MENTOR88/P ONE-YEAR OLD

Diana Santos
Paulo Líbano Monteiro

Grupo Científico IBM-INESC
R. Alves Redol, 9 - 40E Dto, Sala 412
1000 Lisboa
Preface

This document is just the collection of the texts which were written during the first year of development of MENTOR88/P, that effectively started in October 1987 with the PLNLP/PEG course in Hawthorne.
Contents

- "Proposal of a PLNL-based translation system", (DMS and PLM, 2/88) 1
- "Dealing with PEG's hidden pointer attributes in MENTOR88/P", (DMS, 2/88) 7
- "Lexical transfer in MENTOR88/P", (DMS, 2/88) 11
- "Report on a PLNL-based translation system", (DMS and PLM, 3/88) 17
  This is a new (3/88) version of the first paper, dated 2/88.
- "A set of procedures for transfer in MENTOR88/P", (DMS, 4/88) 55
- "The translation of infinitive clauses in MENTOR88/P", (DMS, 5/88) 61
- "Performance of the set of test sentences on INFCLs in MENTOR88/P", (appendix to the previous text) 71
- "Presentation of the work by the Portuguese branch of MENTOR", (DMS and PLM, 6/88) 73
  Copy of the foils presented at the MENTOR meeting in Lisbon.
- "Generation in MENTOR88/P", (PLM, 7/88) 93
- "Machine translation prototyping", (abstract for SJ describing the work in MENTOR on Haifa, Madrid and Lisbon) 109
- "An MT prototype from English to Portuguese", (DMS, 10/88) 113
- "Overview of work done in structural transfer", (DMS, 10/88) 125
  Draft.
Proposal of a PLNLP-based translation system

by Diana Santos
Paulo Libano Monteiro

the Portuguese branch of the Mentor project.

Introduction

The conclusions of the work done from December 1987 till February 1988 will be briefly presented, together with specific examples and discussion of the options that have been taken.

The aim of conducting this work was primarily to test the possibility of using PLNLP as the only software tool in Mentor, as opposed to other approaches, namely that of using the Tree Transducer (Hiroshi Nomiyama Japanese Processing Group, JSI, IBM Japan, NOMIYAMA @ TOKVMS11) for the transfer phase.

The results obtained show that it is actually possible to follow this procedure. The ultimate decision will, however, have to be taken after a careful comparison and discussion of all methods.

We hope to contribute with this work to choose the best final strategy that will eventually be followed by Mentor.

Starting points

Mentor as a translation system can be classified as transfer-based, and not semantically oriented. This means that there will be three distinct phases in the translation process, usually called analysis, transfer, and generation; and that the system will recur only to syntax whenever this is possible.

The analysis phase, which decodes an English sentence and represents it by an internal structure, is done by PEG (Karen Jensen, “PEG 1986: A Broad-coverage Computational Syntax of English”, IBM Thomas J. Watson Research Center), who follows a syntactical approach, and can be defined as of almost unrestricted linguistic coverage.

The task of Mentor is to transform the output of PEG into the corresponding sentences of the target languages, at this moment Finnish, Hebrew, Portuguese and Spanish.

In this framework, we derive the following conclusions:

Although the target languages are considerably distinct, we believe that both the methodology and the architecture of the systems developed should be as similar as possible, not only to allow for a broad collaboration between teams, but mainly to achieve a multilingual system and not four unilingual ones.

This implies, in our opinion, that everything that concerns at least the transfer phase should receive the same treatment in all languages, and that standardization should be tried even for generation. In particular, dictionaries should have similar organizations, and the transfer and generation modules should be independent from each other.

The basis for our work

Being the language in which the first phase of translation was developed (independently of Mentor), PLNLP (George Heidorn, "Natural Language Inputs to a Simulation Programming System", Technical Report NPS-55ID72101, Naval postgraduate School, Monterey CA) was the obvious first candidate for the design and implementation of the whole system. However, time consider-
ations, allied to the unavailability of the language till very late, resulted in the use of other means, namely the Tree Transducer as was already cited, and which was considered the valid alternative for the first Mentor prototypes, in Haifa and Madrid.

Not being by any means our intention to criticize TT, which is undoubtedly a powerful tool, it should be emphasized that its use in Mentor has several drawbacks: It forces the creation of interfaces, and, most definitely, it enforces the change of the representation.

In fact, while PEG creates a graph of PLNLP records, TT works on ordered trees. (In other words, PEG provides us with the representational power of a graph, and we give it up by changing it to tree format). Moreover, while PEG creates a labeled graph, TT works on unlabeled trees. It may not be obvious which approach makes linguistically more sense, a graph or a tree, but from the system design's standpoint, it looks to us clearly incorrect to begin with one representation and destroy it to go on with another... On the other hand, there would also certainly be another better suited grammar for this last representation, so there is an inadequacy from both sides which can only provide a less powerful system in the end.

This being the main reason "against" TT or any other non-PLNLP based tool, others exist, coming from more practical matters. In particular, the question of portability which is endangered by the fact that the only way to make PEG and TT compatible is through LISP! This means very simply that a third language will have to be considered, with the ultimate problems it can bring with, concerning versions and releases. The system developed would be dependent, then, not only of PLNLP, but of TT and LISP.

This fact is still more limiting considering that, while compiled into LISP in the version we have, PLNLP is independent of it, and so would a system built on top of PLNLP alone. Moreover, there are, and more there could be, PLNLP compilers into other programming languages. In a long term approach, and considering there's a PL8 version of PLNLP, there is no doubt this one is much better suited for a commercial product... Also, the Common LISP version of PLNLP is in preparation, which is of great importance, considering this language is really becoming standard LISP.

In conclusion, we believe that many languages in a system are bound to make it unreliable, not portable and suboptimal, especially if, and this is the case, the interface has to be written in yet another language.

These are the reasons that made us choose the other approach, namely the one of trying to write the whole system in PLNLP.

The conceptual design

The priorities we chose to elect were clearness of design, conceptual correctness, and portability and modularity, in the sense that the system may be easily extended and changed, namely to add or remove knowledge.

From now on, and for simplicity of exposition, we use Portuguese as the target language, but it should be noticed that one of the aims of this work was to produce results that could apply to all languages involved.

The system is divided in three phases, each one with a dictionary of its own. PEG is in charge of the first phase, with OD (the Optimized Dictionary, Stephen Richardson, IBM J. Watson Research Center). We are not entitled to do any change or modification of PEG, as is obvious. Our only concern should be report to Karen Jensen any irregularities or information about the English sentence we miss in PEG's analysis. We are sure that she will do any changes in her grammar that appear of linguistic significance in the frame of the English language alone.

The transfer phase has as its goal the producing of a new graph-structure for Portuguese, departing from the English one. Its output should be exactly the one a GPP (PLNLP Portuguese Grammar) would produce, if there were one.

The reason for assigning this goal for this phase, and not any minor one, stems from the fact that the generation phase, in our opinion, should be completely independent of the source language that was used, that is, irrespectively of the former, the same generation module for Portuguese should always be used.

Generation encounters then as its main task the organization of the Portuguese surface structure, from the syntactical structure of a particular sentence and the particular lemmas of the words that take part in it.
The operations to be done at this phase are then mainly morphological ones (maybe later style considerations can be considered). It has been a constant in the development of generation code to make it source- language independent, that is, to use solely Portuguese features, and never any English-specific categorizations.

The work actually developed will now be described.

The actual design of the transfer phase

Transfer is directed by a procedure with a simple algorithm: It travels “down” the graph, copying each English record into a Portuguese one, (created after the access to the English-to-Portuguese dictionary), and delivering this new record to a set of transfer rules. Both copying and transferring are implemented using PLNLP encoding rules, with one record in the left and one record in the right.

The choice for a copy instead of actually changing the English graph seems conceptually more correct to us, although it implies more work. This way, we distinguish between the English attributes (some of which are maintained in the new graph, others not) and the Portuguese ones, that should not crowd nor change the English internal structure. Moreover, the values of the same attribute can differ in the two languages (as they trivially do in their BASE and STR attributes) and so it makes sense the existence of two different entities. Finally, the fact that PEG produces a graph and not a tree is taken care solely by copy, being transfer rules freed of the burden of checking whether the record they are dealing with has already been visited or not.

The actual transfer is performed by rules, indexed by the SEGTYPE2 of the records of the English graph structure. The choice of this attribute instead of SEGTYPE is due to the fact that it represents a finer syntactic classification than this last, consisting so in the ultimate word about the particular segment of text it represents.

The reason to choose rules instead of procedures comes from common sense reasoning: rules have a much superior readiness, together with the fact that rules are modular and stated independently. Instead of changing a procedure anytime we want to add a minor change, it is enough in a rule-based system to add or remove rules. This is fulfil the make the system development easy.

The question of transfer is normally addressed dividing the process in two distinct acts: structural transfer, and lexical transfer.

The first is related to the syntax properties of the elements to manipulate and may produce a syntactical change when executed.

The second is related to the actual words in the sentence, and brings about the problems of meaning and word sense ambiguity.

What has been described so far is the structural transfer. This process, as has been implemented here, although indexed by syntax classes (the SEGTYPE2s) may recur to any characteristics of the PLNLP records, including the actual word (or, more accurately, its lemma). That means that we do not consider structural transfer independent of the words; we classify a transfer as structural whenever any information in the record structure, representing the segment of text we are dealing with, results in a syntactical change.

When, on the other hand, it is the choice of a particular word between many alternatives that is in question, we are dealing with lexical transfer and its result should only produce the right translation for a given word (or idiom). No other changes are to be performed with lexical transfer. This last statement does obviously not mean lexical transfer is trivial...

It may seem strange, however, considering that there are many cases when a particular choice of a translation in Portuguese means changing the whole structure of the English sentence. What has been said so far only separates lexical transfer from the structural one. This last will be performed after the convenient choice of the Portuguese word (with any idiosyncrasies associated), and it will be performed taking those characteristics in account.

This is an important point, since, with this design, there is no need to perform two different structural transfers: the only structural change is done after the access to the transfer dictionary and is based both in the English and in the Portuguese features at once.

We obtain this behavior with two complimentary devices: the use of (structural) transfer rules that deal with all matters pertinent to syntactical change, come they from English syntax or from the
characteristics of the Portuguese words themselves, and the organization of the transfer dictionary accordingly.

In a small parenthesis, we decided to use an OD-based dictionary, again to follow the philosophy of modularity already describe. As the first phase defined its own OD, we chose to create two others, one for transfer, and other for generation. This is not, however, a basic feature of the system, what is important is, obviously, which information is actually stored in it.

To understand how can lexical and structural transfers coexist in one dictionary, its organization will be described.

For each English lemma (stored as the BASE attribute in PEG), there is one or many translations, represented by the corresponding lemma in Portuguese, followed by the part of speech it belongs (as OD requires). Optionally, two different kinds of information may be stated, syntactical features related to the words in Portuguese (e.g., whether an adjective does not conform to the usual syntax, or whether the word is omitted when it has premodifiers); and conditions for choice when there is more than one candidate (e.g. existence or not of attributes in the record; context).

The first group is to be used during structural transfer, the second is what makes possible the lexical one.

When copying the English record into the Portuguese one, the dictionary is accessed and lexical transfer is performed, by calling a procedure that chooses, between the several alternatives available, which word (and consequent information it has associated) to copy from the transfer dictionary.

When (structural) transfer rules are triggered, the choice has already been made. The rules will be activated based in the information got from the E-to-P dictionary, together with all the other information put at our disposition by PEG.

The choice of two different ways of performing the two faces of transfer was carefully considered. While it was clear from the beginning that PLNLP rules were clearly suited for implementing structural transfer, we decided to implement the choice of the right translation with a PLNLP procedure, which runs through the list of the candidates electing one.

Another alternative would be encode all the rules of choice in PLNLP rules, and apply them indexed by the actual word (or better the BASE attribute). This is clearly impractical, although it would favour readability: to have a rule (or more) for each word in English would mean a system not easily expandible or modified, not to speak of resource issues.

On the other hand, a fixed number of rules and an easy-to-improve-and-enlarge dictionary are much more both realistic and user-friendly, and of course more programmer-friendly too...

This, we think, is enough to give up representing words by rules. History backs us too, since there was a time when PLNLP followed this approach, and it changed it to OD when growing up.

However, till now only negative arguments were presented, that is, reasons against the alternative but not for the method chosen. It seems to us that there is also a reason why the information should be attached to the dictionary, and not somewhere else: the rules of how to choose one word can be seen of being related primarily to the word itself, so should be stored with it.

At last, we explain why we decided to separate the two levels of transfer. From a theoretical point of view, it seems to us that the very existence of this distinction is enough to be taken into account when implementing a translation system.

From a practical point of view, it considerably simplifies the problem since it reduces it to two smaller ones, in that way easier to control and define.

From an implementational point of view, it frees our transfer rules from matters of actually choose one word and behave differently according to the one chosen, and it considerably reduces the actual number of transfer rules. On the other hand, to add a new word to the dictionary, with this architecture, does not imply any changes to the PLNLP code (except if another dictionary feature were introduced) while otherwise it enforces changing the code, at least to write a new transfer rule for the new word.

To end the section on transfer, please notice that as an appendix, we present the basic procedures together with some sample rules, and the corresponding entries in the dictionary.
The actual design of the generation phase

As a starting point, the graph produced by the transfer phase is passed to the generation one; this is the only communication between the two phases. This graph includes all relevant information from both analysis and transfer and, obviously, from the corresponding dictionaries.

The algorithm that goes through the graph, in order to perform generation, is somewhat similar to the one used in the transfer phase: the graph, actually with the form of a tree, is traversed top-down and left-to-right. At each node, the generation dictionary is eventually accessed and a rule is triggered by the corresponding SEGTYP attribute. Terminal node rules are different from the others in that they produce a segment of the Portuguese sentence; non-terminal node rules serve as filters of the information that is carried through the nodes below until the "leaf" is reached.

Unless the required word has already been read, the generation dictionary is accessed, at each node, using the BASE attribute, which, as already mentioned, is the linguistic lemma. What is stored in the dictionary concerns essentially morphological features of the words. Each entry in the lexicon has one or more sub-entries, depending on the number of parts-of-speech that the word may assume. Only irregular forms and other relevant information have to be placed; particularly, if a verb is irregular in a specific number and person of a tense, then only that irregularity has to appear in the dictionary. Regular forms are handled by specific rules, so that the majority of the words won't need to have an entry in the generation lexicon. This follows the philosophy of what is called the sparse dictionary.

The attributes chosen for the generation dictionary are, naturally, rather different from those used in the analysis dictionary. Therefore, we had to build a new dictionary table (ODTABLE), whose elements are, typically, attributes to denote the irregular forms of plural, feminine, verb tenses, etc.

At this moment, rules for generation may be separated into three main groups: terminal node rules, non-terminal node rules and morphological rules. All of these rules have one record on the left-hand side - this seems obvious - and one record on the right-hand side - this is somehow strange and will be explained afterwards in this section.

Terminal node rules are responsible for generating a segment of Portuguese text (usually a word). They test some attributes of the node and, if necessary, they trigger morphological rules accordingly. Actually, the morphological rules are triggered only if the word in question is regular on that specific attribute that is being tested (e.g., plural). If the word is irregular, then the necessary information is provided by the dictionary.

Non-terminal node rules act both as filters and sources of the information that is passed to the nodes below. They change, erase or create the attributes that will "flow" towards the terminal nodes. For example, the attribute GNUML, relative to the number in generation, must be passed to following nodes with its previous value by an AJP node, but, on the contrary, must be assigned a new value by each NP node.

Morphological rules are very simple in that they only generate the regular form of some morphological feature of a word, given a specific pattern of that word. A simple example to help clarifying ideas: the rules GPLURS and GPLURUL are used to generate the plural form of regular words whose last letters are S and UL respectively.

The approach followed so far for the generation phase assumes a very important feature about the whole graph that is received from the transfer phase. It assumes that all the information that is needed for a given node exists in the node itself and/or in the nodes over it. This is not at all trivial to achieve, since it implies that both analysis and transfer be directed towards this target. In fact, we do not have all the information we want, either because it simply does not exist, either because it is not in the right nodes (e.g., the indicator for the person of a verb is not present at a VP node).

As a last point, it should be mentioned a strange feature (at least, at a first glance) of the rules described. These rules encode one record into just one record, rather than into several ones. Why is this so? The fact is that, when the graph is received from the transfer phase, it is known its exact syntactic structure. It is just a matter of starting at the top node and going down through its pre-modifiers (PRMODS), its head (HEAD) and its post-modifiers (PSMODS). This is much more simpler and efficient than building a generation grammar, based on rules, which would play the opposite role of what an analysis grammar, as PEG, does. We think that PLNLP encoding rules, used in a one to many fashion, are much more suited for the creation of text or for the generation of text from an internal representation that has no implicitly defined syntactic structure (as would be the case of representations close to an "interlingua" approach).
We should say that the generation phase is still in a very early stage, and that the ideas described here will, definitely, suffer adjustments and modifications as work on this subject goes on.

**Conclusions**

We presented a skeleton of a PLNLP-alone translation system. The reasons for following this path were discussed, together with the reasons to take the chosen options, and the conclusions to this moment.

We find it a good method in what concerns modularity, readability, and simplicity.

We are conscious, however, that many problems will have to be dealt with which we have not yet encountered, and that, although this seemed to us the right time to present some detailed description of our work in Mentor, many statements here written may have to be reconsidered later.

As a last remark, it should be mentioned that nothing we present here or in the appendix is just a mere suggestion or declaration of intentions. On the contrary, all examples were actually developed and are now working in Lisbon.
Dealing with PEG’s hidden pointer attributes in MENTOR88/P

Diana Santos
February 1988

A characteristic of PEG which somehow difficults the transfer process is here described, together with a method to systematically deal with it. A classification of the records and their attributes in PEG is also presented as a starting point to this analysis.

Three kinds of attributes

Any attribute in a PLNLP record can point to a PLNLP record. It can be the null record (0) or a named record (most of the times only having its NAME attribute non-null), or it can be any other kind of record. An attribute can also have as values integers, real numbers or strings.

PEG manipulates PLNLP records, so, what was said holds for a PEG graph. However, it is the meaning PEG gives to the different attributes that matters in the classification we propose.

Looking at the different records in a PEG graph, we can divide them in terminal nodes (or tree leaves), which in PLNLP are represented by a record whose attributes are non records themselves (except for the trivial named records, whose only non-zero attribute is their name, a string); and non-terminal nodes. The last are, from a syntactical point of view, those who have at least an attribute pointing to a non-named record. (In fact, it is enough that they have an attribute that points to a non-trivial named record).

The terminal nodes in PEG are used to represent the words or punctuation marks that appear in the sentence, any other segments of text are “decoded” into non-terminal nodes.

Non-terminal nodes can, in PEG, be of two kinds: those who are the value of the attribute PRMODS, HEAD, or PSMODS of any non-terminal record, and those who aren’t. These who aren’t are auxiliary information that PEG provides, but “hidden” from the naive user, since it is not displayed in the “tree” that PEG creates.

We propose a classification of PEG attributes that follows from the one for records.

Simple attributes - those that do not point to records (with the exception of the trivial named records, that are no more than high level strings).

Open pointer attributes - those that point to records which are also pointed by PRMODS, HEAD or PSMODS. Examples: those three, SUBJECT, TOPIC, etc.

Hidden pointer attributes - those who point to records which are not PRMODS, PSMODS or HEAD of any other records. Example: PPOBJ, that can point to NPs or VPs which do not appear in the PEG tree.

The existence of this third kind of attributes proves that there is structural information that is not displayed in the tree, in addition to the other information associated with each record (like indicators or syntactical features, in a word, simple attributes), that are also obviously not displayed. They can be at once accessed, though, by displaying each individual record.

Hidden pointer attributes bring some problems to a systematic treatment of PEG results based on PRMODS, HEAD and PSMODS, that is, what we call the tree structure.
The PP problem

To clarify the problem at hand, namely the existence of records which do not belong to PEG’s tree structure, one example will be detailed.

To help the exposition, some general PEG rules will be stated:

All non-terminal nodes in PEG have the HEAD attribute non-zero. (or stated differently, at all levels but the lowest of PEG trees there is one HEAD).

The HEAD is always a terminal node, and as a corollary of this, the HEAD is only one record (neither a list nor a tree of records).

However, there are terminal nodes which are not HEADs of any records, like the particle of a phrasal verb or the preposition of a prepositional phrase (PP).

Let us follow carefully this last example, supposing, for illustration purposes, that the PP was built by a rule with the form PREP NP --> PP.

PREP has no HEAD, since it is a terminal node, and NP has one HEAD. This rule does not say that PREP should be integrated in the NP which would remain NP, but, on the contrary, that another segment of a different type should be built.

The “normal” kind of rule would create a new tree level, assigning the HEAD of the new node to the terminal node (that is, PREP), and PSMODS to the NP (which, according to the rules stated before, could not be a HEAD because it was not a simple record).

However, in this case all processing is performed at the same level. PP stays at the NP level, with a new PRMODS (the PREP), but without changing the HEAD, which is NP’s. Not to lose information, and to distinguish this case from a normal NP, two new attributes, PRP and PPOBJ, are created, pointing to PREP and the NP respectively.

While PRP is nothing more than a new link from the PP node to the PREP node, being so an open pointer attribute, PPOBJ points to a record NP which otherwise would be “floating” at the same level of the PP, and that does not belong to PEG’s tree structure.

The consequences for transfer

It may be convenient, at this moment, to explain why this fact may bring problems, or why it deserves special attention. The reason is related to the way transfer is implemented in our system.

As a short description, it can be said that the PEG graph is visited in a top-down way, each record being transferred taking into account the information of its attributes and that of the records pointed by them. In a tree terminology, we would say “only taking into account the records below and not above”. What should be done then when the records are at the same level, like PP and the NP that gave origin to it?

Briefly considering several possibilities, we shall see that transfer must be altered to correctly accommodate this idiosyncrasy of PEG.

Ignoring the hidden pointer attributes, and just following the open ones in transfer would result in that there would be no way to know what to do with the PP (considering that the tasks for transferring VPs or NPs, both possible PPOBJS in PPS, for example, are substantially different).

On the other hand, the problem with following hidden attributes as “normal” pointer attributes would be that, in a top-down approach, the changes done to the record pointed by PPOBJ, which should be equally reflected in the tree structure of the PP (that is, its PRMODS, HEAD and PSMODS) could not have any influence in that PP, since the transfer would be one level down when actually performing those changes.

Not to fall into a non-deterministic tree-traversal, that is, going up and down driven by the graph, giving up simplicity and determinism, some changes to the transfer procedure have to be done.

Discussion of several alternatives

The most straightforward way to deal with hidden pointer attributes would be to change them into open ones, that is, to make them be what we wish they were. For instance, whenever a PPOBJ
were found, transfer would create one more level, HEADed by the preposition, as described before, and then proceed with its unchanged algorithm.

This procedure does not, however, seem correct to us, since it is redoing English analysis, whose correctness we do not question, and so it would actually create some wrong structure for Portuguese in place of a former correct English one.

Another alternative would be to pay attention to the hidden pointer attributes, while traversing the tree, and when processing a record with this kind of attribute, perform as many additional transfers as their number.

This is conceptually the same as recognizing an implicit other level, while not changing the structure. So, the transfer of one PP, centered in its PREP, would be preceded by the transfer of its PPOBJ, a segment of type VP, NP or any other, and the changes it implied would also at once be reflected in the PP, prior to the corresponding changes in its PRMODS, PSMODS and HEAD, performed one level later.

The proposed procedure

The way to deal with hidden pointer attributes, not sacrificing the top-down approach nor randomizing the tree traversal can, in our opinion, be stated like this:

Follow the tree structure top-down when finding simple attributes or open pointer ones; use the hidden pointer attributes to, at the same level, refine the transfer.

Examples of the system performance using this method

This can be considered an addenda to the file EXAMPLES DATA, appendix of our "Proposal of a PLNL-based translation system".

An example of applying NP rules to a PP built from a NP:

I came with all my friends.

<table>
<thead>
<tr>
<th>DECL1</th>
<th>NP1</th>
<th>PRON1*</th>
<th>&quot;i&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERB1*</td>
<td>&quot;came&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PP1</td>
<td>PREP1</td>
<td>&quot;with&quot;</td>
<td></td>
</tr>
<tr>
<td>QUANT1</td>
<td>ADJ1*</td>
<td>&quot;all&quot;</td>
<td></td>
</tr>
<tr>
<td>DET1</td>
<td>ADJ2*</td>
<td>&quot;my&quot;</td>
<td></td>
</tr>
<tr>
<td>NOUN1*</td>
<td>&quot;friends&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Arvore portuguesa

<table>
<thead>
<tr>
<th>TRAD1</th>
<th>VP1*</th>
<th>NP2</th>
<th>PRON2*</th>
<th>&quot;eu&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERB2*</td>
<td>&quot;chegar&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PP2</td>
<td>PREP2</td>
<td>&quot;com&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUANT2</td>
<td>ADJ3*</td>
<td>&quot;todo&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DET2</td>
<td>ADJ4*</td>
<td>&quot;o&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DET3</td>
<td>ADJ5*</td>
<td>&quot;meu&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOUN2*</td>
<td>&quot;amigo&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Geracao

eu CHEGUEI com todoS oS meuS amigoS .

Modification of the TRANSFER procedure

// For each record, it guarantees its unique copy, it selects its right//
// translation and triggers the appropriated structural transfer rules.//
(mods.ne.0,seg=top<mods>,
  =pcopy<seg,ptrattrs>,
  // Now the structural transfer rules are finally triggered, with some //
  // small adjustments in the names of some segtyp2s.            //
  <SEGTYP2.EQ.'DET', SEGTYP2='DETER'>,
  <SEGTYP2.EQ.'QUANT', SEGTYP2='PQUANT'>,
  // The "hidden" attributes are considered, triggering each a new struct//
  // ural transfer.                                       //
  hiddenaux=hidden,
  (hiddenattr=top<hiddenaux>,
   @hiddenattr,
   =premove<seg,@hiddenattr>, =pcopy@hiddenattr, ptratts>,
   =encode<SEGTYP2(@hiddenattr), @hiddenattr>,
   PRMDS=PRMDS...PRMDS(@hiddenattr),
   HEAD=HEAD(@hiddenattr),
   PSMODS=PSMODS(@hiddenattr)...PSMODS>,
   hiddenaux=rest<hiddenaux>, hiddenaux.NE.0),
  // Invoking the structural rules corresponding to the actual segment. //
   =ENCODE<SEGTYP2,SEG>,

Dealing with PEG's hidden pointer attributes in MENTOR88/P
Lexical transfer in MENTOR88/P

Diana Santos
February 1988

We present here the way the basic translation system developed in Lisbon works when dealing with lexical transfer, or word choice.

To fully describe it several points must be emphasized: its delimitation in terms of the whole process, its timing related to the other tasks in transfer, and the actual procedure followed. For this last matter, it is essential to define which information is relevant, and how it is treated by the choosing algorithm.

Relationship between lexical and structural transfers

Lexical transfer, as implemented here, only concerns the choice of one particular Portuguese word among many, given an English word. (As a parenthesis, it should be mentioned that there have not yet been considered multi-word expressions, except for the prepositions associated with verbs, which already receive, at this moment, adequate treatment).

We achieve like this a complete separation between structural and lexical transfer, that is, if a chosen word requires a change of syntactical structure in the Portuguese sentence, that will be dealt with at another time, another place.

More specifically, for each word, that change will be performed after obtaining the right translation, during structural transfer.

Structural transfer has then available all features of interest, coming them from the English syntax, or the Portuguese word. As examples of the two kinds, we can present negation and the word "gostar" (possible translation of the verb "like", and which in Portuguese requires the preposition "de" before the English object).

Although the organization of the whole system was described in "Proposal of a PLNLP-based translation system", it is convenient to describe here with greater detail what we mean by "lexical transfer precedes structural one".

The access to the transfer dictionary is performed, for each record, using the value of its attribute BASE (which in PEG is the BASE (lemma) of the head of the segment of text the record represents). The information which is in the dictionary, connected to the actual situation described by that record and the ones it points to, is used to select the translation. Then, the new record (copy of the English word) which has already as BASE the right Portuguese choice, is sent to all structural rules applicable to its SEGYTP2, taking in account all possible features connected to the specific chosen word.

Only after that record was "completely" transferred, will another one be visited, with the corresponding access to the transfer dictionary, choice among the several possible translations, and triggering of the related structural transfer rules.

So when we say lexical transfer precedes syntactical changes, that is to be understood from a record point of view, and not considering the two kinds of transfer as wholes.

The reason to use this approach is that it allows that each record in the PEG graph will only be visited once.

Information with each possible translation

The information associated with lexical transfer must appear together with the corresponding entry in the English to Portuguese dictionary. Since there may be different translations with different conditions and/or requirements (these for structural transfer), that information must be associated with each Portuguese word in the dictionary.

The information we use is of two different kinds:
Conditions - to be tested and used in the actual choice. They can be negative or positive, that is, they can test whether a particular feature is present or absent in the segment of text dealt with.

Actions - they describe the possible actions that have to be performed later, during structural transfer. These "actions" are stored in the form of ordinary OD features (indicators or attributes with a value) that are interpreted by the rules of structural transfer.

This duality mirrors the two kinds of transfer we distinguish in the system, being the first type of information used for lexical transfer and the second in structural transfer.

**How to choose**

More specifically, the format of the condition information at this moment is the following:

Conditions are stated through the attribute COND and/or NCOND (for Negative CONDitation) in the transfer dictionary. They have as "values" pairs of features, with a special meaning:

The first element of the pair stores the name of the attribute of the current record to be tested, while the second has the value it must have (or should not have, in NCOND) in order to be chosen.

(For instance (COND NPIND TME) means that the word will only be chosen if TME is in the attribute NPIND of the record).

If the first value of the record is the word "SELF", the second value stands for the name of the attribute whose existence is required.

(For example, (COND self OBJECT) means that the word can only be chosen if the current record has an attribute OBJECT not null.

Finally, if the first element of the pair is itself a pointer attribute of the current segment, the second value stands for the attribute which should (or should not) exist in the record pointed by the first element.

(For instance, (NCOND SUBJECT ANIM) only chooses the word it is associated to if the feature ANIM does not exist in the record pointed by the subject; (COND PSMODS LOC) requires that that word should only be chosen if there be the feature LOC in any record pointed by PSMODS of the current record).

The examples at the end of this paper may help to fully understand the syntax.

**The structural information**

Although this is separated from the choice itself, it represents its consequences, so it is here mentioned.

We believe the choice of the necessary markers (specific to the pair of languages English-Portuguese) is not a trivial task, and has to be carried based on a linguistic theory, and not in an ad-hoc fashion as it is being done now. However, the storing and consequent processing of structural attributes was implemented, so that we could test the whole system, as was described in "Proposal of a PLNLX-based translation system".

At this moment, the working features are:

The feature OMISSO for nouns. That kind of nouns in Portuguese is in general omitted in the sentence.

The feature VPREP for verbs. Its value indicates the particle that that verb needs in Portuguese, for its compulsory argument.

The feature ANTES for adjectives. It indicates that they should be placed before the noun they modify, as an exception to ordinary adjectives in Portuguese.

