EUROTRA: A MULTILINGUAL SYSTEM UNDER DEVELOPMENT

Rod Johnson

Centre for Computational Linguistics
University of Manchester Institute of Science and Technology
PO Box 88
Manchester M60 1QD, England

Maghi King

Institut Dalle Molle pour les etudes cognitives et semantiques
Université de Geneve
54 route des Acacias
CH-1227 Geneve, Switzerland

Louis des Tombe

Instituut voor Algemene Taalwetenschap
Rijksuniversiteit Utrecht
Trans 14
3512 JK Utrecht, Holland

ACKNOWLEDGMENT

None of the authors could claim ownership of the work and ideas presented here. They are the result of a collaborative effort over several years. Our thanks should go, therefore, to all our EUROTRA colleagues, and especially to Sergei Perschke for his help in preparing this paper. The authors are of course, solely responsible for misreporting, inaccuracies or errors.

1 INTRODUCTION

This paper tries to give an overview of the state of EUROTRA development at the beginning of 1985. The system is under development in the sense that the project is now an official project of the European community, much basic preliminary work has been done, and work in the Member States has started. However, as will be seen, linguistic work is in a very early stage, and no very concrete results are as yet available.

The paper is composed as follows. In the first two sections, we describe the project's purpose, some of the external factors influencing its organisation, its infrastructure and some of the design principles deriving from these general considerations.

Section 3 discusses in more detail the model of translation on which the project is based and some of the linguistic and metalinguistic issues involved.

In section 4 we turn to the issue of communication between the linguists developing the linguistic modules and the computer. It presents our choice of a software environment that allows rapid development and modification of problem-oriented notations for linguists.

Section 5 describes the architecture of the current software prototype, which has been implemented following the principles outlined in section 4.

Since EUROTRA, as already noted, is barely out of its preparatory phase, the final sections do little more than sketch some factors that will determine the project's future development.

2 PROJECT HISTORY, BACKGROUND, AND ORGANISATION

EUROTRA is a research and development programme of the European Communities. It is intended to produce a comparatively small operational prototype translating operational prototype translating from each of the seven official languages of the Community into any one of the other seven languages. The programme period of five

0362-613X/85/020155-169$03.00

and a half years started in November 1982. The total budget is roughly equivalent to 25 million dollars.

The system is to be developed cooperatively, by independent teams working in each of the Member States. Seven teams, some of them involving more than one Member State, take primary responsibility for work on their own language. Two smaller teams carry out support work used by all the other groups, such as acting as a software clearing house or as expert terminologists. A further group, independent of the organisation in the Member States, ensures coordination, is responsible for the linguistic framework within which the language groups agree to work, and provides documentation both for the software and for the linguistics. Software specifications also are being developed by an independent team.

At the time of writing (May 1985), seven of the Member States of the Community have already signed the contracts of association under which the linguistic work in the Member States is to be done. Signature of the remaining three is expected in the near future.

The first peculiarity of EUROTRA can already be seen from this: it is very much a linguists' project. The average size of a group working on a language will be on the order of 10-12 people. This gives a total of around a hundred linguists – a term *that* should be taken very broadly as including linguisticians, translators, lexicographers and so on – as against perhaps ten people working on software. This fact has had a profound influence on the system design.

Development is broken down into three phases, of which the first is drawing to its end:

1. **1983-1984:**
   The main tasks to be achieved during this period were the definition of a basic software to be used by all the groups and a preliminary definition of the linguistic framework to serve as the basis for work during the second phase.

   The manpower available during this phase reflects the concentration on setting up the basic framework. Some twelve people have been involved in producing software and linguistic specifications, whilst the language groups remained comparatively small, mostly equivalent to two or three people.
2. **1985-1986:**
   During this period a first working system is to be produced, taking all the seven languages as potential source and target languages. This system will be corpus based, and cover approximately 2,500 lexical entries for each of the languages. The corpus will be a Community text in the domain of information technology. The choice of a corpus is constrained by two factors: equivalent versions of the corpus must be available in all of the languages, and, although the subject area should be technical, it should be an area with which the people in the language groups are reasonably familiar in order to minimize dependence on outside help for technical terminology. Certain Community texts fulfil both these conditions, and at the same time provide an interesting range of subtext-types, which is important in demonstrating the generalisability of the first system. It is perhaps superfluous to remark that testing and evaluation of the first system will not be based on the corpus alone. The system's ability to translate similar texts in the same subject area and with the same vocabulary will also be taken into account.

   The construction of the first system serves two main purposes. First it demonstrates the feasibility of the approach taken. A number of aspects are relevant here: it is laid down in the requirements the system must meet that it must, as well as being multilingual, be easily extensible in order to incorporate new subject areas, new text types, new linguistic techniques, and even new languages. To this end the system is designed to be as modular as possible and also puts a great deal of emphasis on the declarative representation of linguistic knowledge. Extreme modularity, besides ensuring easy extensibility, also makes it easy to repair the system, even, if necessary, re-writing sub-parts completely.

   Secondly, the construction of the first system provides the feed-back needed to refine and stabilize the linguistic framework. Of our seven languages – Danish, Dutch, English, French, German, Greek and Italian – most have received comparatively little study within computational linguistics, and for nearly all of the language pairs contrastive work is almost entirely lacking. For this reason, many of the decisions taken about linguistic representation during the first phase cannot be known to be correct until they have been thoroughly tested during the second phase: indeed it is a fair assumption that they will have to be modified. Hence the emphasis in section three on the tentativeness of current proposals. Firm decisions will be made only towards the end of the second phase.
3. **1987-1988:**
   In this phase, the real prototype system will be constructed. Once again, it will cover all the seven languages and will be corpus based, but the vocabulary will be approximately 20,000 lexical entries per language. The time allowed for this phase is only a year and a half (the programme period stops in mid-1988), so the importance of establishing reliable linguistic models by the end of the second phase is clear. Note that the prototype is still only a prototype rather than an industrial system. During this last phase the question of industrial development will be considered for the first time.

   The twin constraints of multilinguality and decentralized development essentially determine some fundamental choices in system design. The system will be transfer based, but in the interests of keeping the transfer components as small as possible an attempt has been made in the definition of the

linguistic representation that is input to transfer, to determine what linguistic information can be considered interlingual across the European languages and to treat it as such. For example, differences in syntactic structure between the languages are effectively neutralized before transfer, emphasis has been put on an attempt to define an adequate set of semantic relations common to all the languages, to define language independent representations of time, modality and so on. In short, the linguistic representation is semantically based, and as close to an interlingua as seems possible, whilst still recognising that a pure interlingua based system is out of the question. The primary aim of defining a representation rich in semantic information is always to reduce the size and complexity of the transfer components. In fact, in the ideal (but unrealistic) case, transfer would consist of simple lexical substitution. The desirability of keeping transfer small is obvious on purely economic grounds (42 transfer modules versus 7 analysis and 7 generation modules). But at the same time, the extraction and use of interlingual information should do much to improve the quality of the translation.

The system is designed to be integrated into an overall text-handling system suitable for use in a heavily pressed translation service. Thus, it is not intended that there should be any significant human intervention during the translation process, although, of course, the possibility of human revision of the output text is allowed for. The amount of post editing to be foreseen depends on the use to which the text is to be put. For simple text, or text to be used for information gathering, there should be little or none. For more sensitive text, careful post-editing will be required. Special input modules are being designed to eliminate any need for pre-editing.

