

# ALP – A Latin Parser

Archibald Michiels  
[amichiels@uliege.be](mailto:amichiels@uliege.be)  
<https://archibaldmichiels.hcommons.org/>

2021

# Table of Contents

ALP – A Latin Parser.....	1
Introduction.....	3
Parsing Issues.....	14
Grammatical Sketch.....	14
Dealing with Multi-Word Units.....	16
Linearity.....	17
Producing, Storing and Retrieving Information.....	24
Mapping.....	26
Matching.....	30
A Voice in the Middle.....	32
Relative Clauses.....	34
Prioritizing Subject-object Order in Accusative-cum-infinitive Clauses.....	39
Binding SE.....	49
Weighting.....	55
Test File.....	57
Additional Test Sentences.....	63
A Few Example Parses.....	65
Morphological Variants: Rogo.....	71
Appendix I ALP tackles coordination ... during a quick ... coffee break.....	74
Appendix II How to Use ALP.....	80
Appendix III Parsing and Word Sense Assignment.....	82
Two word senses of COLO in Forcellini.....	83
Word sense assignment in ALP.....	85
Appendix IV Syncretisme grammatical ou ambiguïté de construction ?.....	89
References.....	99

# Introduction

ALP (A Latin Parser) is a syntactic parser for a small subset of classical Latin. How small the subset is (or how large, size lying in the beholder's eye), can be guessed through a quick perusal of the test files (included in this document), i.e. the collections of sentences that ALP is able to deal with, i.e. to parse. We do not think that ALP can be described as a toy parser, a mere sketch of what could be done on a larger scale (see for instance Covington 2003). A toy system is seldom extensible without a major revision of the very framework it is based on. That is not the case with ALP, as we will attempt to show.

ALP is a true parser, i.e. it delves into the surface strings looking for deep syntactic structure, which means here predicate-argument pairings, which it delivers according to a canonical order that is fully independent of the linearity of the sentences it deals with.

It is worth stressing that this endeavour amounts to more than simple tagging (the assignment of tags associated in a lexicon with the surface elements) followed by the limited amount of surface structure that can be built on the basis of sequences of tags (a good example of what can be achieved with these limited means is Koster 2005).

The argument structure referred to above is not limited to noun phrases (nps), prepositional phrases (pps) and the like, but includes clauses, which can exhibit the complex structure associated with full independent sentences. In a word, we have to do with true parsing.

Of course, the linearity of discourse will be seen to be preserved in the *wordlist* associated with the surface string, but is also taken into account in assessing the weighting assigned to a given parse, its preference ranking. As a matter of fact, most reasonably complex sentences will receive more than one parse (i.e. will be ambiguous with respect to the grammar embodied in the parser, although they may appear – and be – fully unambiguous as utterances, if we leave aside the body of utterances used as grammatical examples or illustratory material for schoolbooks, even if borrowed from the classical writers, precisely because they are deprived of their context of utterance). The weighting procedure aims at keeping only the most promising parses (or in presenting them first to the user, which amounts to the same). A weighting procedure is an essential part of a parser, even if it is often missing.

Latin exhibits a relatively free word order, and is therefore not likely to be amenable to top-down parsing, which is based on the checking of structural hypotheses derived from the grammar (and specifying a given left-to-right ordering) by confronting their expectations with what is found in the string submitted to analysis.

A better candidate is provided by a bottom-up parser, which looks at what it has under hand (the string with its given word order) and tries to put it together somehow on the basis of the structural descriptions the grammar holds for higher elements (for example, putting together a noun and an adjective sharing gender, number and case to produce a np which will be able to fill in a syntactic slot, let's say subject if the case is nominative - or accusative in an infinitival clause - and number is compatible with that of the VP of which it is supposed to provide the subject).

It is such a bottom-up parser that ALP implements, but in a way that takes full advantage of the 'facilities' offered by Prolog<sup>1</sup>, which are in fact the very mechanisms that Prolog is built on, I mean UNIFICATION and BACKTRACKING.

Before moving on to a discussion of those two basic mechanisms, a short and drastically oriented introduction to Prolog as a programming language may be in order.

In Prolog we describe a *world*, very often a *micro-world*, the *objects* that populate that world and the *relations* that obtain between them. We do this by means of *facts* and *rules*. In our case the micro-world is a subset of classical Latin, the facts are mainly to be found in the lexicon and the rules in the grammar.

Note that the notion of the distinction between facts and rules running parallel to that between lexicon and grammar is very much a simplification. It can be argued that the description of *multi-word units* that we need for parsing is much nearer to grammar rules than to standard lexical entries. I have attempted to show elsewhere that most multi-word units, unless they are completely frozen, are best captured by rules that look very much like standard grammar rules, except that they contain lexical material that restricts the openness of the purely structural requirements of standard grammar rules (see Michiels 2016).

Facts in Prolog are like records in a standard data base, but in a much freer format. Rules have conditions of the *if-and-only-if* type, and can lead to the production of new facts, which are then added to the database of facts Prolog is working with and which aims at capturing the basic facts and relations of the world being described.

Once the description of the *world*, its inhabitants and properties, is embodied in a Prolog program, we can submit *queries*, i.e. ask questions. A Prolog query (a question set to the Prolog engine) will be interpreted in the following way: is this provable? If the query contains *variables*, it will involve finding values for the variables that make the query true. So, by the side of very simple queries, which look like queries on a standard data base, we can formulate queries of a much more complex form, asking Prolog to enquire whether the object we propose can be admitted as a new inhabitant of our world (can be *proved* to be such an inhabitant), and what values need to be assigned to the variables in the query to make it so.

In our case (once the parser and the grammatical and lexical resources it draws on have been entered as a Prolog program), we can submit a string to Prolog, and ask whether that string can be read as a Latin sentence (i.e. can the string be *proved* to be a Latin sentence?). We can leave as variables to be instantiated (i.e. given a value) what will turn out to be the parse, i.e. the structural assignments that explain why the string is in fact the embodiment of one or more Latin sentences (there will be more than one *proof* if the string is ambiguous with respect to the grammar and lexicon in use).

To sum up, the variables to be instantiated will be bits of the structural make-up of the sentence, under the structural assignment that made that string a Latin sentence, according to the grammar and lexicon embodied in the Prolog database built as a result of running the program. This data base is partly static (standard lexical entries), but mainly dynamic – applying the rules yields new structures. When Prolog has found a structural description (i.e. a parse) that covers the whole input string, it has found one way of making the string a Latin sentence. It is then ready to start all over again, and find the other possible ways of parsing the string that make it a Latin sentence (and thereby dealing with ambiguous sentences, ambiguous with respect to the grammar and lexicon

---

<sup>1</sup> We use SWI-Prolog, a long-standing high-quality free Prolog. A suitable installation file (i.e. geared towards the OS one is using) can be downloaded from the SWI-Prolog website. See Wielemaker 2003 for an overview.

embodied in the Prolog program, which, as we have said, may or may not reflect perceived ambiguity when the sentence is replaced in its discursive context).

Let us not be mean, let's give at least a toy example... Suppose we feed Prolog the following program:

```
noun(regina, agreement(case:nominative,gender:feminine, number:singular)).
noun(reginam,agreement(case:accusative,gender:feminine, number:singular)).
noun(reginae, agreement(case:nominative,gender:feminine, number:plural)).
noun(reginas,agreement(case:accusative,gender:feminine, number:plural)).
```

```
noun(fatum, agreement(case:nominative,gender:neuter, number:singular)).
noun(fatum,agreement(case:accusative,gender:neuter, number:singular)).
noun(fata, agreement(case:nominative,gender:neuter, number:plural)).
noun(fata,agreement(case:accusative,gender:neuter, number:plural)).
```

```
adjective(clara, agreement(case:nominative,gender:feminine, number:singular)).
adjective(claram,agreement(case:accusative,gender:feminine, number:singular)).
adjective(clarae, agreement(case:nominative,gender:feminine, number:plural)).
adjective(claras,agreement(case:accusative,gender:feminine, number:plural)).
```

```
adjective(clarum, agreement(case:nominative,gender:neuter, number:singular)).
adjective(clarum,agreement(case:accusative,gender:neuter, number:singular)).
adjective(clara, agreement(case:nominative,gender:neuter, number:plural)).
adjective(clara,agreement(case:accusative,gender:neuter, number:plural)).
```

The above are *facts*, the first type of Prolog *clause*. They are made up of a *functor* (here, *noun* or *adjective*) with a given *arity*, i.e. number of elements within its domain, known as *arguments* (in this case, 2). In our bundle of facts, the first arg(ument) is *atomic* (to simplify drastically: a word or a number) and the second takes the form of a functor with its own arg(ument)s. The functor is *agreement*, and the args take the form of *features*, i.e. pairs of *feature name: feature value*, both atomic in this case (e.g. *gender:neuter*).

We add a *rule* to our program (the second form of Prolog clause). It reads as follows :

```
pair(First, Second,Agreement):-
( (adjective(First,Agreement),noun(Second,Agreement)) ;
(adjective(Second,Agreement),noun(First,Agreement)) ).
```

A few words of explanation are in order:

*pair(First, Second,Agreement):-*

% means: we can conclude (regard it as a *fact*)  
% that we have a *pair* of two members, *First* and *Second*  
% (these are variables, opening with a capital letter as variables do in Prolog),  
% exhibiting an agreement triplet referred to as *Agreement*.  
% Note that the latter too is a variable, and that the variable name  
% gives no information to Prolog as to its contents – *X* or *Y* would have conveyed as much  
% information, the program writer being the only one who knows that he means this variable  
% to stand for an agreement triplet such as *agreement(case:accusative,gender:neuter, number:plural)*

% *if and only if* (that's the meaning of ':-')

( (*adjective(First,Agreement),noun(Second,Agreement)*) ;  
 (*adjective(Second,Agreement),noun(First,Agreement)*) ).

% we have (i.e. in our data base) a pair adjective-noun,  
% in either order (the operator 'OR' is written ';' in Prolog)  
% (bracketing being necessary because each branch of the alternative is made up of *two* clauses)

% and the two agreement triplets are UNIFIABLE (here unification boils down to identity – see  
% below for a fuller treatment).

Once the program has been fed into the Prolog database, we can enter queries such as

a) (query type: is this true?) e.g. *pair(clara,regina,\_)*. Answer should be 'true'  
*pair(fata, clara,\_)*. Answer should be 'true'  
*pair(reginas, clara,\_)*. Answer should be 'false'

In the above queries, the underline (    ) lets Prolog know that we are not interested in the value assigned to the argument, in this case the agreement triplet.

b) (query type: what values should the variables get in order to make this true ?)

e.g. *pair(clara,regina,Accord)*.

*Accord* will be instantiated to *agreement(case:nominative,gender:feminine, number:singular)*.

Try to guess what the following query is likely to yield:

*pair(Premier,Second,agreement(case:nominative,\_,\_))*.

The answer being given on the next page, pause a few seconds before moving on.

The answer consists in a series of pairs ranging over the available vocabulary, all in the nominative case. The 'false' at the end of the list means that there are no more answers, as far as Prolog knows (i.e. has been told).

Note that the order in which the answers are given reflects the order Prolog follows in exploring the data base; in fact, it reads the way we do: from left to right and from top to bottom (to get the answers at the terminal we should press the ';' key after each pair).

```
Premier = clara,  
Second = regina ;  
Premier = clarae,  
Second = reginae ;  
Premier = clarum,  
Second = fatum ;  
Premier = clara,  
Second = fata ;  
Premier = regina,  
Second = clara ;  
Premier = reginae,  
Second = clarae ;  
Premier = fatum,  
Second = clarum ;  
Premier = fata,  
Second = clara ;  
false.
```

**UNIFICATION** is a simple and powerful mechanism. It accomplishes two things : verifying structural compatibility and retrieving and assigning information. In ALP we make use of a feature-unification algorithm, which turns straight Prolog unification into a less rigid mechanism but is firmly based on standard unification all the same (see Gal et al. 1991).