As already emphasized, these features were only implemented to prove the feasibility of the system and are not intended to show any completeness or linguistic correctness.
The organization of the dictionary

Just before the actual examples, we would like to refer some characteristics of the dictionary organization: first of all, what we call an "open" representation.

By this is meant that the system does not require that all words have an exhaustive disambiguation method associated, nor that a single word will always be chosen as the only correct one.

On the contrary, the dictionary choice is implemented as the selection of all possible words (and not only one). To produce a single sentence, the first of the list is used.

However, and for instance for stylistic reasons in the generation process, it might be useful that others could be accessible as well. While this may be a rather particular detail, which may even prove to be impossible to implement in the end, the way the dictionary access is done in this system allows for incremental refinement, that is, if only a small set of words has stored disambiguation features, that does not limit the use of any other words.

A second feature we want to describe is another consequence of storing the possible translations in an ordered list. It allows the choice to use probabilistic results, or in other words, when there was not enough information to help select the best translation, to choose the one which was right in the majority of the cases could be a convenient thing to do.

Finally, the concept of order also diminishes the information that is necessary to store, related to disambiguation. For instance, there is no need to state the opposite conditions in the words following, since it is guaranteed that the next word will be chosen if and only if the first did not apply.

Also words with no conditions that appear in the end of the dictionary list are used as default translations. (However, we are aware that this saving can hide some of the information that is stored in the dictionary, implicitly and not explicitly).

Some examples of lexical transfer implemented

stand
suportar (VERB (COND SELF object))
ficar (VERB (NCOND SUBJECT anim) (COND PSMODS loc))
estar (VERB (COND SUBJECT anim))

Above is an example that may result in no possible translation. Also note that "estar" has not the opposite conditions of "ficar", since "ficar" requires both that the SUBJECT do not have the indicator ANIM and that there be the indicator LOC in some PSMODS.

last
demorar (VERB (NCOND SELF object))
durar (VERB)
passado (AJAVPP (COND NPIND time))
ultimo (AJAVPP ANTES)
final (AJAVPP)

Above and in the present situation, the word "final" could never be chosen instead of "ultimo". However, in the future some more information could be added that allow to distinguish between both words, or, as proposed before, a more complex way to choose among many correct ways of saying the same thing could be implemented.

appear
appearan (VERB (COND SELF object))
appeartwo (VERB (COND SELF predcomp) CHNPORDER)
appeartwothree (VERB (COND SELF predicomp))
appearfour (VERB (COND SELF predadj))
appearfive (VERB)

Above is the example presented by Moris Rimco in his "Working Memo" of October 1987. The feature CHNPORDER stands for change NP order, and would be tested in some structural rule for VPs, which would have to put the subject of the sentence after the verb. While this example was implemented and tested in what concerned the choice of the right word, the attribute CHNPORDER is fictitious, and is only here to represent the complete example.
Illustration of the system’s behavior

Any discourses last.

DECL1 NP1 QUANT1 ADJ1+ "any"
   NOUN1* "discourses"
   VERB1* "last"
PUNC1 "."

Arvore portuguesa

TRAD1 VP1* NP2 QUANT2 ADJ2+ "qualquer"
   NOUN2* "discurso"
   VERB2* "demorar"
PUNC2 "."

Geracao

== == > QUASISQUER discursos demorAM.

The process lasted many years.

DECL1 NP1 DET1 ADJ1+ "the"
   NOUN1* "process"
   VERB1* "lasted"
   NP2 QUANT1 ADJ2+ "many"
   NOUN2* "years"
PUNC1 "."

Arvore portuguesa

TRAD1 VP1* NP3 DET2 ADJ3+ "o"
   NOUN3* "processo"
   VERB2* "durar"
   NP4 QUANT2 ADJ4+ "muito"
   NOUN4* "ano"
PUNC2 "."

Geracao

== == > o processo durOU muitoS anoS.

The blue one stands there.

DECL1 NP1 DET1 ADJ1+ "the"
   ADJ1+ "blue"
   NOUN1* "one"
   VERB1* "stands"
   ADV1+ "there"
PUNC1 "."

Arvore portuguesa

TRAD1 VP1* NP2 DET2 ADJ3+ "o"
   NOUN2* "azul"
   VERB2* "ficar"
   ADV2+ "ali"
PUNC2 "."

Geracao

== == > o azul ficA ali.

He stands two weights.

DECL1 NP1 PRON1+ "he"
   VERB1+ "stands"
   NP2 QUANT1 ADJ1+ "two"
   NOUN1* "weights"
Arvore portuguesa

TRAD1 VP1* NP3 PRON2* "ele"
   VERB2* "suportar"
   NP4 QUANT2 ADJ2* "dois"
   NOUN2* "peso"
   PUNC2 "."

Geracao

He stands in the doorway.

DECI1 NP1 PRON1* "he"
   VERB1* "stands"
   PP1 PREP1 "in"
   DET1 ADJ1* "the"
   NOUN1* "doorway"
   PUNC1 ""

Arvore portuguesa

TRAD1 VP1* NP2 PRON2* "ele"
   VERB2* "estar"
   PP2 PREP2 "em"
   DET2 ADJ2* "o"
   NOUN2* "soleira"
   PUNC2 "."

Geracao

They caught the last red train.

DECI1 NP1 PRON1* "they"
   VERB1* "caught"
   NP2 DET1 ADJ1* "the"
   AJP1 ADJ2* "last"
   AJP2 ADJ3* "red"
   NOUN1* "train"
   PUNC1 ""

Arvore portuguesa

TRAD1 VP1* NP3 PRON2* "ele"
   VERB2* "apanhar"
   NP4 DET2 ADJ4* "o"
   AJP3 ADJ5* "ultimo"
   NOUN2* "comboio"
   AJP4 ADJ6* "vermelho"
   PUNC2 ""

Geracao

He came last summer.

DECI1 NP1 PRON1* "he"
   VERB1* "came"
   NP2 AJP1 ADJ1* "last"
   NOUN1* "summer"
   PUNC1 ""

Arvore portuguesa
Geracao
===
= = = > ele chegOU vera o passado.
Report on a PLNLP-based translation system

by     Diana Santos
       Paulo Libano Monteiro

March 1988

the Portuguese branch of the Mentor project.

Introduction

The conclusions of the work done from December 1987 till February 1988 will be briefly presented, together with specific examples and discussion of the options that have been taken.

All work described in this report only reflects the Portuguese opinions and experience, while much distinct work has been done in Haifa, Madrid, and Helsinki, following other approaches.

The aim of conducting this work was primarily to test the possibility of using PLNLP as the only software tool in Mentor, as opposed to other approaches, namely that of using the Tree Transducer (Hiroshi Nomiyama Japanese Processing Group, JSI, IBM Japan, NOMIYAMA @ TOKVMS11) for the transfer phase.

The results obtained show that it is actually possible to follow this procedure. The ultimate decision will, however, have to be taken after a careful comparison and discussion of all methods.

We hope to contribute with this work to choose the best final strategy that will eventually be followed by Mentor.

Starting points

Mentor as a translation system can be classified as transfer-based, and mostly syntactically oriented. This means that there will be three distinct phases in the translation process, usually called analysis, transfer, and generation; and that the system will recur only to syntax whenever this is possible.

The analysis phase, which decodes an English sentence and represents it by an internal structure, is done by PEG (Karen Jensen, “PEG 1986: A Broad-coverage Computational Syntax of English”, IBM Thomas J. Watson Research Center), who follows a syntactical approach, and can be defined as of almost unrestricted linguistic coverage.

The task of Mentor is to transform the output of PEG into the corresponding sentences of the target languages, at this moment Finnish, Hebrew, Portuguese and Spanish.

In this framework, we derive the following conclusions:

Although the target languages are considerably distinct, we believe that both the methodology and the architecture of the systems developed should be as similar as possible, not only to allow for a broad collaboration between teams, but mainly to achieve a multilingual system and not four unilingual ones.

This implies, in our opinion, that everything that concerns at least the transfer phase should receive the same treatment in all languages, and that standardization should be tried even for generation. In particular, dictionaries should have similar organizations, and the transfer and generation modules should be independent from each other.
The basis for our work

Being the language in which the first phase of translation was developed (independently of Mentor), PLNL (George Feldman, "Natural Language Inputs to a Simulation Programming System", Technical Report NPS-55HD72101, Naval postgraduate School, Monterey CA), we chose to investigate the feasibility and advantages or disadvantages of implementing the whole system in PLNL.

The conceptual design

The priorities we chose to elect were clearness of design, conceptual correctness, and portability and modularity, in the sense that the system may be easily extended and changed, namely to add or remove knowledge.

From now on, and for simplicity of exposition, we use Portuguese as the target language, but it should be noticed that one of the aims of this work was to produce results that could apply to all languages involved.

The system is divided in three phases, each one with a dictionary of its own. PEG is in charge of the first phase, with OD (the Optimized Dictionary, Stephen Richardson, IBM J. Watson Research Center). We are not entitled to do any change or modification of PEG, as is obvious. Our only concern should be report to Karen Jensen any irregularities or information about the English sentence we miss in PEG's analysis. We are sure that she will do any changes in her grammar that appear of linguistic significance in the frame of the English language alone.

The transfer phase has as its goal the producing of a new graph-structure for Portuguese, departing from the English one. Its output should be exactly the one a GPP (PLNL Portugal Grammar) would produce, if there were one.

The reason for assigning this goal for this phase, and not any minor one, stems from the fact that the generation phase, in our opinion, should be completely independent of the source language that was used, that is, irrespectively of the former, the same generation module for Portuguese should always be used.

Generation encounters then as its main task the organization of the Portuguese surface structure, from the syntactical structure of a particular sentence and the particular lemma of the words that take part in it.

The operations to be done at this phase are then mainly morphological ones (maybe later style considerations can be considered). It has been a constant in the development of generation code to make it source-language independent, that is, to use solely Portuguese features, and never any English-specific categorizations.

The work actually developed will now be described.

The actual design of the transfer phase

Transfer is directed by a procedure with a simple algorithm: It travels "down" the graph, copying each English record into a Portuguese one, (built with information from the English-to-Portuguese dictionary), and delivering this new record to a set of transfer rules. Both copying and transferring are implemented using PLNL encoding rules, with one record in the left and one record in the right.

The choice for a copy instead of actually changing the English graph seems conceptually more correct to us, although it implies more work. This way, we distinguish between the English attributes (some of which are maintained in the new graph, others not) and the Portuguese ones, that should not crowd nor change the English internal structure. Moreover, the values of the same attribute can differ in the two languages (as they trivially do in their BASE and STR attributes) and so the existence of two different entities makes sense. Finally, the fact that PEG produces a graph and not a tree is taken care solely by copy, being transfer rules freed of the burden of checking whether the record they are dealing with has already been visited or not. The algorithm guarantees that each record will be visited only once, while at the same time no information of PEG is lost.

The actual transfer is performed by rules, indexed by the SEGTYPE2 of the records of the English graph structure. The choice of this attribute instead of SEGTYPE is due to the fact that it repres
ments a finer syntactic classification than this last, consisting so in the ultimate word about the particular segment of text it represents.

The reason that made us choose rules instead of procedures is the following: in our opinion, rules have a much superior readability, apart from the fact that rules are modular and stated independently. Instead of changing a procedure any time we want to add a minor change, it is enough in a rule-based system to add or remove rules. This is essential to make the system development easy.

The question of transfer is normally addressed dividing the process in two distinct acts: structural transfer, and lexical transfer.

The first is related to the syntax properties of the elements to manipulate and may produce a syntactical change when executed.

The second is related to the actual words in the sentence, and brings about the problems of meaning and word sense ambiguity.

What has been described so far is the structural transfer. This process, as has been implemented here, although indexed by syntax classes (the SEGTYPEs) may recur to any characteristics of the PLNLPL records, including the actual word (or, more accurately, its lemma). That means that we do not consider structural transfer independent of the words; we classify a transfer as structural whenever any information in the record structure, representing the segment of text we are dealing with, results in a syntactical change.

When, on the other hand, it is the choice of a particular word between many alternatives that is in question, we are dealing with lexical transfer and its result should only produce the right translation for a given word (or idiom). No other changes are to be performed with lexical transfer. This last statement does obviously not mean lexical transfer is trivial...

It may seem strange, however, considering that there are many cases when a particular choice of a translation in Portuguese means changing the whole structure of the English sentence. What has been said so far only separates lexical transfer from the structural one. This last will be performed after the convenient choice of the Portuguese word (with any idiosyncrasies associated), and it will be performed taking those characteristics in account.

This is an important point, since, with this design, there is no need to perform two different structural transfers: the only structural change is done after the access to the transfer dictionary and is based both in the English and in the Portuguese features at once.

We obtain this behavior with two complimentary devices: the use of (structural) transfer rules that deal with all matters pertinent to syntactical change, come they from English syntax or from the characteristics of the Portuguese words themselves, and the organization of the transfer dictionary accordingly.

In a small parenthesis, we decided to use an OD-based dictionary, again to follow the philosophy of modularity already describe. As the first phase defined its own OD, we chose to create two others, one for transfer, and other for generation. The most important is, however, which information is actually stored in it.

To understand how can lexical and structural transfers coexist in one dictionary, its organization will be described.

For each English lemma (stored as the BASE attribute in PEG), there is one or many translations, represented by the corresponding lemma in Portuguese, followed by the part of speech it belongs (as OD requires). Optionally, two different kinds of information may be stated, syntactical features related to the words in Portuguese (e.g., whether an adjective does not conform to the usual syntax, or whether the word is omitted when it has premodifiers); and conditions for choice when there is more than one candidate (e.g. existence or not of attributes in the record; context).

The first group is to be used during structural transfer, the second is what makes possible the lexical one.

When copying the English record into the Portuguese one, the dictionary is accessed and lexical transfer is performed, by calling a procedure that chooses, between the several alternatives available, which word (and consequent information it has associated) to copy from the transfer dictionary.

When (structural) transfer rules are triggered, the choice has already been made. The rules will be activated based in the information got from the E-to-P dictionary, together with all the other information put at our disposition by PEG.
The choice of two different ways of performing the two faces of transfer was carefully considered. While it was clear from the beginning that PLNLP rules were clearly suited for implementing structural transfer, we decided to implement the choice of the right translation with a PLNLP procedure, which runs through the list of the candidates electing those that do not violate any conditions.

Another alternative would be to encode all the rules of choice in PLNLP rules, and apply them indexed by the actual word (or better the BASE attribute). This is clearly impractical, although it would favor readability: to have a rule (or more) for each word in English would mean a system not easily expandable or modified, not to speak of resource issues.

On the other hand, a fixed number of rules and an easy-to-improve-and-enlarge dictionary are much more both realistic and user-friendly, and of course more programmer-friendly too...

This, we think, is enough to give up representing words by rules. History backs us too, since there was a time when PLNLP followed this approach, and it changed it to O& when growing up.

However, till now only negative arguments were presented, that is, reasons against the alternative but not for the method chosen. It seems to us that there is also a reason why the information should be attached to the dictionary, and not somewhere else: the rules of how to choose one word can be seen of being related primarily to the word itself, so should be stored with it.

At last, we explain why we decided to separate the two levels of transfer. From a theoretical point of view, it seems to us that the very existence of this distinction is enough to be taken into account when implementing a translation system.

From a practical point of view, it considerably simplifies the problem since it reduces it to two smaller ones, in that way easier to control and define.

From an implementation point of view, it frees our transfer rules from matters of actually choose one word and behave differently according to the one chosen, and it considerably reduces the actual number of transfer rules. On the other hand, to add a new word to the dictionary, with this architecture, does not imply any changes to the PLNLP code (except if another dictionary feature were introduced) while otherwise it enforces changing the code, at least to write a new transfer rule for the new word.

To end the section on transfer, please notice that as an appendix, we present the basic procedures together with some sample rules, and the corresponding entries in the dictionary.

The actual design of the generation phase

As a starting point, the graph produced by the transfer phase is passed to the generation one; this is the only communication between the two phases. This graph includes all relevant information from both analysis and transfer and, obviously, from the corresponding dictionaries.

The algorithm that goes through the graph, in order to perform generation, is somewhat similar to the one used in the transfer phase: the graph, actually with the form of a tree, is traversed top-down and left-to-right. At each node, the generation dictionary is eventually accessed and a rule is triggered by the corresponding SECTYPE attribute. Terminal node rules are different from the others in that they produce a segment of the Portuguese sentence; non-terminal node rules serve as filters of the information that is carried through the nodes below until the “leaf” is reached.

Unless the required word has already been read, the generation dictionary is accessed, at each node, using the BASE attribute, which, as already mentioned, is the linguistic lemma. What is stored in the dictionary concerns essentially morphological features of the words. Each entry in the lexicon has one or more sub-entries, depending on the number of parts-of-speech that the word may assume. Only irregular forms and other relevant information have to be placed; particularly, if a verb is irregular in a specific number and person of a tense, then only that irregularity has to appear in the dictionary. Regular forms are handled by specific rules, so that the majority of the words won’t need to have an entry in the generation lexicon. This follows the philosophy of what is called the sparse dictionary.

The attributes chosen for the generation dictionary are, naturally, rather different from those used in the analysis dictionary. Therefore, we had to build a new dictionary table (ODTABLE), whose elements are, typically, attributes to denote the irregular forms of plural, feminine, verb tenses, etc.

At this moment, rules for generation may be separated into three main groups: terminal node rules, non-terminal node rules and morphological rules. All of these rules have one record on the left-hand
side - this seems obvious - and one record on the right-hand side - this is somehow strange and will be explained afterwards in this section.

Terminal node rules are responsible for generating a segment of Portuguese text (usually a word). They test some attributes of the node and, if necessary, they trigger morphological rules accordingly. Actually, the morphological rules are triggered only if the word in question is regular on that specific attribute that is being tested (e.g., plural). If the word is irregular, then the necessary information is provided by the dictionary.

Non-terminal node rules act both as filters and sources of the information that is passed to the nodes below. They change, erase or create the attributes that will "flow" towards the terminal nodes. For example, the attribute GNUMB, relative to the number in generation, must be passed to following nodes with its previous value by an AIJ node, but, on the contrary, must be assigned a new value by each NP node.

Morphological rules are very simple in that they only generate the regular form of some morphological feature of a word, given a specific pattern of that word. A simple example to help clarifying ideas: the rules GPLURS and GPLURUL are used to generate the plural form of regular words whose last letters are S and UL respectively.

The approach followed so far for the generation phase assumes a very important feature about the whole graph that is received from the transfer phase. It assumes that all the information that is needed for a given node exists in the node itself and/or in the nodes over it. This is not at all trivial to achieve, since it implies that both analysis and transfer be directed towards this target. In fact, we do not have all the information we want, either because it simply does not exist, either because it is not in the right nodes (e.g., the indicator for the person of a verb is not present at a VP node).

As a last point, it should be mentioned a strange feature (at least, at first glance) of the rules described. These rules encode one record into just one record, rather than into several ones. Why is this so? The fact is that, when the graph is received from the transfer phase, it is known its exact syntactic structure. It is just a matter of starting at the top node and going down through its pre-modifiers (PRMODS), its head (HEAD) and its post-modifiers (PSMODS). This is much simpler and efficient than building a generation grammar, based on rules, which would play the opposite role of what an analysis grammar, as PEG, does. We think that PLNL encoding rules, used in a one to many fashion, are much more suited for the creation of text or for the generation of text from an internal representation that has no implicitly defined syntactic structure (as would be the case of representations close to an "interlingua" approach).

We should say that the generation phase is still in a very early stage, and that the ideas described here will, definitely, suffer adjustments and modifications as work on this subject goes further.

Problems with PLNL, PEG and OD

This section presents most problems we had with the system, tools and programs our prototype is based on.

The first and most important problem is obviously the lack of documentation, from the compiler characteristics to the switches of PLNL shell, from the problems dealt by PEG to what most attributes mean, and finally from the information that is stored in OD to that that is only in part there or does not exist at all at the moment.

To try to solve them somehow, some pages on PLNL were written (which, by the way, is the less documented of the three), after some trial-on-error work with the system. We are, however, sure, that a lot of characteristics remain to be discovered.

With the other two, PEG and OD, the interaction with their authors has been till now enough to use them without loss of efficiency from our part, but, as we may believe, that interaction has been a burden to Karen Jensen and Steve Richardson, which could be avoided if there were more documentation.

The weakest part of PLNL is undoubtedly its compiler into LISP:

It does not accept some characters as names of rules, which we would like it to, namely our accents and the tilde. (The reason of this "need" can be better understood in the examples section).
It has some idiosyncrasies that may bring problems, such as: the renumbering of (-56789) to ENC5678, for instance, or the fact that it modifies strings (for example, that brings us problems with accents).

Finally, the messages it issues are not readable at all, together with the fact that many errors (like a missing section heading, or a bad position of a procedure's name) are not pointed out.

Also, there are some details that have to be taken into account when mixing several levels of encoding rules (for instance, recursive calls to the encoding algorithm) which would be better if dealt automatically by the system.

To use PLNL in writing a machine translation system based in PEG is, however, in our opinion, relatively simple, and moreover because we developed with this work a small set of basic procedures.

We think that there won't be any need to dwell in the interiors of PLNL to write a transfer rule if using them, or, in other words, most work we had to get to know the language and its implementation details will not have to be repeated by those who want to do a similar task in PLNL.

Concerning PEG, her main problem is maybe the lack of a written systematization of what is done to any class of problems. This means that, in some cases, to know how she would work we have to actually probe it. But we are aware that such a systematization may cost a couple of years' work, and even may prove impossible, in view of the interaction of rules in the parallel decoding algorithm, and her data-driven philosophy. There is, on the other hand, a very good support from Karen Jensen, that makes use of our problems to improve PEG and cases at the same time some of the tasks we have in translation.

Finally, the ordering of the multiple syntactical parses of the same sentence, through a parse metric, frequently leads to a wrong choice. We think that some improvements could be done in that field too, to what the reporting of the problems for translation by MENTOR and even the suggestion of some algorithms could be of help.

As to OD, there are some inconsistencies in the knowledge stored in it, such as features never used, or not appearing where they should, as in phrasal verbs. The work in these fields is continuing, though, so we do not think it necessary to elaborate on that.

Conclusions

We presented a skeleton of a PLNL-alone translation system.

As a conclusion, we think that we not only proved that it was possible to do it like that, but that the method described has as advantages its modularity, readability, and simplicity.

We are conscious, however, that many problems will have to be dealt with which we have not yet encountered, and that, although this seemed to us the right time to present some detailed description of our work in Mentor, many statements here written may have to be reconsidered later.

As a last remark, it should be mentioned that nothing we present here or in the appendix is just a mere suggestion or declaration of intentions. On the contrary, all examples were actually developed and are now working in Lisbon.

It must also be emphasized, that the opinions here stated do not belong to Mentor, but only to the Portuguese team. Different paths have been followed by each team, hoping that in the end the experience of all can contribute to the best choice.

Examples

The following lines are examples of the performance of the system, showing the results of the two phases - transfer (another tree) and generation - one sentence.

The changes to the Portuguese words from transfer done by generation are presented in capital letters, contrasting with their "roots" coming from the dictionaries.

Some PLNL code follows, taken from transfer, and from generation.

Finally, some entries in the dictionaries are presented, where the present (and not the future) information is displayed. It is not much what our dictionaries contain up to this date, but we decided
to display their contents for some words, so that the examples could be understood and the format proposed could be visualized.

**EXAMPLES of English-to-Portuguese translation**

```
I want to go to the office to see my father.
```

---

**Arvore portuguesa**

```
TRAD1 VP1* NP3PRON2* "eu"
VERB4* "querer"
INFCL3 VERB5* "ir"

? PP2PREP2 "a"
DET3ADJ3* "o"
NOUN3* "escritorio"

? NP4PREP3 "para"
VERB6* "ver"
DET4ADJ4* "meu"
NOUN4* "pai"

```

---

Geracao

```
=> eu querer ir a o escritorio para ver meu pai .
```

---

To want him to come and not to see is a way to solve the problem.

---

**Decl**

```
NP1PRON1* "i"
VERB1* "want"
INFCL1 INFTO1 "to"
VERB2* "go"

? PP1PREP1 "to"
DET1ADJ1* "the"
NOUN1* "office"

? NP2DET2ADJ2* "my"
NOUN2* "father"
```

PUNC1 " . "

---

Report on a PLNLP-based translation system 7
Arvore portuguesa

TRAD1 VP1* NP1 DET1 ADJ1* "o" NOUN4* "instrumento"
? PP2 PREP2 "para" VERB3* "usar" NP4 DET5 ADJ5* "o" NOUN5* "espingarda"

VERB4* "ser"
PP3 PREP3 "em" DET6 ADJ6* "o" NOUN6* "cozinha"
PUNC2 "."

Geracao

===> o instrumento para usar A espingarda E# em A cozinha .

*****************************************************
To kill people, the instrument to use is the gun.

He is careful not to execute wrong commands.
She listened to the music to rest, and looked at the boy.

Geracao

----> ele E# cuidadoso em na o executar comandoS erradoS.

*************************************************************************

She sings lovely old songs.

*************************************************************************

Report on a PL-NLP-based translation system
Geracao

elá cantA cancO ES antigAS encantadorAS .

************************

All blue ones wanted him to disappear.

DECL1  NP1  QUANT1  ADJ1*  "all"
   AJP1  ADJ2*  "blue"
   NOUN1*  "ones"
   VERB1*  "wanted"
   NP2  PRON1*  "him"
   ? INFCL1  INFTO1  "to"
   VERB2*  "disappear"
   PUNC1  "."  

Arvore portuguesa

TRAD1  VP1*  NP3  QUANT2  ADJ3*  "todo"
   DET1  ADJ4*  "o"
   NOUN2*  "azul"
   VERB3*  "querer"
   VP2  COMPL1  "que"
   NP4  PRON2*  "ele"
   VERB4*  "desaparecer"
   PUNC2  "."  

Geracao

elóS oS azUIS QUISERAM que ele desaparecESE .

******************************************************************************

They do not like children.

DECL1  NP1  PRON1*  "they"
   VERB1  "do"
   AVP1  ADV1*  "not"
   VERB2*  "like"
   NP2  NOUN1*  "children"
   PUNC1  "."  

Arvore portuguesa

TRAD1  VP1*  NP3  PRON2*  "eles"
   AVP2  ADV2*  "na o"
   VERB3*  "gostar"
   PP1  PREP1  "de"
   NOUN2*  "crianc,a"
   PUNC2  "."  

Geracao

elóS na o gostAM de crianc,aS .

******************************************************************************

Any discourses last.

DECL1  NP1  QUANT1  ADJ1*  "any"
   NOUN1*  "discourses"
Some lines of PLNL code which integrate the system are now presented. The core rule:

```
4010 TRANSL(!NL-PERICAO.EQ.NIL) -->
   PTRANS(HEAD=TRANSL,SEGTP2='TRAD',
           SEGTP2(HEAD)=typechoice<SEGTP2(HEAD),
           transfer<SEG>,SUP='''')
   PPRINTTREE(pmtree<TRANSL>,SUP='''')
   PGENER(%%TRANSL,
           SUP='''',!NL-PERICAO='',
           pgenerat<SEG>)
   OUTPUT(#11=1,#13='''')
   PDISPLAY(SUP=!NL-PERICAO,NL-PERICAO=NIL)
```

**EXAMPLES of the code of transfer**

The main copy rule:

```
4030 ORIGINAL(~COPIA) --> COPIADO(ORIGINAL,
       COPIA(ORIGINAL)=SEG,
       SUP='''',-NODENAME,
       -COPIOF,-RULES,-RULE,-VERB,-NOUN,
       -ADJ,-ADV,-POS,-CLOSED)
```

Some transfer rules:

```
11000 NOUN(~ALTERADO) --> NOUN(NOUN,BASE=BASITOP<ESC>),
       STRI=STR(TOP<ESC>),
       ALTERADO=1)
```

// The following rules delete the auxiliary verbs for negation, while //
// preserving the relevant information they carry.

```
4500 VP(NEG,
       pfnd<seg,'BASE','DO',PRMDS,'FOUND'.NE.0)
       -->
       VP(%%VP,=pdelete<SEG,top<FOUND>,PRMDS'),
```

Report on a PLNL-based translation system
NUMB=NUMB(top<FOUND>),
TENS=TENS(top<FOUND>),
-FOUND)

// The following rules insert a determiner in NPs which begin with the //
// word ALL.

(5700) NP(BASE(top<PRMODS>).EQ.'ALL',
SEGTYP2(top<PRMODS>).EQ.'QUANT',
<pfind<seg,'SEGTYP2', 'DET',PRMODS,'FOUND'>.EQ.0]  
POSS(top<FOUND>))

->
NP(%%NP, SAVETOP=top<PRMODS>,pdelete<SEG,top<PRMODS>, 'PRMODS',
add<SEG,%'ARTI', 'PRMODS', 'before'>,
HEAD(top<PRMODS>)=%'ARTAUXIL',
add<SEG,SAVETOP,'PRMODS', 'before'>,
-SAVETOP,-FOUND)

// The following rule(s) change NPs related to adjectives : except if //
// explicitly written in the dictionary, (with the feature ANTES), the //
// position of adjectives will change from before to after the noun, //
// with the relative closeness not altered.

(5500) NP(~ADJS, ADJS='done',
  pfind<SEG,'SEGTYP2', 'AJP',PRMODS,'FOUND'>.NE.0)

->
NP(%%NP,
   (<-ANTESTM(top<ESC(top<FOUND>))>,
    =pdelete<SEG,top<FOUND>, 'PRMODS',
    =add<SEG,top<FOUND>, 'PSMODS', 'before'>),
    FOUND=rest<FOUND>,
    FOUND.NE.0),~FOUND)

// This rule accounts for the appearance of an infinite clause embedded//
// in the PSMODS of a VP.

// This fact must be taken in account in the following situations :
// * If there is an OBJECT in the VP, the infcl will turn out to be //
// a relative clause in the subjunctive, if the verb is classified //
// as NPTOV and THATCOMP. Otherwise, it will follow normal process//
// ing, which by default is a final clause.
// Examples: I want him to go. Quero QUE ele va'.
// He stopped the car to smoke. Parou o carro PARA fumar.
// * If the verb is COPL, and the infcl has an explicit "to", and the//
// subject is indefinite ("it"), then the infinitive clause will //
// stay as it is (without "to" of course) ...
// Examples: This is impossible to read. Isto e' impossivel DE ler.//
// It is impossible to read... E' impossivel ler.