Although the primary purpose of EUROTRA is to build a working prototype for the seven languages, it is also intended to serve some important secondary purposes. It aims at stimulating research in Europe on machine translation and related areas, including computational linguistics and natural language work in artificial intelligence, whilst at the same time building expertise in these areas. This is reflected in the software design, where an attempt has been made to design a general purpose tool for computational linguistics applications which is then specialized down into a machine translation software. The basic design is best seen as a system generator rather than as one particular software system. As will be seen later, the flexibility of a system generator is very useful even within the specific application of machine translation. It makes it possible to offer the linguist a range of tools with which to express himself, leaving him to choose the tool most appropriate to the linguistic task in hand.

3 Some Fundamental Choices

It is clear from the preceding section that EUROTRA is a very ambitious project with a very special set of initial requirements, which, at the time of writing, distinguish it from all other machine translation projects of which we are aware. In particular, there are three a priori decisions about the general organisation of the project that have strongly influenced our whole methodological approach: decentralisation, diversification, and size.

3.1 DECENTRALIZATION

EUROTRA is intended from the start as a collaborative venture between groups working in all the countries of the European Community. Since the European Community institutions are set up to perform a political and administrative function, there is simply no place in the organisation for establishing large research institutions, particularly to serve R & D projects. In any case, the vast majority of the contributors to EUROTRA are faculty members in Universities and it would just be impractical to attempt to move them to one place for a period of five years.

The consequence of this is that most of the EUROTRA work will be carried out by teams working semi-autonomously on their own language, or in pairs on transfer. If a project of this magnitude is to succeed with such a high level of decentralisation, there needs to be a great deal of advance planning to ensure that all the components developed separately will fit together smoothly and correctly. This is why, in EUROTRA, almost all the linguistic work done to date has been preoccupied with the question of defining interfaces at the points where the different components meet. There are implications, too, for the software design, in that the software provided should facilitate modular construction of systems, even, if necessary, at the cost of other desiderata like efficiency.

3.2 DIVERSIFICATION

EUROTRA is also unusual in that it sets out giving equal priorities to seven languages and 42 distinct language pairs. Some of these languages (especially English) have been the subject of very detailed study by theoretical and computational linguists. Others, like Greek and Danish, have received comparatively little attention. In EUROTRA it would be quite foolhardy to base the entire design of the system on our experience of working with, say, English or French, assuming that the same linguistic and computational strategies will be appropriate everywhere else. In any case, many of the countries of Europe have their own strong linguistic traditions, and we should expect these traditions to be reflected in the way scholars in these countries go about handling their own language on a computer.

These considerations add to the problems we already face as a result of decentralization. On the one hand, such a level of inherent diversity increases the impor-

tance of initial planning and particularly of a well defined, well understood, and adequate interface representation. On the other, we have to try to ensure that the representation chosen is sufficiently general to meet the requirements of all languages (and linguists) in the system, and sufficiently constrained to be practically effective. Here again, there are implications for the software design, of quite similar nature: we want to constrain the software so that it helps users to express what they want to express about their language; and we want to be flexible enough to cope with the idiosyncrasies of all the languages.

This tension between generality, on one side, and a need for constraints on the other is pervasive in EUROTRA, and constitutes one of the most interesting and challenging aspects of the project.

3.3 SIZE

The last of the particular features of EUROTRA is the question of sheer size. Even in the development of the first initial prototype with a lexicon of a mere 2500 words per language, we can expect the number of active contributors to the system to be over 100. Not only do numbers of this magnitude clearly compound the serious problem of diversification, but the mass of linguistic knowledge which has to be encoded in a relatively short time to meet the requirements of the full project term of 5.5 years is quite staggering. In our view it will just not be feasible to try to encode such vast quantities of knowledge using specialised programmers as intermediaries. We must expect that the people who have the knowledge (linguists, lexicographers, translators) will themselves have to interact directly with the system. Therefore we have concentrated on designing a software system intended to perform computations on the basis of (mainly declarative) user knowledge as input.

3.4 SUMMARY

Although the idea of EUROTRA has been with us since 1978, we are only now, at the time of writing, nearing the end of the design stage, and just beginning to experiment with the first prototype software system. Because of the peculiar circumstances of EUROTRA, the gestation period has been extraordinarily long, and we have been forced to reason at length and in some depth about the theory and methodology of machine translation. We believe, in retrospect, that this long preparatory phase has led to better and more secure design, and will lead in its turn to a more manageable and robust machine translation system. By the time this paper is published, the first intensive efforts at linguistic development will have taken place, and we shall know whether or not we are justified in that belief.

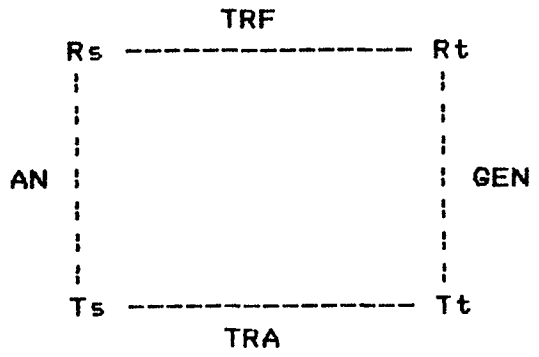
In the remainder of this paper, we discuss first the current state of the EUROTRA linguistic specifications; then we outline our approach to the development of a suitable software environment for a project such as EUROTRA, and finally we finish with some speculations on the future of EUROTRA and on the question of evaluating our results.

4 THE THEORY OF LINGUISTIC REPRESENTATION

4.1 BASIC NOTIONS

The following diagram forms a good basis for the study of representations in a transfer-based translation system:

(1)



In (1), Ts and Tt are texts, where a language is regarded as a set of texts. TRA is a binary relation, consisting of pairs of texts [Ts, Tt] where Tt is a translation of Ts. So, given two languages, SL and TL, TRA $\leq$SL$\times$TL. We introduce, furthermore, ρ, which is a set of representations of some kind. Rs and Rt are both members of this set. We will write R when it is unimportant whether we are dealing with Rs or Rt, or when the context makes it clear which is intended.

AN, TRF, and GEN are all binary relations:

$$AN \leq SL \times \rho,\ GEN \leq \rho \times TL, \text{ and } TRF \leq \rho \times \rho$$

We define the composition AN$\circ$TRF$\circ$GEN as the set of pairs [Ts,Tt] such that there exist Rs and Rt such that Ts AN Rs and Rs TRF Rt and Rt GEN Tt. We will restrict our attention to those cases where AN$\circ$TRF$\circ$GEN = TRA

Analysis, transfer, and generation are systems that compute the relations AN, TRF, and GEN. For the reasons given in the previous section, EUROTRA tries to produce a detailed specification of the contents of the relations as input to the research teams that write the computational systems. For this, we have a theory of representation. This should not be confused with the schemes for representation of knowledge proposed by, amongst others, Hayes (1978). Together with the knowledge linguists and bilinguals have of their languages, this theory will specify the contents of AN, TRF and GEN. The contents of the theory itself depend in part on the aims of the EUROTRA linguistics (cf. section 2 above), and in part on experience gathered by the writers of the various subsystems.