*Feature* here is not to be understood as restricted to *atomic binary feature* such as the *singular-plural* pair to capture *number*. A feature as we understand it in ALP has an atomic feature *name* all right (such as *number*) but can take as *value* any structure recognizable by Prolog, i.e. any Prolog *term*. Such structures include *lists* and *trees*, and well-nigh anything the linguist can dream of ever wanting to use.

Let us give a simple example, not related to linguistics or to ALP. Let's build a structure whose functor-name is *suite* and whose argument is a three-element *list*.

*Lists* are sequences of *elements* enclosed in square brackets. The *elements* can be *atomic*, or themselves be *structures*, *lists*, or any other *Prolog term* (a *term* being anything Prolog recognizes as its own, remember). Examples are:

<code>[albert,bernard,camille,didier, z adig]</code>	(atoms)
<code>[Ten, Nine,One, Two, Three]</code>	(variables)
<code>[first(One,1,Next), [X,Y,z], A,B,C, [z adig], auteur(z adig, voltaire)]</code>	(complex)
<code>[]</code>	(empty list)

Note that lists are explored by means of the operator '|', which divides the list into *Head* and *Tail*.

The *Head* is a list element, or several such elements, separated by commas, as are all list elements; the *Tail* is the remainder of the list, and is itself always a list.

Unification can be used to show '=' at work:

$$[a,b,C,d,e,f] = [A,B|Queue].$$

The unification of the two lists succeeds, with the following variable instantiations:  
 $A=a, B=b, Queue=[C,d,e,f].$

Examples of our *suite* structure would be

$$\begin{array}{l} suite([semel(One), bis(Two), ter(Three)]) \quad \text{and} \\ suite([semel(1), F, ter(3)]) \end{array}$$

Recall that variables in Prolog open with a capital letter, so that *One*, *Two*, *Three* and *F* are variables. If we use the operator for straight Prolog unification, which we have seen to be nothing else than the equal sign, we can write:

$$suite([semel(One), bis(Two), ter(Three)]) = suite([semel(1), F, ter(3)])$$

entering it as a Prolog *query*, i.e. asking Prolog to carry out the unification.

Unification will first check *structural compatibility*. We have two structures here whose functor is *suite* and whose argument is a list, so that they are compatible if the lists themselves are compatible. The *arity* of the lists is the same: 3 (recall that the arity is simply the number of constituents). The two lists will therefore be compatible if each of their elements is compatible. The first is a structure with functor *semel* and a single argument. So far so good, but the arguments must themselves be compatible, i.e. unifiable. This is the case (meaning: they ARE unifiable) because variables (and *One* is a variable) are unifiable with anything, therefore variable *One* is unifiable with the numeric atom found in the corresponding slot, i.e. *1*. The unification succeeds by giving the value *1* to the variable *One* (we say that *One* is *bound* or *instantiated* to *1*). In a similar way, in the second list element, the variable *F* will be bound to the structure *bis(Two)*, the variable *Two* continuing *unbound*. In the third list element, *Three* will be bound to *3*. The resulting structure (i.e. the result of the unification process) will be:

$$suite([semel(1), bis(Two), ter(3)]).$$

It's time to give an example of feature unification from the real world, i.e. in our case parsing.



We need a *lexical look-up* process, which associates information to be found in the lexicon with the words encountered in the text:

```
[lex,words] ->[recorded(pos,position(A,B,Word),_),  
lex(Word,Box,FS),  
map(Box,[from:A,to:B|FS])].
```

In this unusual construction, where left of the arrow (→) we find information that does not lead to any action, we accordingly need to concentrate our attention on what follows the arrow.

Structurally, it is a list. The elements it contains (three) are actions to be performed, i.e. calls to Prolog predicates defined somewhere in the program (facts or rules).

The first such call (to the *recorded* clause) specifies that a feature (the second argument of the *recorded* clause) must have been stored in the data base in the box specified as first argument (*pos* here, a *box* being a named bundle of records), corresponding, unsurprisingly, to the lexical item's *pos* (position).

The feature in question (*position(A,B,Word)*) is a three-arg structure: the start and end positions of the item in the sentence are recorded in the first two arguments, the variables *A* and *B* pointing to the beginning and end positions of the word in the wordlist corresponding to the sentence. The *Word* variable refers to the word found between these two positions in the wordlist representing the string.

For instance, if the sentence is '*Habent sua fata libelli.*' (the third example sentence in our test file, credits to Terentianus Maurus), the corresponding wordlist will be [*habent,sua,fata,libelli*] and *fata*, for instance, will be found to occupy the third position, i.e. from 2 to 3 (the count beginning at 0). As a matter of fact, the production of the wordlist corresponding to the input string will already have inserted the positions in the resulting wordlist (as well as specifying the end position by means of the *endpos* feature):

```
[0/habent,1/sua,2/fata,3/libelli,endpos(4)]
```

In this instance, the positions of the word and the morphological variant will have been stored in a record belonging to the *pos* box ('position' records), yielding here, by instantiation: *position(2,3,fata)* (variable *A* being instantiated to 2, *B* to 3, and *Word* to *fata*).

We then retrieve information on *fata* from the lexicon (*lex(Word,Box,FS)* (i.e. the set of *lex* clauses). The lookup (carried out by straight Prolog unification) yields two such Prolog clauses, one for nominative and one for accusative:

```
lex(fata, noun, [pos:noun, txt:fata, lex:fatum, case:nom, gender:neuter, class:common, number:pl, sem:[abstract]]).  
lex(fata, noun, [pos:noun, txt:fata, lex:fatum, case:acc, gender:neuter, class:common, number:pl, sem:[abstract]]).
```

the pattern being *lex(Textual\_form,Record\_Box\_in\_db,Feature\_List)*

The *lex* clauses have arity 3, i.e. three arguments: the first is the morphological variant itself (*fata*), the second the POS (Part of Speech, this time) and the third is a list of features, with information on part of speech, text form, lexeme, case, gender, number, class, and a list of semantic features, a single one in this case, namely 'abstract'.

The three arguments will instantiate the variables *Word*, *Box* and *FS*:

*Word* = *fata*, *Box* = *noun*, and *FS* will be instantiated to the first feature list (that for *fata* as nominative: [pos:noun, txt:fata, lex:fatum, case:nom, gender:neuter, class:common, number:pl, sem:[abstract]])

Note that such an assignment will eventually lead to failure (its fate...), *fata* being an accusative in our sentence; as will be explained in the next section, backtracking will occur, and the second feature list will come to bind the variable FS.

We then record the information (ending the look-up procedure) in a box (i.e. a collection of records) whose name is that of the POS, i.e. in this case 'noun' : map(Box,[from:A,to:B|FS]).

The feature bundle FS ([pos:noun, txt:fata, lex:fatum, case:nom, gender:neuter, class:common, number:pl, sem:[abstract]] ) will come to be included in a list whose first two elements will be the positions within which the item was found in the user's text. Variable binding will therefore yield:

*map(noun, [from:2, to:3, [pos:noun, txt:fata, lex:fatum, case:nom, gender:neuter, class:common, number:pl, sem:[abstract]]])*

In ALP unification is thus used to associate morphological variants with lexical items, to retrieve the argument structure of predicates within their entries, to ensure gender, case, number compatibility, and well-nigh everything else. Matching the argument list of a predicate will simply mean going down the list, picking each element and trying to unify it with what we find in the string submitted to analysis. Unification, with the added flexibility of feature unification as implemented in ALP, can deal with the assignment of decorated tree structures to utterances, i.e. parsing. The decoration will be mainly lexical (the words as leaves), but any type of added information (e.g. semantic) can be envisaged, as long as it is computable (for instance, in ALP, the weight assigned to each parse, which leads to the selection of the best parse or parses).

**BACKTRACKING** plays an important part in the tracking of all the possible solutions to a given problem. Seeing that language (especially with respect to a given grammar) is highly ambiguous, both globally (at sentence level) and locally (within the structure of phrases that will be included at a higher level, where further choices will operate), it is essential for the parser to be able to come up with all the solutions licensed by the grammar it embodies.

Backtracking is the mechanism by which Prolog keeps track of every single choice point in the search tree. Whenever a goal fails, Prolog backtracks to the last choice it made in its attempt to solve the goal, and chooses another branch of the search tree, if there is any that is still unexplored. If all fail, Prolog moves up one step further up the tree, and tries another branch up there. While doing this, it also uninstantiates any variable that got instantiated while Prolog was exploring the branch that led to failure.

In order to find all possible solutions, we can simply store the current solution, and force failure, and thereby force backtracking to occur. As a matter of fact, Prolog itself provides what are known as *second-order predicates* to build a list (including a sorted list) of all solutions. In ALP we use sorting on the weight in order to get the best solutions first, if there are more than one.

One might wonder what kind of profit to expect from a Latin parser. After all, we do not need such a tool as a first step towards machine translation, the texts we are interested in here (classical Latin texts) having been translated and retranslated, commented and over-commented.

The profit we can derive is directly linked to the absence of pressure of any kind. We do not NEED such a parser, so that we can concentrate on what a parser can teach us about language. To be usable as the basis for a parser, a grammar needs a degree of explicitness which forces it to come to grips with a good number of issues that are likely to have been considered irrelevant or to have been relegated to stylistics, i.e. quirks and idiosyncrasies left over to expressive power and the like. A major issue that has to be dealt with is the amount of freedom in word order – what are the limits that need to come to be part of an algorithmic treatment? How does syntax interact with semantics and pragmatics, in a way that can be shown to improve coverage, i.e. increase the part that can be dealt with algorithmically?

The development of a parser for Latin does not pursue any practical aim. It can be conceived as a contribution to the study of the language, in a spirit of free enquiry, which also means freedom from any pressure that does not directly derive from the subject under scrutiny.

The above considerations militate in favour of a parser whose design keeps grammar and parsing algorithm as separate as possible. We achieve that aim by relying entirely on a *production system* as parsing algorithm. The production system is organized in *passes*. Each pass is implemented as a series of production rules which operate over and over again until they are unable to produce anything new, in which case they pass control over to the next pass.

The production rules are allowed to build structure on the basis of what is available to them. In the first pass, the lexical pass, the words in the text are paired with the information stored about them in the lexicon (we have seen that this is a matter of straight variable instantiation). When we claimed above that the reading of *fata* as an nominative case would lead to failure, we certainly did not mean that this happened at the lexical look-up stage, where there is absolutely no information available to reject the nominative case or prioritize the accusative. The information is simply stored and made available to the next pass (higher in the structure-building hierarchy).

The grammar passes will likewise proceed from simple structures to more complex ones that need the information provided by the simple ones. Again, all the production rules in a given pass are allowed to produce structure over and over again, until they have nothing to add at their level.

Let's look at a very simple production rule to be found in the first grammar pass, that for building one-word nps such as *rex* in *rex scribit epistulam*.

The comments included in the Prolog program provide basic information about the np building procedures :

The core NPS are assembled before the other NPs, for which they can serve as building blocks.

There are indeed two passes for nps: *core* and *finite*.

The core NPs are simple nps that do not involve predications, therefore no relatives, no arg-bearing nouns, just the simple buiding blocks: nouns as nps, names as nps, adj+n as np, and so on...

Each np is associated with an index which refers to the positions it spans in the input string

The index is useful to make sense of *gaps*, i.e. *traces* (*t* or *e* in syntactic parlance) 'left' by elements 'moved out of place' by 'transformations'. The quotes are meant to show distance with respect to the syntactic theory underlining such treatment.

But undoubtedly a similar treatment is needed. If the trace cannot be associated with the relative pronoun, and, via the relative, and more importantly, with the antecedent, all the controls we wish to perform, such as

semantic controls on arg bearers, will prove impossible in relative clauses, to give one example.