(4901) VP(~INFCLS,INFCLS='done',
  pfind<seg,'SEGTYP2', 'INFCL',PSMODS(top<PSMODS>),
   'FOUND'>.NE.0)

->
VP(%%VP,
  <OBJECT, SECTYPE(OBJECT).ISIN.'AJP'...'NP',
  NPTOV,THATCOMP,
PUTCOMPL(OBJECT)=1,
CHICTOHD(OBJECT)=SEGTYPE(OBJECT),
SEGTYPE(OBJECT)='VP',
+SUBJUNCR(OBJECT),
TENS(OBJECT)=TENS>,
<PREDJH, INFTO(top<FOUND>).NE.0,
BASE(SUBJECT).EQ.'it', CHICNOTO(PREDJH)=1>,
-FOUND)

*******************************************************************************

Some procedures of interest for transfer

// This function receives the name of an attribute and a value, and a //
// list of records. It returns a new list composed by the records of //
// the previous one whose attribute ATT has the particular value VAL. //
// The value returned is also assigned to its last parameter, OUTLIST.//

PFIND (seg*PTR, att*CHR, val*ANY, inlist*LST, outlist*LST,
auxil/PTR, auxlist/LST/0,

   (inlist. NE.0,
    auxil=top<inlist>,
    @att(auxil).EQ.val, auxlist=auxlist...auxil,
    inlist=rest<inlist>,
    @outlist=auxlist,
    <-auxlist)

The core procedure for transfer:

// TRANSFER receives a segment (of any type) and defines the form of  //
// navigation in the graph according to it. In each new record that is //
// created, the transfer rules which treat its segment type are invo- //
// cated. On ending, it signals through the global variable NLP-GERACAO//
// that the transfer phase has finished.
// The top of the encoding stack is restored, doing transfer this way //
// no harm to the following records in rule number 4010.  //

TRANSFER (seg*PTR,
   mods/PTR, ptratts/LST, attrs/LST, lista/LST,
   resul/LST, attr/PTR, auxil/PTR,
   saveseg/PTR, hidden/LST, hiddenattr/PTR, hiddenaux/LST,

   saveseg=seg,
   mods=node<seg>,

   // A list containing all the pointer attributes of PEG is built, not to//
   // lose any information from PEG analysis.  //
   ptratts='PRMODS'..'PSMODS'..'TOPIC',
   ptratts=ptratts...'SUBJECT'..'OBJECT'..'INDOBJ',
   ptratts=ptratts...'PREDADJ'..'PREDNOM'..'PREDCOMP',
   ptratts=ptratts...'EXTRAPOS'..'PARTICL'..'INVERT',
   ptratts=ptratts...'POBJ'..'PRP'..'SUBJ1'..'SUBJ2',
   ptratts=ptratts...'FRSTV'..'INFTO',

   // A list containing all "hidden" pointer attributes is built.  //
   hidden='POBJ',
   // The main part of this procedure simple cycles through this loop, that/
// covers all records of one solution of PEG analysis. //
// For each record, it guarantees its unique copy, it selects its right //
// translation and triggers the appropriated structural transfer rules. //

(mods.NE.0, seg=top<mods>,
 =pcopy<seg, ptrattrs>,
 // Now the structural transfer rules are finally triggered, with some //
 // small adjustments in the names of some segtyp2s. //
 <SEGTYP2.EQ.'DET', SEGTYP2='DETER'>,
 <SEGTYP2.EQ.'QUANT', SEGTYP2='PQUANT'>,
 <SEGTYP2.EQ.'INFCL', SEGTYP2='VP'>,
 // The "hidden" attributes are considered, triggering each a new struct //
 // ural transfer. //
 hiddenaux=hidden,
 (hiddenattr=top<hiddenaux>,
 =<hiddenattr,
 =premove<seg, @hiddenattr>, =pcopy@hiddenattr, ptrattrs>,
 =encode<SEGTYP2(@hiddenattr), @hiddenattr>,
 PRMODS=PRMODS...PRMODS(@hiddenattr),
 HEAD=HEAD(@hiddenattr),
 FSMODS=FSMODS(@hiddenattr)...FSMODS>,
 hiddenaux=rest<hiddenaux>, hiddenaux.NE.0),
 // Invoking the structural rules corresponding to the actual segment. //
 =ENCOD<SEGTYP2,SEG>,
 // The new records to visit are appended to the list to visit (MODS) //
 // They are new records, that is, come from the copies performed above. //
 mods=rest<mods>...HEAD,
 attr='PRMODS'...'FSMODS',
 (attr=top<attrs>,
 mods=mods...@attr,
 attr=rest<attrs>, attrs.NE.0)),
 // Finally, the top of the encoding stack is restored, and the end of //
 // transfer is signalled through the global variable NLP-GERACAO. //
 encode<"PRESTORE", saveseg>, !NLP-GERACAO= & T)

// This procedure performs the copy of all pointer attributes of the SEG/ // segment. It treats HEAD first, and then all the others. It calls the// // lexical transfer. //

PCOPY (seg^PTR, ATTRS^LST,
 auxil/PTR, attr/PTR, resul/LST, lista/LST,

<HEAD.NE.0, =encode<"ORIGINAL",HEAD>,
 HEAD=COPIA(HEAD),
 <ESC,ESC(HEAD)=ESC>,
 =pchoice<HEAD>>,
 // Here begins the loop which performs the copy of all records pointed //
 // by the current record. Above, HEAD was treated in a special way, to //
 // avoid a double call to PCHOICE (who does the choice of the right //
 // Portuguese translation (lexical transfer), and the corresponding set// // of the ESC attribute).

(attr.NE.0, attr=top<attrs>,
 =<attr.NE.0, lista=@attr, resul=0,
 (lista.NE.0, auxil=top<lista>, lista=rest<lista>,
 =encode<"ORIGINAL", AUXIL>,
 =proplf<seg, COPIA(auxil), intern<SEGTYP. |."IND">>,
 =pchoice<COPIA(auxil)>,
 resul=resul...COPIA(auxil)),
 @attr=resul)

Report on a PLNLP-based translation system
EXAMPLES of the code of generation

// Encoding rule for the terminal node ADJ.  // Plural generation is activated if necessary.  
(19000) GADJ --> GENNDNODE("><GADJ,
          <GNUMB.EQ.'PLUR', PMAKPLUR<SEG>>)  

// Encoding rule for the non-terminal node NP.  // GNUMB is assigned the correct number for the sub-tree below NP.  
(19110) GNP --> GNULL("><GNP,
          <GNUMB='SING', _SING, GNUMB='PLUR'>)}

// The following rules concern plural generation.  
// corac,a o - corac,o es //
(19200) GPLURA 0 --> GNULL("><GPLURA 0,
          STR=SUBSTR<STR,1,WORDLEN-3>.|"O ES", _WORDLEN)

// agrada#vel - agrada#veis //
(19310) GPLURVEL --> GNULL("><GPLURVEL,
          STR=SUBSTR<STR,1,WORDLEN-3>.|"VEIS", _WORDLEN)

// homem - homens //
(19370) GPLURM --> GNULL("><GPLURM,
          STR=SUBSTR<STR,1,WORDLEN-1>.|"NS", _WORDLEN)

// The following rules concern regular verb tense conjugation.  //
// The 2nd person of the plural is omitted.  //
// GPRES - Presente do Indicativo  
// cantAR - cantO cantAS cantA cantAMOS cantAM //
(19400) GPRESAR --> GNULL("><GPRESAR,
          <SING,
          <<PERS1, STR=STR.|"O">|  
          <PERS2, STR=STR.|"AS">|  
          <STR=STR.|"A">>
          ,
          <<SING,
          <<PERS1, STR=STR.|"AMOS">|  
          <STR=STR.|"AM">>
          )

******************************************************************************

Some procedures of interest for generation

The core procedure for generation:

Report on a PLNLP-based translation system
// PGENERAT receives a segment - a node of the tree gotten from the //
// transfer phase - and "unfolds" it in its PRMODS, HEAD and PSMODS. //
// This procedure is called recursively, traversing the tree top-down //
// and left-to-right, until all terminal nodes are reached. //
// For each node, a rule is triggered according to its SEGTYPE //
// attribute, and the generation dictionary is accessed with the BASE //
// attribute (actually, the dictionary is read just once for each //
// word and its contents, stored in records, become available for //
// future access. //

PGENERAT (SEG*PTR,
    NODELIST/LST, TOPNODE/PTR,
    NODELIST=PRMODS..HEAD..PSMODS,
    (NODELIST.NE.0, 
    TOPNODE=TOP<NODELIST>,
    <SEGTYPE(TOPNODE).NOTIN.'CHAR',
    PORTUG(TOPNODE)=PDIegen1<TOPNODE>,
    GNUMB(TOPNODE)=GNUMB>,
    =ENCODE<INTERN"]"G"[.|.PNAME<SEGTYPE(TOPNODE)>>, TOPNODE>,
    <HEAD(TOPNODE), =PGENERAT<TOPNODE>>,
    NODELIST=REST<NODELIST>))

// This procedure receives a segment - typically, a terminal node of //
// the tree - and performs the plural of its STR attribute. //
// If an irregular plural form was gotten from generation dictionary, //
// it is used. Otherwise, possible plural generation rules are //
// invoked until an existing one is matched an triggered. The name of //
// the invoked rules is built concatenating "GPLUR" with the last 3, //
// 2 and 1 letters of the singular word (STR). The last rule //
// invoked - GPLUR - applies always if all the others fail. //
// In order to recognize if, for each invocation, a rule was actually //
// triggered, all of the plural generation rules have to delete the //
// WORDLEN attribute. //

PMAKPLUR (SEG*PTR,
    <PLURFORM(PORTUG),STR="".|.STRING<PLURFORM(PORTUG)>, <-NIL>,
    WORDLEN=STRLEN+STR>,
    <WORDLEN.GE.4,
    ENCODE<INTERN"]"GPLUR"[.|.UPCASE<SUBSTRG<STR,WORDLEN-2,3>>, SEG>>,
    <WORDLEN.GE.3,
    ENCODE<INTERN"]"GPLUR"[.|.UPCASE<SUBSTRG<STR,WORDLEN-1,2>>, SEG>>,
    <WORDLEN.GE.2,
    ENCODE<INTERN"]"GPLUR"[.|.UPCASE<SUBSTRG<STR,WORDLEN,1>>, SEG>>,
    <WORDLEN.GE.1,
    ENCODE<INTERN"]"GPLUR"], SEG>>)
ultimo(AJAVPP ANTES)
final(AJAVPP)

listen
escutar(VERB (PREPO to OBJECT))

Note: AJAVPP is a mixed category which comprehends adverbs, adjectives, and prepositions, originated from the adaptation of a dictionary which is based in a different linguistic approach, namely Junction Grammar Theory, and that was kindly put at our disposition by Stephen Richardson.

********************************************************************************

Examples from the generation dictionary:

qualquer
qualquer(ADJ (PLURFORM quaisquer))
chegar
chegar(VERB (PRPF1S cheguei))
corac,a o

********************************************************************************

It is important to notice, that no claim of linguistic correctness or completeness is being made. All examples and rules were presented with the only aim of showing the methods and the programming style.

Most rules, and their corresponding results, were chosen in an ad-hoc approach, to illustrate the different abilities the system has got. The only extensive coverage that our system nowadays performs is that of the translation of infinitive clauses, and the treatment of prepositions asked by verbs both in English and in Portuguese.
PLNLP: a Computer Scientist's Introduction

Diana Santos

March 1988

Introduction

This text is primarily intended for teaching PLNLP and its environment to the Mentor project’s branch at Lisbon and it is expected that it will be used at Madrid too.

It was written after a two weeks’ course, in Hawthorne, November 87, organized by George Heidorn, the author of the language, and attended by a member of each country participating in Mentor. In view of the scarce documentation on PLNLP up to now, it is hoped that the following pages may help to fill the gap, contributing to a better use and understanding of the language and its environment.

The text is organized in two main parts:

The description of PLNLP as a programming language, emphasizing its encoding and procedural abilities, in view of its later use in Mentor.

The presentation of the PLNLP environment, which is an interactive program built on top of LISP to use PLNLP, which comprehends predefined functions for displaying and debugging, and several switches that alter the behavior of the system.

Related to the intended use of PLNLP, several other subjects will be also briefly described:

Because of the main use of PLNLP, namely PEG (PLNLP English Grammar), another set of functions and switches is available, which in connection with the PLNLP environment will be called PEG shell. This new layer is useful mainly to change, display and debug PEG.

PEG herself is available as an impressively large PLNLP program, whose description is obviously outside the scope of these lines. Only the implementation details needed to proceed from her results will be mentioned here.

OD, (the Optimized Dictionary), although more general and independent of the PLNLP language, can be accessed from its environment, being another integral constituent of PEG shell. The description of OD and its relationship with PLNLP will not, however, be described here, as some documentation on it is being prepared at another site.

Although this text tries to give a general view of PLNLP as both programming language and environment, it is acknowledged that it was written with the translation application in mind. So, the section on decoding rules is scarcely developed, while several details concerning procedures and encoding deserve considerable attention.

The PLNLP programming language

Since most documentation available on PLNLP consists of introductions, these lines will try to situate it in the general framework of programming languages.

Since PLNLP was created before the new trends in computer science, without relationship with LISP symbolic processing, from one side, and the production systems’ paradigm, from the other, it is hard to classify it as belonging decidedly to a family. It contains, however, many features found later in the languages of the new generation.
PLNLP is a language that, as its name indicates, was conceived for and directed towards Natural Language Processing.

However, it encompasses capabilities that make it general-purpose, in the sense that PLNLP allows the computational power of a Von Neumann machine. You can do with PLNLP whatever you do in Pascal, for example, (but not what you do in LISP, which is not a language of the Von Neumann type). It is obvious, though, that PLNLP is not “another Pascal-like” language: it presents capabilities far beyond general-purpose to deal with natural language, and an environment and programming style very similar to the production system model.

Some distinct features of PLNLP are presented as to illustrate the statements above:

PLNLP provides one basic data structure, the RECORD, an entity with any number of pairs attribute-value, and which appears to be most suitable for storing the kind of information linguists use. There is no need to declare attributes, nor any restrictions concerning their number. Associated with a record may also be indicators, which are binary (or boolean) attributes, and whose presence means that the property they represent holds for the record at hand.

PLNLP programs are written as a set of RULES, which, being either of the type DECODING or ENCODING, assume a different algorithm in their application. Both these algorithms, of the kind bottom-up parallel and top-down serial respectively, were especially conceived to deal with Natural Language. (For the sake of generality, PLNLP procedures can also be written, bringing an imperative flavor to the language.) The rules’ formalism brings PLNLP close to a production system, together with the fact that it implies an “inference engine” belonging to the language. There is, however no such distinction as working and production memory, and the rules’ format was not designed to be general, but to be NLP suited.

Finally, its environment and applications prove, most definitely, that PLNLP was designed as a vehicle to Natural Language programs.

The description of PLNLP as a programming language will be divided in the following parts:

- The record data structure
- Decoding rules
- Encoding rules
- Procedures

An effort will be made to present PLNLP independently of PEG shell, namely by not enforcing any record names which in PEG do have some special meaning, nor giving examples taken from PEG.

Records

A record in PLNLP is the main data structure available. It can be defined as a set of attribute-value pairs. There are some special attributes in PLNLP, related to the way the language works, and others only of significance to the programmer.

A named record has an attribute NAME, which makes it unique, and allows for its reference as an entity.

A record without a name, on the contrary, has to be accessed through other mechanisms, being the most obvious the use of pointers. The large majority of records dealt with in PLNLP are segment records.

These are related to the actual processing of PLNLP programs, or, more accurately, to the meaning of the language when it deals with RULES. The segment record is the basic entity that PLNLP rules talk about. Syntactically, they differ from other records only in the fact that they have a SEGTYPE attribute. They were created to represent classes of objects (each in turn represented by a record), instead of unique entities.

Segment records are created during the application of PLNLP rules, and according to them. They can be described as dynamic, in that, contrarily to named records, their creation depends on the particular execution of one program. In fact, the same program with different inputs leads to the
creation of different segment records. Apart from that, after execution, some of the records created may be no longer accessible to further processing.

The names of the types of segment records, that is, the value of their SEGTYPE attribute, are arbitrarily chosen by the programmer, but should indicate a classification of some sort, to increase readability.

Any other records are also allowed, without a SEGTYPE or NAME attribute. This use of records is much rarer than the two formerly described, but it is important to know that a PLNL P record is a broader concept, describable only by a collection of attribute-value pairs. Examples later in this text will help you understand when and why use each particular kind.

**Named records**

You can define explicitly a named record in the RECORDS section of your PLNL P program, typing its name followed by round parentheses which include at least one PLNL P instruction, most frequently an attribute or indicator assignment.

Examples

GEORGE (CAT = 'NOUN', + PROPER, PORT = 'Jorge')

JORGE (+ MASC)

SET (nlp-ola = &T)

On the other hand, you define a named record implicitly any time you type a single-quoted string in a PLNL P program, provided it does not yet refer to a previously defined named record. This record will only have its attribute NAME non-null.

There is no need to test whether a name already exists or not. A record will automatically be created if it doesn’t.

You can think of named records defined implicitly as PLNL P strings, that is, an object whose only information contents is its name. They are, however, more than simple strings, because nothing prevents you from assigning other characteristics to them (as to any PLNL P record).

**Segment records**

As was stated before, they are the rules’ participants. PLNL P rules refer to types of segment records, and consequently apply to all segment records with that SEGTYPE (apart from having to obey any explicit conditions, too).

You define a type of segment record by simply using it in a rule, and you use it by simply making a reference to it. Any record name in a rule, apart from those of the already defined named records, will be considered a segment record designation. There are no limits for the type of segment records that can exist in your program.

Examples

.... -- > VP(at1 = 'ola', at2 = 2)  (VP is being defined as the name of the attribute SEGTYPE of this record, which means that the preceding rule could be said to mean the following:

.... -- > unnamed(SEGTYPE = 'VP', at1 = 'ola', at2 = 2)  ) You create new segment records simply by referring to its segment type in PLNL P rules, provided you give them at least an attribute. In the right-hand side of a rule, you can only refer to the segment records which match the rule’s left-hand side, since there is no way to distinguish between two segment records of the same type. The programmer does not need to keep track of any specific records which will be created during execution, he/she only has to worry about the rules to apply to each particular class (type) of segments.

In your rules, you can create segment records from scratch, copy them from the left-hand side of the rule or simply using the ones from the left.

Examples :

VERB -- > VP(ui = 'ipoi')  (Creating VP from scratch)
VP --> VP ui = 'likju'
VP --> VP(%%VP, ui = 'lhhjkh') (Two ways of copying a former VP)
VP --> VP(%%VP, gfd = 'ill') (Using the record of the left-hand side of the rule).
VP VP VP --> VP(%%VP#2, jij = 1) (When there are several, you select one by its number).

The compiler distinguishes between records and any other identifiers this way: any name that appears in a PLNL rule (outside the brackets) stands for a segment record type, except if it had already been defined as a named record. Inside the brackets, the same convention holds when the name is preceded by the signs "%%" or "%%". When a name is enclosed in single quotes, it always represents a named record.

Others

To build any other kind of record, you enclose its definition in angle brackets, and just give values to attributes, as in any PLNL record. You cannot refer to them by name or automatically in a rule, so you must know where it is stored to access it, normally having a pointer to it.

Examples

VP --> VP ola = <ijk = 'oljh', yy = 'eed', father = VP>, hjj = 'oiu')

To close the record overview, it should be stated that there are some special names in PLNL, which should not be used as trivial segment names: SEG, which stands for the current record, MEM, a record which is the "memory" of the system, and NIL, which exists for implementation under LISP, and means FALSE for PLNL procedures.

Record structure

Attributes

A record exists as long as it has some non-zero value of at least one attribute. There are special attributes in PLNL in the sense that they determine the treatment given to the record (as SECTYPE for segment and NAME for named) or that convey a special kind of general information, like SUP (the superset of that particular record) or INDIC, which contains the indicators set in the record (or better, INDIC is an attribute that holds all information about indicators).

These "special" attribute names are then inherent to PLNL, because the language has mechanisms to deal with them: NAME and SECTYPE were already described; SUP has also something special about it: it is the "default" attribute, you only need to specify the value, and not the word SUP, in a record definition. Or, stated differently, when the compiler finds a value where normally would stand a pair attribute value, it assigns it to SUP.

Indicators

INDIC, in turn, is used to store efficiently the values of the indicators of a particular record. As always, the trade off between efficiency and ease of programming shows here in the fact that indicators must be declared, and cannot be more than a fixed number (currently 128). Since they are conceptually not more than attributes who may take either zero (do not hold) or non-zero (hold) as their value, they can always be implemented by the programmer as normal attributes. There are, however, some characteristics of PLNL indicators, apart from compactness, which couldn't be used, and they are both powerful and practical... For instance, you can assign a name to a group of indicators, somehow related, and deal with that group as a single entity, setting or deleting it, copying it to another place, etc.

Examples

VERBIND 10-20, TENSE 10-13, MODE 14-15, PRESENT 10, PAST 11, PASSIVE 20
SEX 5-6, FEM 5, MAS 6
Use of attributes and indicators

You set and reset indicators by affecting their name by the prefix + or -. You test them by simply stating their name or precede it of if you want to test its non-existence.

Examples

\( \text{VP}(-\text{fyr}) \rightarrow \text{GH}(-\text{MAS} , + \text{poiuy}) \)

Attributes, on the other hand, are referred also by name, which in turn does not need to be declared, but you must assign them the values you desire and normally test the existence of some value explicitly (except if you are only interested in its existence or not in the record...)

Examples

\( \text{TRG}(-\text{ttt,nuu,eq,ppp},-\text{llll}) \rightarrow \text{OKI}(-\text{nuu, llll} = \% \text{TRG}, \text{ttt} = 9) \)

\( \text{YJ}(-\text{ola,eq,oiuy}) \rightarrow \text{YJ}(-\text{ola} = \text{"oytc"}) \)

The assignment operator in PLNLP is the "=" in infix position, as has been being used above.

Addressing

To get the value of an attribute or indicator, you just write its name (in a place, of course, where PLNLP is waiting for an attribute to appear, namely, inside records or procedures). For some attributes, and for implementation reasons, you can also write @ followed by an integer and get the same result. This means that, somewhere, a correspondence is established between attributes and numbers. This should not, however, worry or interest the programmer.

Examples

\( \text{OLA} \quad @12 \)

You may also not know the name of the attribute you want to access, but only that it is the value of some other attribute. In that case, you reference it using indirect addressing, which is done in PLNLP by an @ followed by the place where the name of the attribute can be found.

Examples

\( @\text{OLA} \) means "the contents of the record pointed by the attribute OLA".

\( \text{MASC}(@\text{OLA}) \) accesses the attribute (or indicator) MASC of the record pointed by the attribute OLA of the current record.

Finally, you can also specify a search in an attribute chain, irrespectively of how many records will have to be visited. You use this addressing simply by preceding the attribute name by $. If you use $ alone, PLNLP follows the SUP chain, since SUP is the default attribute. This feature was specially conceived to deal with inheritance.

Examples

\( \text{MASC}($\text{OLA}$) \quad \text{MASC}($\text{SEG}$) \)

are the values of the attribute (or indicator) MASC, visiting the record pointed by the OLA attribute (or SUP), till some value is found, or then visiting the record in turn pointed by the new record’s OLA attribute, and so on.

In any case, if you don’t specify it otherwise, PLNLP assumes that the attribute (or indicator) referred belongs to the current record (SEG). To access the values of other records’ attributes, you write the name of the record, enclosed in round brackets, after the attribute name.

Examples

\( \text{OLA}(\text{VP}) \) means the attribute OLA of the record VP (which PLNLP would already know that it was a record’s name):

Finally, you can nest references to any extent, being allowed, for example, a sequence like the following:

\( \text{ikj}(@\text{crfc}(\text{uijkn}(@\text{crfct}(\text{nlkhg})))) \)
Values

At last, and so that you can use attributes, you must know which kind of values can be assigned to them. Apart from other records, as has already been exemplified, PLNLP can handle character strings (enclosed in double quotes), integers (as expected), real numbers (in an awkward syntax, preceded by ? and enclosed in double quotes) and LISP objects (preceded by & and obeying LISP syntax).

Examples

"This string will unfortunately be converted to capitals"
89
?"45.89"

&(this is a LISP list)

In the PLNLP literature, attributes which are assigned a record are said to be of type POINTER, since this assignment doesn’t copy the whole structure but only points to it. An empty pointer has the value zero.

PLNLP is a loosely-typed language, in that it does, like LISP, dynamic type checking, or in other words the type is a value’s (not a variable’s) characteristic. This means that there is no need of type declarations, and the same symbol may have different types during execution.

Lists

Another data structure available in PLNLP is the list. A PLNLP list is a list of records. A record in itself is a list of one element. The null list is the null record: ZERO (0).

To access a list, there are the functions TOP and REST, that return the first record in the list, and the list without its first record. To build a list, you have the constructor ’.’, that has as both arguments records or lists (and works so as CONS or APPEND in LISP), and returns a list.

Examples

onelist = 'ola'...pappp
iii = top < onelist >

This data structure is intrinsic of PLNLP, and should always be used instead of LISP lists, that are only accepted to communicate easily with LISP programs.

Decoding rules

A PLNLP program may have INDICATORS and ATTRIBUTES sections, although most of the time you will just be using the defaults of PEG shell, and may or not have a RECORDS section (where named records are defined). But the rules are the core of a PLNLP program: decoding rules, and encoding ones, although you can still enlarge the power of PLNLP by writing procedures to be called from the rules.

A rule in PLNLP has the following format:

(rule number) left-hand side -- > right-hand side

The meaning (that is, the processing they suffer) differs, however, substantially from an encoding to a decoding rule.

The numbers preceding the rules are the form the system uses to identify them. There should not be two rules with the same number (its name). The use of numbers instead of words, by the way, although less readable, has the advantage of allowing some trace capabilities of PLNLP: by specifying a numeric interval, the application of all rules with numbers within it will be displayed.
The left-hand side

Decoding rules have a left-hand side which consists of one or more record identification (segment type name of segment records) followed or not by round parentheses, inside of which various conditions can be expressed, separated by commas. The left-hand side expresses the conditions that must be met so that the rule can be applied. A record identification means "if there is a record in the system which can be identified like this" and the conditions speak by themselves. If there is more than one record in this side of the rule, it is meant "followed (contiguously in the input stream) by..."

Examples:

TREF CVFRE --> ... (If there are two contiguous records whose identification is TREF for the first and CVFRE for the second, then ...)

TREF(hjui,'tgrfd',vgh='ujds') --> ... (If there is a record TREF that has the indicator HJUI, the value of its SUP attribute is TGRFD, and the value of its VGH attribute UJDS, then ...)

The right-hand side

The right-hand side of a decoding rule consists of a single record identification, followed by round parenthesis where actions, separated by commas, are specified.

Although the syntax of both sides of decoding rules is similar, it is important to emphasize that the semantics is not. The right-hand side states the actions that will be performed when the rule is applied. So, the record identification means that a record of that form (a segment record of that type) will be created, with the attributes set as indicated explicitly inside the right-hand side parentheses.

You can nevertheless also perform some conditional actions on the right-hand side of a rule, provided you explicitly indicate it through the use of angle brackets.

Examples

... --> THFRE( + poi,jh, 'loijh', ghgfh = 45)

(Create a segment record of type THFRE, setting its POIJH indicator, its SUP to LOIJH and its attribute GHGFH to 45. If in the left-hand side there was any record of the same THFRE type, all its characteristics (attributes and indicators) would have been copied to the newly created THFRE, as was pointed out before).

... --> HGBF(l = 'OPOPOP', < jikuh, GT.78, gh = poiuy >, -i0jku)

(Create a HGBF-type segment record, set its attribute IL to OPOPOP; if its attribute JIKUH is greater than 78, set its GH attribute to POIUY; clear the indicator or attribute IOJIKU).

This device is used to reduce the overall number of rules you have to write, making the code more condensed. You can also recur to more complex flow of control devices, like loops, for instance. You are, however, warned about the danger of transforming a rule which should be simple and readable into a procedure... in which case you are advised to write it in the adequate section.

A PLNLP loop is stated in round parenthesis, and PLNLP interprets what is inside as in the interior of a left-hand side record. The loop is exited when a condition is found that does not hold.

Examples

(i.GE.6, pl = top < fitt > ...'ola', i = i + 8, gtf.EQ.'ppp')

Example

As an example of a set of decoding rules, let us look at a very simple grammar of English:

DECODING

(2010) NOUN --> NP(%NOUN,HEAD = NOUN,SEGTP2 = 'NP')

(2020) VERB --> VP(%VERB,HEAD = VERB,SEGTP2 = 'VP')

(2030) ADJ NP --> NP(PRMODES = ADJ...PRMODES)
This grammar is so simple that it doesn't even have conditions in the left-hand sides of the rules, but it is useful for example purposes.

Assume for the moment that there were already (as in fact happens in PEG shell) some records in the system whose type could be NOUN ADJ, PREP, VERB, and PUNC. These records were created by the dictionary access that was triggered after you input a sentence to the system. With this grammar, you are trying to make some (syntactic) sense of the words that were input.

Before following closely the output of the grammar for a sample sentence, some explanations are due:

The slash (/) stands for context, in this case left context, which means that the record to its left will only be considered in the rule as a new condition to be met, but it cannot be used in any other way.

Apart from setting attributes you can do any actions in the right-hand side brackets of a rule, and specifically you can apply functions or call procedures. This last ability is done by writing the procedure's name followed by angle brackets enclosing its parameters, as in NVALUES < SENTLIST(MEM) > and will be discussed in greater length in the section concerning procedures.

PLNLP functions can have the same syntax as described above for the procedures, or their name can appear between two dots, between the arguments. In this sample program you have ADJ...PRMODS, which stands for calling the "\" function (to build a list) with the two arguments ADJ and PRMODS.

To make all symbols clear, the records SNTBEG and SNTEND must be mentioned: They are also created by PEG shell during the words' input to the system, and stand for "beginning" and "end" of the input stream.

The decoding algorithm

Instead of giving a detailed description of the decoding algorithm, which can be read in George Heidorn's thesis, a simple example will be followed, to give an idea of the processing and results of a decoding section of a PLNLP program.

Imagine the following sentence being entered: "man loves." Of no concern to us now how it is done; the following segment records will be created by the PLNLP dictionary access functions:

SNTBEG (FW = 0, LW = 0)
NOUN (STR = "man", FW = 1, LW = 1, BASE = 'MAN')
VERB (STR = "loves", FW = 2, LW = 2, BASE = 'LOVE, SING3')
PUNC (STR = ".", FW = 3, LW = 3)
SNTEND (FW = 4, LW = 4)
STR is the attribute that holds the STRing, BASE stores the lemma of the word coming from the dictionary, FW and LW indicate the position of the string they represent in the input stream (FirstWord and LastWord). Other attributes, as for instance INDIC, would also receive values.

These records would feed our rules, in the following way, (the numbers are only for display improvement):

Rules which fire ---------------- segment records created

2010  with NOUN-1       NP-1
2020  with VERB-1       VP-1
2010  with NOUN-2       NP-2
2070  with VP-1 and NP-2 VP-2
2020  with VERB-2       VP-3
2060  with NP-1 and VP-3 VP-4
2080  with VP-2 and PUNC SENT-1
2080  with VP-4 and PUNC SENT-2

Although this example is considerably simple, it illustrates some aspects of decoding. Since this section is both the best documented and the one which will be of least use for us in Machine Translation, we chose to describe it with this brevity.

The encoding section

Encoding rules

Encoding rules in PLNLP have a syntax similar to the decoding ones, but, as was already pointed out, have quite a different semantics associated.

The left-hand side

The left-hand side of an encoding rule is built of only one record, followed by the conditions that have to be met, enclosed in round parenthesis. Actions can nevertheless be done on this part of the rule, provided you explicitly impose it (by preceding calls to procedures by an = sign, for instance). This feature, at first puzzling, becomes at once clear knowing that the encoding algorithm is top-down and serial. That means simply that the rules’ format only “conceals” a more traditional approach to programming, which is that of sequentially applying procedures and performing tests.

