NOTES ON PLNLP GRAMMAR WRITING

March 1989

Diana Santos

IBM-INESC Scientific Group
R. Alves Redol, 9 - 4 Dto
1000 Lisboa
Portugal
# Table of Contents

**Introduction** ................................................. 1  
**Coverage** ................................................. 1  

**PLNLP** .................................................. 3  

**Grammar writing in PLNLP: PEG experience** ..................... 5  
Ingredients for broad coverage parsing .......................... 5  
Primacy of language over theory ................................ 6  
Linguistics... ................................................. 6  
...and AI ..................................................... 6  
Trees? .......................................................... 6  

**The PLNLP system** .......................................... 7  
Procedures in PLNLP ........................................... 7  
Trees again ..................................................... 8  
PLNLP semantics associated to the symbols ",", and "|" ............ 8  
Action / condition distinction .................................. 9  
Enforcing OUR interpretation .................................. 9  
Example ....................................................... 9  

**Language is difficult** ...................................... 11  
Ellipsis ....................................................... 11  
Ambiguity ...................................................... 11  
Example of eliminating an ambiguity source .................... 13  
The concept of CLOSED ....................................... 13  
PP attachment ............................................... 13  
Concluding remarks .......................................... 14  

**Dictionaries** .............................................. 15  
Introduction .................................................. 15  
Capitals ....................................................... 16  
Global variables in CMS ...................................... 16  
The OD files ................................................... 16  
Programming ODLISTER ....................................... 17  
Interaction with LEXIS ....................................... 18  

**General tactics** .......................................... 19  
Which path to follow when the worst happens? .................. 19  
Documentation .............................................. 19  
Don’t make the grammar worse! ................................ 19  

**Grammar checking** ........................................ 21  
Style mode on ............................................... 21  
Error detection on .......................................... 21  
Errors versus style .......................................... 21  

**A morphological problem** ................................ 23  
Requirement of the hyphen in written Portuguese ............... 23  
Phonological changes ....................................... 24  
Systematization of the hyphen collocation ..................... 24  
An algorithmic description .................................. 25  
How to implement the above morphological rules in PLNLP .... 26  
Detecting endings ........................................... 27
Introduction

This text is the result of the six-weeks grammar course held in Bethesda from October 24th to December 9th 1988. It represents my vision and what I learned there, so it is surely incomplete and partial; however, it is intended to organize my thoughts and, especially, to be of help in teaching what I learned to the future grammar writers for Portuguese.

On the other hand, it is not intended to furnish comprehensive descriptions, but more to serve as a reference to what was talked about in the course, what were the main issues, and where are they described or discussed. Thus only in very few subjects a full description will be attempted.

Coverage

There were three main distinct subjects covered in this course, which are required for succeeding in writing a grammar:

- the PLNLP system itself, the programming language and environment that we use to handle Natural Language;
- the philosophy of writing a computational broad-coverage grammar, its conception and purpose;
- and how to actually develop a grammar for a language, with all related problems of measuring, using and debugging it.

No other claims are made about the above division, apart from being a way to organize the text. I’ll try also to follow the order in which the several points were presented in the course, whenever no other reasons impose a different order.
PLNLP

PLNLP is a programming language directed to Natural Language processing, plus an environment for friendly use and interaction with it. It has been used in the development of a broad coverage English grammar, PEG, whose great power is the best proof of the suitability of the language.

There is unfortunately not much documentation available, and most texts are still in a draft form. The main source is George Heidorn's dissertation[5], especially chapters 1, 2.a, 2.b, 3.a and 3.a.4, 4.a and 5. However, what is described in this document is only a subset of the present PLNLP capabilities, which were extended from then on. A lighter introduction from the same period is [6].

There is also an introductory text on PLNLP[16], which unfortunately refers to a PLNLP version who has no OD dictionaries, therefore beginning the grammar at the letter level instead of the word. Having thus remarked it, it should be emphasized that the text was really intended for teaching how to write in PLNLP, therefore it is very clear and has many enlightening examples.

Still in a draft form, [19] gives more emphasis to the encoding section programming in PLNLP as well as to procedures, and presents an overview of the whole system that unfortunately is also outdated in some details (concerning encoding). It should be noted that all examples that appear on this text were meant to be meaningless.

And there are some small notes written by George Heidorn on enhancements on PLNLP such as procedures, global variables and control specification[7].

Some relevant properties of PLNLP for the particular task of grammar writing will nevertheless be mentioned in this document, in "The PLNLP system" on page 7.
Grammar writing in PLNLP: PEG experience

This section is intended to present the advice and opinion of the author of the PLNLP grammar of English, Karen Jensen. I hope her opinions are not too changed here.

Reading the text [11] is assumed before reading this section. The high quality and large coverage of it, together with the vast material (foil copies, examples and exercises) that Karen furnished us throughout the course are the reason for the small extent of this section.

Ingredients for broad coverage parsing

As the large experience of the development of the English grammar allows Karen Jensen to conclude, the following items are necessary:

- **a strong, robust system**: this, though in a certain form obvious, should be stressed, because natural language is too complex a system to be manageable with deficient or unappropriate tools. PLNLP has till now proved able for the task, and this is why it was chosen to start grammars for so many different languages.

- **good dictionary/lexicon**: even though the computer cannot handle yet what we call common sense information, or pragmatics, it was found that more "modest" resources like word class features, for instance, could still provide a considerable power.

Actually, an interesting feature that Karen Jensen pointed out to us is that detailed lexical information results, in her experience, more often in a burden than in a blessing, at least when syntax is concerned: actually, the "smoooshing"7 of several distinct and possibly incompatible features under the same dictionary node (being a word only divided between different parts of speech) has proved beneficial in reducing the number of multiple parses and providing a more rational way of coding ambiguity.

- **binary rules as the preferred parsing strategy**: as a computational device to allow for unrestricted stating of natural language relations, they provide an elegant way of achieving modularity while allowing for free interaction of several linguistic phenomena.


- **relaxed versus rigid approach**, or better, **textual approach**: it is not the grammar that commands the way a sentence is understood, but the text that gives the clues on how the grammar should parse a certain string.

This can be related to the primacy of performance over competence, or to the opinion that it is the context that ultimately defines the meaning (in any sense) of a word or expression.

This point is clearly extremely important for any **application** of natural language as opposed to academic experiments. In a real application the grammarian cannot force the language to be as she/he wants, on the contrary, language must be accounted for as it is.

See for example on this matter Tomita's intervention in [20].

It is important to stress, related to the above considerations, that they are connected to the process of developing a mainly syntactical grammar. According to Karen Jensen, the grammars we are ex-

---

1 PLNLP is an acronym meaning contraction of several kinds of information, even contradictory associated to the same word, as **COPL** (copulative verb) and **TRAN** (transitive verb) in the dictionary entry of GO as a verb.
pected and committed to write are just the first stage (the first "shortcut") in the process of understanding language; by no means the ultimate and definite goal of natural language understanding. However, there is no doubt that syntax can definitely be of much help in this task.

**Primacy of language over theory**

One of the major inspiring notes in Karen's work is not to let any theory decide how language should be, but on the contrary use and accept only those parts of the theories that prove useful in describing language. (See [15] on her position.)

**Linguistics...**

So PEG does not follow a particular theory, but received influences from several schools. As in any generative grammar theory, phrases have always heads (though a noun phrase is not necessarily headed by a noun), and PEG's PRMODS and PSMODS can be somehow identified with GB specifiers and complements. On the other hand, Karen acknowledges some inspiration from dependency grammars.

I would like to emphasize, however, that PEG in itself represents a theory of language, and particularly, many significant linguistic conclusions could be drawn about English. For example, though English is generally presented as the paradigm of a configurational language, many examples of free word order were found.

**...and AI**

Comparing PEG, on the other hand, with AI work on NLP, it should be stressed that PEG aims at broad-coverage language parsing (as opposed being restricted to a particular domain). And, as it has recently been acknowledged by the AI community, "Small systems don't scale up!".

On the other hand, and though PEG incorporates only linguistic knowledge, further uses of PEG in the quest for understanding language (like PP reattachment through parsing of on-line human dictionaries[11][14]) are significant steps forward in the use of world knowledge in NLP.

**Trees?**

One important characteristic of PEG that should be noted, too, is the fact that the tree displayed after the parsing process takes place is NOT what is important.

That is, the "holy tree" concept so dear to some linguistic schools is not worshipped by PEG. On the contrary, it is the underlying (graph) structure that PEG computes - a complex of attribute-value pairs related among them, that really matters.

The first thing that should be noted, is that PEG-like record structures can carry much more information than ordinary nodes in a tree, since those structures can have as many attributes as wanted with values that are themselves complex record structures. (Contrarily to tree nodes, that can at most "point" to their children.)

The second point I would like to mention, is that the tree structures computed by PEG and therefore displayed by PLNLP are not standard in the sense that they do not mirror the parsing process, and are in general flattened trees. This again is discussed in [12] and [11].