4.2 THREE PRINCIPLES

4.2.1 DIVISION OF LABOUR

An important consequence of decentralized development in EUROTRA is that a serious amount of attention must be given to the nature of Rt, because a research group writing generation for language L should have concrete ideas about the class of inputs to expect. Moreover, this class should make sense to people who only know about L. In EUROTRA, we relate Rs to Rt, where these are both representations of texts in L; one would expect there to be some similarity between the Rs of some text T and the Rt that represents the same text.

At first sight, one may be inclined to think that AN and GEN must be each other's "mirror image", that is:

(2) $[T,R] \in AN(L)$ iff $[R,T] \in GEN(L)$

Given (2), there is a lot of structure in the framework. If a given representation theory determines the Rs for some text, it also determines the Rt. This has two advantages:

- First, since the class of Rs for language L is clearly understandable to linguists of L, the class of Rt is also understandable.
- Second, given (2), it becomes possible to relate simplicity of transfer directly to the "depth" of the representations (for discussion, see section 4.2.2).

However, (2) is too strong, and may be in conflict with the idea of simple transfer. For example, if surface constituent structure is taken as (the basis for) a theory of representation, then (2) implies that TRF relates source language surface word order to target language word order, which clearly involves a lot more than substitution of lexical elements.

Therefore, we define an equivalence relation between representations. We will not say anything about the contents of this relation here, since its definition clearly depends on the contents of the representation theory discussed in section 4. However, it should be clear that we can use this relation (called **isoduidy,** which is an invented name) to relate GEN(L) to AN(L) as follows:

(3) Relation between AN(L) and GEN(L):

$[R',T]$ GEN(L) iff $[T,R]$ AN(L) and R' isoduid to R

In practical terms, this means that generation will relate each R' that is isoduid to R to the T that R belongs to.

If we think again of surface constituent structure as a possible basis for a theory of representation, one could imagine that R' isoduid to R if they have the same vertical geometry but not necessarily the same left-to-right order of constituents. In that case, whatever structure transfer is presented with, it would only have to produce a representation with the target language vertical structure. Generation then produces the text from that.

As a consequence of this, the representation theory will, amongst other things, contain a substantive definition of isoduidy.

4.2.2 SIMPLE TRANSFER

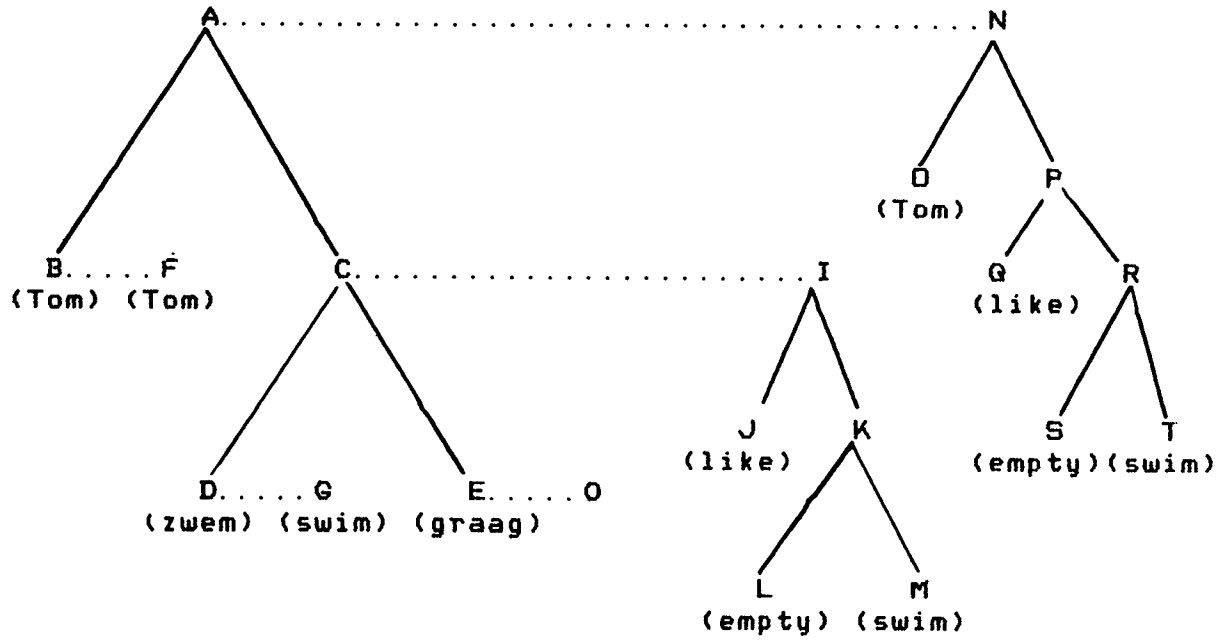
The reasons for the desirability of simple transfer have been stated in section 2. In order to give this notion some content, we have to be rather specific about the nature of the representations.

In EUROTRA, the structure input to transfer (the interface structure) is always a hierarchical structure, that is, a tree. For our present purposes the linguistic interpretation of the nodes of the tree is irrelevant; it could be surface constituents, units of meaning, or whatever.

We then introduce another relation, called **translates-as.** This relation must not be confused with TRA or TRF. Translates-as is a binary relation, probably many-to-many. If we call ρ^* the set of all subtrees of all elements of ρ, then translates-as $\rho^* \times \rho^*$. Its left-hand term is a subtree of the source language interface structure. Its right-hand term is a tree. The relation TRA is entirely different: it is a relation between texts. TRF is a subset of translates-as: it is the set of those elements of translates-as for which the left-hand term is the topmost node of a representation of a complete text.

The following example may clarify the idea of translates-as. Dotted lines indicate instantiations of the relation.

(4)



Translates-as is a straightforward generalization from two notions that have already existed for a long time in various machine translation projects:

- TRF, which is the same relation but only between root nodes of representations of whole texts;
- the bilingual dictionary, which is the same relation but only between terminal nodes.

We now turn to the notion of simple transfer.

The basic EUROTRA view (see section 2, above) is that the relation translates-as is trivially simple in all cases except terminal nodes, which then correspond to the lexical units of the two languages. Actually, if this view were guaranteed to be correct, then the generalized relation introduced here would be just a useless complication of the description of transfer: instead, we then would just say that transfer equals substitution of the lexical unit in a given representation.

However, there are three reasons for modifying the basic view.

First, and most important, there are well known problem cases such as the translation of the Dutch adverb *graag* to the English verb *like*:

(5) *Tom zwemt graag* translates as 'Tom likes to swim'

Many examples like this are caused by lexical holes in the languages; in the example, English does not have the adverb corresponding to *graag*. If we stick to the basic view, examples like these force us to have a rather abstract, "deep" theory of the interface structure. Such a deep theory may cause difficulties to analysis and generation in many cases; it would be motivated on the grounds of exceptions.

Second, we assume that whatever theory we choose at the end of the second phase of the project has to stabilise and undergo no further modification during the third phase. Thus, even if it were feasible to construct this deep theory, it would be hard, if not impossible, to guarantee that problems like the one described will never turn up during the third phase.

Third, one important way in which EUROTRA is said to be extensible is the possibility of adding other languages. But it is really impossible to have a version of the representation theory, based on the current seven languages, guaranteed to be compatible with the basic view on simple transfer when languages like Spanish or Portuguese are added.