Examples

\[ \text{FGT(10, hgbn.BALAAL.tdes, = print < "I'm here"> ) --> ...} \]
\[ \text{YTRED('OYIYU', 1npoi) --> ...} \]

The right-hand side

On the other hand, the right-hand side of the encoding rules can have any number (greater than zero) of record identifications, followed by the usual sequences of actions in parentheses. Everything that was mentioned about the right-hand side of decoding rules holds for encoding ones. There is only an additional subtlety: in the same rule, any right-hand side record can only refer to the left-hand side record of that rule, and not to the preceding right-hand side records. This is somehow different from the left-hand side of a decoding rule, where the preceding records (which in this case mean REALLY preceding from a linear order point of view) can always be referenced.

As another distinctive feature of encoding rules, you are allowed (and even encouraged) to use the “USE IT” (%%) device, when there is no point of creating new records when you just want one output and not multiple ones...
Examples:

... --> TGRF(ghj = 786, + uiyr, rf = yu...ty) VCDE (ty = jhgfd)
... --> YHGFRE(%%DCFRE, yg = 65) UIII (+lokjgh)

Since encoding rules are designed primarily for output purposes, there is a special segment record type in encoding: OUTPUT. An OUTPUT record, when found by the encoding algorithm, performs some output, according to the values of its attributes. The interesting attributes are @11 (number of newlines), @12 (column position of the output, or tabulation), @13 (canned output, i.e., any character string), and @14 and @15 for printing integer and decimal numbers. They are printed in this order (11 < 12 < 13...), e.g., newlines are executed before the text is output.

Examples

... --> OUTPUT(@13 = 'Error message :', @11 = 2) UI (+ error)

Example

A small example of an encoding section of a PLNLP program follows, to illustrate the description above.

ENCODING

(8000) DSPLYFR --> NEWLIN(BLANKS = 1) FR(%TOP < SENTLIST(MEM) >)
(8010) FR(¬HEAD, ¬STR) --> NULL
(8020) FR(¬HEAD) --> DISPLAY(SUP = STR(FR))
(8030) FR -->
PR(HEAD = TOP < PRMODS(FR)>, PSMODS = REST < PRMODS(FR)>)
HD(%%HEAD(FR))
PS(HEAD = TOP < PSMODS(FR)>, PSMODS = REST < PSMODS(FR)>)
(8040) PR --> FR(%%FR)
(8050) HD --> FR(%%HD)
(8060) PS --> FR(%%PS)

The explanation of this sample program is as follows: it navigates in a PLNLP tree till arriving at the leaves, that is, the records which have no HEAD attribute, and simply outputs the string they have associated (stored in their STR attribute, and assigned to SUP for this reason). The other records have a HEAD attribute and possibly PRMODS and PSMODS attributes. In that case, the program recursively goes down through the three records pointed by these attributes.

The encoding algorithm

This algorithm, as opposed to the decoding one, is serial instead of parallel. It applies sequentially the rule which was fired by the record of the top of the stack, and this record will be discarded, not firing any more rules afterwards.

On the other hand, it is top-down instead of bottom-up. This means that it eventually produces, from one record, a big number of them. An encoding rule decomposes some structure into smaller parts, instead of building a structure from its constituents.

The encoding algorithm is triggered by explicitly calling it using the procedure ENCODE and giving it a pair of data to begin with, or by the creation of a record of segment type ENCODING in a decoding rule. This record is used to switch control between the DECODING and ENCODING sections of a PLNLP program.

The algorithm used in the application of the encoding rules is based in a stack, that holds records and their respective segment types:

It begins by taking the record of the top of the stack out of it.
In case the record's segment type is OUTPUT, the value of its attributes 11 to 15 is put in the output stream.

Otherwise, the algorithm applies the first rule it can, producing the records of the rules' right-hand side, and performing whatever actions they stipulate.

These new records are then pushed onto the top of the stack, in such a form that the leftmost record will become the top of the stack. Obviously, together with the record, its segment type is stored in the stack as well.

In case the record that stands in the top of the stack does not have any rules that apply, the value of its SUP attribute is introduced in the output stream.

If the record has no SUP, then the name of its segment type is output.

This procedure is repeated till there are no more records in the stack.

The major goal of encoding rules is to produce some output, so that any record created will give origin to an action eventually leading to it. The choice of the attribute SUP and then of the name of the segment type is coherent with PLNLP features: the SUP is meant to store hierarchical considerations, such as categorization or other sort of classification, so it is a natural alternative in case a record does not lead to specific output. On the other hand, real words of the language may be (and in the beginning actually were) represented by named records whose name was identical to the word. So the desired output was the name of the segment type. This is the reason why the name of the record is still printed by PLNLP as a last resource, even though nowadays, instead of representing the words by named records, OD is accessed.

Although this algorithm seems perfectly straightforward, there are some points which should be made clear in order to prevent the user from receiving unexpected puzzling results. These issues are mainly related to the actual implementation of the algorithm, but their ignorance may cause several headaches to the naive encoding rules' writer.

The order by which rules are accessed

The first point concerns the notion of FIRST rule to be found (see the algorithm description). Rules are simply stored by the order they become known by the system. This implies, first, that the rules' numbers have nothing to do with it, and second and most important, that, if there are some rules in the PLNLP shell about a particular segment type, and you write others, you can be sure that the former will always precede yours... This can be changed, however, by two mechanisms provided by the PLNLP shell :

The first allows you to actually replace the code of a rule, while maintaining its relative order. This is accomplished by simply preceding the number of the rule you are rewriting by a minus sign.

You should be forewarned, however, that rules should only replace others with the same segment type in the left! The reason for this is that the system has already memorized, during input of the first version of that rule, which segment type did it concern, and it won't change that information even if you decided to write a different rule about a different kind of records... so a completely unpredictable behavior can result in case you use the replacing of a specific rule for other tasks than actually changing it.

The second device allows you to change the treatment of a whole segment type, and not simply performing a minor modification inside a rule. For this task, there is a switch in PLNLP environment ('"NLN-REPLACE-RULES"'), that, if set, tells the system to forget any rules about the segment types appearing in the left of the code that will be input from then on. This means all rules input so far will be replaced, if they have the same SEGTYPE as the new ones. You should be sure that that is what you want, so that you do not waste powerful aids of PLNLP shell.

Concerning the relationship between these two devices, you should be aware that the minus sign has precedence above the setting of 'NLN-REPLACE-RULES' to T, which means that if you use both, you won't be clearing all rules with the record type in the left, but only changing the one with the same number...

Finally, in case you neither want to forget all the rules nor just change a single one, as in the case of having decided to insert another one, some mixed strategy will have to be performed to achieve
that goal. This concerns, obviously, only the case when there are already (in PEG shell, or in a previous system of yours you are enhancing) rules that concern the segment types you are using. Some possible strategies follow:

Replace the rules that follow the rule you want to insert so that they test for a condition that should only happen after your new rules are triggered.

Just insert your rules in the numbers of the former rules, and have those copied to another file, with different rule numbers. That would enforce that they would be considered after your new rules.

A subtle distinction between implicit or explicit settings of SECTYPE

Another characteristic of the PLNLP language that, related to the encoding algorithm, may cause some problems, is the fact that you have two ways (which can be used contradictorily) of setting a record's SECTYPE in a rule: by the rule itself, or by actually assigning a value to that attribute...

Consider the following example:

\[ \text{VP} \rightarrow \text{VP}(%\%\text{VP}, \text{SECTYPE} = \text{LTA}) \]

The new segment record which will be created by this rule will have a SECTYPE of LTA, as imposed by the programmer, but will continue its way through rules as a VP... (that is triggering the first rule which has at its left-hand side a VP, and not a LTA segment type name...)

This has nothing to do with %\%VP or %VP specification, the new record could begin without any inheritance of attributes... This behavior has solely to do with the fact that the rule states "VP turns into VP...", and it is with that information that the encoding algorithm will proceed, irrespectively of whatever changes the record will suffer afterwards.

In other words, the user has no access to the encoding stack, which is fulfilled by the rules skeleton before any actions take place during rule application.

It is important to emphasize, though, that this is by no means a PLNLP restriction, since the desired behavior of the preceding example would be fairly simply achieved by rewriting it:

\[ \text{VP} \rightarrow \text{LTA}(%\%\text{VP}) \]

This remark is just here to explain how encoding rules are processed by the compiler, and avoid several erroneous programs to be written.

Precedence of actions

Another characteristic of the processing of the encoding section is worth while being mentioned: when a specific rule is being applied, the actions in the parentheses of the right hand side records are at once performed, while only in the next encoding algorithm's loop will the results of the creation of those records show. In other words, actions inside records precede always the described normal encoding algorithm's sequence.

Not only SECTYPE can index encoding...

Till now you have been kept ignoring one of the most powerful capabilities of PLNLP encoding: that, while by default, associated with a segment, it is its SECTYPE which is processed as the index, you can index it by any other feature!

The encoding stack receives a name and a record, and, from this pair, the algorithm will look for any rules with that name on the left, applicable to the record. So, when you explicitly call the encoding algorithm through the procedure ENCODE, you can specify SECTYPE, any other attribute, or simply a name (in single quotes)...

Examples

encode < 'ORIGINAL', seg >
encode < SECTYP2, seg >

All characteristics focused previously hold for any encoding index, and not only for SECTYPE, of course.


**Procedures**

The **PROCEDURES** section

Another section of a PLNLP program is the procedures' section, where actual "how" information is encoded (that is, information of how actually perform some action).

Procedures in PLNLP are much the same what they are in other languages. They allow recursion, and use call-by-value discipline in parameter binding. They were designed so that their body were as similar to a record as possible, i.e., they only differ from a right-hand side specification of a record in any rule because they allow some other constructs that would have no meaning there.

So, to begin with, the reader will be reminded about what is legal in a right-hand side record: primarily, actions. Setting or resetting attributes or indicators, (of the record itself or of other records); altering global variables; test conditions to do some actions (using angular brackets); call procedures; perform cycles (using round brackets).

All this can be done in the body of a procedure. The only difference up to now concerns the "modification of the record itself or of others". The records that a procedure can modify are those who are known in its interior, that is: those who were passed by parameters when it was called, or those explicitly accessed or created inside the procedure. PLNLP provides you with as much similarity of treatment as it can: when, in a record specification, the name of the record whose attribute is being used does not appear explicitly, it defaults to SEG, the current record. The same mechanism is available in a procedure, that is, if you do not specify which record you are talking about, the compiler will assume SEG. It is of the user's responsibility to provide a sensible meaning to SEG in that procedure (normally by using this name as a formal parameter, or at least as a local variable).

**Extensions of procedures**

The big differences (or extensions) of a procedure compared to a record are the following:

A procedure has formal parameters, which, as in all languages, represent its communication with the world. Since the parameters' binding is by value, they are used only for introducing information inside the procedure. Also, it can have local variables.

A procedure always returns a value (in what it should more conveniently be called a function). When, while calling a procedure, we are not interested in the value it returns, we should precede it by an = sign, as was already stated. Many procedures are in fact written by their side-effects, that is, not because of the values they return, but because of the actions they perform. These actions can be altering of global variables, or, more commonly, changing or creating records. As should be emphasized, the fact that PLNLP employs call-by-value does not prevent one procedure to alter a record passed to it by means of a pointer. The pointer is not changed, but the contents of the record it points to can be.

It allows the specification of labels, and the "goto" construct, to be used as another control structure. A label is syntactically defined as a valid PLNLP identifier ending with a colon. To state a "goto" to one label, the syntax is "+" followed by the name of the label (without the colon).

It allows explicit return from the procedure, expressed by "<" " followed by the return value. In case no return directive is found, the procedure exits with a value of NIL. Although this fact should be of no interest to the programmer, who should use "=" when not interested in the value of a function, and always specify the return value, otherwise, it is worth while remarking that, contrarily to the PLNLP way, in procedures it is NIL the false value (as in LISP) and not 0, as usual. That means that, for instance, one condition form will be exited when one constituent evaluates to NIL, and that the form "/" followed by a condition will be translated into LISP into (NOT ...) instead of (ZEROP ...) as usual. Although this may seem strange, the reason is to give a compatible form to all functions, PLNLP or LISP, that can be called from a PLNLP program.

**Declaring procedures**

The declaration of a procedure consists of its name, followed (at least with one space in between) by round parenthesis, similar to a right-hand side record specification. The parameters of the pro-
procedure follow, described by their name, a "*", and their type, without any spacing, separated by commas. The type is of no concern for the moment, because the language is compiled into LISP, weakly typed and with dynamic type checking. But for reasons of good programming practice, there should be specified PTR (pointer), LST (list), INT (integer) or CHR (character string), instead of ANY, since it increases the readability of the code.

Following the declaration of the formal parameters, local variables are declared. The syntax is the same as of the former, except for using a slash "/" instead of the star "*". You can also specify initial values for them, just adding another "/" followed by the value.

Example

PPPPPPPP (yhy*INT, thr*LST,
aux*LST, ola/INT/3,

<yhy.GT.3,yhy = 3 >,
aux = TOP < thr >,
(yhy, = PRINT < yhy >,
= PRINT < dfdfde(LALALA < aux >) >,
aux = cont(TOP < aux >...REST(aux),
yhy = yhy-1),
<y -aux>

As usual, local variables and formal parameters are only known inside the procedure, and will shadow any other objects identified by the same name outside. So it is strongly unadvised to give them the names of attributes, indicators, or special PLNL names, (except for SEG if you want to have available the behavior described before).

After declaring variables, the actual actions appear, with the same syntax as inside records in the right-hand side of a rule: separated by commas, and recurring to angular brackets when a condition is to be tested.

There is no need to say that a "goto" to a label is only legal if that label is at the same level, that is, you cannot exit a procedure by "gotoing" out of it.

Calling a procedure

To call a procedure, it suffices to write its name followed by its actual arguments separated by commas and enclosed in angular brackets. Since a PLNL procedure always returns a value, when you do not want it to, you should precede its call by a =, and the value is discarded.

There is also another way of calling a two-argument procedure with an infix syntax, as opposed to the more usual prefix syntax that has been used in the examples till now. You call the function by surrounding it by its two arguments, separated from the procedure's name by two dots, one in each side.

Examples

first.O.L.A.second

aaa.HG GU.bbb.HG GU.ccc  results in the second call to HG GU being performed first, with arguments bbb and ccc. Parentheses can be used to force other orders of execution.

Warning: Although it is possible, in the present implementation of PLNL under LISP, to call LISP functions directly, that is strongly not recommended, because of lack of modularity and of rising of conflicts. The PLNL programmer should only write in PLNL, so that the program produces the same results in whatever language it gets compiled, and free her/himself from LISP.

Moreover, you should also be aware, that the fact that you can use this feature has the consequent drawback of the compiler not checking a call to a procedure, concerning the existence of a declaration, and/or compatibility in number of arguments between declaration and call. This means that all errors related to procedure calls will only get known at execution time.

PLNL: a Computer Scientist's Introduction
Recursion

You can use recursion with all its power, that is, nothing prevents a procedure from calling itself again, but be careful to specify whether the value returned is wanted or not... A typical mistake is some piece of code like the following:

PROCEDURE (x*ANY, y*ANY,
...
< ... ,
 procedure <a,b>,
tatata... >)

where PROCEDURE is a true procedure in the sense that what matters are the actions it performs, and not the value it returns. Since PROCEDURE returns NIL, then, the action TATATA will never be performed, contrary to what was intended. The correct program would use the "=" before the call to PROCEDURE.

The ROUTINES section

Related to procedures, there is another section of PLNLP programs called ROUTINES, which is used to declare names for the compiler to recognize procedures which have a PLNLP name different from its compiled name (the LISP name). So you can refer to a procedure in PLNLP by one identifier, but use a different one for LISP so that conflicts do not arise with LISP's own functions, or even use different names for the same procedure in different files, but ensure it will be recognized as the same by the compiler...

The syntax of this section is very simple: first appears the PLNLP name of the procedure, that is, the name you will be referring to in the following code in the file, followed by a space, and then, the name you want it to be translated (into LISP or whatever object language) surrounded by double quotes when the new name does not conform to PLNLP identifiers' syntax.

Examples

ola2 "nlp-ola-2", ola nlp-ola,
gtfre "completely-different"

PLNLP intrinsic functions

The functions a language provides are related to the data types it recognizes.

For integers, PLNLP accepts + and - to perform the usual addition and subtraction.

For conditions, PLNLP uses the comma ",” for conjunction and the vertical bar ”|” for disjunction. Sets of conditions related through AND or OR must in turn be enclosed in angular brackets. The nesting of ANDs and ORs is unrestricted, but OR-OR nesting is not accepted, and AND-AND nesting unnecessary.

Examples

< <A|<B,C> >,D>

(holds if D is true (not zero) and either A is true or B and C are different from zero).

The evaluation of conditional expressions is from left to right, being another form of flow of control specification. Actions are accepted inside them, having a logical value of TRUE.

The unary functions +, - and ¬ were already described, and are related to attributes' and indicators' manipulation.

The call to all other functions conform to PLNLP infix syntax, except for TOP and REST which only expect one argument.
For building lists, there is ".".

For testing, there are EQ, NE, GT, GE, LT, and LE, with the obvious meanings.

To concatenate strings, ";" is used.

Finally, and while the following functions are not basic since they could be written with the help of the ones described above, they are available and very helpful to manage lists: ISIN (testing for membership) and NOTIN (the opposite).

Examples

< verb Neil 'rule',

number.ISIN 'FPS'... 'FPP',

termin = verbbase.

Relationship between the several sections of a PLNLP program

Although we can classify (and actually have to write) a PLNLP program divided in three main sections: encoding, decoding and procedures, they can intermix and communicate through several ways.

First of all, the encoding and decoding algorithms can be invoked as single procedures, namely ENCODE and DECODE. This means that, from wherever a PLNLP procedure can be called, the two algorithms that deal with rules can be triggered.

On the other hand, procedures can be called in both sides of PLNLP rules.

Direct communication between decoding and encoding is achieved through the creation of a special record of type ENCODING. PLNLP switches to the encoding algorithm upon encountering a record with that SEGTYPE. Decoding rules can, however, instead of following the more formal approach of ending their job and creating a special record for encoding to go on, call the encoding section directly using the function ENCODE. Control will return to decoding rules normally when the encoding stack will turn empty.

There is no other connection between encoding and decoding sections in a PLNLP program. Rule numbers, left-hand sides of rules can be the same, that they will not conflict.

There is a closer relationship between procedures and encoding rules than it may appear at first sight, because the encoding algorithm can in fact also be seen as the sequence of application of the "procedures" represented by the records in the rules.

To recall what has been said before, calling ENCODE as a procedure simply requires two arguments: the first the segment type, the second the actual record. As was already stated, a specially interesting possibility of PLNLP is that nothing enforces you to put the real SEGTYPE of the record as the first argument of the call to ENCODE. You can use this fact to start a different group of encoding rules, indexed by whatever attribute may be important, and even by any other kind of identifier.

Several embedded calls to encoding

The free intermixing of these two kinds of programming styles (rules with whose order of application the programmer has not to bother, and procedures, where she/he must define all steps) is not without problems, however. The first consequence may be, for instance, that the system may forget data you were expecting it to process afterwards...

In particular, each call to the encoding process alters the stack (and about that at first sight there is no problem, since eventually everything will be removed from it) and the TOP OF THE STACK (by this what is meant is the segment of the left-hand side of the last rule that was applied, that is, the "current left record" in the algorithm, together with its SEGTYPE or indexing feature).

This is very important, because the encoding algorithm, when resumed, will start with that record and not with the one that was there before a new encoding section was triggered. If the new record
is essentially the same as the late, maybe no effects will arise, but this is obviously to trust luck and no good programming practice at all if you do it in ignorance.

The suggested procedure about nested calls to encoding (using or not procedures as the means to achieve these nested calls) is the following: keep in some attribute of the new record (with which encoding will eventually begin) the old record (the "current left record"). As the last action of the embedded encoding call, create the old record again. Like this you guarantee that, when returning to the outer encoding process, the situation, as involves the following records of the rule, is the same as before.

The PLNLP environment

In this chapter two different things will be described: how to use PLNLP, and the system you get when you install PLNLP at your site.

How to use PLNLP

The first thing to say about PLNLP programs is that they are written in normal CMS files, with filetype RULES or PLNLP. Their syntax is not free formatted, however, so you must follow some directives to write them.

A PLNLP file must begin with the identifier "DECODE:" in the first line and column, and end with two lines, also beginning in column one, with the words "EOF" in the first, and "END-OF-FILE:" in the second and last.

A section name should begin in the first column of a line where it stands alone. In a file there can be any number of sections of the same type. (Remember that the possible section names are ENCODING, DECODING, PROCEDURES, INDICATORS, ATTRIBUTES, RECORDS and ROUTINES).

The order of the sections is free, except for ROUTINES which should stand before the places where the translation of the identifiers it concerns occurs, and for RECORDS, which in case they set some switches or global variables important for the compilation process, (as NLP-REPLACE-RULES, for instance) should be placed, accordingly, before that process takes place. Also, named records should be defined (in a RECORDS section) prior to their reference in rules or procedures, when they own more attributes than their name.

Wherever a line ends in a slash "/", it will be ignored and consequently dismissed by the compiler. The slash is therefore the comment character, provided it is the last character in the line.

Everything else should be written in columns 11 through 79 of the file (except for rule numbers).

The rule numbers should be enclosed in round parenthesis and begin in column 1. Because in most cases 69 characters are not enough to state a rule, rules can be continued in the next lines, starting after column 12.

The segment types in the rules should have no space between their name and their body, otherwise the compiler interprets the name as an empty record.

In the procedures section, the name of the procedure must begin in column 11. The other lines must conform to the continuation line syntax, that is, they must begin not before the 13th. If a symbol appears in a procedure whose name is not declared in it, and it does not begin with an exclamation mark "!" and is not a reserved word like MEM, it will be interpreted as an attribute name, and of the record SEG if not followed by a name in parenthesis.

As opposed to the rules' sections, in the records' there must be a space between the name and its description.

Some other general features should be mentioned at this point: blanks and/or newlines inside records (in RECORDS, DECODING, ENCODING and PROCEDURES sections) are only allowed following left parenthesis and left angular brackets, following commas and "(" ("OR" signs).

As particularities of the compiler, be warned that you can't use the string "-- > " in your code, and that all lowercase characters may be uppercased, even if inside a string. You can't use " - " or " : " as the last character in a line, either. Finally, rule numbers with more than five characters will be
truncated to their first five, with no warning message, as any PLNLP identifier only has relevant its first eight characters.

The system

An attempt to describe the whole PLNLP system, so that the actions it performs can be understood and used, will follow:

Conceptually, it can be divided into three parts: PLNLP environment, the environment for PLNLP programming; PEG shell, a bunch of functions to help PEG understanding, debugging and use; finally, PEG herself. (In fact, there can still be CRITIQUE, or other programs based on PEG, but for this classification, PEG stands for the knowledge about English, with no concern about the actual results and ends of the programs. From a computer science's point of view, all these programs are at the same level: that of the use of PEG shell to store and deal with linguistic knowledge).

A careful distinction between what the system provides as a shell for PEG and what is really intrinsic to PLNLP will be made.

PLNLP environment

The first and essential tool to use PLNLP is the compiler. It won't be described in this text, though, except through the previous description of the language it must recognize.

More will be said about the LISP workspace into which the compiled PLNLP programs will be loaded (EXFed).

The PLNLP environment is in fact, and except for the basic primitives, written in PLNLP itself. The whole system was then translated into LISP, forming a shared LISP workspace. That means that, in the end, we have all the debugging aids of LISP plus those of PLNLP.

The PLNLP environment contains decoding and encoding rules, procedures and definitions of attributes and indicators, as well as an interactive loop which dialogues with the user. It is, however, rather difficult to separate PLNLP environment from PEG shell, so in some cases you may not agree with the classification proposed here.

As a first presentation, it can be said that the user can input to the system sentences in English (which would be parsed by PEG) as well as "PLNLP commands", that is, LISP instructions to change some environment's characteristics. The most usual are undoubtedly the loading into LISP of compiled PLNLP programs (by EXFing them), setting values of global variables (by using SETQ), and calling some functions specially designed for a friendly interface with the system from a user's point of view.

As examples of these last, the functions PRTREC and PRTREE display information from a specific record, or from a graph of records in a "tree-like" way. Both these functions and most of them were intended to satisfy the need for interaction with PEG, and probably will only be useful for that, so are not generic in any sense.

On the contrary, there are some features that are global in the sense they are not attached to any particular application, but are available to all of them. Specifically, some debugging aids available for encoding are extremely helpful for the writing of encoding rules:

The application of the rules whose numbers belong to one interval can be traced, printing the rules and the specific records who triggered and were created by them, by assigning a list of the limiting numbers to the global variable NLP-TERN.

(SETQ NLP-TERN '(4200 4500))

The creation of records of one or several specific segment types can also be traced, by creating a list with the corresponding segment type names and assigning it to NLP-PRSEGTYPES. Each time a record of one of the types is created, its name is printed. (This is in fact not only for encoding, but for all rules).

(SETQ NLP-PRSEGTYPES '(VP OLA TPPP))
The contents of the encoding stack can also be seen, and the same applies to what has been called the "left current record", or "top of the stack". For this you have to switch on the variables NLP-STACK-TR and NLP-TOPSTACKTR, respectively.

Finally, everything that is happening can be seen by switching on NLP-TRACE.

There are corresponding features to decoding rules, also, like NLP-TRSEG, NLP-DEC-FULL-TRACE, and tracing specific to procedures, as NLP-SET-TRACE (for NLP-SET) or NLP-TRACE-DICT (for READREC).

For rules only, the function APLY can apply them to particular records, so that the user can see their performance.

**PEG shell**

PLNLP environment has been described, unconcerned about its primary use: PEG. To it belong the compiler, the primitives of PLNLP written in LISP, the debugging helps and the PLNLP reader (which is significantly different from the LISP reader).

PEG shell contains all PLNLP code (procedures and rules) that were developed because of PEG. Its ends are mainly displaying, so most rules and procedures are for printing trees and records, in different formats and displaying different contents. These functions are based in and created for PEG, so it is wrong to think they are intrinsic of PLNLP. In particular, there are segment types who seem to receive a special treatment, and the same happens with attributes’ names. This has nothing to do with PLNLP, but it is the consequence of the fact that those segment types and attributes/indicators have a definite meaning in PEG. So PEG shell contains also the definition of indicators and attributes used by PEG.

On the other hand, the access to the Optimized Dictionary (OD) is one powerful tool which is embedded in the system provided. Although it is formally unrelated to PEG in that it is a general method of access, it is also unrelated to PLNLP in that the language does not presuppose any dictionary at all. So for the purpose of classification, it will be included in PEG shell, or better, the interface PLNLP-OD will be included there. It is code which is undoubtedly distinct from PEG, but which was introduced in the system because of it. (As it should be noted, nothing prevents the PLNLP user from analyzing the PEG shell, or, on the contrary, from creating another shell, having other display facilities and/or other accesses to OD...)

**PEG**

PEG herself will be considered everything which deals with an English sentence, already in the form of PLNLP records (created by PEG shell) creating one or several graphs representing the result of an extensive analysis done on deep linguistic knowledge. It comprehends several parts, coded in decoding rules (the "core grammar") and in encoding and procedures (like fitting a parse, ordering several alternatives heuristically, critiquing style and syntax). No attempt will be made here to further explain PEG, due to the excellent texts available, and to lack of competence to do it.

Only the features which are needed to proceed from PEG will be described, and to a minimum. From now on it will be assumed that the user gains control after the following processes occur:

Access to the OD English dictionary for the words integrating the input sentence,

PEG performance.

Printing of the resulting analysis’ trees, (by PRTREE), which was done by PEG shell, and subsequent creation of one TRANSL record.

A PLNLP program that uses PEG for translation should begin with this record, and have available all PLNLP features. Attributes and indicators’ names, as well as procedures’ and records’ identifiers should be attended carefully, so that no misuse or conflict arises. The list of indicators is available in the file LINDIC PLNLP, the list of attributes in the paper "PEG1986: A broad-coverage syntactic coverage of English" by Karen Jensen. Segment type names used by PEG can be found in the same reference, and will constitute the raw material on which the new programs will work.

There is no problem in writing encoding rules indexed by the same names, since decoding and encoding rules do not conflict. Redefinition of attributes and indicators, although possible, is not advised.
The rest of segment type names used in rules in PLNLP shell, as well as procedures', is scattered in multiple files, existing till now no listing of them, so the assignment of a particular prefix to every new name is recommended.

The results of the English analysis will be in the attribute SENTLIST of the special record MEM. The last sentence displayed is also available in the record TRANSL.

All procedures defined by PLNLP system are available to the programmer, who must nevertheless understand for which features of PEG they were designed for, and which consequences they may bring.

An example of the kind of unexpected side effects you may get by using already defined PLNLP functions is the call to PRTREE in a program of yours. This procedure calls the encoding algorithm, losing, this way, the "current left record". On the other hand, the execution of PRTREE always creates a new TRANSL record, so all your programs may get triggered by an innocent call to PRTREE from the PLNLP environment.

This example is only meant to warn you that you may have to choose between dwell on the interiors of PEG shell programs, or write your own functions.

Anyway, everything new will have to be defined by you in a new program, a PLNLP program. Good luck!
A set of procedures for transfer in MentorP/88

Diana Santos

April 1988

Transfer from English to Portuguese, from the graph structure created by PEG to a similar Portuguese structure, made us develop a set of PLNLP procedures to perform the basic actions for that task.

These procedures are the primitives used in PLNLP (structural) transfer rules, providing a higher-level language for specifying both actions and conditions in the rules.

No claim is being made that this is the correct set, nor that it is complete. In fact, this text is not more than a description of the procedures used so far in MENTORP/88.

The procedures developed so far

The procedures will be presented first by their declaration and then by examples of their call, but first some questions about their design will be discussed.

Concerning declarations, it should be noticed that a classification of the procedures' parameters was done, to increase readability, indicating the user what is expected from the arguments. That classification shows through PLNLP type declaration. It is important to emphasize, though, that the meaning assigned here to those types does not conform to their true PLNLP sense, since they were originally meant to represent data types, and here they do not.

The convention used is the following:

- PTR represents parameters that are passed to the procedure by means of a pointer to them. So it is possible to change the contents of what they point to inside the procedure and have those changes reflected outside it.
- CIR designates parameters that will receive an attribute name (enclosed in single quotes). The existence of such a parameter together with another one named SEG (of type PTR) makes it possible to use indirect addressing through that name, thus changing the record bound to SEG if an assignment is done.
- ANY stands for any other parameter, and designates in our convention that it is simply an input parameter. This means that the information that travels through it is not going to be altered inside the procedure. Of course, this does not prevent a ANY parameter from being a name, a pointer to a record, a list or whatever else.

In a word, the meaning of data types in procedure declarations was redefined to represent the kind of parameter binding that the procedure will perform for each parameter. So ANY stands for "call-by-value", PTR for "call-by-reference", and CIR for "call-by-name".

(It should, however, be clear, that PLNLP only uses call-by-value, and the present classification does not mean more than the simple fact that these procedures were implemented in such a way that behave "as if" the parameters were passed to them with the kinds of parameter binding referenced).