In this text PLNLP trees will be also mentioned in "Trees again" on page 8.
The PLNLP system

By "the system" I mean here what I used to call "PEG shell" in [19], that is, the whole environment for a grammar only without the grammar rules.

So, it is not only the PLNLP language that we are going to use, but a set of built-in facilities and assumptions that implicitly impose some of our grammar writing style.

As in any real system, there is always a compromise between what is already furnished (and therefore, restricts our freedom to specify it) and what should we program and tailor right to our needs.

Although it is very easy for program-oriented people to change the whole system, that is not the point to be discussed here, on the contrary, my purpose is to describe what is already available.

Procedures in PLNLP

A procedure is the description of some action, as a sequence of sub-actions. It is the description, in the language of the computer, of an algorithm.

In PLNLP, as in most programming languages, there are two levels at which one can be interested in procedures:

- which ones exist, and what they do
- and how can we write others, or change the already existing ones.

While the second point is pure programming, and so it is described on [7] or [19], the first point, namely what the PLNLP environment already furnishes ready to use, is necessary for the understanding of the whole system.

PLNLP procedures that come with our environment (grammar writing environment) can be classified in four different types:

- **System procedures** - those that actually make the system behave like it does, such as the encoding or decoding algorithm code, or the access to the dictionary.

- **Utilities** - a set of procedures that can be invoked directly from the interactive shell and that are generally of use during the actual "run" of the grammar (as opposed to its writing). Examples: PRTELE, APLY, DIFF.

- **Library functions** - these are procedures (or functions) that are already defined and are to be used in the actual building (writing) of the grammar, like BOTRECS, or AGREES. See Appendix for a general list.

- **Linguistic functions** - these procedures are an integral part of the system, plus having a definite linguistic function. I am concerned specifically with:
  1. FITREE (that computes a fitted parse tree after the desired sentence parse has failed),
  2. PMETRIC (that computes a metric for the ordering of several alternatives)

---

2 There is, in my opinion, a difference between primitive functions like TOP or ISIN, on one hand, in the sense that they are required by the language and are coupled to its basic data structures, and library functions, on the other, that can themselves be defined in terms of the primitives.

The only distinction, on the other hand, between library and user-defined functions is that the user can use the first with no additional work.
3. ALTFOKS (that computes the alternative attachment positions for an ambiguous constituent).

It should be noticed that these functions may have to be modified to deal with languages other than English, however, the overall structure and purpose should remain the same.

Trees again

The fact that system procedures of PLNLP have some underlying assumptions should not be forgotten. To illustrate this, I want to point out the display of the result of the parsing process (or equivalently, the result of the call to the procedure PRTREE with a given node as argument).

The order by which PLNLP prints (and what it prints and how) is based in the following assumptions on the structures built by our grammars:

1. Only nodes representing terminal elements of our grammar (typically, words or punctuation) do not have "structure" (that is, attributes whose values point to other nodes).
2. All other nodes have a structure represented by having a HEAD attribute, always, and possibly PRMODS and PSMODS attributes (which have as values lists of nodes).
3. Every node has a type (either given explicitly in SEGYTYP2 or being its SEGYTYPE), from which its name is computed.
4. PRMODS store the constituents that precede the HEAD of a node and PSMODS those that follow it.
5. Each node is only in one position in what concerns these attributes (that correspond to the "tree structure" of the grammar). That is, a node is either HEAD or stands in the PRMODS or in the PSMODS of another one.

Nodes who do not stand in any of these positions, on the other hand, will never be printed.

Therefore, it should be clear that HEAD, PRMODS and PSMODS, independently of the linguistic meaning that the grammarian chooses to assign to them (having completely freedom on deciding what modifies what, how many modifiers should a phrase have, etc.) do already have a very concrete meaning in the PLNLP system.

This is just however an implementation issue, and, provided the grammarian fully understands the assumptions behind these attributes, she/he should not feel his/her power restricted.

PLNLP semantics associated to the symbols "", "|" and "|"

Simple statements in PLNLP are separated by "", or "|". The statements can be either conditions or actions, their interpretation depending on the context most of the times. A series of simple statements can in turn be treated as a single entity (again condition or action) if they are enclosed by "<" and ">". (i.e., <A,B,C> can be either a (complex) action or condition).

Statements cannot be both conjoined by "", and "|" at the same level, that is, "A,B|C" is not valid. However, both "<A,B>|C" and "A,<B|C>" are valid, since they join the statements at different levels.\footnote{\textsuperscript{3} A remark should be done at this point. Even though PLNLP allows for unbounded bracketing to occur, the present compiler has a bug that does not allow for "|" nesting, that is, it doesn't translate correctly "|" inside of "|", as in "<A|<B,<C|D>">". Therefore, it is necessary to "develop" those nested expressions into <A|<B,C>|<B,D>>.

Statements cannot be both conjoined by "", and "|" at the same level, that is, "A,B|C" is not valid. However, both "<A,B>|C" and "A,<B|C>" are valid, since they join the statements at different levels.\footnote{\textsuperscript{3} A remark should be done at this point. Even though PLNLP allows for unbounded bracketing to occur, the present compiler has a bug that does not allow for "|" nesting, that is, it doesn't translate correctly "|" inside of "|", as in "<A|<B,<C|D>">". Therefore, it is necessary to "develop" those nested expressions into <A|<B,C>|<B,D>>.

Conversely, "|" means : execute all statements till one is found that evaluates to TRUE (that is, a non-0 value or a procedure returning non-NIL).
It is easy to see that, if all statements joined by these symbols are conditions, then "" is functionally equivalent to logical AND, and "" to logical OR (with the restriction that they presuppose an order of evaluation, left to right, alien to a logical theory).

**Action / condition distinction**

It only remains to be said what is a condition or an action. Unambiguously (or independent of the context) the following PLNLP statements are always conditions:

- logical operators .EQ., .GT., .LSIN., etc.
- test for negation (¬)

and the following are always actions

- assignment (=)
- setting or deletion of attributes/indicators (+ -)
- the procedures PRTREC, PRINT, etc. that are declared in the indicators file as returning no values

Any other statement is differently interpreted, depending on appearing in a condition or action environment.

**Condition environment**

The left hand side record specification in a PLNLP rule, the interior of a loop, the interior of a complex statement (that is, what is enclosed by "" and "")

**Action environment**

The right hand-side of a record specification in a PLNLP rule, the body of a procedure.

**Enforcing OUR interpretation**

To override the defaults, that is, to force something to be interpreted as an action in a condition environment, or the converse, is obviously also possible in PLNLP.

If A is ambiguous⁴, and we want to make it an action, it suffices to write = A.

If A is ambiguous, and we want to make a test out of it, we write A.NE.0 (or A.NE.nil, if A is a procedure to which we do not assign any return value, with <>).

**Example**

What was described in the preceding exposition makes it possible to have the same in left and right hand sides of a PLNLP rule with a completely different meaning

```
VP(OBJECT, pfld<aaa>)
-->
VP(OBJECT, pfld<aaa>)
```

A fully disambiguated equivalent formulation of the above would be:

```
VP(OBJECT.NE.0, pfld<aaa>.NE.nil)
-->
VP(OBJECT=OBJECT, =pfld<aaa>)
```

Even though this is an extreme case, it is helpful to illustrate the context sensitiveness of PLNLP.

---

⁴ that is, if A is not independent of the context
Language is difficult

Under this section some problems will be discussed, and a PEG-like way to deal with them is described.

"PEG-like" because the reason that should make us follow the way of dealing with problems that PEG solved for English is pure common sense. But it should be emphasized that PLNLP itself does not prescribe us this approach.

Ellipsis

This is a very frequent phenomenon in any natural language, and it takes place the more the meaning of the utterance is clear from the context.

The first instance (and less problematic one) occurs in connection with the conjoined phrase, and its correct analysis (or understanding) must go through the appropriate argument structure.

Dealing with this problem in English, Karen could give us the following clues:

• First of all, parsing of constructs embedded in parentheses, such as

  She was happy (but hesitant)

  presents one example of the advantage of using binary rules for conjunction (this time with "but"). It is clear that the following examples are not valid English in any part of the globe

  (She was happy) but hesitant.
  She was (happy but hesitant).
  She was (happy but) hesitant.

• A typical example of object and subject distribution is the following

  He drove through and completely demolished a plate glass window.

  but a speaker can even go further very easily

  we thought it would happen but not so.

  In these cases, Karen's opinion is that the grammar should provide a fitted parse (because the CONJ ADV ADV "but not so" is difficult to admit as a valid clause in English, after all!), and then process it afterwards.

Actually, these examples of ellipsis are intimately connected with the problem of parsing endocentric constructs (as opposed to exocentric ones), in a framework of heads or dominating constituents. In fact, constructs as conjunctions cannot naturally be described as having a center.

Ambiguity

This is an intrinsic and relevant characteristic of natural languages. But this is too broad a concept, actually, that can be broken down in several ones:

• Syntactical ambiguity. Typical examples are
Time flies like an arrow.
They are flying planes.