We then proceed to the production rules for simple nps, and begin with nps consisting of a single noun. We give below the *lex* clause for our word *rex*

```
lex(rex, noun, [pos:noun, txt:rex, lex:rex, case:nom, gender:masc, class:common, number:sing, sem:[hum]]).
```

The relevant production rule is the following:

```
[core,np1] ->
[mapped(noun,[from:A,to:B|FS]),
constraint([pos:noun,lex:Lex,class:common,sem:Sem,txt:Text,
           number:Nb,gender:G,case:C],FS),
map(np,[pathlist:[p(A,B)],hp:[p(A,B)],index:i(p(A,B)),distance:[0],
        cat:np,sem:Sem,class:common,lextype:full,
        number:Nb,person:3,gender:G, type:core,lex:Lex,txt:Text,
        case:C,w:1])].
```

Which basically means that if we have in our text (we have it *mapped* by a previous *production*), from position A to position B (remember that capitals are reserved for variables in Prolog – A and B are thus variables), an item that was placed in the noun box and is associated with *feature bundle* FS, then we can use the predicate **constraint** to check or retrieve information from that feature bundle: we need here a common noun, a full lexical item; we retrieve the information contained in the feature bundle regarding *number, gender, case, textual form* and *semantic class: sing, masc, nom(inative), rex, [hum]*. We can then allow the production rule to produce (via the predicate **map**) a record to be put into the np (noun phrase) box.

Such a box will include information about the path covered by the np, the head of such a path, the index in case we need it somewhere down in the structure-building process (for instance if *rex* was to be found to be the antecedent of a relative pronoun), the distance within the path (in case of non-contiguity of the constituents), morphological and lexical information derived from the lexical item. We add person (third person), a type (core np) and a weight (1).

This may seem to be a very heavy procedure just to account for what is dealt with in a couple of rewrite rules in a top-down or bottom-up parser, namely

```
np->n, n->rex      or      rex->n, n->np.
```

But in fact all the other information we gather and transport via the production rules based on feature unification will prove to be useful or downright indispensable in any sophisticated parser designed for a nearly free order language such as Latin.

The important design decision is to select a process (such as the one embodied in production rules) that boils down to monotonous incrementation of the available information. This does not prevent us from using specifically designed algorithms for ancillary tasks.

In short, we try to combine a parsing algorithm that is reduced to monotonous structure incrementation through a production system with various procedures that compute the quality rating of the structures licensed by the grammar rules and the information embodied in the lexical items (for instance the argument structure of a predicate, which has predictive power on how the string elements need to be structured into phrases of various levels such as nps and clauses).

A linguist will surely find that there is a huge distance between the type of grammar he is used to

writing and the one he is confronted with in ALP. Well, there is a price to pay – an algorithm embodying a grammar cannot be a grammar written without an idea of how it is to be used in parsing. What we can attempt to do is to make the grammar as *declarative* as possible, i.e. as independent as we can make it from issues of control, of how it is to be used, in what order its rules are to be applied, where structures are to be stored, what should be done in case of failure, and the like. As soon as we attempt to go beyond toy systems, we have to dirty our hands a little and think about issues like the degree of freedom there really is in an 'order-free' language such as Latin, and what to do to assess the quality of the parses delivered by the system. The profit we will draw from such an effort is that we will increase explicitness, and have a much better idea of coverage, i.e. how much of the language can be captured by our rules.

Finally, is there nothing to be said *against* ALP as a parser for Latin? Well, there is one negative point, which, if we were parsing anything else but a dead language, would be rather devastating. The production system sketched here is *inefficient*<sup>2</sup> – its very monotony (it does one single thing: increase available information) is at that price.

We should bear in mind the reasons for which one may want to produce a Latin parser – I can see two main reasons only: teaching and research. The quicker the better shouldn't be our motto, surely.

---

<sup>2</sup> It would seem that at present a fifteen word limit should be imposed on strings to be parsed, unless time does not matter AT ALL, which is seldom the case...

# Parsing Issues

## Grammatical Sketch

The heart of the grammar implemented in ALP revolves around the *predicate* and its *arguments* building a *clause* (a grammatical clause, not to be confused with a Prolog clause). The clauses can be *finite* or *non-finite* (infinitive, participial) and can contain clauses as constituents. The arguments taking the form of *phrases* (adjective phrases, noun phrases, prepositional phrases) can also contain clauses, for instance under the guise of relative clauses attached to noun phrases. Recursivity is also to be found at the level of the phrases themselves, since a prepositional phrase is best defined as a preposition governing a noun phrase, and a noun phrase itself can contain prepositional phrases. The ease with which Prolog handles recursivity is a major pluspoint in its use in the implementation of a grammar for a natural language such as Latin.

The association of a predicate and its arguments is a matter for the lexicon to handle, but the description of the structural make-up of the arguments and the constraints imposed on them (e.g. semantic) must be such that the grammar can tackle them – the interaction between grammar and lexicon must be total, with no piece of information in either that the other cannot 'understand', i.e. register or make use of.

We will attempt to show this by looking at two lexical entries for verbs and the requirements that they impose on a grammar capable to match them.

The lexical entries are those for *obliscor* and *timeo*. We are not concerned here with the mechanisms building all their morphological variants, but with their argument list, which are housed in the relevant *lexarg* clauses:

```
% OBLIUSCI
% Non obliviscar sermones tuos - Pascal, Mémorial.
% Oblita est periculi ancilla fortior dominis multis.
% Obliscitur rex reginam longas epistulas scripsisse ancillae Marci.

lexarg(obliscor,
  arglist:[ws(obliscor_forget,tr_cod,clause:[],mwuw:0,
    args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
    object:[type:np,oblig:no,constraints:[case:or([acc,gen])]])]),
  ws(obliscor_forget_that,tr_inf,clause:[],mwuw:0,
    args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
    object:[type:pred,oblig:yes,constraints:[type:nonfinite]])]).
```

The *arglist* opens with an indication of the word sense being described; here we have two word senses, *forget* and *forget\_that*. The information that follows concerns the *category* (transitive verb with np object, transitive verb with clausal object), the *constraints* it imposes on the clause in which it fits (here, the empty list indicates that there are no such constraints), its *weight* as a multi-word unit (0, since it is not a mwu), and then the *arglist* proper, the feature whose name is *args* and whose value is a list of arguments.

We should emphasize that the members of the *arglist* need not be found in the string to be parsed in the order in which they appear in the *arglist* (a canonical order used for easier maintenance of the

lexicon). As a matter of fact, in our second example for *obluiiscor* the genitive object precedes the subject, and in the first example (the one drawn from Pascal's *Mémorial*), the subject does not appear in the sentence at all, but is projected from the verb phrase (first person subject).

In the first wordsense recorded here, the args are subject and object, the first obligatory and the second optional (in which case it can be argued that it is mostly context-retrievable), and both structurally nps. The constraints on the subject are semantic (the subject must bear the feature +HUM), and the constraints on the object are case-related. The default cases are of course nominative for subject and accusative for object. But the object of *obluiiscor* can also be in the genitive case, so we need an OR-value for case: either accusative (as in the example from Pascal's *Mémorial*) or genitive (as in our second example).

In the second wordsense, the arglist specifies a clausal object, a non-finite clause (accusative cum infinitive), as in our third example.

We see thus that our grammar must be able to structurally characterize nps and assign them functions within the clause they operate in. They must also receive a semantic description, and have been assigned a case. The grammar must also deal with clauses in arg position, both finite and non-finite. For an example of a finite clause as arg we can turn to the entry for *timeo*:

```
% TIMERE
% Timeo Danaos etiam dona ferentes.
% Timeo amicis meis.
% Timeo ne veniant ad urbem capiendam.

lexarg(timer,
  arglist:[ws(timeo_fear,tr_cod,clause:[],mwuw:0,
    args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
      object:[type:np,oblig:yes,constraints:[case:acc]]]),

    ws(timeo_fear_for,tr_cod,clause:[],mwuw:0,
      args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
        object:[type:np,oblig:yes,constraints:[case:dat]]]),

    ws(timeo_fear_that,tr_cod,clause:[],mwuw:0,
      args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
        object:[type:pred,oblig:yes,
          constraints:[type:finite,mood:subjunctive, argbound:yes,subordinator:or([ne,ut])]]))].
```

In the case of *timeo*, we have three distinct word senses, each with its own arglist; we have a general *fear*, a *fear for* and a *fear that*. The constraints on the args have already been discussed, except for the new type of arg associated with *timeo* as *fear that*. The object is again a whole clause (type:pred), with its own list of constraints: it must be a *finite* clause, in the *subjunctive* mood, bound to that argument and opened with subordinator *ne* or *ut*. All requirements that the grammar must handle properly for the word sense to be captured.

The above examples give a very rough idea of the granularity problems that arise in the pairing of a lexicon (whose granularity should extend to the wordsense) with a grammar.

## Dealing with Multi-Word Units

Another relevant example is the treatment of multi-word units. First, it should be clear that the description we give in the lexicon must allow their insertion into the grammatical framework we are implementing. Second, if their behaviour is not constrained in any way that the grammar is able to capture, their being read as mwu's must be given priority over the readings where they are just standard grammatical strings, although the latter readings cannot be excluded.

Consider a simple mwu such as *res nouae* (revolution). We enter it in the lexicon as follows:

```
% RES NOVAE (revolution)
% cupiditate regni adductus novis rebus studebat (Caesar, De Bello Gallico, 1.9.3)
[core,np2aii] --->
[mapped(noun,[from:A, to:B]FSnoun)],
mapped(adj,[from:X, to:Y]FSadj),
constraint([number:pl,gender:fem,case:Case,lex:res],FSnoun), % plural needed, of course
constraint([number:pl,gender:fem,case:Case,lex:nouus],FSadj),
adjacent([p(A,B)],[p(X,Y)]), % adjacency required res nouae or nouae res
append([p(A,B)],[p(X,Y)],Path),
msort(Path, Sorted),
map(np,[pathlist:Sorted,hp:[p(A,B)],index:i(p(A,B)),distance:[0],cat:np,class:common,sem:[abstract],
      number:pl,person:3,gender:fem,type:core,lex:res_nouae,lextype:full,
      case:Case,w:3])).
```

This entry specifies that the noun *res* and the adjective *nouae* should be adjacent (as opposed to the the adjective-noun nexus, where the two elements can be separated from each other: *res inuenit nouas*, 'he found new things'). The number is not free either: it must be plural, as opposed to the unspecified number of a standard adjective-noun nexus: *noua res*, 'a new thing'). Third, of course, the lexemes are specific: the noun must be *res* and the adjective must be *nouus*. If the relevant constraints are satisfied, we build a standard np, to which we assign a specific semantics (standard *res* can be sem:[thing]) and a specific lexical value (*res\_nouae*). And of course we increase the weight assigned to the np. An np made up of a noun and an adjective will have weight 2 and the weight we assign to *res\_nouae* is 3.

Let's now consider a more complex mwu, i.e. one with wider, less local constraints.

```
% ALIQUEM/QUOD (NON) PILI FACERE
% Praetor non amabat milites nec faciebat pili cohortem.

lexarg(facere,
      arglist:[ws(mwu_non_pili_facio_not_give_a_damn,tr_cod_cp1t,clause:[polarity:neg]),mwuw:2,
      args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
      object:[type:np,oblig:yes,constraints:[case:acc]],
      object_cp1t:[type:phrase,frozen:yes,oblig:yes,constraints:[lex:pili]]]),
```

Remember Catullus, 10, 12-14? *praesertim quibus esset irrumator / praetor, nec faceret pili cohortem*. (not least when said praetor was a fuckface / and didn't give a shit for his poor staffers. - translation Peter Green).

*Pili facere* needs to be inserted in a non-affirmative context (provided here by *nec*, but there are other ways of making a context non-affirmative, such as *quid obstat quominus/quid est causae quin*). We therefore introduce a clause-level constraint, on clause polarity: clause:[polarity:neg]. We give a bonus weight to the verb (2). The object complement is a phrase, quite frozen, and whose lexical specification goes down to the word-level: we need *pili*, nothing else will do (the lexicon has an entry for the relevant phrase, namely *string(phrase,[pili],[lex:pili,w:1])*).