Another convention followed, this time for procedure calls, that may be worth while mentioning too, is the writing of names (inside single quotes) in lowercase. This is done to indicate that the name is intended to represent an option, that is, some word that requires a specific behavior from the procedure. The other names, that stand for attribute names for indirect addressing or values for tests, are written in capital letters instead.
Conditions

Only two procedures that test for conditions seemed to be needed at this stage:

PFIND looks for an occurrence of a specific attribute-value pair, in a list of records (usually pointed by a pointer attribute of a given record).

PFINDINDIR has a similar function, browsing the records (in a list of records) that are pointed by still another attribute. It may be said that it performs an indirect search, because it goes down through two levels of pointers.

They are, however, not pure conditions, in that they give origin to side effects, as will be soon shown.

Declarations

PFIND (seg*PTR, attr*ANY, value*ANY, where*ANY, result*CHR,
PFINDINDIR (seg*PTR, attr*ANY, value*ANY, where*ANY,
secondattr*ANY, result*CHR,

Their use in rules

Both PFIND and PFINDINDIR return the list of the records meeting the requirements that were asked for, so that they can be tested as conditions. However, their result is also put in the attribute stated in their parameter RESULT.

pfind < seg,'SEGTYPE1',INFCL,PSMODS,'FOUND'>

This can be read as "find in the list pointed by PSMODS of the current segment (SEG) all records which have a SEGYTP2 of INFCL. Put this information in the attribute FOUND of SEG."

pfind < OBJECT,'BASE','for',PSMODS(OBJECT),'FOUND'>

This means "find in the list that is the value of PSMODS(OBJECT) all records which have a BASE of 'for', and put that information in the attribute FOUND of the record pointed by the OBJECT attribute of the current record".

pfindindir < seg,'BASE',ENGPREP,PSMODS...PRMODS,'PRP','FOUND'>

Or, "look for all records in the union of the lists PSMODS and PRMODS of the current segment, that have a PRP attribute that points to a record that has as BASE the value of the ENGPREP attribute of the current record, and set the FOUND attribute to the result".

Actions

Several procedures were developed to perform specific transfer actions. Working inside a certain record, with attributes having as values PLNL lists of records,

PDELETE simply removes a record from an attribute.

PADD adds a record to an attribute. When this attribute has a PLNL list as value, it may add it as its top or as its end. It can also add it as the only occupant.

PINSET inserts a record before or after another in an attribute.

PCREATE creates a new record from scratch, and assigns it to the attribute indirectly addressed by its WHERE parameter.

PRAISE raises a record one level up, that means, change the current SEG to be this new record, but saving either the former SEG's PRMODS or PSMODS.

PLOWER, at last, performs the inverse function of PRAISE, but, as well as in PRAISE there had to be some precautions (like saving PRMODS or PSMODS of the old record), here, too, not the whole record is to be lowered down one level, but only a part of it (containing always its HEAD), BEFORE or AFTER some reference record.
Declarations

PDELETE (seg*PTR, segm*ANY, where*CHR,
PADD (seg*PTR, segm*ANY, where*CHR, position*ANY,
PINsert (seg*PTR, segm*ANY, refseg*ANY, where*CHR, position*ANY,
PCREATE (seg*PTR, where*CHR, typ*ANY, typ2*ANY, value*ANY,
PRAISE (seg*PTR, wasdown*ANY, savehow*ANY,
PLOWER (seg*PTR, where*CHR, type*ANY, relating*ANY, togolow*ANY,

As a curiosity, it can be mentioned, that the following two procedures were also once thought to be necessary, but the subsequent development of the rules proved (till now) that they are not.

PCHANGE (seg*PTR, segm*ANY, oldattr*CHR, newattr*CHR,
PREPLACE (seg*PTR, old*ANY, new*ANY, where*CHR,

PCHANGE would change one record from one attribute to another (in fact, it is the same as PDELETE and PINsert or PADD), while PREPLACE would substitute one record for another, at the same place.

Their use in rules

Through a detailed view of their calling, it is expected that the use and goals of these procedures become clearer.

pdelete < seg, FRSTV, 'PRMODS'>

This call to PDELETE will remove, from the attribute PRMODS of the current segment, the record which is the value of the attribute FRSTV.

By the way, it is convenient to recall, that there is also a PLNLP procedure called DELETE (and which is translated into LISP as NLP-DELETE) that deletes an item from a PLNLP list. The two procedures should not be confused.

padd < top < PSMODS > ,SUBJECT,'PRMODS','before'>

This call to PADD will add to the PRMODS attribute (of the record which stands at the top of the PSMODS of the current record) the record pointed by the SUBJECT attribute, and in the first position.

padd < seg,bot < PRMODS > , 'HEAD', 'alone'>

Here PADD sets the record which is the bottom of PRMODS of the current record (SEG) as its new HEAD attribute (unique).

PADD could be paraphrased in terms of simple assignment and use of NLP-dot (an intrinsic PLNLP procedure) with one argument always equal to the place where to store the result:

before - WHERE(SEG) = SEGM...WHERE(SEGM)

after - WHERE(SEG) = WHERE(SEG)...SEGM

alone - WHERE(SEG) = SEGM and some details that have to be dealt with.

However it seemed to us a simpler way of specifying these actions, creating the concept of (p)ADDing some record to an attribute in a certain position.

pinser t < seg,PORTPREP,'PRP','PRMODS','after'>

This call to PINsert inserts the record which is stored in the attribute PORTPREP in the PRMODS of the current record (SEG) immediately after the constituent of PRMODS pointed by the PRP attribute.

pinser t < seg,NEWDET,'FOUND','PRMODS','before'>

Here the value of the pointer attribute NEWDET is going to be inserted in PRMODS of the current record (SEG), before the record pointed by FOUND.

PINsert can be considered a more elaborated version of PADD, or better, it could replace this one (by specifying something like 'after' and 'any', for the 'after' option of PADD, and 'before' and
'top', for 'before', for instance) but this would make unnecessarily complex a PADD statement, so it is only used in cases where the specific place is not absolute, but relative to another record.

perate < seg,'NEWDET','AJP','DET','o' >

PCREATE receives a record and the name of an attribute (of that record) to store its result. It also receives the SEGTYPE, SEGTYP2 and BASE of the record it will create.

It must be said, that this procedure is intended to create Portuguese (target language) records to add to the graph, and never records representing English words or phrases.

This could obviously be achieved by the actual specification of the new record's attributes :

WHERE(SEG) = < SEGTYPE = ... , SEGTYP2 = ... , BASE = ... >

but, in fact, some other attributes (like FW, SUP and ESC, for example) are also set for the correct behavior of the system. Their specification would, however, only darken and complicate the rule, not belonging to the linguistic knowledge that should be expressed.

praise < seg,top < FOUND > , 'PRsaving' >

PRAISE raises the record which is the top of the attribute FOUND to the current segment (SEG), saving, however, the former SEG's PRMODS, which will be added to the new segment's PRMODS.

Once again, this is not more than a copy of attributes, but the procedure definition helps to save the user the burden of specifying them each time he/she wants to change the level of a record in the tree.

plower < seg,'PRMODS',CHICTOH,'before',top < FOUND > > ,

The example lowers all contents of the current record (SEG) that appear before the record that stands on top of the attribute FOUND. The SEGTYPE (and SEGTYP2) of the new record created will be the value of CHICTOH attribute. This new record will be assigned to PRMODS.

plower < seg,'PPOBJ',INFCL,'before','end' >

Here the whole record that before was the current record (SEG) is to be lowered, and have a new SEGTYPE (and SEGTYP2) of INFCL. This new record will be assigned to PPOBJ attribute of the current segment.

A comment on the examples presented

A remark should be done here. All examples presented came from our rules, except for the second one on PFIND.

This one was introduced in order to illustrate the possibility of calling the procedures with a first argument different from SEG. In fact, apart from that and one of PADD examples, very few calls use this potentiality.

The reason is, however, straightforward. In the design of MENTORP/88, any actions that change graph constituents located more than one level lower, are forbidden (because they would be changing English and not Portuguese records).

This is the reason why most examples presented have SEG as parameter. In fact, PDELETE, PINsert and PRAISE only can have this argument not to disobey the rule. All others, including PFIND and PFINDINDIR, can have as first parameter a record that is at most a level lower (pointed directly by an attribute) from SEG.

It is important to emphasize, though, that this is not a limitation that comes from the procedures, but a rule for their use dictated by the philosophy of the design of our system.

Real rules that call some procedures

Some rules will be presented, to illustrate the use of this set of procedures, and to show how their use increases the readability and "writability" of the code.
They have suffered no simplifications. It can thus be seen, that the number of simple assignments and deletion of attributes is greater than that of calls to the procedures. This makes us conclude, that the "structural" movements in a rule of ours are as important as the setting of the correct information (attributes and indicators) for the segment of text at hand.

// This rule is applied at the INFCL level, and two things can happen://
//  * The INFCL remains an INFCL in Portuguese, with no preposition, //
//      if the attribute CHINCNOTO exists, or //
//  * The INFCL turns into a PP, with PREP the one that was carried in//
//      the attribute CHICTOPP, if it exists, or "para", otherwise. //
// In any case the INFTO ("to") is always deleted. //

(4940) VP(<INFIN|INFVP>,
    pfind<seg,'SEGTP2','INFTO',PRMODS,'FOUND'>.NE.0)
   -->
   INFCL(%%VP,
      SEGTYPE='INFCL', SEGTP2='INFCL',
      pdelete<seg,top<FOUND>,PRMODS>,
      -INFTO,
      <-CHINCNOTO,
      <CHICTOPP|CHICTOPP='para'>,
      =pcreate<seg,'PPREP','PREP','PREP',CHICTOPP>,
      INFPP=INFIN, -INFIN, -INFTV,
      SEGTP2='PP', SEGTYPE='PP', PRP=PPREP,
      =plower<seg,'PPOBJ','INFCL','before','end'>>,
      padd<seg,PPREP,PRMODS,'before'>,
      -PPREP, -FOUND)

// This rule alters the relative levelling of the constituents in the //
// sentence, turning the INFCL into the head, and sending the former //
// head to constitute the PRMODS. //

(4925) VP(CHICTOHD,
    pfind<seg,'SEGTP2','INFCL',PSMODS,'FOUND'>.NE.0)
   -->
   VP(%%VP,
      plower<seg,'NEWPR',CHICTOHD,'before',top<FOUND>>, PRMODS=NEWPR, -NEWPR,
      praise<seg,top<FOUND>,PRsaving>,
      <<CHICTOHD.EQ.'PP', -PRP, -PPOBJ, -INFPP, INFTV=1>|
      <SUBJECT=top<PRMODS>,
      DELINFTO=1,SEGTYPE='VP',SEGTP2='VP',
      -INFTO, -INFIN>>,
      -CHICTOHD, delete<infcls12',DONE>)

Conclusions

Several procedures were described, and illustrate the needs of this first phase of MENTORP/88.

Apart from two used for finding records with a particular characteristic, most of them were developed to perform simple actions, improving like this the ease of writing transfer rules for translation from English.

They were developed in a necessity-driven way, and will probably be enhanced and their number increased as work goes on.

They may serve, however, to describe the kind of basic transformations that a PEG graph needs to suffer for a Portuguese graph to emerge.

A set of procedures for transfer in MentorP/88
No surprises arose. In fact, there is creation and deletion, there is adding and inserting of new or old but displaced constituents, and there is upward and downward navigation in the graph.

Their writing was in most cases straightforward, having the present procedures' bodies sizes varying between 6 (PADD) and 22 lines (PLOWER). They were as much as possible written to conform to PEG philosophy and conventions, so that their results should be indistinguishable from PEG output.

Following work with this set of procedures will help to perfection them, and eventually to completely adjust them to the writing style necessary to transfer. We hope then to be able to furnish them to all other sites to whom they may be useful for transfer, particularly to the other Mentor developers.

Comments and criticisms are most welcome, particularly about the possible use of these tools for translation from English to other target languages.
The translation of infinitive clauses in MentorP/88

Diana Santos

May 1988

Introduction

This paper has as major goal the illustration of the development and implementation of a set of rules that cover a specific domain of English to Portuguese translation in the MentorP/88 environment.

The architecture of the translation system, together with PEG philosophy, enforces a particular approach, which will be described.

Infinitive clauses were chosen by two kinds of reasons.

From a theoretical point of view, they are relatively complex in that the same pattern in English can give origin to several different constructs in Portuguese, sometimes with a completely different structure.

On the other hand, the fact that the Spanish team was at the moment solving this very problem for their language elected INFCLs as the best set of rules to develop, in that it allows for a comparison between different translation systems’ designs. It should also be mentioned, by the way, that the first set of rules on paper was given to us by Teolfo Redondo Pastor from Madrid’s team.

Organization of this text

First, to try to make the set of rules derive logically from the system, some points will be briefly described:

PEG’s analysis of INFCLs: the information available, and the possible structures it produces.

The transfer philosophy: or the design of the transfer phase.

Then, a description of the informal rules that should be implemented. Simply stating what should be the Portuguese output is obviously essential, not only to describe to non-Portuguese speakers the kind of transformations needed, but also to show the differences between this kind of formulation and the one used in the actual transfer rules.

Next, some of the rules are described at length, while others are just briefly referenced. A discussion of some options is presented.

A description of the performance of the set of rules follows. Their successes and shortcomings are mentioned, together with some details that concern the overall design.

More than to impose this kind of design or assert that these are the correct rules, this text is intended to give an overview of the programming style for translation in MentorP/88, and, we hope, to demonstrate its adequacy to the transfer between two languages.
Starting points

PEG analysis of INFCLs

A standard INFCL in PEG is represented by a non-terminal record with SEGTYPEP2 (and SEGTYPE) INFCL. The vast majority of INFCLs in English do have the word "to" preceding the verb. That is stored by PEG in the INFTO pointer attribute of the INFCL. The HEAD of every INFCL is its verb, while at PRMODS are the INFTO and sometimes adverbs. Anything can be PSMODS in an INFCL (VPs, INFCLs, NPs, AIPs, PP...). Other information is also specifically connected to INFCLs in PEG:

INFIN, in a non-terminal record, points to the INFCL that exists one level down from that record. INFIN has the value one when inside the INFCL itself.

INFTTV is an indicator that signals that the tense is infinitive, and with the "to" too, (note that this is not the same as INF, that indicates that the tense of a particular verb appears in the sentence in its non-marked form).

INFPP, only meaningful inside a PP, indicates that the prepositional phrase has an INFCL in it.

Also, conjunction of INFCLs is also parsed as an INFCL by PEG, together with a conjunction of VPs with only one INFTO (like "to read and write").

Finally, there are some (rare) cases where INFTV is zero. This happens when there is no INFTO, (e.g., the argument of "make" in "he made me come").

Transfer architecture

In MENTORP/88, the navigation in the PEG graph obeys the following philosophy: in each node, given the knowledge expressed in it and in the nodes immediately down, some changes are done, or an alteration is signalled downwards.

There can be no changes of the nodes above (that were already changed before) nor of nodes not immediately (one level) down from the current one, because otherwise we would be changing "English nodes" and not those of the new Portuguese graph (with specific information about the target language, taken from the bilingual dictionary).

This does not mean, clearly, that we cannot get to nodes that are further away than one level down. On the contrary, we can use their information and even raise them to the current level, provided that, in this last case, they get copied to "Portuguese nodes". What is forbidden is the modification of records two or more levels down from the current record, maintaining their position.

For a more detailed explanation on the system's design, see "Report on a PLNLP-based translation system".

An informal description of the translation requirements

As shortly as possible, let us try to formalize the differences between the English sentences that include an INFCL and their Portuguese counterpart.

When the INFCL is the subject, the direct object, or the predicate nominative of a sentence, it remains an INFCL in Portuguese, such as

1. To read is good --- LER e" bom.
2. He hates to read --- Ele detesta LER.
3. My hobby is to read --- O meu passatempos e' LER.

The same happens when we face a "dummy subject" construction in English, where the INFCL is the logical subject of the sentence:

4. It is good to read --- E' bom LER.
5. It is a difficult task to read --- E' uma tarefa dificil LER.
If none of the particular cases that will be described below holds, the default translation of an English INFCL is a purpose clause in Portuguese, that is characterized by the conjunction (preposition) PARA while the tense of the verb remains infinitive. (In PEG’s terms, it is represented by an INPP with PREP “para”).

6. To activate this function you must follow the ... PARA activar esta func,a-o deve seguir ...

7. He came to the room to have dinner --- Ele veio para a sala PARA jantar.

There are, however, several specific situations where translation outgrows these simple norms.

When we face an adjective phrase containing an INFCL, by default this turns into an infinitive PP with preposition “de”, except if the particular adjective that is the head of the AJP asks for another preposition (notice the Portuguese lexicon’s influence).

8. He is ready to dance --- Ele esta’ pronto PARA dançar.

9. He is happy to dance --- Ele esta’ feliz POR dançar.

10. He is easy to cheat --- Ele e’ fa’cil DE enganar.

Similarly, when an NP is found containing an INFCL, four cases may happen:

If that INFCL is really acting like a modifier of the NP’s head, (as is the case of nouns marked with the feature TOV in the English dictionary), the INFCL should be converted into an infinitive PP with default preposition “de” (provided no other preposition is stored in the bilingual dictionary).

11. This is a way to read --- Esta e’ uma forma DE ler.

12. He had the chance to meet her --- Ele teve a oportunidade DE a conhecer.

When the noun is (semantically) the direct object of the INFCL, or its subject, the INFCL gets translated into an infinitive PP with preposition “a”.

13. He is the man to kill --- Ele e’ o homem A matar.

14. The name of the worker to do the job --- O nome do trabalhador A fazer o trabalho.

When none of the three cases above holds, the correct translation is the default already mentioned, that is, a purpose clause, or an infinitive PP with preposition “para”.

15. He entered the shop to buy some clothes --- Ele entrou na loja para comprar alguma roupa.

PEG parses of PPs who include an INFCL, and are started by the preposition “for” will, on the other hand, in Portuguese be translated by a tensed PP, whose tense is the personal infinitive (a tense only belonging to the Portuguese language).1

16. This book is good for you to learn English --- Este livro e’ bom para tu aprenderes ingle-s.

Finally, there are two English constructions that do not have a similar counterpart in Portuguese, and for that reason imply several transformations:

The first covers some verbs in the passive which mean a general but indistinct subject, like

17. I am supposed to do the job --- Supo-e-se que eu fac,o o trabalho.

18. He was assumed to have returned in 1950. --- Assumia-se que ele tinha regressado em 1950.

This implies changes both in the main clause and in the INFCL, e. g., the passive clause becomes an active reflexive one, and the INFCL turns into a new sentence which inherits the first clause’s subject and tense, among other modifications.

The second problem are the verb plus NP constructions that can be paraphrased in English as the verb plus a THAT sentence...

19. I wanted the young man to come -- I wanted that the young man come.

---

1 By the way, the question whether this construct should be classified as a PP or as a VP is not essential for our dealing with the problem, since that would only amount to a trivial change of the attributes SEGTYPE and SEGTYP2.
The Portuguese language only admits this second kind of statement, implying again a structural change, this time turning an NP into a VP representing an integrating clause, in the subjunctive mode, and with the tense coming from the clause above.

To end this description, one of the features of Natural Language should be recalled: its infinitude, in the sense that the rules that accurately describe it must be recursive, not to create artificial limitations. By this we mean that phrases like

- to want the good man to pay to enter the room to have dinner not to starve ...

have already been described here, and, moreover, that they must also be accounted for by our rules.

### The actual set of transfer rules

#### Methodology

The way we dealt with this informal set of rules to obtain the behavior described above is the main subject of this text.

As everyone who ever dealt with real implementation knows, the way a piece of knowledge in a computer program gets implemented is most of the times quite different from its stating in everyday words. The ultimate goal, however, is to make these two descriptions as similar as possible. PLNLNP is quite a good tool for this, we concluded, but we are still forced to a little "twisting".

The first thing we have to think about is at which level (of PEG's graph) do we need to begin creating rules. From a first ordering of all cases described above, we see that the ones we must treat from a highest place above the actual INFCL level are the situations where an NP should be transformed into a complement clause (see example 19) and those that have a dummy 'it' as subject (examples 4 and 5). The "distance" in these cases is two (levels).

They have to be taken care of at the level of the VP that includes them, the first because the kind of verb must be taken into account, and the second because the VP is the lowest place possible to test the verb phrase's subject.

Then come the situations where there is a need to work at the immediately upper level with respect to the INFCL : NPs whose noun imposes a change, AIPs whose head asks for a particular treatment, VPs in the passive which belong to the particular category mentioned (examples 17 and 18) and which will suffer some change, and finally normal VPs which have as subject, predicate adjective or nominative, or object, an INFCL. It should be evident why they have to be considered at this level.

Finally, when instead of a normal INFCL, there stands a conjunction of INFCLs, it is necessary to guarantee that both terms receive the same and adequate treatment. This can also constitute an additional level between the simple INFCLs and the place where the necessary modification is detected.

And only remain those cases where simply at the INFCL level it is necessary to do something.

With this classification in mind, the next question is organization. Should there be one rule for each distance (and for each different SEGTP2, of course) or should a rule be written for each separate case, or, still, should the number of rules be decided by some other strategy ?

We decided to separate the main rules above the terminal INFCL itself in four rules for VPs, one for NPs and another for AIPs. Since the NPs and AIPs have to be separated from the VPs by PLNLNP demand, and the information to associate with each is not enough to justify further fragmentation, only the VP division begs for an explanation.

It was based simply in PEG's features: the distinction between PRMODS and PSMODS, or, in other words, its tree structure. We chose to assign a rule for VPs that own INFCLs as PRMODS, another to VPs that represent a conjunction of two INFCLs (one in PRMODS, the other as PSMODS), and two that handle INFCLs in their PSMODS - one in the PSMODS of the VP's PSMODS (to handle "distance two" problems) and the other that treats INFCLs in PSMODS

---

2 We suggest that PEG's trees be consulted for a clearer perception of these lines.
(which, in PEG's terms, and because of its right-attachment strategy, means exactly the same as "immediately following the verb").

These six rules handle all cases described (with the exception of PPs with preposition 'for', which will be discussed in the next section). There are two more rules that concern the INFCLs themselves, and perform what was destined to them by the transfer rules applied to their parent nodes, or apply the default action. They are separate because they give origin to nodes with different SEGTYPES.

(4901) * If a VP has an INFCL in the PSMODS of its PSMODS.
(4905) * If a VP has an INFCL in its PSMODS.
(4910) * If a VP has an INFCL in its PSMODS.
(4915) * If a VP is a conjunction of INFCLs.
(4920) Changes passive voice into active.
(4925) Changes the infinitive clause into the VP's head.
(4930) Puts a COMPL before integrating clauses.
(4935) Deletes the INFTO from a normal VP.
(4940) Handles infinitive clauses that remain INFCL.
(4945) Handles infinitive clauses that turn into PPs.
(4950) * Rule for NPs that have an INFCL in its PSMODS.
(4955) * The same for AJP's.
(4960), (4965), (4966) and (4970) Handle PPs beginning with 'for'.

Note: The * marks the six rules referenced.

Please note that in the preceding table, the feature VP stands for our general concept of VP, that includes PEG's VPs and INFCLs, among others, (e.g. ABBCL, IMPRs, CMPDs, SUBCLs, etc.).

Another remark is also noteworthy. Though nothing has been said about PPs who themselves include an INFCL (except for the 'for' case), the rules enumerated above automatically handle them. This stems from the fact that we do not need to create any additional rules for PPs repeating what was stated for NPs or VPs, since we perform the transfer of the record pointed by the PPOBJ of a PP first thus making duplication of rules unnecessary.\(^3\)

Of course, some programming cares had to be taken. For instance, some rules are exclusive from one another, and most rules act also as transmitters of information from their parents. Moreover, all rules which do not set particular switches for further processing which will then be reset, have to be prevented to trigger again and again resulting in infinite loops.

**Some sample rules**

To help the understanding of what has been said, some of the most complex rules will get detailed, so that all of them can be inferred.

Let us begin by rule number 4901, that handles "distance two" problems in the PSMODS of a VP.

The PLNLP rule follows:

---

3 See our text on "Dealing with Hidden Pointer Attributes in MENTORP/88".
The right-hand side of the rule is divided in the two cases sketched below, which, by the way, are disjoint (no VP has at the same time a PREDADJ and an OBJECT attribute). We shall only examine the first case, here.

<table>
<thead>
<tr>
<th>English structure</th>
<th>Portuguese structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP(OBJECT)--*VERBI(NPTOV,THATCOMP)</td>
<td>VP--*VERB1</td>
</tr>
<tr>
<td>--NP--*..</td>
<td>--NP--*..</td>
</tr>
<tr>
<td>--INFCL--*VERB2</td>
<td>--INFCL--*VERB2</td>
</tr>
<tr>
<td>VP(PREDADJ)--NP--PRON(&quot;xxx&quot;)</td>
<td>VP--NP--PRON(&quot;&quot;&quot;)</td>
</tr>
<tr>
<td>or</td>
<td>or</td>
</tr>
<tr>
<td>*VERB1</td>
<td>*VERB</td>
</tr>
<tr>
<td>PREDNOM--NP--*..</td>
<td>--NP--*..</td>
</tr>
<tr>
<td>--INFCL--INFTO</td>
<td>--INFCL--*VERB2</td>
</tr>
<tr>
<td></td>
<td>--*VERB2</td>
</tr>
</tbody>
</table>

The rule begins by storing information in the (previously) NP to turn it into a VP. It signals that the particle "que" (translation of "that") should be added, and sets the new attribute CHICTOHD (for Change InfCl TO Heads) to the value of SEGTYP2 of the record that currently stands at the top of PSMODS. It also deletes all marks of infinitive, and copies the tense of the VP into the new one, plus setting the SUBJUNCL indicator, to impose the subjunctive mode.

In the particular case of this rule, we can see that only the son node is changed, by storing in it some information. The changes will be actually achieved by the work of other rules, when that node in turn will be the current one.

Briefly, rule 4930 inserts a complementizer in integrating clauses, and rule 4925 handles CHICTOHD markers. This last lowers down one level everything that appears before the INFCL,
creating a new node having as SEGTYPE(2) the value stored in CHICTOHD, and raises the verb of the INFCL and everything in the INFCL to the new current record. The result should be undistinguishable from PEG analysis of an English integrating clause.

This example illustrates some points of interest: first, that the detection of the conditions that lead to a transformation is likely to be done in a different node (in a different cycle of the transfer process) than the one that will suffer the actual changes. Also, different changes are handled by different rules, achieving like this modularity and the existence of general processes that can be used in several different contexts, instead of solving one case in a single rule, that could only apply in a very particular case. (Rules 4920 and 4930, for instance, can be said to be global, in that they apply in many other cases apart from INFCLs’ processing).

Let us pick the example of rule 4950 now, to emphasize another feature that brings a considerable power to our system, and which can be called as a "cascading" of information.

```
// This rule concerns NPs which have as PSMODS an INFCL. The preposition/
// "de" is used by default, when the word is classified in OD as TOV. //
// Example : The possibility to read ... A possibilidade DE ler... //
// When the NP has as HEAD an adverb (like "how to do ...") no preposit-//
// tion is inserted. //
// When the head of the INFCL is a transitive verb, and there is not yet/
// an object, the preposition to insert should be 'A'. //
// Example : The man to kill ... O homem a matar ... //

(4950)  NP('npins', NOTIN.DONE, DONE = 'npins'...DONE, plind <SEG, 'SEGTYP2', 'INFCL', PSMODS, 'FOUND' > .NE.0) <<

  NP(%%NP,)
  < SEGTYPE(HEAD),EQ.'ADV', CHICNOTO = 1 >,
  < ~ CHICTOPP, ~ CHICNOTO, TOV, CHICTOPP = 'de' >,  
  < ~ OBJECT(top < FOUND >), TRAN(top < FOUND >),
  SEGTYP2(HEAD),NE.'PRON',
  ~ CHICTOPP, CHICTOPP = 'a' >,
  CHICTOPP(top < FOUND >) = CHICTOPP, -CHICTOPP,
  CHICNOTO(top < FOUND >) = CHICNOTO, -CHICNOTO, -FOUND)
```

This rule handles NPs which have an INFCL as PSMODS (by PEG's attachment philosophy, that means that the INFCL is immediately following the NP's HEAD, but that is not required for our rule). Basically, two things can happen: one is that while transferring the NP this rule will serve as a transmitter of information downwards to the INFCL (in case there is already a CHICTOPP or CHICNOTO feature). But it can also be the case that the particular characteristics of the NP (like subcategorization of the noun, or syntactical characteristics of the NP itself) add or impose new information that regards the "daughter" INFCL.

This means that, instead of trying to describe and solve a whole situation completely, we let many factors interact to result in the correct translation. This is also responsible for the fact that there are no complexity limits for the situations we can handle, or, in other words, the recursive property of natural languages is preserved in our treatment.

In conclusion, with these examples we tried to give the flavor of complex translation rules from English to Portuguese in our system. Several points were pointed out: modularity, generality, and cascading of information.

**Dictionary information**

It may also be of interest to have a look at the dictionary information we use for this task, comprehending both English and bilingual features. Of course, English categorization of verbs and nouns is already extensively used in PEG to achieve the analyses it produces.

However, we also test explicitly some English verb and noun features to decide among several alternative translations.

- NPTOV and THATCOMP (for verbs) : to be discussed below.
- THATCOMP (for verbs) : to trigger transformations of sentences like that of example 19.
• TRAN (for verbs): to determine transitiveness, for rule 4950, where something like a "role classification" is being sketched (see also discussion below).

• TOV (for nouns): to know whether the infinitive clause that follows is actually modifying the noun.

On the other hand, we are at this moment only making use of one feature in the bilingual dictionary, to store the preposition asked by the word to precede the Portuguese infinitive PP that will replace the INFCL.

So, in particular, for examples 8 and 9 the corresponding entries in the dictionary are

| ready:pronto(ADJ (INFPREP para)) |
| happy:feliz(ADJ (INFPREP por)) |

Till the moment, there haven't been found any nouns that asked for a different preposition than the ones contemplated in the rules. Also, verbs that ask for a preposition only for INFCL translation (when it is general) are already handled by other rules) were discovered to be only "have to", "be used to" and "be going to", which are not parsed as verbs + INFCL by PEG, but as quasi-auxiliaries. So the feature INFPREP is not being used for verbs, either.

Discussion

Problems

No honest description of a implemented piece of work could hide the problems that remain to be solved. A distinction between two kinds of problems will be done, though. The first includes things that could be improved, and probably there are already solutions known. The second, much more important, refers to the problems that cannot be solved with this approach, and so ultimately define its limits.

In the first category, we have things like the existence of the word "enough". It happens than in a AJP that includes the adverb 'enough' followed by an INFCL (implying a form of comparison or measuring), it is this word, and not the head adjective, that imposes the translation.

it is good enough to buy -- E' suficientemente bom PARA comprar.

We chose to look for the existence of the word 'enough' after the NP's head, but we are conscious that this is not the best approach, because the rule is testing for the existence of a word, and not for a structural feature.

In fact, we could identify this phenomenon by a more general characteristic, such as the attachment of the ADV ('enough') to the AJP instead of to the INFCL. In fact, PEG usually attaches adverbs to the front of other constituents, so we could also test for the existence of an ADV in PSMODS. However, and since there were no other case found that obeyed the same pattern, we let it like this.