One "working" method to detect ambiguity is to automatically rephrase the sentences, like

time does fly like an arrow / an arrow is liked by time flies / etc.
planes they are flying / they are planes that fly

- Attachment ambiguity is well-known subset of syntactic ambiguity.

They dressed the woman in red.
The woman was dressed in red by them / they dressed the woman who was in red

- Lexical ambiguity - one word means normally more than just one clear-cut concept.

My children are good students.
(Children in a classroom, said by the teacher)
(Children of a proud mother)

- Logical ambiguity - in the following example, there's a difference in meaning coming from
  existential or universal quantification performed by the quantifier "a":

John wants to marry a girl his parents do not like
(because he hates them)
(even though he is sad about it)

- And referential ambiguity is the more talked about, this time being generally extraclausal in
  nature.

My parents visited the Smiths. They were angry.

It should be mentioned that the examples above are not intrinsically ambiguous. Actually, context
and/or emphasis make it generally possible for the receiver of the message to select one alternative
only. However, as we are building simple sentence grammars of written language, we have often
to struggle with the problems above.

Our concern in the development of the grammar, actually (personal opinion), should be to
disambiguate as much as possible at the syntactical level, and the least possible at the other levels.
(Of course referential ambiguity is generally even outside our scope).

Two reasons seem to make this methodology preferable: The first is that it is defendable that
lexical ambiguity interpretation is language dependent (as the example above can show for people
who speak languages that do not perform the "criança/filho" distinction for "child"). The other
reason stems from the fact that there are in general extralinguistic factors, such as the intention of
the utterer and the context of the utterance, that are ambiguity removers, and to which we do not
have access, therefore we should not expect to disambiguate at that level in our grammars.

It remains only to be emphasized why should we care about syntactical ambiguity at all. The reason
is straightforward. A different syntactical structure corresponds to a different meaning (otherwise
the sentence would not be ambiguous). We must have a syntactical analyser good enough to see
all syntactic meanings of a sentence, and, still more, to restrict the imagination abilities of the parser.

It was formally proved impossible to decide whether natural language is inherently ambiguous, or
just the grammars that have been built to describe it. It is nevertheless often the case that some
ambiguities are due to the grammars. It is because most grammars describe syntax at too superficial
a level that they produce a combinatory explosion of parses.

Work with unrestricted English in PEG made Karen Jensen aware of several mechanisms of lan-
guage that fight ambiguous readings, or, in another way, that it is possible to diminish considerably
the number of parses generated by a grammar, recurring to syntax alone.

---

5 This statement does not imply the converse, namely, that the same meaning cannot be vehiculated by
different syntactical structures...
Example of eliminating an ambiguity source

Let us take as example the verb "to go". Consider that we would like to parse it as a transitive verb in "go a mile", and as a copulative verb in "go crazy", apart, of course, the most common use of "go" as intransitive. However, it can be observed that, and unlike other copulative verbs (like "be"), if a noun phrase (NP) appears after the verb it is not a predicative complement, but can only be its direct object. Also, and even though "crazy" (as any adjective in English) can be interpreted as a noun, it cannot in positions where it would otherwise be ambiguous, like "he is crazy". (Notice the extremely deviant "he is crazy man"). In "The crazy die first", on the contrary, there is no restriction for "crazy" to be interpreted as a noun.

How can we make use of this observation in building a PLNLP grammar? We block the undesired alternative parses, accepting "go crazy" with "crazy" as predicate adjective (PREDADJ) and not predicate noun (PREDNOM) or direct object (OBJECT). That is, one of our object picking rules could be as follows:

\[ \text{VP(COPL,TRAN) NP(\sim ADJPOS(DICT) | CLOSED>) --- VP(PSMODS=PSMODS...NP, OBJECT=NP)} \]

which can be read: if a verb can be both transitive and copulative, and is followed by an NP, it is assigned only the role OBJECT (and not PREDNOM), provided that that NP could not be an AJP too, or that there's enough evidence that it is a real NP. This last statement can be rephrased by postulating that an NP is considered closed if its HEAD cannot be an adjective (no part-of speech ADJ) or if it can, it must appear in an unambiguously syntactical context (such as having a determiner and not being able to modify what stands in its immediate right).

The same scheme of reasoning happens for the English indirect object - direct object sequence. Not all VP NP NP should be considered as such, but only those who consist of closed NPs.

Summing up, if we take into consideration some language features we are able to reduce drastically the number of parses produced by the grammar, which shows that deep enough syntactical "dwelling" can make the problem of syntactical ambiguity in a grammar tend to the limit of that of the language.

The concept of CLOSED

Related to the examples presented in the preceding section, there was an emerging concept from English PLNLP parsing, called CLOSED, that corresponds to the need to identify a well delimited linguistic constituent, in order to stop completely improbable (if not impossible) parses to pop out.

Thus, an English noun phrase (NP) is considered to be closed, for instance, if it cannot receive more premodifiers (if it has already a determiner, for instance). In that case two contiguous closed NPs have to be parsed as such, while if the second were not closed, the first could (should) be parsed as modifying the second.

An NP is also immediately closed if its head is a pronoun, etc.

More than theorize about the ultimate meaning of this concept, it is important to see it as a development aid in the grammar, marking "complete" constituents versus those who may not yet be. The fact that some phrases are marked "complete" definitely helps in the statement of the rules that deal with these kinds of constituents, as could be seen in the previous section.

PP attachment

This is the standard example of syntactical ambiguity, and one of the main responsible for the exploding number of parses.

---

6 Note that the theory of English we are describing (PEG's) considers "nuts" in "go nuts" to be an adverb, therefore this construct falls outside the scope of this discussion.
PEG's answer is simple and elegant - to produce only one parse (actually that with rightmost attachment), and as a postprocessing phase (performed by ALTFOKS, 3. on page 8) compute and therefore indicate the possible alternative attachments.

How can we implement this same feature in our grammars? Simply by blocking all parses except those who attach the PP to the smallest valid constituent immediately to the left.

\[ \text{NP} \text{ PP(SEGTYPE(last\text{wrd}<\text{PSMODS}(\text{NP})>)\text{.NE.'NOUN'})} \rightarrow \text{NP}(\text{PSMODS=}\text{PSMODS}...\text{PP}) \]

The above can be read as - "the PP attachment to an NP can only take place if that NP has not yet any postmodifiers whose last word is a noun". This restriction can be completely understood if we note that, if that NP had already postmodifiers ending in a noun, then there would have already been an attachment performed by this same rule, and no further attachments should be allowed.\(^7\)

(A common error that should be prevented is, by the way, to write the above rule as

\[ \ast \ast \text{NP} \text{ PP(SEGTY2(last\text{wrd}<\text{NP})>)\text{.NE.'NOUN'})} \rightarrow \]

Like this, this rule (for PP attachment) will never be triggered, since even a simple NP with no postmodifiers at all has as last word a noun - its head.)

The computation of the other attachment alternatives, on the other hand, and as was already pointed out, may imply the rewriting of ALTFOKS for languages not too similar to English.

**Concluding remarks**

Summing up this section, some ways of "thinking in PLNLP" were first introduced:

- The idea of blocking undesirable parses during the parsing process, through additional "clever" conditions in the grammar rules
- The idea of using a postprocessing phase (ALTFOKS)
- The concept of marking constituents "closed" (complete) to help the parsing process.
- The advice looking at language to see how to syntactically disambiguate cases where an impulsive grammarian would say only semantics could help

\(^7\) Note that this condition makes sense for English, but not forcefully to all languages. In Romance languages, for instance, NPs can have adjectival postmodifiers, and so the last word of a valid NP could be an adjective as well.

The preceding rule should then test the following:

\[ \text{NP} \text{ PP(SEGTY2(HEAD bötrees < PSMODS >))\text{.NE.'NOUN'})} \rightarrow \]

Anyway, the above is not supposed to handle all cases of PP attachment in any language.
Introduction

OD dictionaries are the dictionary environment used by PLNLP. The way their content should be specified for building them for the first time is described in the introductory notes[22] that all grammarians received.

Some terminology will however be described:

**OD entry** is the index in the dictionary (a word) plus all the information that is stored there about that word.

**OD node** is one lemma-part of speech combination in the entry. There are no limits for the number of nodes per entry, but we were strongly advised to keep a node per part-of-speech, and using the smooshing of different syntactic senses of the word.

Follows the entry for the word “go”, with two nodes

\[
\begin{align*}
g0 & (\text{NOUN SING}) \\
g0 & (\text{VERB COPL INF PLUR PRES TRAN (OBJTPREP for)})
\end{align*}
\]

The interaction between PLNLP and the dictionaries is such that for each OD node, a PLNLP record is built, and stored in a named record whose name is the index name, in its attribute NLPDCTRECS. For each part-of-speech considered in ODTABLE, another attribute will be created, that has as value the record(s) after the OD node with that part of speech.