## Linearity

Let us consider the *contiguity check* algorithm as an example. In Latin, elements that belong together need not be found side by side in the string. A standard example is the Vergilian *patulae recubans sub tegmine fagi*, where the adjective *patulae* belongs to the noun *fagi* (both singular, both feminine, both genitive – the agreement triplet that we require to hold on the adjective-noun nexus). If we allow elements belonging together to be dispersed in the string, we won't find it too difficult to account for the np, precisely on the basis of the triple agreement required on the adj+n group. But notice that there is nothing in the intervening material (*recubans sub tegmine*) that would offer a better link for the adjective *patulae*. Just as *fagi* is justifiable as genitive attached to *tegmine*, and *tegmine* as governed by *sub*, and *sub tegmine* as place adjunct for *recubans*, *patulae* is justifiable as adjective attached to *fagi* – there is no other attachment to compete with the one that involves non-contiguity between adjective and noun.

In the case of competing links we need to assess the weight to be attached to each link and select the heavier of two or the heaviest among more than two. How do we proceed?

The first thing to notice is that first and foremost we need to keep track of where each string element is to be found in the string. We have seen how to do this with the algorithm computing positions while turning the string into a *wordlist*.

Now, to be able to use that information the way we should, we need to keep track of the position of the material covered in the case of any structure that gets superimposed on lexical material. We store this information in the *path* feature. If, in *bonus rex scripsit epistulas malas*, we build a np to cover *bonus rex* and another one to cover *epistulas malas*, we need to record in the first np a path extending from position 0 (in front of *bonus*) to 2 (at the end of *rex*), and in the second np a path extending from 3 to 5.

We also need to be able to say where the head of a structure is to be found. In the case of our two nps, the head is the noun and we record positions 0 to 1 for the first head (*rex*) and 3 to 4 for the second (*epistulas*).

We can now discuss the algorithms relating to path, distance and contiguity. We go straight to the Prolog program, where the comments are supposed to do part of the job of explaining how the algorithms work. We add a few more comments to make the procedure as explicit as we can.

```
% these procedures examine which parts of the string are covered by various elements
% they are meant to measure properties like adjacency, contiguousness and distance
```

```
% the path is a list of p(X,Y) structures, where X and Y stand for positions in the string,
% as computed when the string is entered in and processed.
```

## FINDING A PATH THAT COVERS THE WHOLE SENTENCE

```
% all the words of a string must be used up for a parse to be considered valid for the string
% no gap left and the end of the sentence must be reached
```

```
% Begin and End are the extremities of the pathlist (0 and whatever fin(Fin) records)
```

The *fin(Fin)* predicate stores in the variable *Fin* the last position in the string. It is taken care of by the process turning the string into a wordlist.

```
path(Begin,End,Pathlist):-
    pick(p(Begin,Next),Pathlist,RPaths),
    path(Next,End,RPaths).
```

```
path(E,E,[]).
```

The procedure applies recursively to its third argument. The second clause for the predicate (i.e. `path(E,E,[])`) gets us out of recursion: when all the elements have been picked out, we are left with an empty list (`[]`) whose beginning and end are the same (single variable *E*, the end of the path). More on this below.

As for *pick*, it is a three-arg procedure. It arbitrarily picks up an element (first arg) in a list (second arg) and returns what remains of the list once the selected element has been taken out (third arg). It can be defined as follows:

```
pick(H,[H|T],T).
pick(X,[H|T],[H|T1]) :- pick(X,T,T1).
```

In both *path* and *pick* use is made of the data structure *List* and the core of the procedure is recursive, i.e. calls on itself in its very definition. This is quite a standard way to proceed in Prolog. Lists are explorable by means of the operator '|', whose second argument, remember, is always a list. We can recur on that list until we are left with the empty list, which we use in a defining clause which gets us out of recursion. Consider *pick* in that light. We begin with the simplest case one can think of: picking an element out of a list is achieved by selecting the element which is easiest to grasp, i.e. the head of the list, the element to the left of the operator '|'. The element picked is thus *H*, and the remainder of the list, which is a list, is *T* (the tail of the list). But the element to be picked can be anywhere in the list; we can select it by picking it out of the remainder of the list, which is what the second defining clause says: leave the head(*H*) well alone, and pick your *X* element in the remainder of the list, *T*. The list you will get as a result of picking out element *X* out of tail *T* we will refer to with variable *T1*. Therefore the list that should be returned as result of the picking is head *H* and tail *T1*, that is to say `[H|T1]`. Here the end of recursion clause does not refer to the empty list, because there is nothing to pick in an empty list – we need at least one element, and the pattern `[H|T]` therefore applies to the list in the clause that gets us out of recursion, in this case the first one.

In *path*, however, we need to explore the whole length of the pathlist. We need to pick elements until we reach the very last position referred to as *End* in the first clause. At that juncture the Pathlist must be empty, and we get out of recursion by means of the second defining clause for *path*. The *Begin* and *End* point must be the same, since we then start from the end-position of the very last element in the path. We stay put, having reached our goal. The path is really a path because we cannot pick a new element (a new p-structure) unless its beginning point correspond to the endpoint of the preceding p-structure. Backtracking ensures that a path will be found if there is a path to be found, i.e. all the p-structures can be joined by sharing a position (endpoint of one is start of the following).



Notice the `\+`, which is used in Prolog for negation. `\+mapped` is used to indicate that no element of the box noun exists that covers the intervening element, i.e. that the intervening element is not a noun.

% with respect to CASE GENDER and NUMBER

% when we try to relate adj and noun

% we are not likely to be allowed to jump a noun with all the right properties in terms of

% case gender and number:

% *putabas malas servas amicas reginae fuisse*

% *malas* is not likely to link with *amicas* by 'jumping' *servas*

% in the code below we use the cut-fail pair (`!,fail`). The cut (`!`) prevents backtracking, so that the *fail*

% that follows cannot be undone; the predicate being defined fails if a noun is found where it shouldn't be.

% Both the cut and the predicate *fail* are pre-defined

```
relaxedadjacent1_cgn(PL1,PL2,Case,Gender,Nb):- member(p(_,Y),PL1), member(p(X,_),PL2),
      succ(Y,X),
      mapped(noun,[from:Y,to:X|FSnoun]),
      constraint([case:Case,gender:Gender,number:Nb],FSnoun),
      !, fail.
```

```
relaxedadjacent1_cgn(PL1,PL2,Case,Gender,Nb).
```

```
relaxedadjacent2_cgn(PL1,PL2,Case,Gender,Nb):- member(p(_,A),PL1), member(p(C,_),PL2),
      succ(A,B),succ(B,C),
      ( mapped(noun,[from:A,to:B|FSnoun1]),
        constraint([case:Case,gender:Gender,number:Nb],FSnoun1) ) ;
      ( mapped(noun,[from:B,to:C|FSnoun2]),
        constraint([case:Case,gender:Gender,number:Nb],FSnoun2))),
      !, fail.
```

```
relaxedadjacent2_cgn(PL1,PL2,Case,Gender,Nb).
```

```
relaxedadjacent3_cgn(PL1,PL2,Case,Gender,Nb):- member(p(_,A),PL1), member(p(D,_),PL2),
      succ(A,B),succ(B,C),succ(C,D),
      ( mapped(noun,[from:A,to:B|FSnoun1]),
        constraint([case:Case,gender:Gender,number:Nb],FSnoun1) ) ;
      ( mapped(noun,[from:B,to:C|FSnoun2]),
        constraint([case:Case,gender:Gender,number:Nb],FSnoun2));
      ( mapped(noun,[from:C,to:D|FSnoun3]),
        constraint([case:Case,gender:Gender,number:Nb],FSnoun3))),
      !, fail.
```

```
relaxedadjacent3_cgn(PL1,PL2,Case,Gender,Nb).
```

## PATH CONTIGUITY

% the various elements follow each other without leaving a gap

```
contiguous([]).
```

```
contiguous([One]).
```

```
contiguous([p(X,Y),p(Y,Z)|Tail]):- contiguous([p(Y,Z)|Tail]).
```

In words : an empty list is contiguous, i.e. does not feature non-contiguity...

A one-element list does not feature non-contiguity either

If the first two elements in a list are contiguous (the extremity of the first being the start of the second), then if the list made up of the second element and the tail of the list (all the remaining elements) is also contiguous, then the whole list is contiguous.

% in *quasicontiguous* we allow one element to be out of place

quasicontiguous(L):- contiguous(L), !. % *Qui peut le plus...*

quasicontiguous(L):- pick(EI,L,L1), contiguous(L1).

The *quasicontiguous* predicate can be applied as a check on non-finite clause constituency when dealing with poetry. The structures building such a clause must be found together, with the exception of a single word. This relaxed check on contiguity allows the parsing of the Horatian *Me tabula sacer votiva paries indicat uvida suspendisse potenti vestimenta maris deo*, where *Me* belongs to the non-finite complement clause of *indicat*: *Me ... uvida suspendisse potenti vestimenta maris deo*.

## DISTANCE BETWEEN TWO PATHS

% we first determine the end points of the two paths

% we determine the order in which they appear

% and then the distance between extremity of the first one and start of the second

```
distance(Path1,Path2,Distance):- extremity(Path1,Ext1), extremity(Path2,Ext2),
    start(Path1,St1), start(Path2, St2),
    ifthenelse(Ext1 =< St2, % IF
        Distance is St2 - Ext1, % THEN
        Distance is St1 - Ext2). % ELSE
```

% *extremity*: last position in pathlist

% we select the very last position registered, i.e. the second element of the p(X,Y) structure that ends the path

```
extremity(PathList, Ex):- last(PathList,p(_,Ex)).
```

% (*last(List,Last)* is true if *Last* is the last element of *List*)

% *last*, although *pre-defined* (i.e. part of the Prolog language) can be re(?)defined as follows:

```
last([Last],Last). % Last is the last element of a list which does not contain anything else
```

```
last(_|Tail],Last):- last(Tail,Last). % If there is more than a single element in the list, then Last is the
    % last element of the Tail of the list
```

% *start*: first position in a pathlist

% the first element of a list is easy to find by simple unification:

% we select the first element of the relevant p structure

```
start([p(Start,_)]_,Start).
```

% precedes(Path1,Path2)

```
precedes(P1,P2):- extremity(P1,Extremity), start(P2,Start), Extremity =< Start.
```

Consider the following line from Martial, 2.78:

*Aestivo serves ubi piscem tempore quaeris?*  
(You want to know where to keep fish in summertime?)

[0/aestiuo,1/serues,2/ubi,3/piscem,4/tempore,5/quaeris,endpos(6)]

```
vg
  selected_reading:quaero_ask
  polarity:pos
  cat:vg
  pos:v
  lex:quaerere
  voice:act
  tense:present
  mood:indicative
  number:sing
  person:2
subject
  source:context_retrievable
  number:sing
  gender:or([masc,fem])
  person:2
  cat:np
  index:i(0,0)
  constraints_to_be_met:[sem:[hum]]
  case:nom
object
  cat:pred
  illocutionary_force:question
  number:sing
  person:2
  mood:subjunctive
  tense:present
  polarity:pos
  argbound:no
  add:no
  flagint:wh_question
  c_str
    vg
      selected_reading:servo_keep_safe
      polarity:pos
      cat:vg
      pos:v
      lex:servare
      voice:act
      tense:present
      mood:subjunctive
      number:sing
      person:2
    subject
      source:context_retrievable
      number:sing
      gender:or([masc,fem])
      person:2
      cat:np
      index:i(0,0)
      constraints_to_be_met:[sem:[hum]]
      case:nom
    object
      index:i(p(3,4))
      cat:np
      sem:[thing]
      number:sing
      person:3
      gender:masc
      lex:piscis
      case:acc
    clause_level_adjunct
      cat:advp
      value:place
      lex:ubi
      c_str
        lex:ubi
        sem:location
        cat:advp
    clause_level_adjunct
      cat:np
      value:time
      number:sing
      person:3
      gender:neuter
      lex:tempus
      case:abl
      c_str
        head
          lex:tempus
          sem:time_when
          cat:np
          number:sing
          gender:neuter
          case:acc
          index:i(p(4,5))
      adj:aestiuus
```

Such a sentence seems to exhibit a completely free word order. But it suffices to run it through a string generator to come to realize that this is far from the case. A six-word sentence generates 6!, i.e. 720 strings, most of which are totally ungrammatical:

aestivo piscem quaeris serves tempore ubi .  
aestivo piscem quaeris serves ubi tempore .  
aestivo piscem quaeris tempore serves ubi .  
aestivo piscem quaeris tempore ubi serves .  
aestivo piscem quaeris ubi serves tempore .  
...  
aestivo tempore quaeris ubi serves piscem .  
aestivo tempore serves piscem quaeris ubi .  
...  
piscem quaeris tempore ubi aestivo serves .  
piscem quaeris tempore ubi serves aestivo .  
piscem quaeris ubi aestivo serves tempore .  
piscem quaeris ubi aestivo tempore serves .  
...  
quaeris piscem ubi aestivo serves tempore .  
quaeris piscem ubi aestivo tempore serves .  
quaeris piscem ubi serves aestivo tempore .  
...  
serves aestivo piscem quaeris tempore ubi .  
serves aestivo piscem quaeris ubi tempore .  
serves aestivo piscem tempore quaeris ubi .  
...  
tempore quaeris piscem aestivo ubi serves .  
tempore quaeris piscem serves aestivo ubi .  
tempore quaeris piscem serves ubi aestivo .  
...  
ubi aestivo quaeris piscem serves tempore .  
ubi aestivo quaeris piscem tempore serves .  
...  
ubi tempore serves quaeris aestivo piscem .  
ubi tempore serves quaeris piscem aestivo .