Another problem whose solution we are not particularly happy with, is the construction "for <NP> to <INFCL> <anything,> « •. While we achieve a correct behavior in every case tried so far, again it is the design that should be improved. In fact, PEG parses this construct as a normal PP, although it has a more clause-like appearance. We needed to write a separate rule for PPs beginning with 'for' and including an INFCL. However, weren't it for a particular situation, one single rule would be enough. Recall that these constructs are translated into Portuguese into infinitive clauses that agree in person and number with their subject.

• For him to come here a huge problem must have happened --- PARA ele VIR aqui um grande problema deve ter acontecido.

However, in sentences of the sort :

• It is good for him to read a book --- E' bom PARA ele LER um livro.

the analysis provided by PEG, joining what follows the adjective ('good', in this case) in a single PP fails to separate "to read a book" (which is the "logical" subject of the clause) from "for him" (which is modifying "good"). In Portuguese, these different structures have a different translation
(exactly like examples 4 and 5), and so, two more rules had to be added to account for this case, which, by the way, is a "distance-three problem".

The question here is whether this is a true INFCL translation problem, or, instead, it is an INFCL attachment issue. We could as well argue that the lack of elegance of this particular design (in that instead of a rule covering several cases there are several rules to handle an exception) would automatically disappear if PEG distinguished between the two cases.

So, once again, we are conscious of a design problem but not sure if it is in the INFCL domain that it should be handled.

And now the real limits of this transfer method will be presented. They originate from syntactical ambiguity that can only be removed by understanding the sense of the sentence (which might in fact even not be possible for a human being without further information...)

The cases detected are related to the assignment of syntactic arguments in the INFCLs.

Most English verbs can be both intransitive or transitive. In the example below, "the man" can be considered the object of "to kill", while "the gun" cannot. Also, "the man" in the third sentence is to be the subject of "to kill", while in the first it is its object.

I chose the man to kill. (it was to be a known politician)
I chose the gun to kill. (it was the firearm I preferred)
I chose the man to kill (the best assassin I knew)

We can therefore see, that the same constituent can be either the subject, the object or a simple optional argument of the verb.

Since the third sentence looks a little weird, we give some others, to explicit that, in the three sentences above, there is not one ambiguity (the choice of one role for the INFCL), but two (because both (semantic) subject and object may be dubious).

I chose the man to die. (SUBJECT-the man)
I chose the weekend to die. (SUBJECT-I)

the name of the entry point to receive control is XZZZZ. (SUBJECT-the entry point)

write the name of your password to receive control. (purpose clause).

In these cases, it is the meaning of the constituents that decides the right translation. Our system simply cannot distinguish them, so it employs the following default rule (see rule 4950 above) : if the NP is followed by an INFCL whose head is a transitive verb with no PEG object associated, it is assumed that the INFCL constitutes the object of that verb. Otherwise, it is considered to be a purpose clause as was described above. Never the possibility that the NP be the subject of the INFCL is taken into account in our system.4 Though it fails to translate correctly example 4, for instance, this strategy gives considerable good results, as can be appreciated in the next section.

Another issue that may be of interest to discuss is how we distinguish between the verbs where the NP-INFCL will turn into an integrating clause (see example 19) and the others. We are relying on the verb classification stored in OD, assuming that verbs marked with both NPTOV and THATCOMP have this property regarding translation into Portuguese, while others do not.

This approach seems intuitively correct, by the way, since we are going to change the NPTOV construction into an integrating (THAT-)clause... Moreover, it works for the great majority of the verbs, with, unfortunately, at least two exceptions, namely "want" and "phone". In fact, "want" does not belong to this set (not being THATCOMP), while "phone" does not admit an integrating clause translation in Portuguese. Both these examples pose an interesting question : is this subcategorization meaningful to English, or only to the translation into Portuguese ? If we assume this category is valid because it implies a different meaning for the sentences, we have to ask for a new OD feature that holds all THATCOMP and NPTOV verbs plus the verb "want". However, this will not solve the problem with "phone" which has a different case frame in Portuguese, (since it does not admit OBJECT and so no integrating clause could fill the object argument).

4 We are assuming that PEG syntactical arguments correspond to semantic roles when formulating this rule.
So, it looks as if the feature CHIFTOTH (CHange InFel TO THat clause), for instance, should be stored in the bilingual dictionary. However, if the order of the rules is conveniently followed, never will "phone" get a wrong translation, because, before treatment of INFCLs, the convenient case change will be performed. On the other hand, and looking at the example of the verb "tell" (also in the same class), we argue that the parse PEG furnishes is wrong (being the correct output, in our opinion, that the INFCL be parsed outside the NP, filling the object argument, while the NP be considered the INDOBJ). Like this, we are again getting back at the conclusion that this is not, in the end, a bilingual matter.

This discussion was brought here mainly to give an account of some problems that probably would never arise without an actual implementation, and whose final answers are not known yet.

Performance

All cases thought by us were tried and are succeeding, with the exception of the ambiguities described before. However, the real test we consider is the set of 120 sentences of the REXPLI manual that include 141 instances of INFCLs. Of this set, we claim that the translation of 104 are completely correct, while 26 are wrong because of wrong PEG analysis (but consistent with it) while only 11 sentences were not correctly translated.

Of these last, 5 concern the verb "have to" and 2 "to be to be" constructs. We have purposely not handled "have to", because while we consider (from a Portuguese point of view) that they should be parsed as HAVE + INFCL, PEG doesn't do it, converting them instead in a only node with base HAFTA. Like this, they have no place in the set of rules we developed.

"to be to be" constructs, on the other hand, is a multiword expression (considering its translation into Portuguese) which is not considered as such by PEG. So, and while acknowledging that it can be a problem, we include it in the more general MWE problem and not in INFCL (where if it had a literal translation would arise no problem).

Finally, we must acknowledge our failure in dealing with the three remaining sentences. Two of them are due to undecidability of subject versus purpose clause constructs by syntactical analysis, as was extensively described, while the other comes also from ambiguity, this time in the INFCL attachment to adjectives versus purpose clauses.

In appendix, both the set of test sentences used (all sentences that include INFCLs from a REXPLI manual, and which was kindly furnished to us by the MENTOR Spanish team), and the set of actual results achieved are available, together with a more extensive description of the performance.

Conclusions

We claim that the good results obtained with the rules described in this report prove both PEG's power and the adequacy of our system to deal with its output.

We believe, too, that this also supports the idea that syntax, together with a relatively small number of attached features to lexicons (both English and bilingual) can make considerably more (at least in the restricted domain of computer manuals) than current linguistic trends seem to consider.
Performance of the set of test sentences on INFCLs in MENTOR88/P

This is a description of the results presented in the file INFTO EXF, and constitutes an appendix to "Dealing with INFCLs in MENTOR88/P".

The sentences will be referenced by the number appearing in the file INFTO TEO. Whenever more than one sentence appears with the same entry, the sentence meant is the one that includes an infinitive construct. When there are more than one occurrence of (independent) infinitive clauses, a number inside parenthesis indicates whether it is the first, second or third one.

It should be emphasized, that all analyses (including those on PEG) are always about the infinitive clause subject, and so both wrong parse trees or strange translations will be considered correct if their treatment of infinitive clauses is correct.

The test sentences were not prepared by us in any way, resulting in a repetition of several constructs while others never arose. To see all cases we considered, see our paper on this matter, and also the file INFS EXF, that includes the sentences we created during the rules' development.

First of all, PEG's performance on infinitive clauses appears to be very good, with the exception of one particular matter: no assignment (contrarily to all other verb phrases) of the feature OBJECT in relative clauses (RELCL). This is the major source of what we considered wrong analysis, and it happens in five sentences, namely 69, 180, 193, 204 and 218 (in this last, ironically, it is a deficient assignment of OBJECT that results in a wrong translation).

The other incorrect analysis we detected were the following, where an X means that they were also classified as ungrammatical by PEG: 4(1), 33(X), 57, 297(X), 132(X), 332(X), 454, 459 and 473.

We also included as incorrect the sentence 413, related to the verb "tell", though it may be considered correct from other point of view. Our proposal is to parse a "tell" sentence assigning the INFCL as its direct object, and what precedes it as INDOBJ. However, we are aware that if we assume that PEG's strategy is correct, we would have to change our rules to handle CMPXTRAN verbs differently.

Another analysis we have also considered incorrect (but which could easily be handled if we didn't) is that of the sentence 440, in what concerns the verb "seem". The infinitive clause should, in our opinion, be considered the PREDOM, as PEG does with other COPL verbs (namely "be").

These are 16 out of 141, which we consider very good, specially if we take into account that the greatest majority of the sentences are by no means simple. To add to this number, there are 10 fragments of text (namely 181, 182, 210, 211, 212, 228, 229, 240, 241 and 242), that PEG parses as XXXXs and fails to attach the INFCL to the NP that precedes it. Considering, however, that this would be extremely difficult (and maybe impossible) to do (and considering that, moreover, the switch NLP-FORCE-NPs was not even tried) we do not consider these sentences as showing bad PEG's performance.

The sentences 18, 64, 396, 432(1), 438(2), 461, and 480(2) include the verb "have to", that was not dealt by us as pertaining to INFCL problems. We only added the word "HAFTA" to the dictionary and translate it. By the way, in the cases where the word "have" was separated from "to", PEG failed to recognize it as HAFTA, and the translation turned out completely wrong. We added these cases to PEG failures above: 28 and 57.

Another set of sentences not dealt with are those with "to be to be" expressions, namely 105 and 394.

As a curiosity, one sentence seems to us wrong (in English). PEG also classifies it as XXXX : sentence number 288. We are, like this, unable to classify its translation.

Sentences 336 and 347 have INFCLs that are wrongly assumed to be purpose clauses, while they are "untensed" clauses whose subject is the preceding NP. This was described at length in the text, so we only want to call attention for the much lower percentage of occurrence of these kind of sentences compared to purpose clauses.

Finally, sentence 329 fails to give a correct translation because it assumes that the INFCL following the adjective is related to it, when it is simply a purpose clause. Since "X makes it unique to avoid... is syntactically undistinguishable from "X makes it easy to avoid..." we can only think of a (semantic?) categorization of adjectives (like adjectives of manner distinguished from the others) to
handle this problem. We are convinced, however, that this cannot be handled by syntactical features alone.

At last, all numbers we can think of:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of sentences</td>
<td>120.</td>
</tr>
<tr>
<td>Number of infinitive clause occurrences</td>
<td>141.</td>
</tr>
<tr>
<td>Number of PEG incorrect parses</td>
<td>14 + 2 + 10.</td>
</tr>
<tr>
<td>Number of not dealt with situations</td>
<td>7.</td>
</tr>
<tr>
<td>Number of translation failures</td>
<td>3.</td>
</tr>
</tbody>
</table>

Percentage of correct output: \( \frac{(141 - 36)}{141} = 74.5\% \)
Percentage of correct translations: \( \frac{(141 - 10)}{141} = 92.9\% \)
Percentage of translation error: \( \frac{3}{141} = 2.13\% \)

Note: To the sharp eye who noticed that two sentences from INFTO TEO are missing from our results: this was due solely to the fact that they gave origin to unexpected LISP execution errors. They (158 and 453) have not been removed by any other reason.
PRESENTATION OF THE WORK BY
THE PORTUGUESE BRANCH OF
MENTOR

Diana Santos
Paulo Libano Monteiro

Grupo Científico IBM-INESC
R. Alves Redol, 9 - 4ºE Dto, Sala 412
1000 Lisboa
# Table of Contents

Introduction to MENTORP ........................................ 1
MENTORP history .................................................. 1
Departing points .................................................... 1

Generic options taken ............................................. 2
PLNLP as the only programming language ..................... 2
  Reasons for .................................................... 2
  Conclusions .................................................... 2
OD dictionaries .................................................... 2
  Reasons for their choice .................................... 2
  Conclusions .................................................... 2
Distinction between transfer and generation ................... 3
  Transfer ....................................................... 3
  Generation ..................................................... 3
  Transfer ....................................................... 3
  Generation ..................................................... 3
No restrictions on domain or semantics ......................... 3

Design of transfer phase ......................................... 4
Distinction between lexical and structural transfer ............ 4
Algorithm description ............................................. 4
Assumptions that base the overall design ....................... 5
Structural transfer ................................................ 5
Lexical transfer .................................................... 5
  The actual implementation .................................. 6
  The existence of several correct translations .............. 6
  Interactive approach .................................... 6
A simple example involving the two transfers ................... 6
Bilingual dictionary .............................................. 6
  Examples of bilingual entries ............................... 7
Tense transfer .................................................... 8
  Algorithm ..................................................... 9
  Example ..................................................... 9

Design of the Generation phase ................................ 10
General description ............................................... 10
  Keeps the overall homogeneity and philosophy, in what concerns: .... 10
  Consists in: ................................................... 10
Algorithm ......................................................... 10
Example ............................................................ 10
Rules ............................................................... 11
  Non-terminal node rules .................................... 11
  Terminal node rules ....................................... 11
  Morphological rules ...................................... 11
Examples .......................................................... 12
Dictionary contents .............................................. 13
Dictionary markers ................................................ 13

Overview of work done ............................................ 14
Structural transfer ................................................ 14
  VPs ............................................................... 14
  NPs ............................................................... 14
  PPs ............................................................... 14
  Tense transfer .............................................. 14
Coordination ................................................................. 15
Lexical transfer ............................................................ 15
Disambiguation by above information .............................. 15
Disambiguation referencing self information ................... 15
Disambiguation referencing information one-level down ...... 15
Generation .................................................................. 15
User interface ............................................................... 16
Numerical data .............................................................. 16

Perspectives and conclusions ........................................ 17
Main problems to solve ................................................ 17
New features to incorporate .......................................... 17
Our possible contribution to MENTOR ............................ 17
Introduction to MENTORP

MENTORP history

The IBM-INESC Scientific group is born the 1st July 1987, after long conversations between managers of both entities. MENTORP is the first project the group is going to work in.

July 1987 Stay in Madrid Scientific Center, where for the first time the two Portuguese "scientists" took contact with MENTOR, more particularly with PEG, CRITIQUE and TT, together with MT general problems.

September 1987 Attendance of IBM NLP conference in Copenhagen by Diana Santos

November 1987 PLNLP/PEG course at Hawthorne, USA, and first meeting of a member of our team with elements from all countries involved in MENTOR.

March 1988 Again a fortnight in Madrid Scientific Center, to discuss and exchange work and opinions

June 1988 Two new researchers, a linguist and a computer scientist, were chosen to work for MENTOR in Lisbon, and will officially begin working in next July.

Departing points

In our view, these were the basic requirements of MENTOR

- It is an unidirectional, English-to-many translation system
- It should use PEG in the analysis phase
- It should be transfer (as opposing to "interlingua" approaches) based
- It should develop a methodology for dealing with several (and quite different) natural languages
- It should develop NLP knowledge inside IBM centers and eventually become a commercial product that performs translation in the field of computer manuals.
- It should be better than the tools and systems already available. In this sense, MENTOR can be considered a research project.
Generic options taken

PLNLP as the only programming language

Reasons for
- PEG was written in it
- Its shell provides an environment specially built to deal with NLP
- PLNLP was written primarily to handle natural language programs
- An unique programming language makes the system more portable and modular, avoiding interface problems
- There is already a PL8 version of PLNLP, that can provide a more efficient implementation if needed in the future

Conclusions
- PLNLP is very powerful in dealing with PEG output
- It is easy to program, and provides a considerably high level of abstraction.
- It presents a large number of mechanisms, like dictionary access, automatic output, and several debugging aids, that make the programming task rather easy
- It is very flexible in control specification, which is a rare and invaluable characteristic in rule-based languages

OD dictionaries

Reasons for their choice
- Automatically integrated in PLNLP environment
- Designed to suit PEG philosophy
- Friendly and efficient
- No problems with interfaces

Conclusions
- Easy to handle and modify
- Can be used for both tasks (transfer and generation)
- However, their choice is probably not important, since what matters is the information they store and not their organization...
**Distinction between transfer and generation**

This was one of the strongest decisions of MENTORP. We divide the tasks to perform after PEG in two big areas:

**Transfer**

Changes the graph representing the English sentence (produced by PEG) into an equivalent Portuguese graph.

Its goal is, in a word, to **produce a graph that could not be recognized as having come from English** (it could as well have come from Hebrew or Finish, or even from the direct application of a Portuguese grammar).

**Generation**

Produces a Portuguese sentence from the contents of the graph created by transfer.

Its goal is, in a word, to **obtain a correct sentence from the basic structure and lemma of the words**

To give a clearer idea of the meaning of this distinction, we present some of the tasks in each phase:

**Transfer**

- Get the Portuguese structures that correspond to the English ones
- Select the words or expressions that correctly translate the English ones
- Compute the Portuguese tenses that correctly express the English ones

**Generation**

- Inflect words
- Produce agreements wherever they are asked by the language
- Produce a string correctly hyphenated, and contracted

**No restrictions on domain or semantics**

Following PEG philosophy, we aim at first at **universal coverage**, which enforces the following methodology:

- Try to solve the most general problems first
- Try to solve everything we can **before** handling particular cases
- Try to come to a definite and accurate statement of the limits of our methods before "patching" some problems that could even not need any "ad-hoc" treatment.
- Try to cover general problems, categories, or classes of words first, and only then handle exceptions.

We are nevertheless aware that to improve the performance of our system we shall have to

- restrict the domain of high-quality coverage (probably to the field of computer manuals)
- solve style and pragmatics problems connected with the kind of discourse of those texts, and not in all discourse areas
- probably restrict Portuguese vocabulary to IBM standards

*Generic options taken*
Design of transfer phase

*Distinction between lexical and structural transfer*

An important option in the transfer design was to distinguish and clearly differentiate two aspects of transfer:

- **Lexical**: the choice among several possible translations of the correct Portuguese word or expression.
- **Structural**: the changes of the sentence’s structure from the English to the Portuguese way.

The reasons for this approach are manyfold:

- Experience suggests a “divide and conquer” strategy to simplify the resolution of a complex problem.
- The actual distinction between the two tasks, both in goals and in required information, makes the idea appealing.
- The search for modularity in the overall design is always an ultimate goal.

This separation of the two tasks does *not* mean, however, that one is performed completely before the other begins. On the contrary, the interleaving of the two phases is, in our opinion, one of the most powerful features of MENTORSS/P.

The separation does *not* mean, either, that one task cannot influence the other: on the contrary, this organization was exactly conceived to allow and ease their interaction.

*Algorithm description*

For each node built by PEG analysis, from the top node to the leafs, in a top-down right to left manner, the following is performed:

1. Each immediately dominated node
   a. Is copied, selectively cleaned from spurious information, and guaranteed that it is the only copy.
   b. Is fulfilled with relevant information for lexical transfer, coming from above.
   c. Is used to access the bilingual dictionary (with its BASE) to choose the right Portuguese word.

2. The type (SEGTYP2) is reevaluated, being transformed in a more general class in several cases.
3. The structural transfer rules are triggered with the node itself.

The first conclusion, is that the lexical transfer is done one-level before (down) the structural transfer, or, in other words, all nodes immediately dominated by the node which suffers structural transfer already carry the target language information.

Also, the new Portuguese graph is being created at the same time that the overall transfer is done.

To notice, too, that each node of the graph is only visited once.
Assumptions that base the overall design

There are some underlying assumptions that it is convenient to make clear

- The structural information relevant to describe a phrase can always be found at the record that describes it, being in their scalar or pointer attributes.
- The higher nodes of the phrase tree represent higher (or the same) level but more complex structures than the ones below.
- The choice of a particular word sense can alter the structure of the phrase of which it is the head, or that immediately above it, and not any that is more than one level higher.

Structural transfer

Looking a little closer to the details of structural transfer, several points should be emphasized

- Selective copy: instead of actually dealing with English nodes, structural transfer handles Portuguese records, without morphology information and with bilingual distinctions already set.

- Phrase reclassification: though the refinement that PEG achieves is of utmost importance in a high-level transfer, there are several basic actions that each superclass (VPs, for instance, irrespective of being relative clauses, or questions or whatever else) must suffer, notwithstanding idiosyncrasies the smaller class they belong may imply...

  For this reason, more general “classes” are considered, being then the old (PEG computed) SEGTYPEs restored.

- Transfer rules are indexed by the SEGTYPE.

- A small set of transfer procedures was developed, which considerably ease the writing of the transfer rules, freeing the programmer of PLNLP implementation details and furnishing a set of basic actions.

Lexical transfer

We consider that lexical transfer can be categorized into the following items, of which only the first four are provided by our system at this moment...

Direct transfer
There is only one translation for the word.

LT based in syntax
The most powerful the syntactic analysis, the better disambiguation it can provide

LT based in universals
Lexical categorization of the kind “animate”, “human”, “time” or “space” is general enough to be considered a kind of universal classification.

LT based in context
Some translations are direct if in a specific restricted domain.

LT of idioms
The meaning of expressions is sometimes different from the conjunction of the meanings of the individual words that take part in it. So, an “expression-to-expression”, as opposed to “word-to-word” translation, must be achieved.

LT based on semantics
Nevertheless, there are some words which only taking into account the social context and the other particular words in the sentence can be correctly translated.

Other issues related to lexical transfer should also be mentioned:

Design of transfer phase
The actual implementation

The information for lexical transfer has a fixed pre-defined format as declarations in the bilingual dictionary. It is the procedure PCHOICE that interprets this information, building the list of the possible translation words, and possibly calling a style module, or the user intervention.

The existence of several correct translations

We build a list of the possible translations, beginning by the most particular, to the most general. It is the job of a style module, to handle conveniently that information in an elaborated way.

Interactive approach

Though not a basic feature, we allow for the time being the interaction with the user on which is the right word to choose, as one option of MENTOR88/P.

A simple example involving the two transfers

I came last summer.

---

TRAD1 DECL2* NP3 PRON2* "eu" VERB2* "vir" NP4 NOUN2* "ver o" AJP2 ADJ2* "passado" PUNC2 "."

---

eu vim ver o passado.

He caught the last train.

---

TRAD1 DECL2* NP3 PRON2* "ele" VERB2* "apanhar" NP4 DET2 ADJ3* "o" AJP2 ADJ4* "ltimo" PUNC2 "."

---

ele apanhou o ltimo comboio.

Dictionary information :

last
passado(ADJ (COND NPIND TME) DEPOIS)
ltimo(ADJ)

Bilingual dictionary

Here we shall only describe the information we store in our bilingual OD, unconcerned with technical organization.

We defined a set of structural markers for the translation from PEG into a Portuguese graph structure, which are not presented as complete or the most adequate, but which we believe will be included in the finite number that will be needed.
At present, we make use of the following features:

PREPO
OMISSO
DEPOIS
CHTOADJ
ADJPREP
REFLEX
SEMCOMP

It is important to notice, that these markers only contain information about the differences between the two languages, and, moreover, about the differences that constitute exceptions to a general difference rule.

On the other hand, the declarations that constitute our "language" for disambiguation only constitute the skeleton of the knowledge to store there, which in turn depends on the particular word. It is, of course, by no means guaranteed that a finite set of disambiguation markers could be found.

So to TRANS ODTABLE we just added two other feature names, COND and NCOND, which test for several kinds of conditions:

- An indicator is on, (or an attribute exists).
- The same, but in a record pointed by some attribute.
- An attribute has a certain value.
- The contrary of each point above.
- The explicit conjunction of any cases above

Disjunction is represented by repetition of the same translation under other conditions.

**Examples of bilingual entries**
BE

haver(VERB (COND self THEREBE))
dever(VERB (COND PREDCOMP INFTO SUBJECT HUM))
estar(VERB (NCOND self PREDNOM self PREDADJ self PREDCOMP) (COND SUBJECT ANIM))
ser(VERB (NCOND self PREDNOM self PREDADJ self PREDCOMP))
ser(VERB (COND self PREDNOM))
ser(VERB (COND PREDADJ EVAL))
estar(VERB)
ser(VERB)

LAST

passado(ADJ (COND NPIND TME) DEPOIS)
passado(ADJ (COND AVPIND TME) DEPOIS)
ultimo(ADJ)
final(ADJ DEPOIS)
demorar(VERB (NCOND self OBJECT))
durar(VERB)

TIME

vez(NOUN (COND PRMODS ORD))
tempo(NOUN)
hora(NOUN)
poca(NOUN)
ocasi o(NOUN)
momento(NOUN)
marcar(VERB)

RETURN

voltar(VERB (NCOND self OBJECT self PASSIVE))
devolver(VERB)
retorno(NOUN)

LOOK

olhar(VERB (PREPO at para))

LISTEN

escutar(VERB (PREPO to OBJECT))

Tense transfer

- Tense transfer is accomplished using mainly (for the moment) the attributes and structural information provided by PEG.

- Each non-terminal node, related to a verb (or verbs), defines its own tense (unless it has already been defined by a higher node) and sometimes defines also the tenses of the nodes just below it.
  - Some nodes are able to define its tense, independently of the nodes around it, i.e., the node itself implies a tense (e.g., IMPR node).
  - Most nodes use an empiric algorithm, described next, to define its tense and possibly the tenses of the nodes below.
Algorithm

The algorithm is applied to a node X. There is a set of 5 conditions that search for the existence or not of certain attributes in X. Each condition, if satisfied, defines a tense.

1. An ordered list is built with the VP and VERB nodes immediately below X.
2. The conditions are tested in order until one holds.
3. The tense implied by the condition is assigned to:
   - the X node if this node has not yet defined a tense.
   - the node in the top of the list except if this is the first node of the whole original list.
4. The node in the top of the list is skipped. If the node was a VERB then continue in step 2 with next condition; else continue in step 3.

Example

Simplified set of conditions:

1 \textbf{INFIN} \implies \text{Infinitivo}
\textbf{FUTURE} \implies \text{Futuro}
\textbf{PRES} \implies \text{Presente}
\textbf{PAST} \implies \text{Pret rito}
\ldots
2 \textbf{AUX1} \implies \text{Infinitivo}
3 \textbf{PERFECT} \implies \text{Partic pio Passado}
4 \textbf{PROG} \implies "a" + \text{Infinitivo}
5 \textbf{PASSIVE} \implies \text{Partic pio Passado inflectido}

the function must have been written and tested

\begin{tabular}{cccccccc}
\textbf{TRAD1} & \textbf{DECL2*} & \textbf{NP2} & \textbf{DET2} & \textbf{ADJ2*} & "o" & \textbf{NOUN2*} & "fun o" \\
\hline
\text{(Condit 1)} & \text{VERB6} & "dever" & \text{must} \\
\text{(Condit 2)} & \text{VERB7} & "ter" & \text{have} \\
\text{(Condit 3)} & \text{VERB8} & "ser" & \text{been} \\
\text{(Condit 5)} & \text{VP3} & \text{VERB9*} & "escrever" & \text{written} \\
& & \text{CONJ2*} & "e" & \text{and} \\
\text{(Condit 5)} & \text{VP4} & \text{VERB10*} & "testar" & \text{tested} \\
& & \text{PUNC2} & "." & \\
\end{tabular}

\implies \text{a fun o deve ter sido escrita e testada} .

\textbf{DECL2 attributes: PRES, AUX1, PERFECT, PASSIVE}
Design of the Generation phase

General description

Keeps the overall homogeneity and philosophy, in what concerns:

- Using the PLNLP language
- Using the same data structure
- Travelling just once through the tree, in an ordered form

Consists in:

- Encoding rules
  - Non-terminal node rules
  - Terminal node rules
  - Morphological rules
- Procedures (for control purposes)
- Small sparse dictionary

Algorithm

- The tree is traversed once, top-down and left-to-right. The traversing order coincides with the "natural" flow of information required for generation (there are exceptions).
- The dictionary access is made one level before the node is actually processed, that is, in each node the dictionary is accessed for the nodes below it (and, sometimes, for some terminal nodes).
- In each node, the generation attributes are passed to the nodes below, unless these already have them.
- For each node, a specific rule, based in its SEGTYPE attribute, is triggered
- The rule triggered performs the actions necessary to process the corresponding node

Example
the rules perform the necessary actions

<table>
<thead>
<tr>
<th>TRAD1</th>
<th>DECL2*</th>
<th>NP3</th>
<th>DET3</th>
<th>ADJ4*</th>
<th>&quot;o&quot;</th>
<th>NOUN3*</th>
<th>&quot;regra&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERB2*</td>
<td>&quot;executa&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP4</td>
<td>DET4</td>
<td>ADJ5*</td>
<td>&quot;o&quot;</td>
<td>NOUN4*</td>
<td>&quot;ac o&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AJP2</td>
<td>ADJ6*</td>
<td>&quot;necess rio&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUNC2</td>
<td>&quot;.&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

===> as regras executa\m as ac es necess rias.

The order by which the nodes are processed is the following:

DECL2
NP3
  DET3
  ADJ4
  NOUN3
VERB2
NP4
  DET4
  ADJ5
  NOUN4
  AJP2
  ADJ6
PUNC2

Rules

Non-terminal node rules
- Act both as filters and as sources of information
- Gather the information needed for the nodes below
- May trigger morphological rules (e.g., gender definition)

Terminal node rules
- Provide the inflection of the words according to the available information
- May trigger morphological rules (e.g., plural generation)

Morphological rules
- Are applied by default, i.e., if the required information is not present at the generation dictionary
- Define morphological features of words (e.g., gender of nouns)
- Inflect the words, concerning the following subjects:
  - Gender
  - Number

Design of the Generation phase
- Person
- Tense
- Reflex particles
- Pronoun case
- Adjective -- Adverb generation

**Examples**

- Illustrating node rules

  In the following examples the PP1 node acts differently in each case, in what concerns gender and number attributes.

  **she likes to be tall**

  \[
  \text{she} \quad \text{likes} \quad \text{to be tall}
  \]

  \[
  \begin{array}{llllllll}
  \text{TRAD1} & \text{DECL2*} & \text{NP2} & \text{PRON2*} & "ela" & \text{she} \\
  & & \text{VERB3*} & "gostar" & \text{likes} \\
  \text{PP1} & \text{PREP1} & "de" & \text{VERB4*} & "ser" & \text{to be} \\
  \text{NP2} & \text{AJP2} & \text{ADJ2*} & "alto" & \text{tall} \\
  \text{PUNC2} & "." & \\
  \end{array}
  \]

  \[\Rightarrow \text{ela gosta de ser alta .}\]

  **she likes to be an enigma**

  \[
  \begin{array}{llllllll}
  \text{TRAD1} & \text{DECL2*} & \text{NP3} & \text{PRON2*} & "ela" & \text{she} \\
  & & \text{VERB3*} & "gostar" & \text{likes} \\
  \text{PP1} & \text{PREP1} & "de" & \text{VERB4*} & "ser" & \text{to be} \\
  \text{NP4} & \text{DET2} & \text{ADJ2*} & "um" & \text{an} \\
  \text{NP4} & \text{NOUN2*} & "enigma" & \text{enigma} \\
  \text{PUNC2} & "." & \\
  \end{array}
  \]

  \[\Rightarrow \text{ela gosta de ser um enigma .}\]

- Illustrating morphological rules

  **I can really give it to you.**

  \[
  \begin{array}{llllllllll}
  \text{TRAD1} & \text{DECL2*} & \text{NP3} & \text{PRON4*} & "eu" & \text{I} \\
  & & \text{VERB3} & "poder" & \text{can} \\
  & & \text{AVP2} & \text{ADV2*} & "real" & \text{really} \\
  & & \text{VERB4*} & "dar" & \text{give} \\
  & & \text{NP4} & \text{PRON5*} & "ele" & \text{it} \\
  & & \text{PP2} & \text{PREP2} & "a" & \text{to} \\
  & & \text{NP6*} & \text{PRON6*} & "tu" & \text{you} \\
  \text{PUNC2} & "." & \\
  \end{array}
  \]

  \[\Rightarrow \text{eu posso realmente dar-o a ti .}\]
Dictionary contents

- The generation dictionary, as all others in MENTOR88/P, is in OD format.
- Holds only the irregular forms:
  - Irregular plural forms
  - Irregular feminine forms
  - Irregular tense inflections
  - Adverbs that do not end in "mente"
- Holds the pronoun forms for its various cases.
- Holds the root verb for the verbs which have one (e.g., the verbs "manter" and "obter" have as root verb "ter").