???????? Figure here >>>>>>>>>>

**Figure 1.** The word “man” after accessing the dictionary

Conversely, every record built from the dictionary has an attribute DICT pointing to the name record that represents the word. From this follows that whenever and wherever we want, in our program, to inspect the contents of the dictionary, we can go through the DICT attribute of the record we are manipulating.8

In order to test whether the transitive feature of the verb homonymic with the noun we are dealing with does not occur, we could write

\[
\text{NOUN(\sim\text{TRAN(VERBPOS(DICT))})}
\]

This example, by the way, shows another capability of PLNLP grammars to block spurious parses: by testing in the rules other possible part-of-speech interpretations.

---

8 Every record, if created through an implicit or explicit copy from another record coming from the input string, has a DICT attribute. Only in the case of it being a low-level record, or a record created from scratch, it won’t have a DICT attribute, except if you assign it explicitly.
Capitals

If a word in the dictionary is to be identified if capitalized only, then it should still be written (as index) in lowercase, but with the special OD feature CAP. Then only a capitalized string matches.

\[
\text{march(VERB ...)} \\
\text{march(NOUN CAP (MONTHNO 3) ...)}
\]

Identically, the ALLCAP feature indicates that the word should only be recognized if all letters that form it are capitals.

Global variables in CMS

OD dictionaries test the variable LANGUAGE of the PLNLP set to switch to the correct language (dictionary files). The command to set the language (that for a grammarian must only be done in his/her profile) is

\[
\text{GLOBALV SELECT PLNLP SET LANGUAGE ENGLISH}
\]

where instead of ENGLISH it should be the name of the OD files, for instance POR for Portuguese, from POR ODTABLE, POR OD, etc...

OD programs always test the value of this variable before any action.

The OD files

The most important file related to OD is that of type ODTABLE, where stands the declaration of the features (or characteristics) that can be associated with a dictionary item. Again, the way to do it is described in the preliminary notes[22].

The ODSOURCE is how we wrote the first set of data to be stored in the dictionary.

The OD file and the ODDATA file are the result of building an OD dictionary, the first is the compacted form for the actual access, the second is a human-readable form for listing and editing purpose (used by the utilities described next).

Finally, the ODUPDATE contains the modifications we introduced interactively while working with the dictionary, since the last creation (or merging) of the dictionary. The full power of this device can only be realized in an environment where several distinct users are allowed to change the dictionary, however, even for the alone grammarian there are advantages in having this file as a “buffer” to the main file. Only when the changes in the ODUPDATE become stable should they be incorporated in the main file.

The utilities provided for OD dictionaries are the following

**OD**

invokes the dictionary, entering in XEDIT mode. We can then look at, change or delete one or several entries, using normal editor commands.

\[
\text{OD hello}
\]

It can be called from the PLNLP environment too, with the following format

\[
\text{(OD "hello")}
\]

**ODBUILD**

builds an OD/ODDATA pair from an ODSOURCE, with the help of the ODTABLE.

\[
\text{ODBUILD POR}
\]

It first compresses all entries, then really builds the OD file.
ODGEN merges two different sources of dictionary information, typically one consisting of updates and the other being the main file. It creates, as ODBUILD, an OD and an ODDATA file, and has the additional care of renaming the old ODDATA and ODUPDATE into ODDATAn and UPDATEn, being "n" the next integer (if there are already ODDATA2 and UPDATE2, then "n" will be 3). This allows the grammarian to keep a trace of all the changes she/he has ever made - and probably run out of space quickly...

ODGEN POR ODUPDATE C POR ODDATA

C is just the output mode desired. It should be referenced that whenever the two files contain the same index, the first overrides the second, so the order above is most convenient.

ODBATCH allows the automatic addition of a list of OD nodes to a dictionary. Those OD nodes should be stated as in an ODSOURCE file. It is useful when many changes should be made, avoiding editing a large number of different entries.

ODBATCH filename filetype (POR

The file with the additions to perform could, for example, have the list of words to add the feature INTRAN:

word1: base1(POS1 INTRAN)
word2: base2(POS2 INTRAN)
word3: base3(POS3 INTRAN)
word4: base4(POS4 INTRAN)
word5: base5(POS5 INTRAN)

ODLISTER allows the listing of the dictionary entries (actually either from the ODDATA file, or from the ODUPDATE)

ODLISTER POR ODUPDATE

To indicate what should be listed, see next section.

It should be mentioned, by the way, that all these programs produce an explanatory message if a question mark is issued as their argument.

**Programming ODLISTER**

To actually list the dictionaries, we must change the file ODLISTER EXEC accordingly. To do that, perform the following steps:

- **Edit ODLISTER EXEC**
- Go to the line where the string "process:" is. (PROCESS is the name of the part of the ODLISTER program that specifies what should be done to each entry. "process:" indicates the beginning of that part.)
- Change the file according to what you want, that is, insert some lines of REXX code:
/* List all entries */
call writentry
   count = count + 1

/* List all OD entries that are marked with TOVCOMP */
if index(nodes,'TOVCOMP')> 0 then do
call writentry
   count = count + 1
end

/* List all OD entries that are marked both with (VERB and TOVCOMP */
if index(nodes,'(VERB)') > 0 & index(nodes,'TOVCOMP')> 0 then do
call writentry
   count = count + 1
end

/* List all dictionary entries that are marked either with THATCOMP or TOVCOMP */
if index(nodes,'THATCOMP') > 0 | index(nodes,'TOVCOMP') > 0 then do
call writentry
   count = count + 1
end

- Exit from the file, and invoke the program (as described above).

Interaction with LEXIS

OD will also be our interface to LEXIS. That means that we shall be able to see LEXIS morphology through an OD interface, while morphological information will also be available in the PLNLP records that we handle with the grammar. For that, we must define how and what we shall see from LEXIS in our ODTABLE, in the following way:

We assign a name for each combination of features in the LEXIS table (MATRIX) that we want to access, and then we simply write the numbers of the columns of that table in the ODTABLE.

054 0 1SNGPRES 1 3 4 this indicates first person singular present
055 0 MASC 8 masculine

There are no a priori reasons to assign more than one LEXIS feature per OD feature, but this is a powerful capability.
General tactics

In this section some miscellaneous points concerning the proper development of the grammar will be focused. A brief report on grammar development can be found in [11].

Which path to follow when the worst happens?

If, inputting a new sentence to our grammar, we get several parses, none of them correct (don’t desperate, this is the worst possibility we could have), Karen advises us to FIRST, block all the undesired parses, and only THEN, try to get the correct one.

Documentation

There’s no point on documenting each rule, writing in our languages what is already written in PLNLP. But some indication of the general problem (and if possible of the solution intended) should be there.

Also (personal opinion) we should keep track of all linguistic decisions we do (probably not in our RULES files), so that we know what should be the outcome of our grammars, and that we know (afterwards) why did we choose that way of representing language and not another.

Don’t make the grammar worse!

It is very important to be able to evaluate a grammar change (hopefully an improvement). Though interactions between rules should be kept to a minimum, they should always be tested in order to verify any undesirable side effects.

For this task, the KAT (keep and test) facility is extremely useful. See [2] for an extensive description of it.
Grammar checking

The main (pragmatic) purpose of these grammars is text critiquing, that is, given a text in our language, produce some comments on its grammaticality and style.

The first remark that should be made, is that there is no clear cut distinction between style mistakes or real errors, depending more on the linguistic background of the critic and his/her/its pedagogical attitude, than on a set of well agreed-upon rules. (See [17], [13] and [21] on this subject).

Therefore, it will also be the responsibility of the grammarian to decide, among deviant input, which should be punished with a grammatical error and which should receive a friendly suggestion. To make this choice, it is necessary, on the other hand, to know the consequences of both options, be it in terms of grammar writing, or in terms of actual output.

The main distinction between a grammatical error and a style weakness is that the first prevents successful parsing (thus displaying a fitted tree) while the second doesn’t. Thus running the grammar without either error mode or style mode on, the difference can easily be seen.

Style mode on

Style mode (triggered by the simple setting of the global variable NLPSTYLE) results in the following: whenever a poor style construction is encountered that was contemplated in our system, after the parse an advice message is uttered.

Style detection programming consists therefore in the writing of procedures that travel down the parses computed by our grammars, and, whenever encountering some particular situation, produce a pertinent message plus a suggestion concerning the problem at hand.

The actual practical requirements to use style detection are described in the appendix concerning “Implementing style”.

Error detection on

To recover from grammatical errors, the global variable NLPERRS must be on9, and in the very same rules that would normally fail because of those errors, a second alternative must be written, to allow them to succeed consistently when error detection is on. The net result for the user is that the previous failing parse now succeeds with a warning message and the corresponding advice.

See [4] to know more about this process.

Errors versus style

Why not allow all typical errors (those our grammar is more interested in criticize - the ones that really happen in everyday writing) and then produce style messages of corresponding severity? - could be asked. Because, answers Karen, there are some errors that, if allowed, will result in an unrestricted number of parses: for example, subject verb agreement in English. (The test for agreement is one of the most important sources of blocking capability in PEG).