For these reasons, we believe that the possibility of some "complex transfer" will have to be accepted, where "complex" means that the relation translates-as is not always trivial. However, the intention of the basic view remains valid. That is, the number of complex transfer relations must be minimized.

A more precise formulation is the following.

If A translates-as A′, then we will call A′ a TN of A. We now call an element [s,t] of the set defined by translates-as a simple element iff

either
s and t are both terminal nodes,
or
(i) s is a subtree, whose root node is the nonterminal node A.
and
(ii) t is a tree, whose root node is A′,
and
(iii) A′ is a copy of A,
and
(iv) the immediate daughters of A′ are copies of the TNs of the immediate daughters of A.

The principle of simple transfer then says that the proportion of simple elements in translates-as must be maximal.

The generalised relation translates-as makes it possible to put some order into complex transfer. It localises it in a natural way, based on a tree structure. In (4), only the pair [C,I] is complex; all the others are simple. This view on transfer is easy to implement by means of an built-in strategy that simulates recursion. (See also section 4.4 below).

4.2.3 SPECIFICITY

The problem of this section is the relation between the representation theory on the one hand and AN and GEN on the other.

Clearly, the representation theory must be "specific", in the following sense: given the representation theory, a language group must be able to produce a device that characterizes the right set of pairs [T,R]. That is, a language group must be able to infer from the representation theory

(i) what the Ts are,
(ii) what the Rs are,
(iii) what the pairings between Ts and Rs are.

It is likely (but not entirely obvious) that (i) will be unproblematic (it might be problematic if some special restriction on text type were imposed). But the same cannot be said for (ii) and (iii). A language group needs clear guidelines in order to be able to analyse its language in a way that is compatible with the principle of simple transfer.

So, the representation theory must be rather specific. The obvious way to specify the pairings of texts and representations is to provide some full grammar that enumerates all the pairs [T,R]. However, it is certainly not the task of the representation theory to define each of the languages in full. Its task is only to enable the language groups to do just that. So, the representation theory must be not only specific but also global.

In the ideal case, the representation theory should strongly suggest a "discovery procedure"; that is, given the representation theory and some set of relevant data of some language L, it should be possible to construct the "right" grammar of L:

(6) the representation theory + knowledge of L yield correct grammar of L

However, the state of the art of discovering useful discovery procedures or even evaluation metrics in linguistics is not such that we may hope to achieve this in EUROTRA. So, in practice, there will be a representation theory that is not specific enough, and the grammars of the languages will not necessarily be "right".

In EUROTRA, we try to partly overcome this problem by relating the representation to the text via some intermediate representations. That is, some "level" of representation is defined with respect to some other "level", for which it is easier to relate it to the text. Note that this only applies to the definition of the relations AN and GEN, and not necessarily to the working of analysis and generation. Furthermore, we try to stick as much as possible to traditional linguistic analyses and provide a lot of exemplification. Finally, there is a central linguistic team that can bring into agreement the work in the language-specific centres and the general representation theory.

4.3 THE GENERAL THEORY OF REPRESENTATION

4.3.1 LEVELS OF REPRESENTATION

Texts are represented in different ways in EUROTRA; these are called **levels.**

First, there is the level of normalized text, roughly a sequence of words and special symbols. A word is any string of characters. In what follows, by "text" we will always mean "normalized text".

Second, there are the intermediate levels. In EUROTRA, there are three of them:

- the EUROTRA Morphological level (EM)
- the EUROTRA Surface Syntactic level (ESS)
- the EUROTRA Deep Syntactic level (EDS)

Third, there is the Interface structure, intended to be the level at which transfer (i.e., simple transfer) takes place normally.

One may wonder what all these levels are for.

While the purpose of the level of normalized text is clear, as well as the purpose of the interface structure, this is less obvious for the intermediate levels. Their purpose is threefold:

- They serve specificity, in that they make it possible to talk about the interface structure and its relation to texts in a well-defined way;
- They may be the basis for some stratificational analysis or generation strategy; while such a strategy may ultimately turn out not to be the most desirable one, especially in the case of analysis, it could serve as a "safety net" if a more "intelligent" strategy fails to yield a result;
- Representations at the intermediate levels may serve as safety nets for transfer; this holds certainly for the "morphological level", as at least the lexical units have been identified there.

4.3.2 THE NOTION OF CONSTRUCTION

The fundamental notion in EUROTRA representations at all levels is the construction. A construction is a structure, made out of primitive elements and other constructions; a text is considered a construction itself.

4.3.2.1 DEPENDENCY CONSTRUCTIONS

The primary principle of the EUROTRA representation theories is dependency. In this section, we describe the intuitive basis for this notion.

A dependency relation is said to hold between two elements in certain circumstances. One member of this

relation is called the **gov**(ernor), the other element the **dependent.**

Two intuitive ideas determine the existence of a dependency relation between a gov and some dependent.

- The gov expects to find certain types of dependents in its neighbourhood. This expectation is inherent to the element that constitutes the gov, and may be a piece of dictionary knowledge.
- The gov is felt to be modified by the dependent.

Thus, to say that there is a dependency between two items is to say that the dependent element is either a slot filler or a modifier of some other element, called the gov.

One further principle determines the nature of EUROTRA dependency grammar: the gov must always be a primitive element (with respect to the level involved). The nature of what are called primitive elements may vary amongst levels of representation.

A dependency construction is the collection of elements consisting of one gov and all its dependents.

The description given here is rather abstract; each distinct level of representation instantiates dependency in its own fashion.

4.3.3 CATEGORIES

Constructions as well as primitive elements can be categorized in various ways. For example, if we take words as primitive elements, then we can classify them with respect to lexical class. Constructions can be categorized according to syntactic category, like sentence, or noun group, which cross-classifies with the distinction between dependency and coordinate constructions.

Dependency relations can also be categorized. For example, notions like "subject" or "object", and also "agent" or "patient" can be taken to be types of dependency relations. One classification of dependency relations must of course be fundamental; since dependency is said to be based on the notions slot filler and modifier, there will always be two categories associated to them. We call them **complement** (for the slot filler) and **modifier** (obviously, for the modifier); abbreviations are compl and mod.

There is an intimate relation between the category of the gov and the category of the construction of which it is a gov. As the nature of this relation varies across levels, we will not discuss it here.

4.3.4 THE BASIC REPRESENTATION THEORY

The representational device used in EUROTRA is the labelled tree.

The representational principles are straightforward:

- There is a one-to-one correspondence between leaves of the tree and primitive elements of the construction represented by the tree;
- There is a one-to-one correspondence between nonterminal nodes of the tree and constructions of the text;
- Labellings on the nodes express the categories of primitive elements, constructions, and dependency relations.

This embodies a very strong theory about language: it says that each text is a construction, and that each construction is a straightforward hierarchy of primitive elements and constructions. Elegant though this idea is, we will shortly see that it is empirically wrong. Consequently, the representation theory will have to be augmented.