Dictionary markers

The dictionary markers currently in use are:

- PLURFORM
- FEMFORM
- ACUSFORM
- DATIFORM
- PREPFORM
- GENDER
- ROOTVERB
- COMPFORM

Markers defining tense, person and number (e.g., PRES1S)

Some dictionary entries:

- enigma: enigma(NOUN (GENDER MASC))
- longe: longe(ADV)
- m o: m o(NOUN (PLURFORM m os))
- mal: mal(ADV (COMPFORM pior))
- manter: manter(VERB (ROOTVERB ter))
- mau: mau(ADJ (FEMFORM m ) (COMPFORM pior))
- ter: ter(VERB (PRES1S tenho) (PRES2S tens) (PRES3S tem) (PRES3P t m))
- tu: tu(PRON (ACUSFORM te) (DATIFORM te) (PREPFORM ti))
Overview of work done

*Structural transfer*

**VPs**
- prepositions attached to verbs
- infinitive clauses
- relative clauses
- negation
- change from passive to active
- change position of adverb phrases
- treatment of expletive pronouns ("dummy" IT and THERE)

**NPs**
- insertion of a determiner
- handling of adjectives
- simple handling of noun compounds
- omission of head noun
- conversion into a prepositional phrase

**PPs**
Everything above is automatically dealt for PPs too.

*Tense transfer*
- Simple tense conversion
  - present
  - past
  - future
  - subjunctive
- Compound tenses
  - modals
  - passive
  - perfect
progressive

- Tense transfer
  - infinitive
  - subjunctive

**Coordination**

Was taken care in all above situations

**Lexical transfer**

**Disambiguation by above information**

- *last* in "last summer" uses the value TME of the feature NPIND.
- *any* in "I didn’t see any children" uses the value NEG of the feature VPIND.
- *what* in “What are you seeing?” uses the value QFORM of the feature SENTIND.

**Disambiguation referencing self information**

- Test on the existence of pointer attributes:
  - OBJECT for *last*
  - PREDCOMP for *appear*
- Test of existence of indicators:
  - PASSIVE for *return*
  - COMP for *more*
- Test on specific values of attributes

**Disambiguation referencing information one-level down**

- ANIM in the SUBJECT for *pay*
- PREDADJ in PSMODS for *find*
- LOC in PSMODS for *stand*
- ORD in PRMODS for *time*

**Generation**

1. Verb inflection
   - Inflection of all the transferred tenses (except for the 2nd person plural, seldom used)
   - Gender and number agreement of the main verb in passive and PTPRTCLs
2. Number
   - Default rules for plural generation
   - Number agreement, including coordination and predicative cases
3. Gender

Overview of work done
4. Pronouns
   - Treatment of pronoun cases
   - Handling of reflexion

5. Adjective --> Adverb generation
A very small generation dictionary is used.

**User interface**

Among the several user-friendly features that MENTOR88/P provides

- The access to other alternative parses, through the call to the ALTERNATIVE function, whose argument is the number of the alternative parse.
- The correct treatment of "ungrammatical" sentences, following PEG philosophy.
- The option of interactive disambiguation and/or input by the user of a missing translation.
- The option of automatically access and prepare a LISP file with selected sentences from the main test files (REXPLI and the Haifa set)
- The option of running the whole system in "batch mode", instead of interactively.

**Numerical data**

- Total code line number
  - PLNLP 1622
  - LISP 2981
- Total number of rules
  - Transfer 91
  - Generation 92
- Total number of procedures
  - Transfer 30
  - Generation 7
- Total number of entries in the bilingual dictionary : 16800
- Number of entries with disambiguation features : 24
- Number of entries with structural features : 29
- Total number of entries in the generation dictionary : 112
Perspectives and conclusions

Main problems to solve

At this moment, we feel that the main problems to solve are:

- **MWE (multiword expressions)**: one of the most delicate and difficult problems of MT.
- **Noun Noun constructs**: more than the correct English attachment, there are many issues concerning Portuguese only that must be studied in order to achieve a reasonable translation.
- **Dictionary organization**: the coherence of all features of the bilingual dictionary is a huge lexicographic work that has to be taken care as soon as the general structure is agreed upon.
- **Lexical transfer coverage methodology**
- **Tense transfer**: the same tense in English may have to be translated into different Portuguese tenses
  
  I think that he goes.
  I hope that he goes.

New features to incorporate

The logical continuation of MENTORP would have to perform the following tasks

- String treatment
- Development of structural rules for the untreated domains (like present participle clauses, question tags, either/or...)
- Development of specific lexicographic work related to words whose disambiguation is not trivial.
- Decision on a "micro-lexicon" related to computer manuals using IBM Portuguese terminology.
- Proper nouns handling
- Style rules development

Our possible contribution to MENTOR

- The overall design and framework can be used for several languages, which would only have to develop the set of rules for their language, and fulfill the convenient entries in the dictionary.
- A set of transfer functions can also be provided, allowing a higher-level writing of the transfer rules.
- When the treatment is similar, our rules can be adapted, saving us time and effort
- The results of our system can be used to compare with other approaches, and have a measure of the limitations of both.

All work developed in MENTORP is at the other teams' disposition if and when desired.
GENERATION IN MENTOR88/P

July 1988

Paulo Libano Monteiro

Grupo Científico IBM-INESC
R. Alves Redol, 9 - 4o
1000 LISBOA
Table of Contents

Introduction .................................................................................................................. 1
Overview ..................................................................................................................... 2

General Design ......................................................................................................... 3
Algorithm ................................................................................................................... 3
Rules .......................................................................................................................... 4
   Non-terminal node rules ......................................................................................... 4
   Terminal node rules ............................................................................................... 5
   Morphological rules .............................................................................................. 5
   Dictionary organization ......................................................................................... 5
Adverbial inflection .................................................................................................... 7

Gender ....................................................................................................................... 8
Gender definition ...................................................................................................... 8
Feminine generation .................................................................................................. 8

Number ..................................................................................................................... 9
Number definition .................................................................................................... 9
Plural generation ..................................................................................................... 9

Tense ........................................................................................................................ 10
Tense definition ....................................................................................................... 10
Person definition .................................................................................................... 10
Tense conjugation ................................................................................................... 10

Pronoun case selection ............................................................................................ 11

Contractions ............................................................................................................. 12

Conclusion ............................................................................................................... 13
Introduction

This report describes the generation phase in MENTOR88/P. Generation is the last of the three main phases into which a machine translation system, following the transfer approach, may be divided. The first two phases, analysis and transfer, are out of the scope of this text, although it is not always possible to define a sharp frontier between transfer and generation.

Generation is responsible for the synthesis of the target language, Portuguese in this case. Its input is a graph representing the structure of the Portuguese sentence, and containing relevant information for generation. The output is a string that is (or should be) the Portuguese translation of the original English sentence which served as input to the analysis phase.

Emphasis will be put mainly in practical issues of generation rather than theoretical ones. In fact, this report constitutes essentially a description of the work done up to date on generation by the Portuguese team of MENTOR. The subjects considered are, obviously, closely related to the design of the current implementation. They reflect the options that have been being taken in an ongoing, necessarily limited work.
Overview

In MENTOR88/P an effort is being made in order to keep the homogeneity of the translation system. This seems important to obtain, with simplicity and clearness, the compatibility among the tree phases and to improve the efficiency of the system. On the other hand, it allows for a greater concentration of efforts in the resolution of specific problems of translation, not wasting much time in defining interfaces between the phases.

Homogeneity concerns the programming language, the data structure and even the philosophy inherent to the algorithms of transfer and generation.

According to this general idea, MENTOR88/P has been following an all-PLNLP approach. The record-based graph produced by PEG is used as the basic data structure. The same dictionary format is shared by the dictionaries of all the phases, although this is not a very important feature.

The target of generation is to produce a correct Portuguese sentence after the graph received from the transfer phase. This graph defines the syntactical structure of the sentence. Each node represents a constituent and contains information that may be used in generation (e.g., number, tense). The specific tasks of generation consist then, essentially, in making the necessary gender, number and person agreements, choosing the right cases for pronouns, inflecting the words, contracting and hyphenating the final Portuguese sentence. This work is accomplished mainly through the use of a set of rules that apply by default. Exceptions are handled with the aid of a small dictionary that contains only the words which are irregular in some morphological issue (e.g., the adjective "qualquer" constitutes a dictionary entry where its irregular plural form is stated).

Generation in MENTOR88/P may be conceptually (but, in the present implementation, not temporally) divided into two parts:

1. Defining and passing information from the non-terminal nodes down to the terminal nodes. This action must account for the needs of each terminal node, in that these ones have to receive all the necessary attributes to correctly inflect the words associated to them. So, the proper attributes must be defined and passed from node to node until the leaf is reached. The order by which this is done, and the way of doing it will be described in detail afterwards.

   There is an algorithm that defines how to traverse the graph (actually a tree, for traversing purposes); a specific rule is triggered for each non-terminal node, allowing for the control of information flow through the tree.

2. Inflecting, contracting and hyphenating the words, at the terminal node level.

   Once again, rules are used in order to perform these actions; the information received from upper nodes determines how each word, and the final sentence, will be generated.

It should be mentioned that no structural changes are carried out at this phase, which means that the Portuguese words are definitely ordered in the graph received from transfer.

In the remaining of this report, the graph structure that constitutes the basic data on which generation - and the whole system - works, will be often referred as tree. This stems from the fact that the graph is traversed by using the PRMODS, HEAD and PSMODS attributes of each node; these attributes are related to the (more limited) tree form of the graph. Nevertheless, other pointer attributes (e.g., SUBJECT) are also used, but not for traversing purposes.
General Design

An important feature of generation in MENTOR88/P is the order by which the nodes of the tree are processed. This seems important to achieve an organized an efficient system.

- Organized, because there is a well defined order for processing the nodes and this is good practice in what concerns software development and updating. It would be certainly better to process nodes independently of their order, but this is not possible (e.g., before inflecting an adjective, it is necessary to know the gender and number of the noun it modifies).

- Efficient, because each node is processed just once.

The processing of a node depends whether it is a terminal node or not, and consists essentially in:

- **for non-terminal nodes**, defining all the information that will be needed by the nodes below. This is done either passively, by inheriting attributes from upper nodes, or actively, by assigning new values to the attributes.

- **for terminal nodes**, inflecting the words according to the information available, and perform contraction and hyphenation where needed.

Algorithm

Generation is guided by an algorithm that goes over the tree and triggers a rule for each node. It is also responsible for passing down the attributes from one node to the nodes below it, and by accessing the dictionary for each node. The algorithm is implemented in a recursive procedure. A detailed description follows:

- The tree is traversed once, top-down and left-to-right, using the PRMODS, HEAD and PSMODS pointer attributes. The traversing order coincides with the "natural" flow of information required for generation. Although there are exceptions (e.g., subject - predicative adjective agreement), it seems that most information needed by a leaf is present, or may be easily obtained, in the nodes above it; thus, being one node processed before its PRMODS, HEAD and PSMODS, it is just a matter of these inheriting the relevant attributes from their parent.

- The dictionary access is made, for a certain node, just before its parent is processed, that is, the access to the dictionary is advanced one level over processing. The reason for this is that a node, to be processed, often needs to recur to (dictionary) information present in the nodes immediately below it. Once again, the subject - predicative adjective agreement is a good example; in this case, the node above both the subject and the adjective, for instance a DECL node, must get the gender from the subject node in order to pass it down to the adjective. Sometimes the dictionary is accessed for terminal nodes more than one level before they are processed. This happens in the case of conjoined noun phrases, where, to determine the gender of the main NP node, it is necessary to define the gender of all the conjoined nouns (this is achieved with the aid of the HEADS attribute).

Anyway, the dictionary may be accessed for any node at any time. Naturally, the actual access to the dictionary files is done only once per word, and the gathered information becomes available for later use.

- In each node, the relevant attributes are passed down to the nodes just below, unless these already have them. The attributes considered are:
  - PTENS for tense feature
  - GPERS for person
  - GGEND for gender
  - GNUMB for number
GSYNFUNC for syntactical function

For each node, a specific rule, based in its SEGTYPE attribute, is triggered. This rule will carry on the actions necessary to process the corresponding node.

The following example illustrates the traversing order of the tree:

```
this is an example

TRAD1 DECL2* NP3 PRON2* "isto"
VERB2* "ser"
NP4 DET2 ADJ2* "um"
NOUN2* "exemplo"
PUNC2 "."

====> isto um exemplo .
```

The order by which the nodes are processed is the following:

```
DECL2
NP3
PRON2
VERB2
NP4
DET2
ADJ2
NOUN2
PUNC2
```

Rules

Encoding PLNLP rules are used extensively in the generation phase. They can be divided into three main groups: non-terminal node rules, terminal node rules and morphological rules.

All these rules share the same format: one record on the left hand side and one record on the right hand side. Although PLNLP encoding rules allow for several records on the left hand side, this possibility is not actually used. In fact, considering that the structure of the Portuguese sentence is thoroughly and explicitly defined after the transfer phase (in the PRMODS, HEAD and PSMODS attributes of each node), there is no need to use the encoding facility of the rules. Indeed, it would be much more difficult and inefficient to perform generation using true (in the sense of profiting from all its power) encoding rules. Nevertheless, the use of rules seems to be adequate to the generation task, because they implicitly provide a simple and high level way of stating conditions. Moreover, the software so developed is easily modifiable, because one may change, add or delete rules as needed, and each rule is bound to a specific task.

There is one rule associated with each type of node (terminal or non-terminal). The name of these rules is obtained from the concatenation of the letter "G" with the name (SEGTYPE) of the respective node. So, the rule GNP is associated with the NP nodes, and the rule GNOUN is associated with the NOUN nodes.

The name of morphological rules also begins with a "G", and the rest of the name identifies the rule's purpose. For example, GDEFGEM is a rule that defines the gender of a word ending in "EM", GPLURCIL is a rule that determines the plural of a word ending in "CIL" and GPRESAR is a rule that conjugates the "presente do indicativo" of a verb ending in "AR".

Non-terminal node rules

Non-terminal nodes must gather all the information that the lower nodes will need. Throughout this text, the word "information" simply means relevant attributes for generation. The overall idea is that there must be an information flow downwards. This does not disable a node to pick up information from some node below, that is, a node reachable through a descending path.

There are three ways of a node getting its attributes properly defined:
• Inheriting them from the parent node (in fact, as was already mentioned, one of the steps of the generation algorithm is to pass attributes from one node to all nodes immediately below).

• (Re)defining them, using the information present in the node itself. This information may either be the result of analysis and transfer phases or may have come from the generation dictionary.

• (Re)defining them, picking up information from some node below. As in the previous case, the information considered may proceed from analysis, transfer or the generation dictionary. Examples of information used from these sources are the number, tense and irregular plurals, respectively.

The rules associated with the non-terminal nodes provide a way of controlling the flow of information through the tree. So, depending on the SEGTYPE of a node and, possibly, on some other conditions, the triggered rule acts as a source and/or filter of information to the lower nodes.

Terminal node rules

Terminal node rules provide for the inflection of the words they represent, according to the information available.

The inflection may be a plural or feminine generation, a tense conjugation, an adverbial suffix addition or a pronoun case selection.

In order to actually inflect the words, morphological rules are triggered, depending on the specific morphological features required. All terminal nodes, after being processed by the correspondent terminal node rule, actuate a last rule, called GENDNODE, which is described next.

Morphological rules

Morphological rules may be divided into two groups:

• rules that identify the value of some morphological attribute. At present, the only set of rules in this group is the one used for the definition of the gender of NOUNs and PRONs.

• rules that actually inflect the word on some morphological issue (e.g., plural).

In this last group may also be included the GENDNODE rule. This rule, applied ultimately to all terminal nodes, builds the target sentence and makes the necessary contractions. Each time this rule is triggered, by a terminal node, the word (or expression) of the previous terminal node is either concatenated to the global variable !NLP-GENERSTR (progressively building, this way, the final string), or contracted with the word of the current node. In this last case, the resulting contracted word (or expression) will be appended to !NLP-GENERSTR when the next terminal node is processed by GENDNODE.

Each set of morphological rules is controlled by a procedure. This procedure determines which rules (in the set) and by which order will apply. Sometimes, when the evaluation of some morphological attribute is quite simple, the procedure itself is enough. This happens in the definition of person and number, where the need to use rules does not arise.

Dictionary organization

The generation dictionary is in Optimized Dictionary (OD) format. It contains only the irregular forms and some restrict features of Portuguese words. Regular morphology is carried out by default rules concerning each morphological aspect.

The dictionary markers currently in use are:

• PLURFORM plural of irregular NOUNs and ADJs

• FEMFORM feminine form of irregular ADJs

• ACUSFORM direct object case of PRONs

• DQATFORM indirect object case of PRONs

• PREPFORM case of PRONs preceded by PREP
- REFLFORM reflex PRONs
- GENDER gender - MASC or FEM
- ROOTVERB root verb (e.g., the root verb of "manter" is "ter")
- COMPFORM comparative and superlative form of ADJs and ADVs
- CONTFORM contraction form of PREPs (e.g., "por" -> "pel")
- CONTPREP list of PREPs with which contraction is allowed
- There is also a set of markers concerning tense + person + number (e.g., PRESIS is the "presente do indicativo", first person, singular)
Adverbial inflection

The adverbial inflection consists in the addition of the suffix "mente" to an adjective in order to get the correspondent adverb. Adverbs that do not obey this rule must appear in the generation dictionary.

The ADVs and adverb related nodes in the graph received from the transfer phase contain, if possible, an adjective as the value of their BASE attribute. So, if the BASE of one of those nodes does not appear in the dictionary, it will be treated as an adjective; that is, rules will be applied to change the adjective into an adverb.

The set of rules that perform adverbial inflection is controlled by the padvflex procedure. The name of the rules is obtained by concatenating "GADVFX" with the last 1 and 0 letters of the adjective.
Gender

Gender definition

Gender definition is needed by all noun and pronoun related nodes. It consists in the assignment of the FEM or MASC value to the GGEND attribute of the node being processed.

There is a set of rules to determine the gender. The name of these rules is formed by the string "GDEFG" plus the last 3 to 0 letters of the word (noun or pronoun). The procedure pdefgend controls the triggering of the relevant rules for each case. Irregular words must have an entry in the generation dictionary, where the GENDER attribute contains the correct FEM or MASC value.

Gender definition is correctly handled in the case of conjoined noun phrases.

Feminine generation

Feminine generation is needed by adjectives that agree with feminine nouns or pronouns.

There is a set of rules that modifies the STR attribute of the node being processed, changing its value from a masculine to a feminine adjective. The name of these rules is formed by appending the last 3 to 0 letters of the (masculine) adjective to the string "GGEND". The rules that must apply and the order by which they apply are controlled by the pmakfem procedure.

Adjectives whose feminine form is irregular have to appear in the generation dictionary: the FEMFORM attribute must hold the irregular feminine form of the adjective.
Number

Number definition

The number is defined through the procedure `plefnumb`, which assigns the value SING or PLUR to the GNUMB attribute of the node being processed.

The number definition is based, essentially, in the SING and PLUR indicators provided by the analysis phase.

Plural generation

Plural generation is required for adjectives and nouns. There is a set of rules that handle the regular plural forms, changing the STR attribute from a singular to a plural word. The name of these rules is obtained by concatenating "GPLUR" with the last 3 to 0 letters of the word. The set of rules is controlled by the `pmakplur` procedure.

Words whose plural form is irregular must appear in the generation dictionary, with PLURFORM attribute set to the plural form of the word.
Tense

Tense definition

The tense is defined in the transfer phase. Each VERB node has (or will have, after attributes reach the terminal nodes) a PTENS attribute that defines a tense feature. At present, the tense feature may be one of the following:

- PRES presente do indicativo
- IMPF pretérito imperfeito do indicativo
- PRPF pretérito perfeito do indicativo
- FUTU futuro do indicativo
- PREC presente do conjuntivo
- IMPC pretérito imperfeito do conjuntivo
- FUTC futuro do conjuntivo
- CNDI condicional
- IMPR imperativo
- INFI infinitivo impersonal
- INFN infinitivo pessoal
- PTPD participio passado
- PTPDFLEX participio passado flexionado em gênero e número

Person definition

The person is defined by the procedure pdefpers. One of the values PERS1, PERS2, PERS3 is assigned to the GPERS attribute, according to the indicators PERS1, PERS2 and PERS3 provided by the analysis phase.

Tense conjugation

For the conjugation of a verb, the values of the PTENS, GPERS and GNUMB attributes are used (if PTENS defines an impersonal tense then GPERS and GNUMB are, obviously, not needed).

There are four rules for each tense, mapping with the four possible endings of the infinitive form of verbs in Portuguese: "AR", "ER", "IR" and "OR". These two letter endings correspond to the last two letters of the name of the rule. The first letter is always a "G", and the following four letters are provided by the PTENS attribute, identifying the specific tense. The triggering of the correct rule is regulated by the pconjug procedure.

Irregular tenses have to appear in the generation dictionary.
Pronoun case selection

In the graph received from transfer, all personal pronoun related nodes have, as value of their BASE attribute, the subject form of the pronoun.

Generation is responsible for choosing the right form of the pronoun. This selection is made by the GPRON rule (a terminal node rule), and is based in the GSYNFUNC and AFTPREP attributes. The latter is set to 1 in nodes that are preceded by a preposition. The former holds one of the following values:

- SUBJECT for subject pronouns
- OBJECT for direct object pronouns
- INDOBJ for indirect object pronouns
- REFPRON for reflexive pronouns

The various cases of personal pronouns must be defined in the generation dictionary, according to the following markers:

- ACUSFORM for direct object pronouns
- DATIFORM for indirect object pronouns
- PREPFORM for non-subject pronouns, following prepositions
- REFLFORM for reflexive pronouns
Contractions

Contractions are handled by the GENDNODE rule. This rule maintains a list of the terminal nodes already processed, performing contractions, if necessary, between the last node of that list and the node being processed.

At present, contractions are allowed only between PREPs and another node. The word that this last node represents must have an entry in the generation dictionary, where the CONTPREP attribute contains a list of the prepositions with which contraction is performed. The contraction is obtained either directly from the dictionary as the BASE of the entry “preposition&word” (e.g., com&mim:comigo(PRON)), or concatenating the value of CONTFORM (available in PREP) with the word (e.g., “pel” + “a” --> “pela”). “Crasc” is handled automatically (e.g., “a” + “a” --> “â”).
Conclusion

The development reached so far in the generation phase, is fairly reasonable in that the Portuguese sentences are satisfactorily generated. Anyway, the work to be done is by no way lesser than the work done. There is a need to generally enhance generation, upgrading the performance of its several tasks.

Specifically, it is necessary to improve contractions, mainly when pronouns are involved. The problem of hyphenation must be handled. Tense conjugation has to be enlarged to new tenses. Object pronouns close to a verb whose tense is the future or conditional, must be correctly placed. Style rules may be added (e.g., omission of subject pronouns, use of 3rd person instead of the 2nd). And, finally, the generation dictionary should be enhanced, in what concerns its size and its organization.
Machine Translation Prototyping

by Diana Santos, IBM-INESC Scientific Group, Portugal
Luis Sopena, Madrid Scientific Center, Spain
Mori Rimon, Haifa Scientific Center, Israel

Abstract

The suggested paper will describe the "Multitarget English Translator" project, a joint effort by groups in several European IBM Scientific Centers, collaborating on research in Machine Translation (MT) and Machine Assisted Translation (MAT). Currently the work focuses on English as a source language, and Spanish, Portuguese, Hebrew and Finnish as target languages. The project began in 1987, with a five-year plan, of which the first phase was completed in June 88. The major outcome of this phase are preliminary prototypes, capable of producing output at a reasonable level of understandability, within certain lexical and syntactic bounds.

Following decades of research worldwide, which included some too ambitious and thus disappointing attempts to automatically translate unlimited text, M(A)T is now considered feasible and several systems have already reached the commercial stage (see &lbrk.5&rbrk. for a survey of MT history). On one hand, the technology has matured and significant work has been done on natural language grammars, knowledge representation, formal semantics, etc.; at the same time, developments in computer technology have facilitated large scale computing, which is required for this application. On the other hand, the limitations of M(A)T are better understood now, and compromises are made as to the level of automation, the text domain, and the output quality.

Our translation prototypes follow the general principles of the Transfer approach to Machine Translation. In this approach, the translation process consists of three major phases, trying to achieve independence between source language and target language processing:

1. Source Language Analysis: Analysis of the English text in all three prototypes is based on Yorktown's PLNLP/PEG package &lbrk.3&rbrk. which is now in use by several internal IBM projects. PLNLP is a general purpose language and parser, and PEG (PLNLP English Grammar), a wide coverage grammar of English relying on a comprehensive lexicon, the Optimized Dictionary (OD). PEG parses the input text one sentence at a time, and provides for each one a "flattened" tree, and a record structure of attribute-value pairs describing the characteristics of each tree node.

2. Bilingual Transfer: The mission of the Transfer phase is to restructure the parse according to the requirements of each target language and to substitute words and phrases by their equivalents in the target language. All bilingual information is therefore concentrated at this stage. Two kinds of processing are mainly dealt with at this stage:

   • "Structural transfer", where the English syntactic structures are mapped into their target language equivalents, by means of a set of transformation rules defined for all language pairs. In the implementation, different approaches were taken: For Spanish and Hebrew, transformation rules are coded using LISP functions of the Tree Transducer package of Tokyo Research Lab.&lbrk.4&rbrk.. In the Portuguese prototype, all the work is done at the PLNLP level.

---

1 The authors represent larger groups in the three centers, collaborating on this effort. Credit should equally go to D. Ben-Ari, E. Bentur, I. Golan, A. Kolsky, S. Lappin, P. Monteiro, T. Redondo, and I. Zapata.
• "Lexical transfer", where the translation for each English word is chosen among several possible senses (target words), by means of rules included in the bilingual lexicon entries. These rules refer to the syntactic/semantic environment and to grammatical relations between words in the parse tree. This is done by different methods in the different prototypes: In the Hebrew one, a concept of an "active bilingual lexicon" was developed; differentiation rules are specified in a special language and are directly executable in a LISP environment. While in that prototype lexical and structural transfer are kept strictly separated, in the Portuguese one care has been taken in the development of a methodology that interleaves lexical and structural transfer; the choice of a particular target word is executed through the interpretation of a set of declarations (stating positive and/or negative conditions) in the bilingual dictionary. In the Spanish prototype, a set of primitives including conditions for meaning selection and actions associated with selected translations was defined; definition of entries is done through an interactive interface, which provides a mapping between the declarative description of the entry, as it is input by the user, and the procedural interpretation to be used by the system.

3. Target Language Generation: The Generation phase finally constructs the target language sentence, by traversing the structure created earlier in the Transfer phase, inflecting words according to information contained in the tree and target language morphological rules, adjusting gender, number and voice, attaching prepositions, etc. Each language in the set of our target languages poses different challenges for Generation. In Spanish, in addition to the surface structure reorganization (which includes such problematic items as clitic pronouns), word inflection is carried out through an interface with the LEXIS dictionary, which allows for morphological analysis and synthesis. The Portuguese prototype uses PLNLP rules for morphology in addition to a sparse dictionary containing exceptions. To produce agreements it traverses the Portuguese graph, which is the Transfer's output, in an ordered way. A novel approach to Hebrew morphological synthesis was developed, based on the "full script" convention. It is basically rule-based, with minimal marking of exceptions in the dictionary.

The translation process is automatic for the most part, with some provisions for human intervention in the selection of target language word equivalents. Work started on interactive disambiguation by rephrasing (see \&lbkb.1\&rbkb.).

The project focuses on a specific, yet linguistically general enough, domain of source texts - the domain of software manuals. The restriction to a given domain mainly applies to the vocabulary and to some extent to the level of sense distinction, but it does not limit the linguistic value of the work, as no domain-dependent assumptions are made in the general structure or methodology. In particular, no domain specific limits are assumed in the syntactic coverage of the system.

The full version of the paper will elaborate on selected technical topics, emphasizing novel approaches taken to different problems in the different prototypes.

It will also include an appendix with Sample Translations. (In fact, such an appendix - including a few sets of source sentences with the machine translations to the three languages - is available as part of this abstract but was not included in this file, as it will not print properly on a standard 38xx printer because of different font requirements. It can be provided in a hard copy.)

References


7. --- a few, selected, external pubs ?? ---
An MT prototype from English to Portuguese

Diana Santos

IBM Portugal
Praça de Alvalade, 7
1799 Lisboa Codex

PODIANA at LISHQ

Abstract

The goal of this text is to describe the machine translation work at the IBM-INESC Scientific Group, Lisbon, in the framework of Mentor.¹

MENTOR88/P is the resulting prototype of one-year work on machine translation from English to Portuguese. The first goals were to develop a convenient architecture, and fulfill it with knowledge from all final system's properties.

Its results seem to show that the architecture and the kind of knowledge for the task are suited to, in the near future, produce an effective automated translation from English technical documents into Portuguese.

Introduction

In short, what we shall present is our choice of an architecture for machine translation, emphasizing some points that seem to us more interesting, and referencing shortly the work already done.

In this paper we look at the system from three different viewpoints:

• the design of the system as a whole, describing its more striking or characteristic features

• the use made of it for machine translation proper, or which translation knowledge is stored in the prototype

• some practical performance measure of the system's today form

At last, we conclude with some ideas about future work, where we claim that the architecture and the current implementation can be enhanced powerfully.

General description

As most general traits, the system can be considered as transfer-based (as opposed to an interlingua approach), unidirectional, and with an overall architecture that should be independent of the target language.

We proceed by emphasizing some of its relevant features.

¹ This paper describes the work done in collaboration with Paulo Libano Monteiro, at the IBM-INESC Scientific Group.

² MENTOR is not a product name and is only used as an internal project name in IBM.
PEG-based

- As a most important feature, MENTORP uses PEG [4] for the analysis of the English text. So, being the graph produced by PEG the departing point of our system, one of the general options was to follow its philosophy, namely both in the way and type of linguistic knowledge being used and in the programming language chosen, PLNLP [3].

In fact, although there are several current translation projects [7] [10] using PEG as the analysis phase, we think that we are unique in trying to use all information inside PEG. As far as we know, other approaches have mainly used PEG’s output trees, or a selected subset of PEG’s internal information as the starting point to a new representation.

