D, A_2) $\in$ CE-GEN(G_i, ce_2, D_1, t_1) and
A = A_1 or A_2 }

$\cup$ { (t, D, A) | $\exists$ ce_1, ce_2:
ce = ce_1|ce_2 and "ce_1 is not a disjunction" and
(t, D, A) $\in$ (CE-GEN(G_i, ce_1, D_2, u)
$\cup$ CE-GEN(G_i, ce_2, D_2, u)) }

$\cup$ { (t, D, A) | $\exists$ ce_1:
ce = $\lfloor ce_1 \rfloor$ and
((D_2 = D and t = u and A = false) or
((t, D, true) $\in$ CE-GEN(G_i, ce_1, D_2, u) and
A = true)) }

$\cup$ { (t, D, A) | $\exists$ ce_1:
ce = {ce_1} and
((D_2 = D and t = u and A = false) or
($\exists$ D_1, t_1, A_1:
(t_1, D_1, true) $\in$ CE-GEN(G_i, ce_1, D_2, u) and
(t, D, A_1) $\in$ CE-GEN(G_i, ce, D_1, t_1) and
A = true)) }

$\cup$ { (t, D, true) | $\exists$ k, n, R_k, d_2,...,d_n:
ce = R_k and

$R_k \in$ MF-RULES$_i$ and
$D_2 = D[(G_i, R_k)<D_{\not\in}, d_2,\ldots,d_n>]$ and
$t \in F_k(u,t_2,\ldots,t_n)$ and
$t_2 \in$ M-GENERATOR(d_2) and
. . .
$t_n \in$ M-GENERATOR(d_n) }

$\cup$ { (t, D, true) | $\exists$ k, R_k:
$R_k \in$ TR-RULES$_i$
$D = D_2$ and
$ce = R_k$ and $t \in F_k(u)$ }

(ce, ce_1, ce_2 are control expressions, D, D_1, D_2 are open derivation trees, $d_2,\ldots,d_n$ are closed derivation trees, $D_{\not\in}$ is the empty derivation tree, t, t_1, t_2, t_n, u are S-trees, R_k an M-rule, F_k is the compositional function defined by rule R_k, A, A_1, A_2 are booleans)

Remarks

In case of a recursive transformation class there is the possibility of infinite recursion during application of CE-GEN. This must be prevented by defining a measure on S-trees in such a way that each application of a transformation of the class yields a smaller S-tree according to this measure.

The definition of M-GENERATOR is symmetric to the definition of M-PARSER. (There is one apparent exception: the condition on EXPORTCATS in M-PARSER and the condition on HEADCATS in M-GENERATOR. However, these conditions are redundant from a formal point of view, because they must follow from the applicability conditions of the rules in the control expression.) Thanks to this symmetry it is simple to prove that M-GENERATOR and M-PARSER are each other's reverse. One of the virtues of this way of controlling rule applications is that the One Grammar Principle can still be obeyed.

7 Conclusion

In section 2 we enumerated three types of problems with the free M-grammar formalism used for the development of the Rosetta1 and Rosetta2 systems.

The first problem was the lack of structure in free M-grammars. This was solved in section 3 by introducing a modular approach, where M-grammars are divided into subgrammars in a way that was inspired by the programming language Modula-2 on the one hand and by the notion projection from $\overline{X}$-theory on the other hand.

The second problem was that there is no way of explicitly controlling the application of rules in free M-grammars and that it is not obvious how this kind of control could be introduced in a compositional grammar, where rules may have more than one argument. The insight that was important to the solution of this problem was that application of a subgrammar comes down to following a projection path, from the imported head to the exported projection. This implies that defining control in a subgrammar comes down to specifying a set of possible sequences of rule applications, which can be done by means of a control expression, a regular expression over rule names. An important advantage of this way of controlling rule applications is that the One Grammar Principle is still obeyed: the same grammar (i.e. the same subgrammars: the same rules, the same control expressions, etc.) can be used for the compositional and the analytical definition of a language. This is proved by the formal definitions in subsection 6.2.

The third problem concerned the consequences of defining the translation relation by means of isomorphic grammars. The introduction of an explicit distinction between meaningful rules and syntactic transformations in section 4 avoids unnecessary complications of the grammars without affecting the Principle of Isomorphy. Because the applicability of syntactic transformations is restrained by the control expressions, they do not cause problems with effectivity or efficiency. The introduction of rule classes gave more insight into complex translation relations. In section 5 it was shown that category mismatch problems can be handled more systematically by a combination of subgrammars and rule classes.

Acknowledgements

The authors would like to thank all the members of the Rosetta team for their constructive criticism. In particular we want to mention the invaluable contributions of Jan Odijk with respect to linguistic matters.

References

Appelo, L. and J. Landsbergen (1986), *The Machine Translation Project Rosetta*, Philips Research M.S. 13.801, Proceedings First International Conference on State of the Art in Machine Translation, Saarbrücken, pp. 34-51.

Chomsky, N. (1970), *Remarks on Nominalisation*, in R.A. Jacobs and P.S. Rosenbaum (eds), Readings in English Transformational Grammar, Georgetown University Press, Washington DC, pp. 184–221.

Isabelle, P. and E. Macklovitch (1986), *Transfer and MT Modularity*, Proceedings Coling 1986, Bonn, pp. 115–117.

Landsbergen, J. (1984), *Isomorphic grammars and their use in the Rosetta translation system*, Philips Research M.S. 12.950. Paper presented at the Tutorial on Machine Translation, Lugano. To appear in M. King (ed),

Machine Translation the state of the art, Edinburg University Press.

Leermakers, R. and J. Rous (1986), *The Translation Method of Rosetta*, Computers and Translation, Vol. 1, Number 3, pp. 169-183.

Nagao, M. and J. Tsujii (1986), *The Transfer Phase of the MU Machine Translation System* Proceedings Coling 1986, Bonn, pp. 97–103.

Stowell, T. (1981), Origins of Phrase Structure, Ph. D. dissertation, MIT.

Thomason, B. (1974), Formal Philosophy. Selected Papers of Richard Montague, Yale University Press, New Haven.

Vauquois, B. and C. Boitet (1985), *Automated Translation at Grenoble University*, Computational Linguistics, Vol. 11, Number 1, pp. 28-36.

Wirth, N. (1985), Programming in Modula-2, Springer-Verlag, third corrected edition.