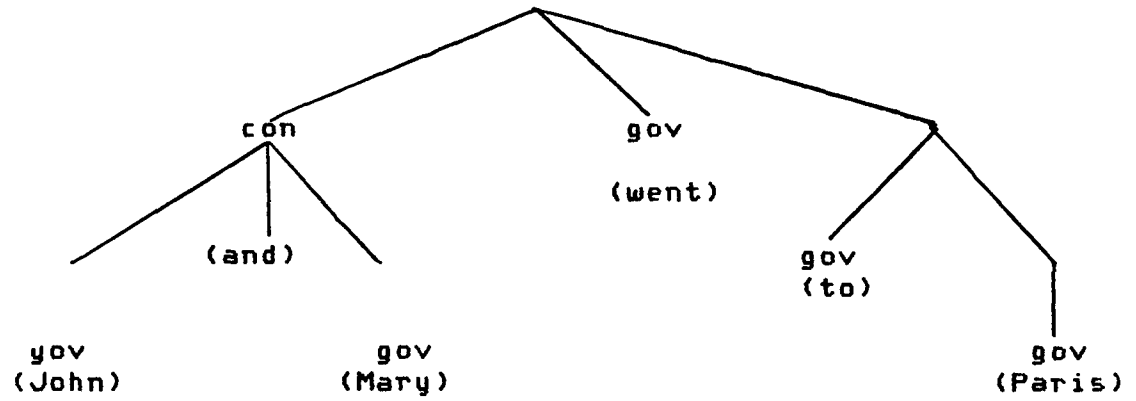
But first, we give one example of a representation:

Given the text:

(7) John and Mary went to Paris

We may have the following tree:

(8)



Note that we have omitted all the labellings, except two: we have indicated the govs, and the coordinate construction has got the label con.

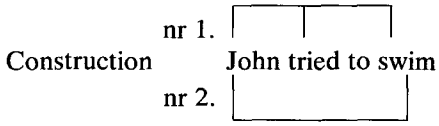
4.3.5 THE AUGMENTED REPRESENTATION THEORY

There are two classes of problems to the basic representation theory. First, the theory implies that no unit

(primitive or construction) can be a member of more than one construction. Second, it implies that every unit must be a member of at least one construction (with the exception of the text itself). Both implications turn out to be wrong.

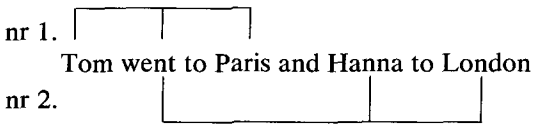
4.3.5.1 UNITS THAT ARE PART OF MORE THAN ONE CONSTRUCTION

The following is a possible type of counter-example to the basic representation theories. Suppose that in a sentence like *John tried to swim* the element *John* is felt to be a dependent of *tried.* It is then a member of two constructions, as described informally:



The element *John* is a member of two constructions; this is impossible to represent in the form of a tree.

The second example shows that one unit can also be the gov of two dependency constructions.



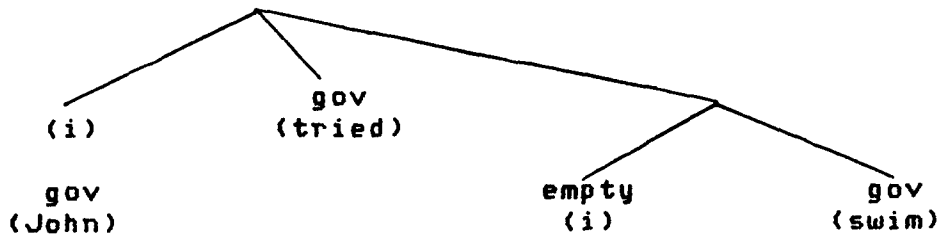
The unit *went* is the gov of two dependency constructions.

The representation theory is now changed in the following way. We introduce the notion of empty elements. These are meant to be "shadow elements" of their antecedents, i.e. the elements that participate in more than one construction. Obviously, the relation between an empty element and its antecedent must be expressed in some way other than tree geometry; in pictures of trees, we will use coindices, but that should not prejudice in any way the representation method chosen in the actual computational systems analysis, transfer, and generation.

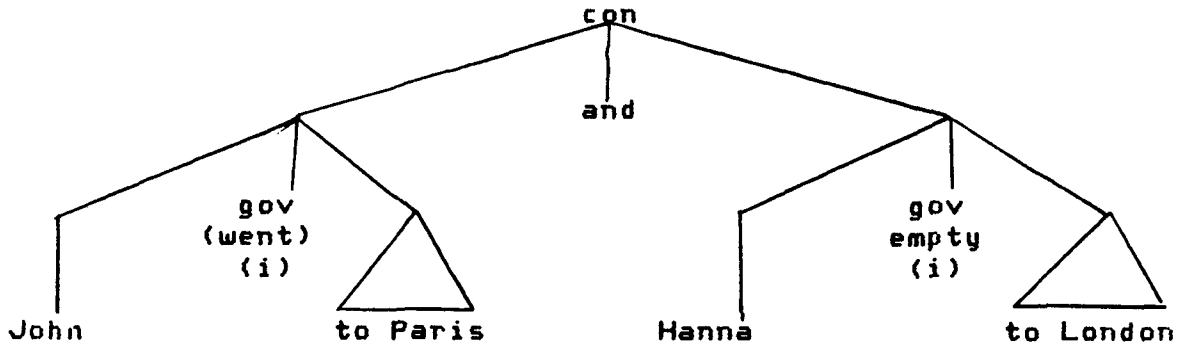
As a consequence of this, the idea of a one-to-one relation between leaves of trees and primitive elements of texts is now no longer valid: the empty elements in trees are leaves that do not correspond to anything at all in the text.

Given empty elements, we can now represent the two problem cases as follows:

(9)



(10)



4.3.5.2 UNITS THAT ARE NOT PART OF ANY CONSTRUCTION

The typical example is:

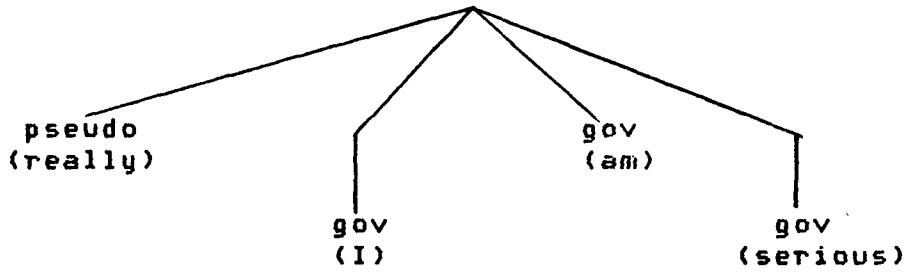
(11) Frankly, I do not care a bit

Here, the element *frankly* cannot be said to be a dependent of any gov. The problematic elements are called **transconstructionals.** They are counter-examples to the basic representation theory. This theory is augmented as follows. Transconstructionals are represented as if they

were dependents in the construction they are related to intuitively. However, in order to indicate that they are not real dependents, a special label is attached to them, indicating pseudodependency. The actual pseudodependency label depends on the representation level.

The following is an example tree:

(12)



4.4 THE INTERFACE STRUCTURE

At this moment, EUROTRA has only preliminary proposals about the contents of the interface structure; this is natural, since decisive arguments about it must come from experience with the writing of transfer systems for all the language pairs. On the current proposals, the interface structure will be a tree,where left to right order of the sub-trees is irrelevant, with a geometry that is 'deep syntactic' in that its leaves are lexical units and surface word order is not necessarily mirrored by the geometry.