So, the task does not boil down to letting the 'anything goes' principle make havoc of Latin word order, but to open up the range of possible grammatical strings by relaxing the contiguity we expect structures to exhibit, while taking care not to allow the production of strings that would turn out to be impossibly ambiguous. The various path procedures enable us to contain freedom within reasonable (i.e. grammatical) limits.

## Producing, Storing and Retrieving Information

A few words may be in order about the *data bases* ALP uses. A first data base consists of the Prolog program itself, made up of clauses embodying both facts and rules.

This data base is increased by running the **makelex** program, which involves expanding its macro-clauses. This process yields new lexical clauses, resulting from the generation of the morphological variants for regular lexical items such as adjectives, nouns and verbs. The irregular or invariant forms are entered directly as *lex* clauses. Let us look at examples of both:

a) directly entered as *lex* clauses:

*semper* (invariable):

```
lex(semper,adv,[lex:semper,pos:adv,type:clausal, sem:time]).
```

*simus* (irregular)

```
lex(simus,v,[pos:v,class:v_esse,type:finite,lex:esse,
            voice:act,txt:simus ,tense:present,kind:std,mood:subjunctive,
            number:pl,person:1]).
```

b) generated on the basis of a macro-clause:

```
lex(rogabis, v, [pos:v, class:tr_cod, type:finite, lex:rogare, voice:act, txt:rogabis, tense:future, kind:std, mood:indicative, number:sing, person:2]).
lex(rogabit, v, [pos:v, class:tr_cod, type:finite, lex:rogare, voice:act, txt:rogabit, tense:future, kind:std, mood:indicative, number:sing, person:3]).
lex(rogabit, v, [pos:v, class:tr_cod, type:finite, lex:rogare, voice:act, txt:rogabit, tense:future, kind:std, mood:indicative, number:pl, person:2]).
```

If we go to the macro-clause itself, we see that it uses the root provided by a *verb* clause (in this case for the verb *rogo*) and the list of endings suitable for that verb, to generate the morphological variants. Each variant is then turned into the first argument of a *lex* clause, the remaining arguments being the Part of Speech and a list of features (tense, mood, person, etc.) to be associated with that particular variant. The resulting new bunch of *lex* clauses (such as the three above) are then asserted by the macro-clause, i.e. added to the Prolog data base.

The verb clause for *rogo* (entered as such as part of the Prolog program) reads:

```
verb([v(rogare,1,rog,rogau,rogat)],tr_cod,std).
```

```
% the v functor encompasses infinitive, conjugation and the three roots. We then have the verb class, and
% the indication that the verb behaves 'standardly' with respect to the production of morphological variants
```

The macro-clause involved is much too long for it to be given in full in this introduction. Suffice it to say that it needs access to the relevant roots and endings, and performs atom-concatenation to produce the morphological variants. The atoms it concatenates (i.e. chains together in the order specified) are simply the relevant roots and endings.

The process results in the production of 213 *lex* clauses for the morphological variants of *rogo*, among which the three given above. The full list is to be found at the end of this document, just before the Appendix.



As for the predicate-argument structure associated with the verb, it is the object of a specific clause for each verb. Such are the *lexarg* clauses, whose first argument is the arg-bearing element (in this case the verb *rogare*) and the second the *arglist feature*, whose value is a list of *word senses* accompanied by the arg structure they require.

% ROGARE

% Examples of the word senses accounted for in ALP  
% Eo auxilium rogatum. Ask for  
% Rogebant quae fortuna exercitus esset. Ask

```
lexarg(rogare,  
  arglist:[ws(rogo_ask_for,tr_cod,clause:[],mwuw:0,  
    args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],  
    object:[type:np,oblig:yes,constraints:[case:acc]]]),  
  
    ws(rogo_ask,tr_cod,clause:[],mwuw:0,  
    args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],  
    object:[type:pred,oblig:yes,  
    constraints:[type:finite,mood:subjunctive, flagint:or([yes_no_question,wh_question])])])]).
```

In the example for *rogare*, we distinguish between two wordsenses, to capture the distinction between *rogo* meaning *ask* and *rogo* meaning *ask for*. *Rogo* meaning *ask for* has an np as object, whereas *rogo* meaning *ask* governs a clause, with its own constraints: it must be finite, its mood must be subjunctive, and it must embody either a yes/no question, or a wh-question. The other requirements we hope do not need further explanation.

We now move on to discuss the second type of data base used by ALP.

This data base has a limited life-span. It is specific to each sentence being parsed, and is erased as soon as the sentence has been parsed. It houses all the productions that the production rules embodying the parser have generated all through the parsing process of that sentence, from working out the positions of the items within the string, down to the structures corresponding to a full parse of the input string.

It should be clear that while processing a sentence, no information yielded by the production rules is ever erased. The process is strictly incremental, producing bits of structure that will or will not contribute to the final parse or parses. That will be decided by the higher levels materialized by higher passes, the structures retained as valid parses having to span the whole string.

In such a system the linguist *describes* rather than sets about specifying the way the information he gives in the grammar and lexicon should be used (in what order, under what conditions, etc.). As soon as the coverage is to be more than strictly minimal, it makes a hell of a difference for the linguist's job if he hasn't to turn into a programmer each time he wants to add new lexical items, and, above all, new constructions.

The predicates for adding information to and for removing information from that second type of data base, are specific to it. Instead of the *assert/retract* pair (used for the permanent db), we have the *record/erase* pair. No confusion is possible.

## Mapping

To illustrate this process, we can look at the treatment of simple np groups made up of a noun and an adjective phrase as in *agricola doctior Petro*. We take up the analysis at a point where the adjective phrase has already been built and need to be attached to the noun to build the resulting np.

We give the commented Prolog code, adding some more comments in an attempt to persuade the reader to stay with us a few pages more...

```

% adj phrase following the noun

[finite,np2a] -> % category and name of the rule

[mapped(noun,[from:A, to:B]FSnoun)], % we have a noun in the word list corresponding to the input string
% it extends from position A to position B
% its feature bundle has been read off the lexicon
% in the case of agricola (taking it as a nominative), it would be
%[pos:noun, txt:agricola, lex:agricola, case:nom, gender:masc, class:common, number:sing, sem:[hum]]

mapped(adjp,FSadj),
% we have an adjective phrase (here covering doctior Petro) with its own feature bundle, sth along the
% lines of

% [cat:adjp,pathlist:Sorted,distance:Distance,hp:[p(A,B)],
%     case:Case,number:N, gender:G,lex:Lex,type:Type,w:Weight,
%     c_str:[Lex,comp_cplt:FSnp]]

% with the variables duly instantiated

constraint([number:Nb,gender:Gender,case:Case,class:Class,sem:Sem,lex:LexNoun],FSnoun),

% we select from the feature bundle associated with the noun
% the values for number, gender, case, etc.

constraint([number:Nb,gender:Gender,case:Case,lex:LexAdj,type:Type,w:W],FSadj),

% we select values from the feature bundle associated with the adjective phrase,
% leaving unification to check that the agreement triplet Number, Gender, Case holds

Type \= int, % interrogative adjectives dealt with separately - they need heavy weight

constraint([pathlist:Padj],FSadj),
% we retrieve the pathlist of the adjective phrase

append([p(A,B)],Padj,Pnp),
% we append i.e. concatenate the path for the noun and the path for the adjective phrase

extremity(Padj,Ext), % we select the endpoint of the path covered by the adjective

Ext > B, % here the adj phrase follows the noun, B being the endpoint of the noun

distance([p(A,B)],Padj,Distance), % the distance between the noun and the path of the adjective
% determines the straining factor as well as helping to decide
% whether noun and adj DO belong together
% the straining factor will contribute negatively to the weight assigned
% to the parse

ifthen(LexAdj=is, Distance=0), % is/eadid adjacent - this requirement is probably too strong

msort(Pnp, Sorted), % merge sort
\+dup(Sorted), % no duplicates – recall that \+ is the negation operator in Prolog

% we standardly apply this couple of procedures to paths
% first, we sort them ; second, we check that they do not contain duplicates

Distance < 4, % 3 is thus the maximum distance
% between adj and noun
% a Prolog call that acts as a barrier – if it fails, the whole thing fails

```

```

% we still have to exclude the occurrence, within the gap, of nouns to which the adjective could be
% attached with priority, because they agree in the well-known agreement triplet
% we use relaxedadjacentN_cgn
% (where N=1,N=2,N=3, and cgn means that case gender and number are checked for agreement)

```

```

ifthen(Distance=3, relaxedadjacent3_cgn([p(A,B)],Padj,Case,Gender,Nb)),
    % three in between, neither of them a noun with relevant triplet
ifthen(Distance=2, relaxedadjacent2_cgn([p(A,B)],Padj,Case,Gender,Nb)),
    % two in between, neither of them a noun with relevant triplet
ifthen(Distance=1, relaxedadjacent1_cgn([p(A,B)],Padj,Case,Gender,Nb)),
    % one in between, not a noun with same [gender,number,case] triplet

```

```

% Weight is W+1,
% we increase the weight of the adjective phrase (as computed when the phrase was parsed)
% with the weight assigned to a single noun, i.e. 1
% myplus is the same as plus, but does not fail if it meets with a variable instead of a number
myplus(W,1,Weight), % means W+1=Weight, a formulation that would lead to disaster in Prolog,
    % since the equal sign (=) is used for unification, not addition !!!
    % standard Prolog requires Weight is W+1

```

```

% we can now build the resulting NP
% note that the head of the NP is the N,
% which also yields the index reference used for binding traces (as in relative clauses)

```

```

map(np,[pathlist:Sorted,hp:[p(A,B)],index:i(p(A,B)),distance:[Distance], % distance is recorded as
    % straining factor
    cat:np,class:Class,sem:Sem,
    number:Nb,person:3,gender:Gender,type:core,lex:LexNoun,lextype:full,
    case:Case,w:Weight,
    c_str:[head:FSnoun,adjp:FSadj]]).