Why not handle everything in the rules (why bother to traverse the graph once computed if we can detect all errors/style weaknesses in the rules too)? First of all, it is more modular to separate tasks.

9 Just issue (SETQ NLPERRS T).
First build, then look at it from other prisms. Second, the kind of processing implied by error correction, working in a twopass mode, makes the system much heavier computationally (and consequently slow). And, finally, there may be interactions not contemplated in our rules in error mode. It is not more than natural that while writing the grammar the grammarian is implicitly imagining that "correct" things are "correct", otherwise no parse is produced (therefore not taking into account the cases where suddenly wrong parses are allowed...)

For all those reasons, there must be a delicate compromise between what we treat in errors or in style processing. As a personal opinion based on non-linguistic factors, I would advise STYLE before ERRORS, because before our programs are perfect, think about their users: it is easier to accept disdainfully a wrong suggestion than being hit by a "Grammatical Error" message in the wrong place!
A morphological problem

The problem to be handled is that of enclitic pronouns plus split verbs in future or conditional tense, in Portuguese.

It is an example of solving non-trivial morphology issues with PLNLP, that can be more or less carried to other fields, such as numeric strings, money, or label processing.

Requirement of the hyphen in written Portuguese

In written Portuguese, personal pronouns and the passivating particle "se" (see below), when after the verb, are joined by a hyphen.

While the pronoun form is the same for the object, indirect object and reflexive cases for the first and second person

\[
\begin{align*}
\text{dei-me} \\
\text{dei-te} \\
\text{dei-nos} \\
\text{dei-vos}
\end{align*}
\]

the situation is different for the third person, where the pronoun forms are distinct

\[
\begin{array}{c|c|c}
\text{D.O.} & \text{I. O.} & \text{Reflex} \\
\text{deu-o} & \text{deu-lhe} & \text{deu-se} \\
\text{deu-a} & \text{deu-lhes} \\
\text{deu-os} & \text{deu-lhes} \\
\text{deu-as} & \text{deu-lhes} \\
\end{array}
\]

When both arguments are pronouns, they get contracted whenever the direct object is in the third person.\(^\text{10}\)

\[
\begin{align*}
\text{dei-mo} & \rightarrow \text{dei-ma} & \text{dei-mos} & \rightarrow \text{dei-mas} \\
\text{dei-to} & \rightarrow \text{dei-ta} & \text{dei-tos} & \rightarrow \text{dei-tas} \\
\text{dei-lho} & \rightarrow \text{dei-lha} & \text{dei-lhos} & \rightarrow \text{dei-lhas} \\
\text{dei-no-lo} & \rightarrow \text{dei-no-la} & \text{dei-no-los} & \rightarrow \text{dei-no-las} \\
\text{dei-vo-lo} & \rightarrow \text{dei-vo-la} & \text{dei-vo-los} & \rightarrow \text{dei-vo-las} \\
\text{dei-lho} & \rightarrow \text{dei-lha} & \text{dei-lhos} & \rightarrow \text{dei-lhas}
\end{align*}
\]

Reflexive pronouns can also be added to the others, functioning as purely reflexive, reciprocal, or of "interest" (also called "ethical dative"), etc. (See [3].) Finally, the pronoun "se" as a "passivating particle", meaning an undetermined subject (playing the role of passive in an active context), is also positioned in the place of the homonymous reflexive pronoun.

\[
\begin{align*}
\text{esqueceu-se-lho} \\
\text{deu-se-no-la}
\end{align*}
\]

\(^{10}\) Actually, this contraction happens also when the elides appear in preverbal position, but in that case they do not carry a hyphen, therefore not falling under the scope of this chapter.
The future and conditional tenses, when with clitics, split their endings, which appear after the pronouns:

\[
\begin{align*}
\text{dar-mo-ás} & \quad \text{perdoar-se-1hos-ia} \\
\text{dá-lo-ei} & \quad \text{esquecer-ta-iam}
\end{align*}
\]

**Phonological changes**

Another issue related to post verbal clitics is the fact that their existence brings several other changes both in writing and orally. For example, verbs in the infinitive

\[
\begin{align*}
\text{amá-lo} & \quad \text{esquecê-lo} \quad \text{abri-lo} \quad \text{repô-lo}
\end{align*}
\]

or in the imperative

\[
\begin{align*}
\text{fá-lo} & \quad \text{põe-no}
\end{align*}
\]

change their termination when followed by a direct object pronoun in the third person.

This fact imposes that "non-words" must be dealt with in the Portuguese grammar, which have, to be accepted, to obey both morphological AND syntactical rules. (for instance, "fá-te" or "deu-no-lhe" are impossible).

**Systematization of the hyphen collocation**

We can resume the facts described above, even though they may seem complex at first sight, by the following rules:

- The order of the pronouns is reflexive, indirect object, direct object, followed by the future or conditional termination.
- When there is both direct (in third person) and indirect object personal pronouns, they are contracted.
- When the direct object personal pronoun in the third person follows
  - a verb form ended in "r" - the infinitive (or the main verb part of future or conditional), the "r" disappears, the last letter gets an accent (except in some cases for "i") and an "I" is inserted before the pronoun.
    Thus, "amar-o" gets "amá-lo", "esquecer-a" turns into "esquecê-la", "trair-os" is "traí-lôs" but "vestir-os" is "vesti-los" and finally "repor-as" is "repô-las".
  - a verb form ended in "z" gets transformed the same way.
    "faz-o" is "fá-lo", "fiz-a" should be "fí-la".

- a verb form ended in "s" (which happens in all cases for first and second persons plural) loses the final "s" and also gets an "I" after the hyphen.
  "fizemos-o" is "fizemo-lo"; "fizestes-os" is "fizeste-los".
- a verb form ended in "m" (every third person plural), gets an "n" inserted instead of an "I", while the "m" does not disappear:
  "fizeram-os" turns into "fizeram-nos". This happens always in the third person plural.
- a verb form ended in "õe" or "ões", at last, an "n" is required before the direct object.
  "põe-o" should be "põe-no".
An algorithmic description

The above description suggests that the first task needed is the recognition of the "true" verb and pronoun forms behind the surface forms, which can be expressed by the following rules, illustrated with the direct object pronoun "o" (replaceable by "a", "os" or "as").

Infinitive
á-lo --> ar-o
ê-lo --> er-o
i-lo --> ir-o
i-lo --> ir-o
ô-lo --> or-o or õr-o

Split future and conditional
á-lo --> ar-o or azer-o
ê-lo --> er-o
i-lo --> ir-o or izer-o
ô-lo --> or-o or õr-o

Present of indicative, 2nd person singular; imperative, same person
a-lo --> as-o
â-lo --> âs-o
e-lo --> es-o
ê-lo --> ês-o
ôe-no --> ôës-o

Present of indicative, 3rd person singular; imperative, singular
á-lo --> az-o
i-lo --> iz-o

Simple past perfect, 3rd person singular
ê-lo --> ez-o

Simple past perfect, first person singular
i-lo --> iz-o
u-lo --> us-o

1st person plural
o-lo --> os-o
o-nos --> os-nos

2nd person plural
e-lo --> es-o

Simple past perfect, 3rd person singular
ô-lo --> ôs-o

Third person plural, present and past
m-no --> m-o

The above can in turn be abbreviated into the following rules, where the 'I' cannot be followed by an 'h' (or alternatively, only 'lo', 'la', 'los', 'las' are possible).

á-1 --> az-
ê-1 --> az-

azer-
ir-

ô-1 --> or-
o-1 --> ôs-
ôs-

i-1 --> ir-
e-1 --> es-

A similar set can be presented for 'n' after the hyphen, with the exclusion of "nos" or "no-l" (first person plural), which correspond to a different situation.

ôe-n --> ôës-  m-n --> m-  o-nos --> os-nos

It should be noticed that when there is more than one alternative, generally only one corresponds to a valid verb form in Portuguese, therefore in most cases not implying multiple results.
The second task is to separate the direct from the indirect object pronouns, when contracted. The rules are straightforward.

-\text{-mo} \rightarrow -\text{me-o} \\
-\text{-to} \rightarrow -\text{te-o} \\
-\text{-lho} \rightarrow -\text{lhe-o} \text{ or lhes-o} \\
-\text{-no-lo} \rightarrow -\text{nos-o} \\
-\text{-vo-lo} \rightarrow -\text{vos-o} \\

The third task is to identify the future and conditional endings.

-\text{-ei} \rightarrow -\text{ia} \\
-\text{-ás} \rightarrow -\text{ias} \\
-\text{-á} \rightarrow -\text{ia} \\
-\text{-amos} \rightarrow -\text{iãmos} \\
-\text{-eis} \rightarrow -\text{iéis} \\
-\text{-ão} \rightarrow -\text{iãm} \\

Summing up, to handle all hyphenated forms of Portuguese (except for compound words), it is necessary to be able to identify the verbs and their inflection (1st task), to identify the criticized pronouns (2nd task) and to identify the split endings for future and conditional when existing (3rd task).