The trees will be labelled extensively.

As well as the more conventional labels indicating morphological and syntactic features of constructions, there will be labels indicating what in the introduction was called interlingual information. There is no space here to do more than indicate what kinds of information are included in this category.

For example, there will be labels expressing time and modality.

Modality labels are necessary for disambiguation of at least modal auxiliaries. One well-known example:

(13) E : can
translates as
F : pouvoir or savoir

The choice is made on the basis of a modal label differentiating between *ability* and *possibility*.

Time meaning is certainly one of the more difficult matters in linguistics. Without some semantically-based time label, EUROTRA would have to translate from morphological tense to morphological tense. That would already mean a deviation from simple transfer as defined in the framework, as tenses are normally not considered lexical units. However, even if the project accepts such a deviation, then there is a severe problem of choice in transfer. One example is the translation of 'present' from Dutch to English:

(14) NL : present
translates as
E : present or present progressive or future

The list indicates the kind of problem transfer writers would meet if there was no time label. Example translations are:

(15) NL : Ik lees een boek
E : I am reading a book

(16) NL : Ik lees vaak boeken
E : I often read books

(17) NL : Morgen lees ik zes boeken
E : Tomorrow I will read six books

These examples are relatively simple, but as soon as we take complex sentences and sequences of sentences into account, the problem for the transfer writer gets really difficult. Therefore, there is good reason for a label that captures the "time meaning" of a text; this may even be interlingual. In any case, we expect that such a label will considerably simplify the task of the transfer writer.

With similar justification, i.e. the need to simplify transfer and to produce good translation, semantic relation labels (such as agent, patient), Katz-Fodor type markers, morpho-syntactic class markers, and representations of anaphoric relations are included in the interface structure.

5 COMPUTATIONAL TECHNIQUES

In the present state of the art, the problem of machine translation is not fully understood. In some sub-domains (e.g., English syntax, English-French lexical equivalences) we have a good deal of experience, a rich theoretical literature, and, hence, the confidence to predict in some detail the behaviour of the program to do the job. In other areas, (the synthesis of Greek texts, mapping Italian representations to equivalent Danish text representations), we have virtually no experience and can only make informed guesses about the "right" way to do the job by computer. In the worst case we are still (at the time of writing at least) hopelessly at a loss when it comes to characterizing precisely what is preserved in translation if more than two languages are involved. In

other words, we do not have, as yet, anything like a complete theory of multilingual machine translation. We have argued elsewhere, and at some length (Johnson, Krauwer, Rosner, and Varile 1984, Johnson and Rosner 1984) that it is in the nature of problem-oriented software to embody some theory of the problem domain, and we shall not repeat the detailed arguments here.

We simply restate our view that no existing solution to the question of finding an appropriate problem-oriented programming language for machine translation seems to us to be acceptable for EUROTRA. These solutions fall roughly into three categories:

(a) Assume some theory and implement it directly; this approach is fairly rare, but seems inherent, for instance, in Jan Landsbergen's Rosetta project (1984).

(b) Use an existing programming language, perhaps extended by a library of purposely-built macros, sub-routines or functions (depending on persuasion): examples of this approach are the IBM macro assembler in SYSTRAN (Bruderer 1978) and FORTRAN in SUSY (Maas 1984).

(c) Invent a new programming language, embodying a very weak, low-level theory of machine translation – usually based on explicit tree-to-tree mappings as in ROBRA (Boitet and Nedobekjine 1982), Q-systems (TAUM 1973), and GRADE (Tsujii 1983). The underlying thesis is normally sufficiently weak, in such cases, to allow the claim that the language has universal or near-universal application for all or most of machine translation tasks.

We have not adopted (a) because there is not sufficient practical evidence of a single theory that encompasses translations between all pairs of the Community languages. We reject (b) on the grounds discussed at some length in section 2. above: ordinary programming languages are just too unconstrained to be reliably handled by a large, loosely-linked community of users, many of whom are unskilled in their use; and they obscure some of the true issues of linguistic knowledge representation and use in the detail of managing a von Neuman machine (or lambda calculus or Horn clauses or what have you). The last option, (c) is more interesting. In principle, we reject (c) also, although in the short term we have adopted a form of it for reasons of expediency, as we explain below in section 6. We are sceptical about any kind of universal programming language for machine translation, because we believe that the tasks involved in machine translation are essentially heterogeneous in nature. If we are constrained to use the same language to describe syntactic parsing, "semantic" interpretation, lexical and structural transfer, resolution of structural and lexical ambiguities, in and between seven different languages, it follows that either all of these are comparable or that the language of description gives us very little help in saying what we want to say.

To give a very simple example, suppose we have a strategy for parsing English that uses phrase structure recognition to construct a network of syntactic relations (SUBJECT, OBJECT, etc.) and then maps these relations to case relations like AGENT, PATIENT etc. In the homogeneous view of the world, we might have to write something like:

```
given A+B where cat(A)=NP and cat(B)=VP
build C(A+B) setting cat(C)=S
and
given A+B where cat(A)=V and cat(B)=NP
build C(A+B) where cat(C)=VP
```

followed in a later process by:

```
given A(B+C(X*+D+Y*+E+Z*)
   where cat(A)=S and cat(B)=NP
   and cat(C)=VP and cat(D)=V
   and cat (E)=NP
build P(Q+R+X*+Y*+Z*)
   where srel(P)=pred and srel(Q)=subj
   and srel(Q)=obj and lex(P)=lex(D)
   and lex(Q) =lex(B) and lex(R)=lex(E)
   and semf(P)=semf(D) and semf(Q)=semf(B)
   and semf(R)=semf(E)

   /* semf stands for "semantic feature", X*,Y*, Z*
   are intended to stand for variables over sequences
   of trees */
```

followed again later by

```
given A(B+C) where srel(A)=pred and srel(B)=subj
   and srel(C)=obj and action-process in semf(A)
   and animate in semf(B)
build A(B+C) adding case(A)=pred and case(B)=agent
   and case(C)=patient
```

While the above notation is very informal, it is worth noting the very arbitrary semantics that underly it. For example, there are clearly conventions about the use of identical variable names on the left and right hand side of rules; in some cases, right hand nodes may be understood as copies of corresponding nodes on the left (indicated by the use of *where*), in others they may be interpreted as identified with their left hand counterparts (indicated by *adding*). The arbitrariness is not accidental – indeed, since the linguistic theory underlying the notation is so weak, the meaning of the notation cannot but be arbitrary to the user. Their arbitrariness, however, is not the biggest defect of notations of this kind. Where they really fail is in being intolerably cluttered, since the user is forced to be explicit about every detail of the operations, precisely because in the absence of any strong linguistic theory, none of the responsibility for details can be left to the machine.

Consider now the same statements in a more perspicuous notation:

```
S   →   NP[iSUBJ=*] VP[i=*]
VP  ←   V[i=*] NP[iOBJ=*]
```

and elsewhere (in the lexicon perhaps),

if action-process in semf(PRED)
and animate in semf(SUBJ)
then [SUBJ→AGENT,OBJ→PATIENT].