```

the *c\_str* is the constituent as it appears in the parse tree : (we use the parse produced by ALP for the sentence : [0/agricola,1/doctior,2/petro,3/misit,4/reginae,5/epistulam,endpos(6)], selecting the bit assigned to the subject (*agricola doctior Petro*) :

```

subject:[pathlist:[p(0,1),p(1,2),p(2,3)],hp:[p(0,1)],index:i(p(0,1)),distance:[0],
    cat:np,class:common,sem:[hum],
    number:sing,person:3,gender:masc,
    type:core,lex:agricola,lextype:full,case:nom,w:3,

```

```

c_str:[head:[pos:noun,txt:agricola,lex:agricola,case:nom,gender:masc,class:common,number:sing,sem:
[hum]],

```

```

    adjp:[cat:adjp,pathlist:[p(1,2),p(2,3)],distance:[0],hp:[p(1,2)],
        case:nom,number:sing,gender:masc,
        lex:doctus,type:std,w:2,

```

```

c_str:[doctus,
    comp_cplt:[pathlist:[p(2,3)],hp:[p(2,3)],index:i(p(2,3)),distance:[0],
        cat:np,sem[hum],class:proper,
        lex:petrus,lextype:full,number:sing,person:3,gender:masc,
        type:core,case:abl,w:1]]]]

```

which pretty-prints as:

subject

index:i(p(0,1))

cat:np

sem:[hum]

number:sing

person:3

gender:masc

lex:agricola

case:nom

c\_str

head

pos:noun

lex:agricola

case:nom

gender:masc

number:sing

sem:[hum]

adjp

cat:adjp

case:nom

number:sing

gender:masc

lex:doctus

c\_str

doctus

comp\_cplt

index:i(p(2,3))

cat:np

sem:[hum]

lex:petrus

number:sing

person:3

gender:masc

case:abl

## Matching

Since we have been looking at *arglists*, we'll now say a few words about the process by which the argument requirements are satisfied, i.e. matched with structures to be found in the string to be parsed. We have already pointed out that the args do not have to be found in the canonical order in which they appear in the *arglist*. It also stands to reason that the args marked as optional need not be instantiated, but if they are, they contribute to the weight assigned to the predicate-arg nexus.

Consider the matching of the *subject* arg. We have already seen that the subject could be projected from the verb group, which is the standard case when the subject is first or second person, but a third person subject may also be textually retrievable.

Voice will affect the arglist. In the passive voice, the object arg will be assigned the subject function, and the subject arg will be demoted to a prepositional phrase status (ab+ablative) or will be assigned the ablative case, and will in all cases be optional. Such transformations to the arglist must be accomplished as soon as we have ascertained the predicate's voice, which should be early enough in the parsing process (but remember a very important property of the production system: rules fire automatically when the material they need is ready, i.e. has been made available by lexical look-up or the previous firing of grammatical rules and their production of the required structures. It is NOT the linguist's task to worry about sequence in the application of rules. Considering the very highly recursive nature of grammar, this is a key property of production systems).

We'll now look at the code for updating the args to be matched in the case of a passive voice being found in the arg-bearer, i.e. the predicate, in a non-finite clause (*puto reginam ab ancilla marci amari*).

```

mapped(vg,FSverb),
constraint([type:nonfinite,mood:Mood,voice:Voice,tense:Tense,      % nonfinite verb form (in our case: amari)
           pathlist:PathlistVerb,lex:Clex, w:WVerb],FSverb),

lexarg(Clex,arglist:ArgList),                                     % connection with the args via lexarg
                                                                % the args are those for amo:
           lexarg(amare,
           arglist:[
           ws(amo_love,tr_cod,clause:[],mwuw:0,
           args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
           object:[type:np,oblig:no,constraints:[case:acc]]])

pick(ws(Lex,Class,clause:Clause_Constraints,mwuw:MW,args:Args), ArgList,_),
           % picking a word sense to see if it is appropriate ...
           % remember that in general there will be more than one
           % ws (i.e. word sense) for a given arg-bearer

ifthenelse(Voice=pass,      % outer THEN      % PASSIVE, as here amari

           % selecting the object to remove it from the arglist and turn it into a subject

           (pick(object:ObjectSpecs,Args,Args1), % note that there must be an object if a passive was produced !!
            pick(oblig:Oblig, ObjectSpecs,OS1), % but perhaps it was not obligatory, as in the case of amare
            pick(constraints:Oconstraints,OS1,OS2), % and we must also update the constraints
            pick(case:Caseobj,Oconstraints,Oconstraints1),
              % recall that the predicate pick selects an element and
              % removes it from the list
              % case is buried within the constraints associated with the arg
              % we pick it out

            append([case:acc],Oconstraints1,NewOconstraints), % subjects are accusatives in nonfinite clauses !!
                                                                % here reginam

            append([constraints:NewOconstraints],OS2,NOS),
            append([oblig:yes],NOS,NewObjectSpecs), % there must be a subject if a passive is used !!!

           % turning the subject into an optional (a+) abl pp arg
           pick(subject:SubjectSpecs,Args1,Args2),
           pick(constraints:Sconstraints,SubjectSpecs,SS1),

           ifthenelse( Sconstraints=[],
             % no constraint on subj: both types of agent are OK      IF-CLAUSE

             ( NewArg=[type:pp,oblig:no,constraints:[prep:ab,sem:[hum]]] ;
               % THEN-CLAUSE note the OR operator (;)
               NewArg=[type:np,oblig:no,constraints:[case:abl,sem:[thing]]] ) ,

             % ELSE-CLAUSE:

             % ab+hum vs simple abl for non-hum

           ifthenelse( constraint([sem:[hum]],Sconstraints), % IF2-clause      case of amare
                                                                % there are constraints: we act accordingly
               NewArg=[type:pp,oblig:no,constraints:[prep:ab,sem:[hum]]], % THEN2-clause
               NewArg=[type:np,oblig:no,constraints:[case:abl]]), % ELSE2-clause

           append([agent:NewArg],Args2,At),
           append([subject:NewObjectSpecs],At,Argstomatch) ), % reconstructing the arglist

Argstomatch=Args), % outer ELSE: ACTIVE : leave the args as they are in the arg specs

           % the remainder of this bit of code concerns the matching of the new arglist
           % and is not discussed here

```

## A Voice in the Middle...

Since we have been discussing voice, this might be the right place to point out that ALP works with a *middle* voice, by the side of the active and the passive.

Morphologically, we generate forms that look like passives, but concern third person singular of intransitive verbs, e.g. *insanitur*, *insaniatur*, *insaniebatur* and *insanietur* for *insanio*, for instance. This middle voice is much nearer to active than to passive. The passive touch is found in the generality and impersonality of the process, which can convey a sense of inevitability: *Ibatur in caedes...*

```
vg
  selected_reading:eo_go          % lbatur
  polarity:pos
  cat:vg
  pos:v
  lex:ire
  voice:middle
  tense:imperfect
  mood:indicative
  number:sing
  person:3
prep_cplt                         % in
  case:acc
  prep:in
  sem:[abstract]
  lex:caedes                      % caedes
  index:i(p(2,3))
  cat:pp
c_str
  prep:in
  head
    index:i(p(2,3))
    cat:np
    sem:[abstract]
    number:pl
    person:3
    gender:fem
    lex:caedes
    case:acc
```



We also register *middle* voice in two other cases:

% 1. **pugnatum est**

```
[verb,vg5amiddle] --->
[ mapped(v,[from:X, to:Y|FSverb]),
  constraint([lex:esse,type:Type, person:3, number:sing, tense:Tense,mood:Mood],FSverb),    % esse
  mapped(v,[from:C, to:D|Supine]),                                                       % supine verb form
  adjacent([p(X,Y)],[p(C,D)]),                  % adjacency, but either order
  append([p(X,Y)],[p(C,D)],Path),
  msort(Path, Sorted),
  constraint([type:supine,lex:Lex,kind:std],Supine),                                     % does not apply to deponent verbs
  ifthen(Tense=present,Tenseout=perfect),                                               % working out tense assignment on the basis
                                                                                           % of what we find in the 'esse'

  ifthen(Tense=imperfect,Tenseout=pluperfect),
  ifthen(Tense=future,Tenseout=future_perfect),

  map(vgpos,[cat:vg,type:Type,pathlist:Path,hp:[p(C,D)],lex:Lex,
            person:3,mood:Mood,tense:Tenseout,
            voice:middle,number:sing,gender:neuter,w:3])).    % middle voice
```

% with gerunds

% 2. **insaniendum est** (also with deponent verbs: **hortandum est**)

```
[verb,vg5amiddle1] --->
[ mapped(v,[from:X, to:Y|FSverb]),
  constraint([lex:esse,type:Type, person:3, number:sing,tense:Tense,mood:Mood],FSverb),    % esse
  mapped(v,[from:C, to:D|Gerund]),                                                       % gerund
  adjacent([p(X,Y)],[p(C,D)]),
  append([p(X,Y)],[p(C,D)],Path),
  msort(Path, Sorted),
  constraint([case:acc,type:gerund,lex:Lex,kind:Kind],Gerund),                          % accusative form of the gerund
  map(vgpos,[cat:vg,type:Type,pathlist:Path,hp:[p(C,D)],lex:Lex,
            person:3,mood:Mood,tense:Tense,
            voice:middle,number:sing,gender:neuter,                                     % middle voice
            value:obligation,w:3])).    % semantic force : obligation
```

## Relative Clauses

We deal with relative clauses in a way that may prove somewhat surprising to a linguist not used to working with *indices* and *gaps*. Consider a relative clause from which the relative pronoun has been removed:

*(qui relinquit reginam / quem relinquit rex) → relinquit reginam / relinquit rex*

we can look at such structures as a *pred-arg nexus missing an argument*, subject in the first case, object in the second (this is not the only reading, it's the reading that we give these structures when we know that they are incomplete, that they miss something). The missing arg is of course the relative pronoun, but the relative pronoun on its own does not give sufficient information to guarantee that all the constraints on the missing arg are met. The relative pronoun must be put into relation with its antecedent, and then we can complete the checking of a number of constraints, for instance of a semantic nature. The relation between antecedent and pronoun, and thereby between the antecedent and the gap in the incomplete predication discussed above, is implemented by *index sharing*. The index is no more than a pointer and can be coded in Prolog by a one argument structure, *i(Index)*, where *Index* is a variable to be shared by all instances of an index pointing to the same thing. The variable *Index* would then be shared by the triplet *missing arg/relative pronoun/antecedent*. We can increase the readability of our parses if we use a pair of values instead of the *Index* variable. This pair of values can be assigned as index each time we posit a noun phrase head: the values will be the start and end positions of the NP head in the string. To give an undoubtedly welcome example, in the sentence

*rex qui relinquit reginam malus est*

*rex* will be assigned *i(0,1)* as an index, which will come to be shared by *qui* and the missing arg of the *relinquit*-arg nexus (where it is often known as a *trace* or a *gap*). If *relinquit* needs a human subject (it does!), the constraint will be placed on the gap, passed on to the relative and to its antecedent, where it will be found to be satisfied.

The assigned parse follows:

vg  
 selected\_reading:sum\_be EST  
 polarity:pos  
 cat:vg  
 pos:v  
 lex:esse  
 voice:act  
 tense:present  
 mood:indicative  
 number:sing  
 person:3  
 subject  
 cat:np  
**index:i(p(0,1))** REX  
 number:sing  
 gender:masc  
 sem:[hum]  
 person:3  
 case:nom  
 lex:rex  
 c\_str  
 head  
 rex  
 rel\_clause  
**index:i(p(0,1))** QUI  
 number:or([sing,pl])  
 gender:masc  
 case:nom  
 mood:indicative  
 tense:present  
 c\_str  
 vg  
 selected\_reading:relinquo\_leave RELINQUIT  
 polarity:pos  
 cat:vg  
 pos:v  
 lex:relinquere  
 voice:act  
 tense:present  
 mood:indicative  
 number:sing  
 person:3  
**subject** pointer to REX  
**index:i(p(0,1))**  
 object  
 index:i(p(3,4))  
 cat:np  
 sem:[hum]  
 number:sing  
 person:3  
 gender:fem  
 lex:regina REGINAM  
 case:acc  
 predicative  
 cat:adjp  
 case:nom  
 number:sing  
 gender:masc  
 lex:malus MALUS

We will briefly go into the index assignment and index sharing procedure, a procedure considerably facilitated by Prolog unification.