\textbf{How to implement the above morphological rules in PLNLP}

Though generally the PLNLP records representing a word come directly from the dictionary, in a process that is transparent to the grammarian, we can perform a letter by letter processing in PLNLP as it is described in [16]. Several examples of that are available in PORDECO (for other languages, in their DECO file), where morphology is done.

The barest rules that have to do with hyphenation simply state that a verb can be followed by an hyphen, a pronoun can be followed by an hyphen, and that a pronoun can be anteceded by an hyphen.

```plaintext
// The test is to distinguish between "vesti-o" and "vesti-lo" // between "vesti-lho" and "vesti-lo-ei" and between "fazem-nos" and // "fazem-no". In the second cases, this rule should not apply, nor in // "damo-nos".
VERBP(top<!NLP-INPUT>.EQ.'-')
    ,<second<!NLP-INPUT>.NE.'L'|third<!NLP-INPUT>.EQ.'H'>
    ,<second<!NLP-INPUT>.NE.'N'|fourth<!NLP-INPUT>.EQ.'-'
    ,<substr<STR,strlen<STR>,1>.UNEQUAL."a"
    ,<substr<STR,strlen<STR>,1>.UNEQUAL."m">) -->
    VERBP(?NLP-INPUT=?NLPSPACE...?NLP-INPUT)
```

```
// This rule adds the hyphen to the following pronoun, and signals, with/ // HIFEN, that it as linked to the previous word with an HIFEN. It also// separates, if existent, the following hyphen.
PRONP(~CONTRAIDO) --&gt; PRONP(HIFEN=1,
    <top<!NLP-INPUT>.EQ.'- ', ?NLP-INPUT=?NLPSPACE...?NLP-INPUT>)
```

```
// This rule separates the following hyphen from the pronoun at hand, as// do the preceding two rules, and moreover it signal the pronoun as // being contracted (since it is not marked HIFEN). //
PRONP(~HIFEN, ~CONTRAIDO, top<!NLP-INPUT>.EQ.'-') --&gt;
    PRONP(?NLP-INPUT=?NLPSPACE...?NLP-INPUT, CONTRAIDO=1)
```

These rules account for the simple cases:
Introduza uma frase em Português ou um comando:
eu dei-me-te.

-----------------------------
DECLI NP1 PRON1# "eu"
  VERB1# "dei"
NP2 PRON2# "-me"
NP3 PRON3# "-te"
PUNC1 "."

So that the rule presented above can be clearly understood, it should be explained that VERBP (verb part) and PRONP (pronoun part) are segments created during PLNLP morphological processing, and may be (or not) transformed in VERB and PRON. ?NLP-INPUT is the PLNLP global variable that contains all characters of the input string that haven’t been used by the rules yet. Finally, ?NLPSPACE holds the character for blank (or space) in PLNLP. Insertion of a blank character in ?NLP-INPUT by a rule has as result signalling that what follows should not be considered part of the current constituent, which should in consequence be looked up in the dictionary or die.

The reason for the tests in the first rule have to do with the fact that, for instance, “vesti-lo-ei” and “vesti-lo” and “vesti-o” and “vesti-lhe” have exactly the first 6 characters equal, but the first two should undergo a transformation (to “vestir-o-ei” and “vestir-o”) while the others not. Therefore what comes after the hyphen must still be tested so that in the first two cases, the “simple” rule will not apply (and therefore a more complex, involving more characters, will in turn apply afterwards).

To handle the three tasks referred in the preceding section, a rule of each kind will be presented, in reverse order.

Detecting endings

Easiest is a rule of the third kind:

- E I --> WORD4(+FUTURO, +PESS1, +SINGU, SECTYPE='NOUN', BASE='EI', SEGTYP2='VENDING')

Where the ending “-ei” is detected, and some features are assigned to it, namely a SECTYPE of NOUN (any would do, provided it is one of the eight POSs recognized by PLNLP) and a SEGTYP2 of VENDING (short for Verb ENDING), which is handy for display and for ready recognition that this “noun” is special (for instance giving no origin to an NP in the syntax rules).

The other settings are no longer arbitrary, since they have to do with the number, person and tense carried by the ending.

Undoing contraction

A rule that detects contracted pronouns is also straightforward:

- M(top:?NLP-INPUT>.NE.'E') -->
  WORD4(?dictrec<"me", 'PRON'>,
  ?NLP-INPUT=?NLPSPACE...?NLP-INPUT)

This rule only applies when a contraction has been performed (that is, it would not apply if “me” were the actual word).

Identifying the underlying verbs and pronouns

Finally, I present a rule of the first kind, namely the one for the “ir” termination (which is more ambiguous than the other conjugations).
// These rules undo the phonological changes that exist in Portuguese in/ // connection with elicitized pronouns, creating, for the "non-word" // verb parts, the corresponding verb. // For the particular case of the termination "ir", as "fizer" is not an/ // infinitive, but "dizer" is, the first occurrence, that is, the one // marked CONJUNCT (subjunctive), must be prevented, deleting that record/ CHARSTR I - L(top<?NLP-INPUT>.NE.'H') --> // WORD4(XDICTREC<STR(CHARSTR)."iz", 'VERB'>), // WORD4(XDICTREC<STR(CHARSTR)."izer", 'VERB'>, // <CONJUNCT>-SECTYPE>, // WORD4(XDICTREC<STR(CHARSTR)."ir", 'VERB'>, // ?NLP-INPUT=?NLPSPACE...?NLP-INPUT)

This rule makes use of the record whose SECTYPE is CHARSTR, which contains all characters found after the last delimiter (a space or a special character). WORD4 is the name of a tentative version of a word, to be used by further processing. DICTREC is the PLNLP procedure that returns the POS interpretation (specified as the value of its second parameter) of the character string which is specified as the first parameter. For instance, ddictrec "vestir", NOUN would return the PLNLP record corresponding to the information stored in OD under the part-of-speech NOUN (and which is displayable by the function PRTDICT\[1\]).

To conclude, it should only be mentioned that in the rule above the ability of PLNLP of specifying in parallel several different records as output of a single decoding rule is used. In case there is more than one record that gets created, it is the role of syntax to decide which one is actually legal.

The role of syntax

How morphological processing is done was thus described entirely in the above section.

Syntax will in turn use its results to decide about the well formedness of the strings. It should be noted that these morphological rules alone would allow, for instance, the (impossible) combinations:

deu-o-me

deu-me-o

deu-me-lhe

deu-me-me-me-me

Syntax, determining the role fulfilled by each pronoun, provides for the filtering of the strings above.

The following rules are therefore necessary:

- picking the indirect object,
- picking the direct object,
- picking a reflexive pronoun (or the passivating particle).

These rules have to provide also for uniqueness, and correct order.

They could also check for the correct hyphenation and contraction required by morphology, but this was decided not to be done at this stage so that it could be treated as a possible error for the grammar to detect and advise.

Finally, the split conditional/future endings have to be dealt with also at the syntactical level, because they can only be joined to a syntactically correct complex, (that is, a verb in the infinitive being at least followed by one pronominal argument). Here is the syntactical rule devised:

\[1\] prtdict "vestir" prints all records (one per OD node) associated with "vestir".
Dealing with discontinuous future or conditional verb endings. //
- SUBJECT is for multiple parse blocking. NUPES covers the indicators //
that carry number and person. //
VP(INFINI, -SUBJECT,
   <<INOBJ, SEGYTP2(HEAD(INOBJ)).EQ. 'PRON'>
   <OBJECT, SEGYTP2(HEAD(OBJECT)).EQ. 'PRON'>>
   NOUN(SEGYTP2.EQ. 'VENDING') -->
   VP(-INFINI, PSMODS=PSMODS...NOUN,
      <FUTURO(NOUN), +FUTURO>,
      <CONDICI(NOUN), +CONDICI>,
      NUPES=NUPES(NOUN).ORBITS.NUPES)

And follow some illustrations of the application of the above rule:

Introduza uma frase em Português ou um comando :
eu vesti-lo-ei.

DECLI NP1  PRON1* "eu"
   VERB1* "vesti-1"
   NP2  PRON2* "o-"
   VENDING1"ei"
   PUNC1 ".

Introduza uma frase em Português ou um comando :
eu vestir-to-ei.

DECLI NP1  PRON1* "eu"
   VERB1* "vestir"
   NP2  PRON2* "-t"
   NP3  PRON3* "o"
   VENDING1"-ei"
   PUNC1 "."

Discussion

In this process there were several alternatives that could be considered and that should thus be debated.

The first one concerns mirroring the input string. This was actually a choice, namely that places of constituents should not be changed while displaying the result of the grammar.

Another option taken was not to treat cliticized pronouns differently from object or indirect object NPs. That is, a VP has the same form independently of its arguments being cliticized and/or contracted or not.