Again the notation is informal, but not totally arbitrary (the debt to Lexical Functional Grammar (Kaplan & Bresnan 1982) is obvious). What is significant, though, is not so much the syntax of the notation as its semantics. Because we have a theory of parsing, we can include in the user's machine a large chunk of the meaning of what it is to parse within that theory. As a result, the user is left with a much clearer view of the task in hand: to provide the details of specific cases within the theory.

The ideal goal of the EUROTRA software design should be to provide just such a theory sensitive system for machine translation. Unfortunately, and we have made the point many times here, we just do not have sufficient knowledge of the domain to provide the necessary theoretical input, and the problem is magnified in the special circumstances of EUROTRA.

What we have therefore built is an environment in which new theories and/or sub-theories of machine translation can be implemented very rapidly on an experimental basis. The environment consists essentially of four parts, not including the usual editing and debugging facilities. Two of the parts are quite standard: a compiler compiler, which we use to write compilers for the languages of a new theory; and a kernel interpreter that runs the outputs from the compiler. What is interesting is that we contrive to make the process of compilation as much as possible a purely syntactic one, mapping statements in the user language into a simple tuple language. Statements in the tuple language are not, however, executable directly by the kernel interpreter, since they contain as yet uninterpreted symbols. The interpretation of the symbols is given by external definitions, which are of two types: control definitions and data definitions. As the names suggest, data definitions are essentially instructions to a pattern matcher which acts as a slave to the main interpreter; control definitions define how and when calls to the pattern matcher are made. By judicious choice of the definition languages we are able to use these external definitions in two ways – to make rapid implementation of new theories, and to serve directly as specifications for a more efficient implementation, should the user agree after experimentation to include a new theory in the system. A more detailed description can be found in Johnson, Krauwer, Rosner, and Varile (1984).

This device is already proving very effective in allowing users to try out new ideas. More important, it frees us from the dangers of committing the user community too early to a small number of particular strategies, which may turn out to be unsuitable in the medium term, without making ultimate commitment impossible by imposing monolithic homogeneity from the start.

Nonetheless, we clearly need to make some decisions now, however provisional, so that we can get started. The remainder of this section describes the first user-language prototype implementation which is being handed over to users for preliminary experimentation.

All our software prototyping has been done under Unix,[1] both for reasons of easy portability and because of the rich set of available software tools. The original prototype was developed on a VAX-11/780 under bsd version 4.2, and successful ports have been made to a bsd version 4.1 on a VAX 750 and to a Dual Systems 83/20 running Unisoft Version 7. We are about to attempt a port to a Sun Workstation and anticipate no serious difficulty.

It should be noted that our decision to adopt Unix as a software prototyping environment (and therefore necessarily as a linguistic prototyping environment in the short term) does not necessarily of itself commit the anticipated industrial implementation to any particular hardware/software combination. The main purpose of our own software prototypes is to help us derive more reliable specifications for the industrial implementation, and to provide temporary short term support for linguistic experimentation.

6 The First User-language Prototype

6.1 Processes

The overriding design criterion we have followed is that of modular construction. Not only is this generally desirable, it is virtually essential given the organisational framework of EUROTRA. The basic unit of a user "program" is called a **process.** Since we want it to be possible for users to test parts of a system independently of others, and indeed to combine parts together in a reliable way, we have been particularly careful to provide ways of limiting or even excluding the propagation of unexpected side effects between processes. We achieve this by defining a process as a quintuple

process=[name, expectation, focus, body, goal]

The name is just a symbol used to identify the process. The expectation and the goal are pattern descriptions that serve a number of desirable functions. The most important of these is to guarantee that the domain and range of the process can be known when the process is defined. They achieve this by acting as filters over the currently active data configuration. A process may only operate on data that satisfy the expectation; correspondingly, only data that satisfy the goal are allowed to be output from the process. Operationally what happens is: the system attempts to apply the process by matching the expectation against the currently active data set; the process is invoked only if a match is found, in which case the process is applied in parallel to all data subsets that match; on termination (we assume that the process terminates) all results are matched against the goal; in all, and only, the cases where the match succeeds, the new

[1] Trademark of AT&T Bell Laboratories.

results are added to the active data set, and the system proceeds to the next task.

The focus gives a way of narrowing down application, to a subset of the data set yielded by the expectation; this is necessary, for example, when a process invokes itself recursively.

The process body may be either primitive or non-primitive. Processes with primitive bodies are also called grammars, and we shall return to them later. Non-primitive bodies consist of expressions over the names of processes, where the meaning of the expression can be varied by external definitions. In the current version, we allow regular expressions over processes, interpreting the concatenation operator as sequential application, the union operator as parallel application and the closure or star operator as all paths combinatorial application. The principle underlying this general scheme of controlling pattern directed invocation via a formal control language owes much to the work of Georgeff (1982). Thus, in the body of a process, a user might write

```
body
  p1,p2,(p3 | p4)
```

with the meaning "apply p1, then p2, then p3 and p4 in parallel". Our current compiler is defined to translate this into the tuple

```
[sequence, p1,p2,[parallel, p3,p4]]
```

And, in our control definition language (we currently use FP, Backus 1978), the definition of *apply* includes:

```
apply=atom→execute;
eq ∘[1,'sequence]→/apply ∘ tail;
eq ∘ [1,'parallel]→apply ∘ tail
```

where, with some simplification

```
execute =
  integrate ∘ filter-goal ∘ apply ∘ filter-expectation.
```

It should be emphasised that the user is only concerned with writing (and understanding!) statements like

```
body p1,p2,(p3 | p4)
```

6.2 GRAMMARS

The process interpreter continues to try to apply processes until it bottoms out at grammars (processes whose body is a primitive). The structure of a primitive depends on the theory it implements: thus a general rewrite primitive will be organised – and defined – differently from a dictionary primitive, which in its turn will differ from a transfer primitive, and so on. We currently have very few primitives, since the system is still in an experimental stage. The most important is a non-deterministic tree transducer, implementing a general re-write system, which does not differ in any interesting way from Colmerauer's Q-system (1971) or Kay's powerful parser (1967). Its main purpose is to provide users with a very (excessively) powerful tool for experimentation, and to provide fall-back for those cases where there is no adequate computational linguistic theory. We also have an analysis dictionary (a device that maps strings to nodes with complex collections of attributes and features) and a phrase structure parser. We are about to start on a transfer device to implement the proposal outlined in section 3, and, as a more searching test of the capabilities of the basic tools, an implementation of a multilevel parser inspired by LFG. Once the basic tools were built, we found it very easy to build prototype implementations quickly. For example, the general re-write system took about two man-months. The first dictionary implementation took less than a man-week. We expect that the transfer device will take around two to three weeks; the multilevel parser will almost certainly take longer – perhaps a month to six weeks.

6.3 DATA STRUCTURE

In our system, there is no data "structure" as such. The same effect is achieved through interaction between a pattern matcher and a data base of primitive objects called nodes. The behaviour of the pattern matcher is defined externally through statements in a data definition language, much in the same way as the meaning of system control constructs is defined in FP. At the present time, we are using Prolog to supply both the data base manager and the definition language. This is not totally satisfactory, and we expect to have a more appropriate "in-house" data definition language shortly. To give a flavour of our data definitions, we give a single example of the definition and use of a tree, in pseudo-Prolog.