First, whenever we build an np, we assign an index corresponding to the path of its head, as in (the simplest possible case of an np built out of a single noun):

```
[core,np1] --->
[mapped(noun,[from:A,to:B|FS]),
constraint([pos:noun,lex:Lex,class:common,sem:Sem,txt:Text,
            number:Nb,gender:G,case:C],FS),
map(np,[pathlist:[p(A,B)],hp:[p(A,B)],index:i(p(A,B)),distance:[0],
        cat:np,sem:Sem,class:common,lextype:full,
        number:Nb,person:3,gender:G, type:core,lex:Lex,txt:Text,
        case:C,w:1])].
```

Second, we map relative clauses as consisting of a relative pronoun followed by a predication which misses an argument, and we relate the missing argument to the relative (in the code given below, the relative pronoun is subject, as in *uir qui epistulas ad Marcum misit*)

```
% Recall that the function of the np is independent from its function in the relative clause:
% "liber quem rex legit ..." : liber is subject in the main clause and quem is object in the relative
% The index is shared; it reports the positions spanned by the np.
% A relative clause is an S displaying a [gap:Gap] feature corresponding to the antecedent:
% same category (np, pp) and shared index
% The gap site can specify any type of constraints on the constituent structure of the antecedent NP;
% this power is necessary to deal with mwus where the deletion site
% can point to an NP that must be lexically described,
% not just in terms of features such as number and broad semantic category
```

```
%% with a relative pronoun filling an np slot
% "(vir) qui epistulas ad Marcum misit" ; "(librum) quem ancilla legit"
```

```
% subject
```

```
[finite,rel_clause_1] --->
[mapped(relative,[from:X,to:Y|FS1]),                                     % a relative pronoun

constraint([number:Nb, case:Case,gender:Gender, function:Functions],FS1),

mapped(pred,FS2),                                                     % a clause

constraint([type:finite,                                             % relative clauses are finite
            mood:Mood, tense:Tense,pathlist:Pathlist,distance:Distance,w:Weight],FS2),

msort(Pathlist,Sorted),
start(Sorted,Y),
contiguous(Sorted),                                                  % the relative clause cannot bind structures outside of itself
                                                                    % this restriction is important
                                                                    % strict contiguity seems to be called for

constraint([gap:GAPARG,c_str:C_str],FS2),                            % there MUST be a gap in the clause

nonvar(GAPARG),                                                       % the built-in predicate nonvar checks that its argument
                                                                    % is an instantiated (bound) variable when the call to
                                                                    % nonvar is made
                                                                    % without such a test the danger is that the following test
                                                                    % be no test at all, but should merely result in unification
                                                                    % of the GAPARG variable with whatever comes its way
```

```

GAPARG=[gap:[type:np,           % GAP specifies type
        index:Index,          % opens a place for the Index of the antecedent to fill
        function:subject,     % function must be compatible with relative pronoun
        subject:[e:Index],    % info for the parse tree - e (empty, trace) followed by the Index
        constraints:Constraints % we put the constraints to be checked on the antecedent in this box
        ]],

member(subject,Functions),     % remember that the acc rel pronouns must bear the 'subject' function as well as the
                               % object one

append([p(X,Y)],Pathlist,Pnew), % appending the relative pronoun to the path

map(relative_clause,[pathlist:Pnew,
                    distance:Distance,
                    gap:GAPARG,      % gap info carried by the clause
                    index:Index,
                    number:Nb,
                    gender:Gender,
                    case:Case,
                    type:finite,
                    mood:Mood,
                    tense:Tense,
                    constraints:Constraints,
                    w:Weight,
                    c_str:C_str]) ].

% similar treatment for the other functions (object, etc.)

```

We should now look at the building of a gapped predication. Whenever we are trying to match an argument with a phrase in the string to be parsed, we allow that argument to be matched with nothing at all in the string, with the proviso that it create a GAP, a structure which houses the requirements that the arg would have filled if it had been found in the string. Again, we look at the subject in finite clauses:

```

% in finite clauses
% the subject is the one of the potentially main clause, i.e. the 'up' one
% rex qui // [subject-gap] amat reginam

match(subject:Specs,
      subject:[index:Indexup],
      [number:Nsubj,gender:Gendersubj,person:Psubj,index:Indexup],
      [],
      [0],
      finite, % important !
      gap:[gap:[type:Type,
              index:Indexup,
              function:subject,
              subject:[e:Indexup],
              constraints:GapConstraints]],
      w:1,
      Int) :-

constraint([type:Type,constraints:Constraints],Specs),
Type \= dummy,
append([number:Nsubj,gender:Gendersubj,person:Psubj,case:nom,index:Indexup],Constraints,GapConstraints).

% remember that a gapped constituent simply puts its constraints in the Gap feature,
% to be satisfied when the pred is connected to the antecedent

```

It remains for us to look at how the junction between the relative clause and its antecedent is accomplished. The index of the np is projected into the relative clause; the constraints housed in the relative clause should be met, with the exception of case, which need not be shared between antecedent and relative pronoun (and thereby the GAP structure of the relative).

```
% NP WITH RELATIVE CLAUSE
```

```
[finite,np6] --->
```

```
[
    % the NP
    mapped(np,FS1),
    constraint([pathlist:PL1,hp:HL1,distance:[Distnp1]],FS1),
    constraint([cat:np,index:Index,number:Nb,gender:G,
               sem:SemNP,lex:Lex, case:Case],FS1),

    % the relative clause
    mapped(relative_clause,FS2),
    constraint([number:Nb,gender:G,pathlist:PL2,distance:[Distrel], constraints:Constraints,w:W],FS2),

    constraint([index:Index],FS2), % index sharing with the NP - essential to link relative and antecedent

    cleanc(Constraints,CC),
    % the Constraints should not include Case (to be removed - other constraints to be kept)
    ifthen(CC\=[],constraint(CC,FS1)),
    % apply the constraints, e.g. semantic constraints passed on to the antecedent noun

    msort(PL1,PLnpSorted),
    extremity(PLnpSorted,X), % contiguity test np and rel clause
    succ(X,Xplus), % successor function (succ(X,Xplus) is equivalent to Xplus is X+1
    (start(PL2,X); start(PL2,Xplus)), % room for only one word to fit between antecedent and relative clause

    append(PLnpSorted,PL2,PL),
    msort(PL, Sorted),
    \+dup(Sorted),
    % contiguous(Sorted), % not applicable on account of possible non-contiguity in the NP constituents
    % although at first sight the restriction looks reasonable... but:
    % imperatores timeo qui a pace abhorrent

    % Distance is Distnp1+Distrel,
    % Weight is W+1,
    myplus(Distnp1,Distrel,Distance),
    myplus(W,1,Weight),

    map(np,[pathlist:Sorted, hp:HL1,distance:[Distance],
           cat:np,type:full,class:common,lextype:full,
           index:Index,number:Nb,gender:G,sem:SemNP,person:3,case:Case,
           lex:Lex,w:Weight,c_str:[head:Lex,rel_clause:FS2]])
].
```

## Prioritizing Subject-object Order in Accusative-cum-infinitive Clauses

In nonfinite dependent clauses, namely in accusative-cum-infinitive clauses, we might encounter accusative pairs, and even accusative triplets in the case of double-accusative verbs of the *doceo*-type:

*Putabas reginam regem amare.*

*Putabas ancillam pueros grammaticam docere.*

The accusatives can play the parts of subject, object, and even indirect object in the argument structure of *doceo*-type verbs.

Suppose we wish to prioritize the standard order of args in the clause, with the subject preceding the object(s), without rejecting readings in which the subject follows either or both of the objects:

*Putabas reginam regem amare:*

Preferred reading: *reginam* subject, *regem* object:

*You thought the queen loved the king.*

Deprioritized but possible reading: *regem* subject, *reginam* object:

*You thought the king loved the queen.*

*Putabas ancillam pueros grammaticam docere.*

Preferred reading: *ancillam* subject, *pueros* indirect object, *grammaticam* object:

*You thought the servant was teaching the kids grammar.*

Deprioritized but possible reading:

*ancillam* indirect object, *pueros* subject, *grammaticam* object:

*You thought the kids were teaching the servant grammar.*

Rejected readings (on semantic grounds: both the subject and indirect object must bear the feature +HUM):

*grammaticam* subject, *pueros* indirect object, *ancillam* direct object:

\* *You thought the grammar was teaching the kids the servant.*

*grammaticam* subject, *pueros* object, *ancillam* indirect object:

\* *You thought the grammar was teaching the servant the kids.*

The parser should come up, and does come up, with the following ranking of the two retained parses:

[0/putabas,1/ancillam,2/pueros,3/grammaticam,4/docuisse,endpos(5)]

5--->

```
lex:puto_think_that
  cat:vg
  polarity:pos
  pos:v
  lex:putare
  tense:imperfect
  mood:indicative
  number:sing
  person:2
  subject
    source:context_retrievable
    number:sing
    person:2
    index:i( G3839)
    constraints_to_be_met:[sem:[hum]]
    case:nom
  object
    cat:pred
    number:sing
    case:or([nom,acc])
    person:3
    i_ss:i(p(1,2))
    c_str
      lex:doceo_teach
        cat:vg
        polarity:pos
        pos:v
        lex:docere
        tense:past
        mood:infinitive
        subject
          index:i(p(1,2))
          cat:np
          sem:[hum]
          number:sing
          person:3
          gender:fem
          lex:ancilla
            case:acc
            c_str
              head
                pos:noun
                lex:ancilla
                case:acc
                gender:fem
                number:sing
                sem:[hum]
            object
              index:i(p(3,4))
              cat:np
              sem:[abstract]
              number:sing
              person:3
              gender:fem
              lex:grammatica
              case:acc
```



c\_str  
  head  
    pos:noun  
    lex:grammatica  
    case:acc  
    gender:fem  
    number:sing  
    sem:[abstract]

i\_object  
  index:i(p(2,3))  
  cat:np  
  sem:[hum]  
  number:pl  
  person:3  
  gender:masc  
  lex:puer  
  case:acc

c\_str  
  head  
    pos:noun  
    lex:puer  
    case:acc  
    gender:masc  
    number:pl  
    sem:[hum]

4-->

lex:puto\_think\_that  
cat:vg  
polarity:pos  
pos:v  
lex:putare  
tense:imperfect  
mood:indicative  
number:sing  
person:2  
subject  
  source:context\_retrievable  
  number:sing  
  person:2  
  index:i(\_G4775)  
  constraints\_to\_be\_met:[sem:[hum]]  
  case:nom  
object  
  cat:pred  
  number:sing  
  case:or([nom,acc])  
  person:3  
  i\_ss:i(p(2,3))  
  c\_str  
    lex:doceo\_teach  
    cat:vg  
    polarity:pos  
    pos:v  
    lex:docere  
    tense:past  
    mood:infinitive  
    subject  
      index:i(p(2,3))  
      cat:np  
      sem:[hum]  
      number:pl  
      person:3  
      gender:masc  
      lex:puer  
      case:acc  
      c\_str  
        head  
          pos:noun  
          lex:puer  
          case:acc  
          gender:masc  
          number:pl  
          sem:[hum]  
    object  
      index:i(p(3,4))  
      cat:np  
      sem:[abstract]  
      number:sing  
      person:3  
      gender:fem  
      lex:grammatica  
      case:acc  
      c\_str  
        head  
          pos:noun  
          lex:grammatica  
          case:acc  
          gender:fem

```

        number:sing
        sem:[abstract]
i_object
    index:i(p(1,2))
    cat:np
    sem:[hum]
    number:sing
    person:3
    gender:fem
    lex:ancilla
    case:acc
    c_str
    head
        pos:noun
        lex:ancilla
        case:acc
        gender:fem
        number:sing
        sem:[hum]

```

It should be borne in mind that the accusative-cum-infinitive construction might be found inside a relative clause: in such cases, a correct parsing procedure must identify the nature of the trace (gap) in the relative clause, and ensure the proper indexing of it by index-sharing between antecedent and trace.

Consider:

*Amas grammaticam quam putabas ancillam pueros docuisse.*

In the accusative-cum-infinitive clause [*ancillam pueros docuisse*] inside the relative clause [*quam putabas ancillam pueros docuisse*] we need to posit a gap (symbolized in the parse tree with an *e* for *empty*) for the object of *docuisse* and associate that gap, by co-indexing, with the antecedent of the relative, i.e. *grammaticam*.

We do this by assigning an index referring to the position occupied by the antecedent in the word list derived from the input string:

[0/amas,1/**grammaticam**,2/quam,3/putabas,4/ancillam,5/pueros,6/docuisse,endpos(7)]

The antecedent *grammaticam* spans from 1 to 2:  $p(1,2)$ .