On the contrary, we use the whole PEG graph as the first intermediate structure for the transfer model. We believe that there are advantages in this choice, namely that most information that was necessary to perform analysis is also very useful to deeply characterize the English sentence, and consequently to achieve a better translation of it.3

Transfer model

- We imposed complete separation between the three phases of the system, namely analysis, transfer, and generation. Particularly, this means that transfer should create an internal representation for the Portuguese structure, analogue to the one created by PEG.

This is a direct consequence of a transfer architecture, in the sense that an intermediate representation in the target language should be defined, and moreover, we believe it should be at the same (representational) level of the one for the input sentence.

In MENTOR88/P, the transfer input is the PEG graph, only devoid of information concerning English only (morphology) or historical information (useful for debugging of the parsing process). Therefore, at the beginning of generation too, there must be a graph structure for Portuguese bearing the same kind of features and information. This means that uniquely information pertaining to Portuguese only (like morphology and surface forms) should be missing.

On the other hand, this second intermediate structure should have no trace of having come from English or whatever language it was transferred from.

Transfer architecture

One of the points we believe most interesting in the system being described, is

- Clear distinction between the two aspects of transfer: structural and lexical, since they access different information and produce different results. It is however important to emphasize that while they are conceptually separated, the design was conceived to allow for their interaction, interleaving the triggering of the two tasks in a certain order.

Their definition in our system is the following:

lexical the choice among several possible translations of the correct Portuguese word or expression

structural the changes of the sentence’s structure from the English to the Portuguese way

On one hand, modularity is achieved, since the two tasks are conceived and implemented separately.

On the other hand, they are allowed to influence each other in several ways: different Portuguese structures can arise because of the selection of different translations of a word, while, on the other hand, the structural environment can (and most frequently does) influence the lexical choice.

The algorithm

This process is better explained by the global algorithm of the transfer phase, which, in a top-down right-to-left manner, performs the following for each node:

3 Namely, we are referring to subject/object pointers, “hidden” nodes that convey important structural information, like the PPOBJ (object of a preposition), conjunction-related attributes, and several scalar data such as lexical or sentential subcategorization, that cannot obviously be displayed in tree form.
1. Each immediately dominated node
   a. is copied, selectively cleaned from spurious information, and guaranteed that it is the only copy.
   b. is fulfilled with relevant information for lexical transfer, coming from above.
   c. is used to access the bilingual dictionary (with its BASE) to choose the right Portuguese word.
2. The node type is reevaluated, being transformed in a more general class in several cases.
3. The structural transfer rules are triggered by the current node.

As can be seen from above, the graph is only traversed once (and in an ordered way), while we assume lexical transfer can at most influence structural transfer one level above, and not higher levels.

**Structural transfer implementation**

- To allow for this, the statement of structural rules obeys the following restriction: transformations can only concern the nodes immediately below.

This is not, however, a constraint on the linguistic rules that can be expressed in the system (which can account for changes of arbitrary depth), but only on the programming style of these rules.

- To make the writing of the structural rules easier and more high-level, a set of primitive transfer functions was implemented, that take care of uninteresting details and constitute the basis of the rules actually written [9].

Some examples of structural rules are presented in appendix.

**Lexical transfer implementation**

Lexical transfer is performed by a simple procedure connected to the bilingual dictionary access, that interprets the declaration stated in the dictionary entry. This procedure treats the several candidates assembled in the dictionary as follows:

1. Matches part of speech
2. Tests positive conditions, in the order they are stated
3. The same for negative conditions
4. Returns the list of the candidates that can be translations
5. In case the system is working in "NLP-INTERACTION" mode, and the size of the former list is different from one, it interacts with the user

We are convinced, however, that a much higher-level language for specifying lexical transfer is convenient, and therefore we only present this (and some bilingual entries in appendix) so that the complete prototype at this stage can be described and fully understood.

**Generation model**

Generation has to produce a correct Portuguese sentence after the graph received from the transfer phase. This graph defines the syntactical structure of the sentence. The specific tasks of generation may be divided, in the actual implementation, into two groups:

1. Process the existing information in order to get the necessary number, gender and person agreements, and the right cases for pronouns.
2. Inflect the words, contracting and hyphenating the final string.

**Generation architecture**

In our system, the generation architecture is very similar to that of transfer, as we are engaged in creating an homogeneous system, from every point of view. We use thus for generation the same
data structure, the same general philosophy, and the same programming tools (language and dictionary system).

We traverse the graph again only once from top to down, and access the dictionary one level down from the current node.

However, and while we distinguish also two separate tasks, they do not interact in both directions, as in transfer, since the second comes logically after the first, and is moreover only performed for the terminal nodes (the leaves of the graph).

Concerning the programming style of generation rules, it is the same as for transfer: changes are propagated one level at a time.

Declarativeness

1. Structural differences between the two languages are stated in rule form,

2. The information necessary to select the right word (lexical transfer) is stored in the bilingual dictionary as a set of positive and/or negative conditions,

3. Portuguese morphology is also stated in several sets of rules.

We believe that a declarative knowledge representation makes both the statement of the knowledge easier to human eyes and more independent of how to actually implement it. In this way it is also simpler to abstract the information stored in our system (and consequently use it for other purposes) than whether it were “hardwired” in control flow.

Dictionaries

- Dictionary information (except for lexical transfer) is reduced to the storing of exceptions: in the bilingual dictionary, exceptions to the general structural transfer rules; in the target language dictionary, morphological irregularities.

It should be acknowledged at this point that the transfer lexicon data that we are using in our system was derived from the ITS Portuguese transfer dictionary developed at Brigham Young University in Provo, Utah [8], using for the moment only its part-of-speech information. Other features and abilities of our own design were only incorporated in a very small scale.

However, we want to mention the methodology chosen (for both bilingual and generation dictionaries), which consists of coding the general rule in rule form, and only signal in the dictionary when some specific words imply a different processing.

Description of the knowledge stored in the system

Instead of describing particular pieces of knowledge implemented, we prefer to focus on a more general way of describing the knowledge, referencing the interested reader to other more specific texts [9] [6].

We shall go through several components of our system, namely structural transfer, lexical transfer, tense transfer, agreement generation, and surface generation.

Some justification on linguistic grounds for the options described in the previous section on the system’s design will also be tried.

Structural transfer

Since the transfer design gives structure a prominent role, as can be seen by its imposing the flow of the whole translation process and, more specifically, the order by which lexical transfer is performed, some linguistically motivated explanation should be attempted.

Justification

Apart from the computational adequacy there seem to be some reasons to support this approach to transfer. First of all, the fact that an organized and finite classification as syntactical structure may govern instead of the unpredictable and probably uncoverable universe of lexical transfer,
makes the system design more reliable. The pillars of the system are defined on solid and well-known bases: the set of different structures handled and represented by the parser.

But the theory that presided at the creation of the structure itself implies some assumptions that support, in our view, the path followed. In fact, the representation PEG creates is not isomorphic to the one stated in its rules [5]. Therefore, the different levels of its graph hold a much greater significance than if they reflected simply the order of application or form of the rules in the grammar.

We can thus postulate, that there is a meaning associated to the structure of the graph itself, and it is by taking it into account that the design is justified. We assume that, at any particular node in PEG’s graph, there is enough information to characterize the phrase it represents. Therefore, rules centered at the nodes deal with language itself and not with the way analysis was led; one node and not a cluster of nodes represents a given structure. These facts motivate the way we state transfer.

In short, we hope to have made clear, in the above lines that the algorithm used is not only a computational device, but it is justified by the way PEG looks at language and therefore arrives at its representation.

Some of the (meta)rules implemented

These are general patterns followed in the system

- Phrase reclassification: we defined a supertype for the current PEG classification of nodes (expressed in their attribute SEGTYEP2) to handle general translation rules, refining afterwards the classification (through restoring of the original values). This last action enables us to handle then the specific translation problems connected to each type computed by PEG.

- Handling of “hidden pointer attributes” [9]: we handle the object of the prepositional phrase first, and then apply the rules to the PP as a whole. This allows for the treatment of the body of a PP in the same way as any other phrase.

- Treatment of constructs of arbitrary length: the way rules must be stated (either changing the current node or signalling down the change) allows for an elegant (recursive) treatment of language, with no constraints on relative closeness of structures.

Lexical transfer

Identification of problems

In general, the choice of a corresponding term in the target language can be based on syntax, on universal “semantic” markers, on semantics, pragmatics or even world knowledge. It can also be approached in a restricted domain context, or treat explicitly (and separately) idioms.

For the moment we have only concentrated on syntax and universals (that is, those characteristics, like animate, time or space, that are found in every human language).

The reason for choosing lexical transfer based on syntax is obvious, taking into account that PEG provides such a detailed description of the syntactical structure of the sentences it analyses, and that we are determined to do the best use of PEG.

However, there is a more general reason for choosing these two criteria as departing points and also as a research subject: the fact that we are aiming at broad coverage. In this situation, we try to develop a processing that may be as general and broadly applicable as possible.

Identifying of a few general patterns used in lexical transfer

From the small experimenting conducted with lexical transfer in the building of this prototype⁴, we were able to identify some patterns:

- Transitive/intransitive use of a verb

⁴ For this matter we are strongly indebted to some very interesting discussions with Shalom Lappin of Haifa Scientific Center, and in general to the ideas presented by the MENTORH team concerning lexical transfer as a whole [2].
While in English the existence or not of the (deep) direct object conveys some meaning, in Portuguese that distinction is vehicle by different lexical realizations. So, the determination of grammatical roles seems to play an important part for lexical transfer (for instance in "he returned / he returned the book"\(^5\), or "he quit / he quit the job"\(^6\)).

- **Passive/active context**
  
  On the other hand, the same verb used in passive or active form is sometimes translated into different verbs in Portuguese, which seems to indicate that the surface syntax must also be taken into account. (This is also related to the point above and/or to the existence of an animated subject or actor, but it definitely shows this way (for instance in "I intend / the program is intended"\(^7\), or "the problem shows / the problem was shown"\(^8\), or even, "the program runs / the program is run"\(^9\)).

- **Human or animated subject (normally related to other conditions as well)**

  This feature was found to be one of the most used in the statement of the conditions for lexical transfer (for instance in "he works / the program works"\(^10\), "he pays / crime pays"\(^11\)).

- **Existence of a particular preposition connected to a verb**

  Even the actual (string) preposition that is associated with a verb must be considered for correct translation of the verb (see "look at/look for/look after/look like"\(^12\)).

We must acknowledge that we are in what concerns lexical transfer in a very early stage, so that we can only sketch some of the problems and solutions we tried, without no conclusive data to back them.

**Several correct translations**

An important point concerning lexical transfer in our system is that one translation is not chosen, but the set of all possible ones in a given situation (see 4. on page 3).

This means that, although we currently use the first one, other more complex mechanisms can be incorporated later, for instance to avoid unnecessary word repetitions, or to test some stylistic criteria in word choice.

Also, it stands in the line of declarativeness and modularity that reigns in the whole system, since an entry associated with one particular Portuguese translation is independent of the others, having its own conditions for selection and its own structural features.

**Tense transfer**

This is a very important issue in translating from English into Portuguese, since though some syntactic devices are similar (like the use of the auxiliaries "be" for passive or "have" for past perfect), the Portuguese language presents a much broader tense choice, and specially there is not a direct mapping in many cases.

This is one of the most important issues for translation, and thus a tense transfer module (also written in PLNLP rules) is incorporated in MENTOR88/P.

Several tasks are performed, from the purely syntactical ones, as deleting auxiliaries for future or getting the correct auxiliaries for passive, progressive or perfect aspects, to others that we call conceptual, such as:

5 In Portuguese, "ele voltou / ele devolveu o livro".
6 "ele desistiu / ele deixou o trabalho".
7 "eu tenciono / o programa destina-se".
8 "o problema aparece / o problema foi mostrado".
9 "o programa funciona / o programa é executado".
10 "ele trabalha / o programa funciona".
11 "ele paga / o crime compensa".
12 "olhar para/procurar/cuidar de/parecer-se com".
• Creation of new tenses (in the sense there are no corresponding ones in English)
• Changing voice (e.g., passive into active voice)
• Changing mood (e.g., indicative to subjunctive)
• Changing the whole tense (e.g. present perfect to ‘imperfeito’, negated imperative to present subjunctive, etc.)

The task is far from complete, and only a very restricted set of cases is yet covered in our system. However, enough has been done to assess the adequacy of the overall system, given that at least one situation of each point of the list above is handled.

Generation

Justification

Also on this subject there seems to be language-based reasons for the processing we present. Particularly, the top-down approach seems to coincide with the “natural” flow of information [6]. (For example, the dominating noun imposes the gender and number of the dominated nodes.)

Motivation

Since generation is target-language specific, some information will be given here about the actual tasks performed, which we divide conceptually between agreement and surface generation.

To motivate the need for agreement processing, we mention that in Portuguese
• adjectives agree in number and gender with the noun they modify (and in case of a conjunction of nouns with different gender, obey a specific algorithm for gender determination)
• past participles in passive voice agree in gender and number with the subject
• pronouns agree in number, gender and person with the entities they refer to
• verbs agree in number and person with their subject

So there is the need to derive and appropriately store the necessary information for agreement, after which the following tasks are performed:
• Define gender for nouns
• Compute feminine form for adjectives (if necessary)
• Compute plural form for nouns and adjectives (if necessary)
• Inflect a verb according to tense, person and number
• Compute case of a pronoun and inflect it
• Compute an adverb from an adjective
• Compute surface forms like contraction or hyphenation of clitics

The algorithm followed for surface generation is, however, very simple:

1. Look in the generation dictionary to see if the information needed is stored there (as it should be in case of an exception)
2. If not, apply the appropriate (morphological) rules to get the result

Dictionary contents

Apart from the irregular forms of the cases described above, the generation dictionary still holds all pronoun cases (indexed by its base form) and other information concerning the contraction of determiners and pronouns with the preceding prepositions.
Description of some practical aspects

In this section we present the system as a working device.

Development methodology

A short description of how the work proceeds in our system follows:

1. The departing point is the English text and the way PEG analyses it.
2. A straightforward rule or set of rules is written to handle the general case for a given problem (for instance, infinitive clauses, pronoun position in the sentence, or noun compounds)
3. Incremental growth follows: by elaboration and by refinement, that is, by contemplating new cases and also by making clearer distinctions
4. Studies on the corpus are done, creating a test set and running the system on it.
5. The appropriate corrections and improvements are done, until
6. the failures that represent the limits of our method are acknowledged (for instance, true ambiguities, or lack of information to distinguish between different cases)

An important guideline in the development of MENTOR88/P is the aim of universal coverage, which implies the following strategies:

- Try to solve the most general problems first
- Try to solve everything we can before handling particular cases
- Try to come to a definite and accurate statement of the limits of our methods before “patching” some problems that could even not need any “ad-hoc” treatment.
- Try to cover general problems, categories, or classes of words first, and only then handle exceptions.

Performance

Some results of MENTOR88/P will be presented, to allow for our conclusion that the overall design is fitted for translation from English. They concern the evaluation of two different kinds of problems, and were done on original corpus’ sentences (in the first case, every sentence containing at least an occurrence of an infinitive clause, while in the second case, the first 143 sentences where appeared the verb “be”).

The first test had as goal to evaluate a rather deep and extensive work on the translation of the infinitive clauses based on syntax. In 120 sentences containing a total of 141 occurrences, a percentage of 93% of correct translation was achieved.

The second test was intended to evaluate both the relative importance of the lexical transfer of the verb “be”, related to its use as a syntactical auxiliary, and the performance of the set of conditions (stored in the bilingual dictionary) for selecting between the two alternatives “ser” and “estar”. This problem is, by the way, also the deepest work on lexical transfer addressed nowadays in the system. The use of “be” as the main verb was present in 41% of the cases, and the overall translation of “be” was correct in 94.5% occurrences of the set.

To allow for some evaluation by the reader, some sample translations are presented in appendix, that illustrate some MENTOR88/P capabilities.

Future work

One of the most important issues a prototype must be concerned with, is, in our opinion, the way it can be expanded and made to incorporate bigger quantities of knowledge, but especially, qualitative improvements without becoming awkward or totally unsuitable.

---

13 an IBM technical manual with more than 1000 sentences, the REXPLI manual [1].
Also in this aspect we believe MENTOR88/P is able to receive a lot of improvement with no significant changes of philosophy or implementation.

The first issue that should be described is the (almost) all-syntactic approach. There is nothing, either in the architecture or in the rules we have been programming, that enforces or stresses this (present) characteristic.

In fact, we are deeply convinced that several kinds of different rules could be described at a higher level, as soon as we get the new layers that are being incorporated in PEG (or post-PEG processing) that will allow for other description levels in the sentence (like deep grammatical roles) and/or discourse characterization (of the kind topic/focus).

However, the syntactic rules should still be there to work as a safety net if the other information is not available. Also, and while we would need more generation processing when transferring at a higher level, it would have to be added to the already existing generation module, which remains necessary.

What we are trying to say is that, although we intend later to formulate transfer rules at different conceptual levels than the syntactic one, we need to go through the intermediate representation for Portuguese as it is now (namely with all syntax specified). Therefore, improvement of the system is an addition of new layers (both in transfer and in generation) but not at all a replacement.

Another important question that at this moment is only planned but we hope to begin working on as soon as possible, is a style component, consisting of several rules for transfer and for generation, and also in the lexical transfer domain (processing the several candidates).

Finally, the addressing of idioms and the development of a consistent and adequate bilingual dictionary are a must, and will certainly have to be dealt with in the future development of this project.

**Conclusions**

In conclusion, we believe that the system presented is appropriate for the task of machine translation, and we attempted to demonstrate it for the English-to-Portuguese case. We believe it to be modular, easily expandable, and expressive enough to encompass the large range of phenomena covered by the translation task.

We are convinced that the relatively large amount of problems dealt with in the small time frame of the project can be presented as a sign of hope. But we are also aware that much work still remains to be done before a prototype can be turned into a real working system, for whose purpose the building of a vast bilingual dictionary is determinant.

In this paper, we tried to justify the architecture presented both from a pragmatic and a linguistic point of view. A presentation of some of the general problems handled and some performance measures were intended to illustrate the translation task that is our goal to perform. Finally, we stated what was in our opinion the logical continuation for this work.

**Acknowledgements**

I am indebted to all other participants in MENTOR for fruitful and interesting discussions, and particularly to Lauri Carlson for his remark on the importance of the linguistic knowledge involved in a system independently of organization issues, and to Shalom Lappin for long and very detailed discussions on lexical transfer.

I would like to thank both Karen Jensen and Steve Richardson for their constant support during the development of this prototype answering my questions and doubts.

And finally, I would like to make clear that the work presented here is mainly a team work, to which I am most grateful to Paulo Libano Monteiro.

**References**


Some sample translations

I want the functions to have been correctly tested the next time

eu quero que as funções tenham sido correctamente testadas a próxima vez.

He listened to no excuses and never returned, but the money was eventually returned after some time

ele não escutou nenhumas desculpas e nunca voltou mas o dinheiro foi finalmente devolvido depois de algum tempo.

basically, to see if it is safe to execute a command, read the manual

basicamente, para ver se é seguro executar um comando, le o manual.

to want the man to pay to enter the house to buy some cigarettes was to kill him

querer que o homem pagasse para entrar na casa para comprar alguns cigarros foi matar o.

the meeting was in Paris, but she wasn't there

a reunião foi em Paris mas ela não esteve lá.

this program set allows users to avoid many problems

este conjunto de programa permite que os utilizadores evitem muitos problemas.

Appendix

This appendix is intended to illustrate the internals of the system, including some simple transfer rules and lexical transfer information, together with some more detailed examples of its output.

Some structural transfer rules

This first rule transforms a noun phrase into a prepositional one, in case there is an attribute called CHTOPP (holding a node representing the preposition). This rule can be triggered in several environments, like the translation of Noun Noun phrases, or of verb arguments asking for a preposition in Portuguese.
It should be noted that the rule only contains knowledge about changing the structure conveniently.

(5700) NP(CHTOMP) --> PP(%%NP, -DONE,
      plower<seg,'PPOBJ','NP','before','end'>,
      PRP=CHTOMP,
      padd<seg,PRP,'PRMODS','before'>,
      SECTYPE='PP', SECTYP2='PP',
      -CHTOMP)

The next rule handles infinitive clauses premodifying another clause. The only processing done is to mark it to be translated as an infinitive, in case it is also the subject of the main clause. Other rules at other levels will deal with that same infinitive clause (and if its translation was not determined here, it will be by other criteria).

(4910) VP('infcl3'.NOTIN.DONE, DONE='infcl3'...DONE,
      pfind<seg,'SECTYP2','INFCL','PRMODS','FOUND'>.NE.0)
      -->
      VP(%%VP,
         <SUBJECT.EQ.top<FOUND>,
         CHINCOTO(top<FOUND>)=1,
         TENS(top<FOUND>)=TENS>,
        -FOUND)

Bilingual entries

Some bilingual entries, both with structural markers and with lexical transfer information, are presented. Very simple entries were chosen.

RETURN
voltar(VERB (NCOND self OBJECT self PASSIVE segtype PTPRTCL))
devolver(VERB)
retorno(NOUN)

LOOK
procurar(VERB (COND PREP for) (PREPO for OBJECT))
olhar(VERB (PREPO at para))

ANY
nenhum(ADJ (COND NPIND NEG))
qualquer(ADJ)

FORGET
esquecer(VERB REFLEX (PREPO OBJECT de))

HAPPY
feliz(ADJ (INFPREP por))

Some more detailed examples

The two examples below can be useful to distinguish the tasks done in transfer and generation, being the intermediate structures built by analysis and transfer printed in a simplified way (PEG's output trees).

In both sentences, differences in structure and some generation processing can be seen.

I want the girl to go.

<table>
<thead>
<tr>
<th>DECL1</th>
<th>NP1</th>
<th>PRON1*</th>
<th>&quot;I&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERB1*</td>
<td>&quot;want&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP2</td>
<td>DET1</td>
<td>ADJ1*</td>
<td>&quot;the&quot;</td>
</tr>
<tr>
<td></td>
<td>NOUN1*</td>
<td>&quot;girl&quot;</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>INFCL1</td>
<td>INFTO1</td>
<td>&quot;to&quot;</td>
</tr>
</tbody>
</table>
Some Portuguese dictionary entries

Only exceptions are stored, as mentioned.

enigma:enigma(NOUN (GENDER MASC))
longe:longe(ADV)
mão:mão(NOUN (PLURFORM mãos))
mal:mal(ADV (COMPFORM pior))
mau:mau(ADJ (FEMFORM mã) (COMPFORM pior))
ter:ter(VERB (PRES1S tenho) (PRES2S tens) (PRES3S tem) (PRES3P têm))
tu:tu(PRON (ACUSFORM te) (DATIFORM te) (PREPFORM ti))
Overview of work done in structural transfer

Diana Santos
October 1988

VPs

Prepositions attached to verbs

If there is the attribute PREPO in the selected translation exactly what is stated in that attribute is performed. Examples:

(PREPO OBJECT de)
(PREPO at para)
(PREPO to OBJECT)
(PREPO OBJECT INDOBJ)

by creating a PP from an NP, changing the preposition in the PP, turning a previous PP into an NP or just altering the grammatical role assigned to a constituent. Rule 6100 in TRANSF is the main one.

Infinitive clauses

See the text "The translation of infinitive clauses in MENTOR88/P".

Relative clauses

If the relative pronoun is missing one gets created. For the moment only a “que” is inserted. However, if the verb asks for a certain preposition and the pronoun has the given role, it is automatically handled by the verb/particle combination rules.

The man I like is here --- O homem DE QUE eu gosto está aqui.

Also, in case the RELCL modifies a noun with feature TOV (for instance, “way”, “chance”, “opportunity”), the Portuguese clause should be introduced by “de”, while the verbal tense personal infinite (INFP) gets signalled.

The chance the program works is minimum -- A hipótese de o programa funcionar é minima.

(Rule 7600 in REFIN chooses the pronoun; rule 6000 in transfer inserts it).

Note: However, the last decision brings problems in the case of The way we did it was... -- A maneira de o fazermos... instead of A maneira como nós o fizermos...

Negation

When the verb is COPL, the English negation appears as an adverb related to the PSMODS (PREDADJ or PREDNOM), as “never”, “not” or “no”. We must negate the whole sentence in Portuguese, and deleting the negation in the PSMODS (except in the case of “no”). This is done
through rules 6210 and 6220 (at the VP level), and rules 6300 to 6400 do the actual deletion in the negated constituents.

The other task is the treatment of <auxiliary> NOT <main verb>, where if the auxiliary is “do”, it is deleted, on the other cases the relative ordering of the negation and the auxiliary is inverted (rule 6230).

Finally, 6240, 6260 and 6270 treat the contracted forms of “have”, “be”; “do” and “will” respectively, doing the same as was described earlier. (The only reason to have different rules is because PEG assigns a different tree structure).

**Handling auxiliaries**

For the moment, “do” as emphasis is simply deleted (rule 6260 in NEG).

Future and conditional are also simply treated by deleting “will” and in the case of the latter, the feature PTENS is set to ‘CNDF’ (conditional). (Rule 6270)

However, much remains to be done on the distinction of conditional and pure modal uses of “shall” (in which case the processing above is wrong), like “I should never have done...” (duty), etc.

**Question order**

The order in questions is made the same as in declarative sentences by rule 8200 in REFIN, while the interrogative pronoun is always translated in its emphatic form, “o que é que”, for instance.

It is not OK the fact that the clause “é que” comes englobated in the direct translation of the pronoun (though PEG uses some sort of similar parse for “that is”, for instance), but it is something to bear in mind to change in the future.

**Treatment of expletive pronouns (“dummy” IT and THERE)**

A very simple attempt is performed in rule 6010, where “it” is considered dummy if it is subject and the main verb is COPL. (This, obviously, does not work most of the times. A more “correct” test would be to test whether there was another candidate for subject in the sentence, like an INFC, a that-clause or even an inverted VP. However, this is somehow a problem of analysis, and so no special work has been done in this field, except for taking this into account when dealing with INFCs).

“there” is easily dealt with in rule 7100 in REFIN. The only action taken is the deletion of the adverb.

**Change from passive to active**

When there is a passive conjugation with a “dummy it” as subject, in Portuguese it is performed by an active sentence with indefinite subject “se”. This is dealt by rule 6050 in TRANSF.

**It is assumed that... -- Assume-se que...**

This is obviously related to the correct determination of “dummy it”... which is not done in this case, being only tested the subject as being the word “it”... Discussion of this matter was already done above, anyway.

**Change position of adverb phrases**

The very little treatment in this field amounts to one rule : 7400 in REFIN, and only moves pre-modifying adverbs in a clause (provided they are not sentential, as indicated by being followed by a comma; and that the clause only has a main verb). Of course it does not handle negative adverbs (since in Portuguese they should premodify the verb), neither does it change interrogatives in the beginning.
Treatment of reflexive conjugation

Here there may be an inaccuracy!!! The same treatment is performed for the use of “se” as indefinite subject, and for the verbs that ask for a reflexive conjugation! In both cases the VP is marked REFLEX, and so a “reflexive pronoun” is created in case the verb is not in the passive. This pronoun agrees in number and person with the subject (and by default is third person singular).

The origin of the marking REFLEX can come from two rules (6050 as mentioned above, and in 4905 of INFCL that handles “it < verb THATCOMP in passive > INFCL”) and means in that case “indefinite subject”. When the verb needs a reflexive pronoun, it is stored in the bilingual dictionary, exactly in the REFLEX feature, which is copied automatically into the new record.

esquecer(VERB REFLEX)

Position of pronouns (“clíticos”) in the clause

The rule 7540 in REFIN moves pronouns from after the verb to before it, in the following situations: in a relative clause or subordinate clause or in an INFPP (that is an infinitive tensed clause), or when the clause is negated, or when the clause is begun by an universal (UNIV) or existential (EXTL) quantifier, or in an SPECIF “quantifier”, like “only”, “even”, etc.

The pronouns referred can be indirect object, reflexive and/or direct object, and the order is DO-IO-REFLEX.

Handling of indirect object

In English there is an unmarked form of the indirect object which has no pronoun and follows immediately the verb. In Portuguese this must be changed (except for pronouns) into a PP begun by “a” (rule 7510 of REFIN).

he gave the young girl a kiss. -- Ele deu um beijo à rapariguinha.

When, on the contrary, the indirect object is in English in its marked form and it is a pronoun, and the same happens to the direct object (rule 7500) or the clause is in the passive (rule 7520), it should be unmarked in Portuguese, and, in the first case, precede the direct object.

he gave it to me -- ele deu-mo.
it was given to me -- foi-me dado.

NPs

Insertion of a determiner

The subject is identified for potential need for a determiner in rule 6020 of TRANSF. Similarly, an AVVPNP (noun phrase having an adverbial role) is also signalled for that purpose by rule 6030 of TRANSF.

Rule 5650 of TRANSF inserts the article whenever it is lacking, provided that the first premodifier of the NP is not a negative adjective (like “no”) or an existential quantifier (like “several” or “many”) and that the number of that quantifier is greater than one (to prevent “each”). Also, when the first modifier is a possessive adjective, it always inserts the determiner.

See NPS EXF for the performance of this rule.

Handling of adjectives

English adjectives are moved into a postmodifying position by rule 5600 of TRANSF when they are not true ordinal (that is, ordinal without the bilingual dictionary DEPOIS attribute) and also when they are not evaluative, that is, having a connotative instead of a denotative value. This last distinction is obviously very difficult to draw, so for the moment it is only considered the OD classification, that holds EVAL for “good”, for example.
A subtlety is inserted in this rule that might be in the scope of a style rule instead, and which is: any premodifying adjectives remain in place whenever the noun is postmodified, to furnish translations like:

Move the white tables and chairs. -- Move as mesas e cadeiras brancas
Move the white tables and chairs that are there -- Move as brancas mesas e cadeiras que estão ali.

**Simple handling of noun compounds**

To this moment we are only performing the default action for NP NP constructs, which is creating a PP with preposition "de" postmodifying the second NP, from the first NP. If the first NP is a proper noun, however, only the order change should be done. Also, treatment of the possessive case is missing. The rule is 5550 of TRANSF.

**Omission of head noun**

Rule 5500 omits nouns marked in the bilingual dictionary with the feature OMISSO and which are inserted in an NP with more than the simple noun itself, resulting in a "adjective" noun phrase whose head is an adjective.

There is a problem connected to generation of the correct Portuguese sentence, which has to do with the fact that gender and number of adjectives depend, in generation, on the noun that they modify, being therefore not correctly treated when they are the head (this is however, a generation problem to be corrected in the future).

Till now, only the noun "one" is marked as OMISSO in ODT.

**AJsPs**

**Comparatives**

English comparatives, depending mainly on the word size, are formed either by the suffix "er" or by the "more" premodifying the adjective. In Portuguese, we have always the counterpart of "more", "mais", except in some rare exceptions, that should be stored accordingly in the bilingual dictionary, through the feature SEMCOMP. So rule 7200 in REFIN just inserts a "mais" when "more" is lacking, except in those exceptions.

**Simple actions used by several rules**

- Conversion into a prepositional phrase (5700)
- Changing from passive into active (4920)
- Conversion from a PP into an NP (6110)
- Changing a preposition in a PP (6120)

**Change of some PEG results**

There are some rules that try to fix some problems of PEG, mainly those recognized as such by Karen, and that will be fixed in the new version.

These rules are the first for VPs, and go from 6000 to 6006 in TRANSF. No description will be attempted here.