Illustrating the opposite idea, it would be conceivable that a single node were created to account for the string "vesti-lo-ei". However, it would be far less modular, in that two different structures would be built by the parser having the same syntactical structure, and that just because of morphology. Moreover, the possibility of interacting with the dictionary to get the features associated with the pronouns would be lost, therefore implying that the information about them would have to be redundantly coded in the morphology rules as well.

Another choice that could have been made was to store the parts of the words (like "amá" or "fí" or "ei") in the dictionary, and have a small subgrammar to make sense out of them. We would thus have "words" like "lo", "pô" or "emos" in the dictionary, which could be argued as strange. Even if these aesthetic reasons were not strong enough, it should be acknowledged that this method
would require more processing, in that the surroundings of these "non-words" could not imme-
diately be checked to discard several possibilities.\textsuperscript{12}

Conclusion

In this rather large section a specific problem of Portuguese grammar was described and our pre-
ferred solution sketched.

This example can be interesting for two reasons. First, because it illustrates morphological proc-
essing in PLNLP. Second, because it argues for the separation of tasks between morphology and
syntax, or, putting it in another way, for the use of syntactical parameters to finish up morphology.

In fact, the solution presented did not handle the whole problem as syntax (if using "non-words")
nor as morphology (creating single units for verbs plus clitics), but used a mixed approach where
morphology contributed to identify the syntactical units, joined by syntax rules, which were also
in charge of legalize morphological phenomena, as split conditional or future verb forms.

\textsuperscript{12} It should nevertheless be noted, that the method presented in this chapter requires PLNLP rules with re-
cords with names having accented characters, which is nowadays not possible. If that restriction cannot
be overcome, then the storing of "non-words" in the dictionary is a must.
A syntactical problem

The problem to be illustrated is that of agreement concerning adjectival complements in a clause.

Agreement is a phenomenon that occurs in many languages, and in one language there can be several different kinds of agreement. By describing one way to solve one subproblem of agreement, we hope that this text can be more general than if it were dedicated to a very particular detail of Portuguese syntax.

General description

There are several agreement phenomena in Portuguese:

1. subject-verb agreement in a clause
2. adjective-noun agreement in a noun phrase
3. subject-past participle agreement in the passive voice
4. agreement of adjectival phrases (not contained in an NP) with the subject of the clause, or with its direct object.

Only 4. will be discussed on this chapter.

As a first distinction, we can separate adjectival phrases (AJP) by their force as an argument of the clause. That is, we can divide adjectival phrases between those who are required by the verb (a copulative one) and those who are optional arguments (in clauses headed by non copulative verbs).

In the first case, we can still divide the clauses between those where the AJP must agree with the subject (and headed by simple copulative verbs, such as "be")

ela é bonita

and those who must agree with the direct object, in clauses headed by verbs "complex copulative"¹, such as "find".

ele acha-a bonita.
ele acha a Maria bonita.

The second case, where the adjectival phrase, optional, must agree with the subject, can be illustrated by the following examples

eles morreram felizes.
elas viveram felizes aquela experiência.
ele bateu à porta, esperançado.
eles deram o livro, envergonhados, à senhora da biblioteca.
ele olhou gulosso para a mesa posta, admirado por terem esperado.

¹³ PEG calls this verbs CMPXTRAN.
The proposed solution

We chose to handle the two cases by two different rules:

1. Testing for the absence of another PREDADJ or PREDNOM, and requiring the verbs to copulative (COPUL) or complex copulative (COPL2), the first rule adds the AJP as the PREDADJ, as only argument.

   Tests are done to prevent it from being an article - an article is not a "true" adjective - and the AJP cannot be parsed differently (for instance, modifying a preceding noun phrase already included in the VP).

   The passive voice (syntactically equal (COPUL verb plus past participle / adjective) is also prevented.

   Finally, if the verb is complex copulative, agreement between the object and the predicative adjective is required.

   (8740) VP(~PRMDS, ~PREDADJ, <COPUL|COPL2>)
   // Articles can never be considered normal adjectives
   AJP(~ARTIGO),
   // Test for a real AJP, not integrable in preceding NPs...
   <PSMOS(P)\{|QUANTIF\{CLOSED\}
   SEGTP2(HEAD(bot<PSMOS(VP)>)),EQ.'PRON',
   nconcorda<SEG,BOT<PSMOS(VP)>>,EQNIL>,
   // If the verb is SER, the AJP can't be a past participle (passive)
   // as in "o livro foi dado à Luísa"
   <BASE(VP).NE.'SER'|~PARTIPAS(VERBPOS(DICT))>,
   // If the verb is COPL2, there must be agreement with the OBJECT
   <COPL2(VP)|~OBJECT(VP)|
   nconcorda<OBJECT(VP),seg>.NE.nil>,
   // Preventing "a mais simpática" from PREP NP AJP
   -> VP(PREDADJ=AJP,
   PSMODS=PSMOS...AJP)

2. The second rule parses the AJP as APOSTO, and for that it is only necessary that the verb is not copulative, nor is it the auxiliary for perfect aspect ("ter" - have).

   Except for the particular tests of copulative verbs of the previous case, namely agreement with object and passive detection, all the remaining ones are shared by this rule.

   // Preventing past participles from being parsed as adjectives
   (8750) VP(~PRMDS, ~PREDADJ, ~COPUL, ~COPL2, BASE.NE.'ter')
   AJP(~ARTIGO,<PSMOS(VP)|QUANTIF|CLOSED|
   nconcorda<SEG,BOT<PSMOS(VP)>>,EQNIL>,
   // Preventing "a mais simpática" from PREP NP AJP
   -> MODADJ(ADVPOS(DICT(lastword<VP>))))
   // Preventing "a mais simpática" from PREP NP AJP
   -> VP(APOSTO=APOSTO...AJP,
   PSMODS=PSMOS...AJP)

The agreement between the subject and the PREDADJ or APOSTO will have to be checked during the subject picking rule, as we follow a binary rule approach. (See "Error detection" on page 39 for a simplified version of that task).

With it, these two rules succeed in parsing all examples above, as can be appreciated by the following examples:
Introduza uma frase em Português ou um comando:
ele acha-a bonita.

DECL1 NP1 PRON1* "ele"
      VERB1* "acha"
      NP2 PRON2* ";-a"
      AJP1 ADJ1* "bonita"
PUNC1 "."

Figure 2. An AJP as argument of a complex copulative verb: The translation reads: "He finds her beautiful", where "beautiful" agrees with "her".

Introduza uma frase em Português ou um comando:
elas viveram felizes, alegres, maravilhas.

DECL1 NP1 PRON1* "elas"
      VERB1* "viveram"
      AJP1 ADJ1* "felizes"
      AJP2 PUNC1 "."
      ADJ2* "alegres"
      AJP3 PUNC2 "."
      ADJ3* "maravilhas"
PUNC3 "."

Figure 3. Free adjectival phrases in non-copulative clauses: The three adjectives following the verb agree with the subject. Translation: "They (feminine) lived happy, joyful, wonderous".

However, other rules contribute directly to this, namely the one for PREDNOM (predicative noun) and/or OBJECT picking, which has clearly to have been triggered before, for clauses headed by complex copulative verbs. And of course, there should be mentioned the other rules that deal with AJP's, be it to add them to noun phrases or to accept adverbial phrases added to them.

Nevertheless, the more important thing to be described is undoubtedly the function NCONCORDA, since it is this function that tests for agreement between the number and gender (noun features) of its two arguments. This function is largely used all throughout the grammar, it is however so simple that it describes itself:

NCONCORDA (PAL1*PTR, PAL2*PTR,
             // This function should compute noun-adj agreement between two records.//
             <NUMERO(PAL1).*AGREE.NUMERO(PAL2),
              GENERO(PAL1).*AGREE.GENERO(PAL2), <&T>)

Figure 4. The function NCONCORDA: Tests for agreement in number and gender.

NUMERO is the cover indicator in the INDICATORS file for number, and GENERO is the one for gender.
A style problem

The problem to be described is the lack of commas before the adjectival modifiers dealt with in the preceding section.

In Portuguese, and as the last chapter illustrates, there should be a comma before APOSTO (free adjectival modifier), whenever it does not come right after the verb.

In the Portuguese grammar, it was decided to accept APOSTOs in all places without commas, and produce a style message when the comma is missing.

Style processing in PLNLP

Style processing in PLNLP grammars is best done through independent procedures which are invoked after the bottom-up parsing process has taken place, and whose result is the display of a message after the tree has been displayed.

```
Introduza uma frase em Português ou um comando:
ela deu o livro envergonhada.
```

```
DECL1 NP1 PRON1* "ela"
VERB1* "deu"
NP2 DETP1 ADJ1* "o"
NOUN1* "livro"
AJP1 ADJ2* "envergonhada"
PUNC1 "."
```

PROBLEMA DE ESTILO
Aposto sem vírgula...:
ela deu o livro envergonhada.
QUE TAL inserir uma vírgula antes deste segmento?
```
Figure 5. Application of a style procedure: After the display of the tree, a message is printed out.
```

The procedures are applied to the whole tree(s) produced by the grammar, and should test, in the places we specify, for the parameters we require. The tree is traversed down from top to bottom, left to right.