First we define some basic relations, using built-in higher-order relations:

```
antisymetric(dom)
intransitive(dom)
irreflexive(dom)
$dom(x,x)
$dom(x,y):-
        dom(x,z),
        $dom(z,y)
                    /* reflexive transitive closure */

tree(R,x):-
      $dom(R,x)          /* tree x with root R */
```

If the notation #x in the user program means "bind x to a tree", then we define our compiler to translate #x to [tree x]. The control interpreter simply performs elementary syntactic manipulation on data requests and passes them directly to the data manager, [tree x] is transformed to tree (-,- x). Repeated calls to the data manager will yield all possible trees x in the currently active data set.

6.4 DISAMBIGUATION

The system potentially has a number of ways of dealing with ambiguity. Which ones are used depends on the extent to which disambiguation strategy is embedded into an implemented theory.

The simplest device is an extension of the use of goals to allow the user to supply an ordered list of goal descriptions. The system simply continues to try to match goals, in order, until it finds one which succeeds. The output from that goal is the result of the process. This rather cumbersome device is actually quite useful, for example in constructing elementary preference strategies painlessly. It is, however, not particularly subtle.

More interesting are strategies that exploit the inherent parallelism of the system – defined, for example through the (apply to all) functional of FP. Normally, the results of a parallel process application are all added to the current data set "in the same place". We could, however implement a primitive that allows the user to state criteria for selection between competing representations, and to exclude less favoured ones on the basis of linguistically motivated judgements. This would only be sensible, however, if the user were able to formulate such judgements in a general way.

Finally, we also have the option of implementing a relation **alt** (for alternative) directly in the data definitions (we have, in fact, done a simulation of a chart parser in this way). The problem here is that an alt relation between nodes is easy to handle, but an induced alternative relation between sets of nodes is not, unless the process that constructs it is very well behaved (for example, only building alternatives between simple constructs like trees). We do not know of any practical method of guaranteeing that such a relation can be maintained in a system which can perform transformations of arbitrary complexity.

6.5 EFFICIENCY

The system we have described here is not particularly efficient – indeed it can be dramatically inefficient when presented with only moderately large and complex computations to perform. We are not (yet) unduly concerned by this inefficiency, for two reasons. First, we are still at the experimental stage where correctness is still more important than speed; there are no plans for an industrial implementation before 1988. Second, the experimental device we have described here has two equally important functions: the first is indeed to permit us to generate implementations of new theories rapidly for experimentation in the field; the second is to provide the basis for a formal specification of the semantics of that theory. If we can construct prototypes using precise definition languages, with the benign side effect that the same prototypes perform tolerably well for experimental purposes, we can be confident that an optimized implementation derived from the same specifications has a good chance of being both correct and operationally efficient.

7 CRITERIA FOR SUCCESS

It is obvious from what has been said already that EUROTRA is to be regarded as a research and development programme, rather than as either a pure research project or a pure development project. This affects the criteria that will be used in evaluating its success or failure. Main emphasis, even at the end of the five and a half year programme period, will be put on quality of translation, with speed and efficiency playing a relatively minor role. (Of course, certain minimum limits of speed must be reached if only to allow the large amount of linguistic development work necessary to be accomplished). Quality will be judged in terms of ability to cover the corpus texts and other texts in the same general class. The testing procedure will probably bear a strong resemblance to that used for evaluating METAL (Slocum et al. 1984).

Apart from the quality of translation, one of the main criteria in evaluating the system design will be ease of extensibility. Exact procedures for evaluation will be decided by the programme's management committee towards the end of each phase, when a checkpoint must be passed before permission is given to pass to the next phase.

8 FUTURE DIRECTIONS

EUROTRA's future falls into two distinct parts: the future covered by the programme period itself, up to mid-1988, and the future after that. During the programme period itself, two further languages (Spanish and Portuguese) will be added, increasing the number of language pairs to 72. (It goes without saying that not all 72 language pairs would be fully treated by mid-1988.) Apart from this extension, it is planned to increase the variety of text-types dealt with, although remaining within the general area of Community texts. There will also be some experimentation with subject areas other than the one initially chosen. Throughout the programme period, the flexibility and modularity of the system design will encourage experimentation with different linguistic techniques, as well as making it possible to expand and repair the system with ease, since the extremely modular approach taken makes it possible for any single module, for instance a primitive process treating noun groups or carrying out dictionary look-up, to be modified independently of other processes, even, in the limit, being replaced by a completely different process. When a process is modified, the goal attached to each process becomes important in that by specifying the results to be delivered by the process, unforeseen interactions with other processes in the system are prevented.

During the third phase, planning of the future after the programme period will begin. The feasibility and desir-

ability of an economic development of the working prototype system will be investigated, and specifications drawn up if it is decided to go ahead. At the same time, by Community conventions for research and development projects, both the prototype system and the kernel software will be distributed at cost to Government institutions, and to Universities and research institutes in the Member States. Indeed, the software can be distributed to these parties before the end of the programme period to be used as a research tool. This is one of the ways in which EUROTRA hopes to stimulate research outside the immediate environment of the project itself.

It could, with some justice, be said that all of EUROTRA lies in the future: given the time and resources invested in defining a sound linguistic basis and a flexible, heavily problem oriented software, it is our hope that the future can be faced with confidence.

References

Backus, J. 1978 Can Programming be Liberated from the von Neumann Style? *Communications of the ACM* 21(8).

Boitet, C. and Nedobekjine, N. 1982 Russian-French Machine Translation at Grenoble: A General Software Used for Implementing a Particular Linguistic Strategy. *Linguistics.*

Bruderer, H.E. 1978 *Handbuch des maschinenunterstutzten Sprachubersetzung.* Munchen, New York: 100.

Colmerauer, A. 1971 Les Systemes-Q: un Formalisme pour Analyser et Synthetiser des Phrases sur Ordinateur. Groupe TAUM,, Université de Montréal.

Georgeff, M. 1982 Procedural Control in Production Systems. *Artificial Intelligence* 18: 175-201.

Hayes, P.J. 1978 The Naive Physics Manifesto. ISSCO Working paper No. 34.

Johnson, R.L.; Krauwer, S.; Rosner, M.; and Varile, G.B. 1984 The Design of the Kernel Architecure of the EUROTRA System. Proceedings of COLING-84.

Johnson, R. and Rosner, M. 1984 Machine Translation and Software Tools. In: King, M., Ed.

Kaplan, R.M. and Bresnan, J. 1982 Lexical Functional Grammar: A Formal System for Grammatical Representation. In Bresnan, J., Ed., *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts.

Kay, M. 1967 Experiments with a Powerful Parser. In *Proceedings of the 2eme Conference International sur le Traitement Automatique des Langues* Grenoble.

King, M., Ed. to appear *Machine Translation: the State of the Art.* Edinburgh University Press.

Landsbergen, J. 1984 Isomorphic Grammars and their Use in the Rosetta Translation System. In: King, M., Ed.

Maas, H.D. 1984 The MT System SUSY. In: King, M., Ed.

Slocum, J. et al. 1984 METAL: The J Machine Translation System. In: King, M., Ed.

TAUM, Le Systeme de Traduction Automatique de l'Université de Montréal (TAUM). *Meta* 18: 227-289.

Tsujii, Jun-ichi 1983 Technical Outlines of Japanese National MT Project. Paper given at the Joint EUROTRA-Japanese Workshop, Brussels.