Common to all style processes is the following information (which pertains also to error detection, to be described in "Error detection" on page 39):

ERRMSGS This record, defined in the RECORDS section, should have the following information - <NAME> = "<error message>", where <NAME> stands for an identifier representing the kind of style weakness, and <error message> should be a string identifying the general kind of style weakness (the diagnosis).

ERRORS In any record built by our grammar, the attribute ERRORS, when exists, is special in that it points to a list of records whose information describes an error/style weakness each.
Its SUP should be ‘ERROR’, and the value of its attribute TYPE should correspond to one kind of error that is listed in ERRMSGS.

Other attributes of each component of the ERRORS attribute should also be described in detail:

- **STYLERR**: should store an identifier referring to a broad classification, like ‘PHRLEV’ for "weakness at phrase level".

- **ESEGB**: literally, the Error SEGment to Brighten, is the segment whose corresponding string is to be brightened (highlighted) in the display message.

- **MSG**: This attribute should have as value a string with a specific suggestion in order to improve the text.

The next attributes will only be taken care if an error (opposed to a style weakness) was detected:

- **CSEGS**: Another attribute of an ERROR record, it provides finer help by storing, as a lower-level record, the modifications suggested. This record is to substitute the erroneous one.

- **REPLSTR**: Is an attribute of the CSEGS record, and stores the string to appear in a “CONSIDER” message, that is, a more complex message that furnishes the right alternative.

## Detecting and critiquing the lack of commas before APOSTO

The procedure that we present here is called ApostoSemVirgula (literally, “APOSTO without comma”), and simply adds, to the ERRORS attribute of the record representing a clause, one ERROR record for each record belonging to the list APOSTO and which does not have a comma as its first element. (With exception of the case when the top of the list is exactly the first word after the main verb, as was already pointed out).

The contents of those ERROR records will exhort the user to insert a comma before the highlighted words (those who belonged to the APOSTO list and were added to ESEGB).

```plaintext
ApostoSemVirgula (seg*PTR,
apostol/LST, apostop/PTR, apostopal/PTR,

<SEGTYP2.ISIN.!nlp-vtyp2, APOSTO, ~COPUL, ~COPL2,
  <BASE(top<APOSTO>).NE.BASE(top<PSMODS>), apostol=APOSTO>
  apostol=rest<APOSTO>>,
  (apostol.NE.0, apostop=top<apostol>, apostol=rest<apostol>,
   BASE(top<PRMODS(apostop)>).NE.',',
   apostopal=frstwr<apostop>,
   ERRORS=ERRORS...<‘ERROR’, STYLERR='nivel'sint',
     ESEGB=apostopal, TYPE='APSV1',
     MSG=downcase"inserir uma virgula ",
     MSG=MSG.|.downcase"antes deste segmento?">)
```

**Figure 6.** The function ApostoSemVirgula: The calls to the function DOWNCASE have to do with a system particularity in Portugal that should not be relevant for other countries.

The corresponding ERRMSGS record simply stores the following:

```plaintext
ERRMSGS (Z271="++56A/SERICOMA - MISS COMMA (CONJ PH'S)",
  APSV1="A".|.downcase"posto sem virgula...")
```

**Figure 7.** The record ERRMSGS: It will have a value per each kind of error treated by the grammar. Z271 is a fictitious error.
Finally, here is where the call to the style procedures is inserted. It is important to note that the number 6000 is in an ENCODING section of PLNLP, and should not be changed as it can be considered to belong to the PLNLP system.

```
(6000)  ERRTST --> NULL(%%ERRTST,
    --Invoke style procedures here:
    ApostoSemiVirgula<ORIGREC>,
    other procedures here ///
    ERRORS(ORIGREC))
```

Figure 8. Place to call style procedures

To activate or deactivate style processing, there’s the PLNLP switch NLPSTYLE, which should have a non-NIL or NIL value, respectively.
Error detection

The example of error dealt with is the lack of agreement between subject and adjective modifiers, already discussed in the previous two sections. It should be noted, on passing, that the reason to have chosen again the problem described in "A syntactical problem" on page 31 stems from the desire for a clear exposition, and is not meant to argue that this problem for the Portuguese grammar is special in any way.

Some comments will also be done on the interaction of error detection with parsing failure because of other problems.

Subject picking

There are at this moment three subject picking rules in the Portuguese grammar, one for simple active sentences, other for simple passive sentences, and still another for “active passive” ones.

In every case, however, the tests for agreement and subsequent error processing are the same in these three rules, and, due to their complexity, only what is pertinent to agreement between subject and the adjectival complements (being it PREDADJ or APOSTO), therefore being common to the three tasks, will now be displayed.

\[
(8780) \quad NP,
\]

\[
\begin{align*}
& \text{VP(-SUBJECT,} \\
& \quad \text{<-PREDADJ[nconcorda<NP,PREDADJ> | SETERR<`CONCAP'>]}, \\
& \quad \text{<-APOSTO[nlconcorda<APOSTO,NP> | SETERR<`CONCAP'>})} \\
& \rightarrow \text{VP(PRMODS=NP...PRMODS, TEMPORAR=NP,} \\
& \quad \text<TSTERR<`CONCAP'>,} \\
& \quad \text{ERRORS=ERRORS...<`ERROR', TYPE=`CONCAP',} \\
& \quad \text{ESEG=HEA(D(NP)...HEA(D(PREDADJ(VP)),} \\
& \quad \text{ESEG=ESEG...HEA(top<APOSTO(VP)>),} \\
& \quad // \text{PREDADJ and APOSTO are mutually exclusive...} \\
& \quad \text{CSEG=HEA(PREDADJ(VP))...HEA(top<APOSTO(VP)>,} \\
& \quad \text{MSG=downcase<"Estas duas palavras">,} \\
& \quad \text{MSG=MSG.|.downcase<"devem concordar em gênero">,} \\
& \quad \text{MSG=MSG.|.downcase<"e número">>>)}
\end{align*}
\]

Figure 9. Subject picking: What is common to the three rules and has to do with agreement of the subject with adjectives

Error detection in PLNLP

The most important concepts for error detection are those of SETERR and TSTERR. SETERR is a function that returns NIL (therefore acting as false) if the flag NLPERRS is not set. Therefore, and being SETERR the last clause of an OR, the parse will fail (or at least the rule in question will fail to apply). However, SETERR changes its behavior if the flag NLPERRS is on, giving a true value and allowing the parse to succeed.

It moreover sends a 'signal' to its twin function TSTERR through the value of its argument. TSTERR evaluates to 'true' whenever SETERR was invoked with the corresponding signal. This
case means that there's an error that was deleted. It should then be processed by creating an ERRORS list similar to the one described before for style.

Let's watch the result of error detection in the Portuguese grammar:

```
DEGL1 NP1 PRON1* "ela"
   VERB1* "é"
   ADJ1* "bonitas"
   PUNC1 "~"
```

ERRO EM
adejctivo não concorda com sujeito.
ela é bonitas.

CONSIDERE:
ela é BONITO.
estas duas palavras devem concordar em gênero e número

Figure 10. Application of the rule of Figure 9 on page 39: The lack of agreement is pointed out, and its solution is proposed.

It should be noted, in the example above, that the suggestion is not completely correct, in that the BASE of the word is displayed, and not the correct inflection. To do this last thing, we should need to access LEXIS morphology in a "generation mode", which is not yet possible...

As a possible provisional solution, the correct gender and number could be appended to the string, which would be done through REPLSTR:

```
CSEGS=HEAD(PREDADJ(VP))...HEAD(top<APOSTO(VP)>)
	<<FEM(SUBJECT),
   REPLSTR(CSEGS)=pname<BASE(CSEGS)>.|:"(feminino "|"
   REPLSTR(CSEGS)=pname<BASE(CSEGS)>.|:"(masculino "|
   <<PLURA(SUBJECT),
   REPLSTR(CSEGS)=REPLSTR(CSEGS).|:"plural)">|
   REPLSTR(CSEGS)=REPLSTR(CSEGS).|:"singular)">,
   MSC=downcase="Estas duas palavras">,
```

Figure 11. Specifying more detailedly the corrections: Through creating a string corresponding to the uninflected word plus the inflections required.

The result will be the following:
Interaction between style and error critiquing

One thing that can happen is that style messages appear also in erroneous sentences, that is, in those which should receive an error message.

It is therefore advisable that whenever an error of the same type appeared no style message pops up.

Also, if some sentence fails to parse, it is generally ridiculous that the grammar indulges in issuing style advice. In this case, I believe no messages should be issued either.
References


[4] Harriehausen, Bettina. 1988 "Don’t give up on me or Why grammars need to expand their scope of parsable input”.