The index for the missing arg within the infinitive clause (the object arg) must therefore bear the index  $p(1,2)$ , as it does in the parses returned by the parser. Once again, the parse tree where the order subject-object is maintained is prioritized

5-->

lex:amo\_love  
cat:vg  
polarity:pos  
pos:v  
lex:amare  
tense:present  
mood:indicative  
number:sing  
person:2  
subject  
source:context\_retrievable  
number:sing  
person:2  
index:i(\_G1737)  
constraints\_to\_be\_met:[sem:[hum]]  
case:nom  
object  
cat:np  
**index:i(p(1,2))**  
number:sing  
gender:fem  
sem:[abstract]  
person:3  
case:acc  
**lex:grammatica**  
c\_str  
head  
head  
pos:noun  
lex:grammatica  
case:acc  
gender:fem  
number:sing  
sem:[abstract]  
rel\_clause  
index:i(p(1,2))  
number:sing  
gender:fem  
case:acc  
c\_str  
lex:puto\_think\_that  
cat:vg  
polarity:pos  
pos:v  
lex:putare  
tense:imperfect  
mood:indicative  
number:sing  
person:2  
subject  
source:context\_retrievable  
number:sing  
person:2  
index:i(\_G2272)  
constraints\_to\_be\_met:[sem:[hum]]  
case:nom  
object  
cat:pred  
number:sing  
case:or([nom,acc])

person:3  
i\_ss:i(p(4,5))  
c\_str  
lex:doceo\_teach  
cat:vg  
polarity:pos  
pos:v  
lex:docere  
tense:past  
mood:infinitive  
**subject**  
index:i(p(4,5))  
cat:np  
sem:[hum]  
number:sing  
person:3  
gender:fem  
**lex:ancilla**  
case:acc  
c\_str  
head  
pos:noun  
lex:ancilla  
case:acc  
gender:fem  
number:sing  
sem:[hum]  
**object**  
**e:i(p(1,2))**  
i\_object  
index:i(p(5,6))  
cat:np  
sem:[hum]  
number:pl  
person:3  
gender:masc  
lex:puer  
case:acc  
c\_str  
head  
pos:noun  
lex:puer  
case:acc  
gender:masc  
number:pl  
sem:[hum]

4-->

lex:amo\_love  
cat:vg  
polarity:pos  
pos:v  
lex:amare  
tense:present  
mood:indicative  
number:sing  
person:2  
subject  
  source:context\_retrievable  
  number:sing  
  person:2  
  index:i(\_G445)  
  constraints\_to\_be\_met:[sem:[hum]]  
  case:nom  
object  
  cat:np  
  **index:i(p(1,2))**  
  number:sing  
  gender:fem  
  sem:[abstract]  
  person:3  
  case:acc  
  **lex:grammatica**  
  c\_str  
  head  
  head  
  pos:noun  
  lex:grammatica  
  case:acc  
  gender:fem  
  number:sing  
  sem:[abstract]  
  rel\_clause  
  index:i(p(1,2))  
  number:sing  
  gender:fem  
  case:acc  
  c\_str  
  lex:puto\_think\_that  
  cat:vg  
  polarity:pos  
  pos:v  
  lex:putare  
  tense:imperfect  
  mood:indicative  
  number:sing  
  person:2  
  subject  
  source:context\_retrievable  
  number:sing  
  person:2  
  index:i(\_G980)  
  constraints\_to\_be\_met:[sem:[hum]]  
  case:nom  
  object  
  cat:pred  
  number:sing  
  case:or([nom,acc])  
  person:3

i\_ss:i(p(5,6))  
c\_str  
lex:doceo\_teach  
cat:vg  
polarity:pos  
pos:v  
lex:docere  
tense:past  
mood:infinitive  
subject  
index:i(p(5,6))  
cat:np  
sem:[hum]  
number:pl  
person:3  
gender:masc  
lex:puer  
case:acc  
c\_str  
head  
pos:noun  
lex:puer  
case:acc  
gender:masc  
number:pl  
sem:[hum]  
**object**  
**e:i(p(1,2))**  
**i\_object**  
index:i(p(4,5))  
cat:np  
sem:[hum]  
number:sing  
person:3  
gender:fem  
**lex:ancilla**  
case:acc  
c\_str  
head  
pos:noun  
lex:ancilla  
case:acc  
gender:fem  
number:sing  
sem:[hum]

It might be worthwhile taking a quick look at the way the Prolog program embodying the parser deals with the subject-object order prioritization in the accusative-cum-infinitive construction:

```
% prioritize normal subject-object order

ifthenelse(constraint([subject:[hp:Pathsubj]],ST),
            % we have a non-gapped subject, we record its head path
            true, % and abstain from doing anything else
            Pathsubj=[p(0,0)],
            % otherwise the subject is higher up and therefore necessarily precedes

ifthenelse(constraint([object:[hp:Pathobj]],ST), % IF-CLAUSE we have an object
            ifthenelse(precedes(Pathsubj,Pathobj), NW is Weight+1, NW=Weight), % THEN-CLAUSE
            NW=Weight), % ELSE-CLAUSE

ifthenelse(constraint([i_object:[hp:Pathiobj]],ST), % we have an indirect object
            ifthenelse(precedes(Pathsubj,Pathiobj), NW1 is NW+1, NW1=NW),
            NW1=NW),
```

To understand this bit of code, one needs to know that the **hp** (HeadPathList) feature above records the positions spanned by the head of the noun phrase filling in the arg position of the parent feature: subject, object, or indirect object.

Remember that if the subject is gapped, it does not have a **hp** feature since it does not occur in the clause: there is no subject present in [*epistulam missime Marco*] as object of *putas* in [*putas epistulam missime Marco*], itself included in the relative clause [*quem putas epistulam missime Marco*] appended to the antecedent *rex* in [*Rex quem putas epistulam missime Marco*] which functions as subject of the whole S(entence): [*Rex quem putas epistulam missime Marco*] *amat ancillam reginae*.

In the case of such a gapped subject we assign it a dummy headpathlist, namely [p(0,0)], which is sure to precede the pathlist of any arg found in the clause. Otherwise we record the value of the *hp* feature and we can then compare it with the values of the headpathlists for the object and indirect object, if available, and proceed to the prioritizing by increasing the current Weight assigned to the parse tree being built.

We have seen that the *precedes* predicate is trivially simple to code; we know that the *constraint* predicate implements feature unification. It is used here to retrieve values for features possibly instantiated in the feature bundle *ST*, which records the structure and properties of the arguments filling in the arg slots opened up by the predicate.



## Binding SE

The issue is well-known. The SE family (*se, sui, sibi*) offers the possibility of multiple binding: a local binding in the clause of which SE is a constituent, and outer bindings in higher clauses, i.e. clauses having the SE-bearing clause as an argument or as the argument of one of their arguments, and so on climbing up the parse tree.

We wish to insist on one point: reference belongs to discourse. The only thing grammar can do is to suggest candidates and provide limits to the exploration of referents.

Remember that in ALP each *NP* is associated with an *index* which records the *string position* of its *head*. In the string we are going to parse, namely

*Romani sciunt*  
    *regem credere*  
        *reginam putare*  
            *se a Venere amatum iri.*

The np heads *Romani, regem, reginam* et *Venere* will be assigned an index. Since the process turning the string into a wordlist yields

[0/romani,1/sciunt,2/regem,3/credere,4/reginam,5/putare,6/se,7/a,8/uenere,9/amatum,10/iri],

the indices will be the following:

*romani* i(0,1) *regem* i(2,3) *reginam* i(4,5) *uenere* i(8,9).

The reflexive pronoun SE (and members of its family such as *sibi*), instead of having an index reflecting its position in the string (here, it would be i(6,7)), is assigned an index waiting to be bound: *index:Index*.

The binding of this index is meant to reflect the assignment of referents to the reflexive pronoun. The binding will result in the production of competing parses, where the index is bound locally, i.e. within the clause of which SE is a constituent, or externally (coming to be bound with a constituent higher up in the hierarchy).

The binding cannot take place until we have available a complete parse. We then examine its structure and proceed to the binding(s).

In our example string, the possible referents are *romani, regem* and *reginam* (the first is nominative, and the other two accusative; *venere* is not a subject, and is therefore excluded). The three candidates pass a structural test (nphood) and a semantic test (+HUM) and are 'positionally' OK, i.e. are at the right level and have been matched as candidate binding arguments.

Looking at the parses delivered by ALP, we note that SE is subject of the infinitive (future and passive voice) and is assigned different indices in the top parsings: i(p(4,5)),i(p(2,3)),i(p(0,1)) pointing to *reginam, regem* and *romani* as candidates for the referent of the subject. The parse is to be found below.

Once again, parses should not be rejected on the basis of the candidates they put forward for reference. We need other tools to tackle the issue, which is only partly a matter for linguistics to handle, and certainly not one to be exclusively assigned to the syntactic component that is the core business of parsers.

vg

selected\_reading:scio\_know\_that  
polarity:pos  
cat:vg  
pos:v  
lex:scire  
voice:act  
tense:present  
mood:indicative  
number:pl  
person:3

subject

**index:i(p(0,1))**

cat:np  
sem:[hum]  
number:pl  
person:3  
gender:masc  
lex:romanus  
case:nom

object

cat:pred  
mood:infinitive  
tense:present  
number:sing  
gender:neuter  
case:or([nom,acc])  
person:3  
polarity:pos  
argbound:no  
flagint:no

c\_str

vg

selected\_reading:credo\_believe\_that  
polarity:pos  
cat:vg  
gender:masc  
pos:v  
lex:credere  
voice:act  
tense:present  
mood:infinitive

subject

**index:i(p(2,3))**

cat:np  
sem:[hum]  
number:sing  
person:3  
gender:masc  
lex:rex  
case:acc

object

cat:pred  
mood:infinitive  
tense:present  
number:sing  
gender:neuter  
case:or([nom,acc])  
person:3  
polarity:pos  
argbound:no  
flagint:no

c\_str

vg

selected\_reading:puto\_think\_that  
polarity:pos  
cat:vg  
gender:fem  
pos:v

lex:putare  
voice:act  
tense:present  
mood:infinitive  
subject  
**index:i(p(4,5))**  
cat:np  
sem:[hum]  
number:sing  
person:3  
gender:fem  
lex:regina  
case:acc  
object  
cat:pred  
mood:infinitive  
tense:future  
number:sing  
gender:neuter  
case:or([nom,acc])  
person:3  
polarity:pos  
argbound:no  
flagint:no  
c\_str  
vg  
selected\_reading:amo\_love  
polarity:pos  
cat:vg  
lex:amare  
mood:infinitive  
tense:future  
voice:pass  
gender:or([masc,fem])  
subject

**index:i(p(4,5))** (/ **index:i(p(2,3))** / **index:i(p(0,1))** in the other 2 top parses)

cat:np  
sem:[hum]  
lex:pp3refl  
number:or([sing,pl])  
person:3  
gender:or([masc,fem])  
case:acc  
agent  
index:i(p(8,9))  
case:abl  
prep:ab  
sem:[hum]  
lex:uenus  
cat:pp  
c\_str  
prep:ab  
head  
index:i(p(8,9))  
cat:np  
sem:[hum]  
lex:uenus  
number:sing  
person:3  
gender:fem  
case:abl

We should add that in the current version of ALP we use a somewhat more sober approach: we bind SE locally, unless there is a governing clause above the one in which SE occurs, in which case we bind 'one up', i.e. to the subject of the governing clause. In *laudant se reges*, we bind *se* to *reges*; in *putat rex reginam se laudare*, we bind *se* to *rex* (preferably as object, due to the heavier weight given to Subject-Object order):

*Laudant se reges*:

```
vg
  selected_reading:laudo_praise
  polarity:pos
  cat:vg
  pos:v
  lex:laudare
  voice:act
  tense:present
  mood:indicative
  number:pl
  person:3
subject
  number:pl
  gender:masc
  index:i(p(2,3))
  cat:np
  sem:[hum]
  person:3
  lex:rex
  case:nom
object
  number:pl
  gender:masc
  index:i(p(2,3))
  cat:np
  sem:[hum]
  lex:pp3refl
  person:3
  case:acc
```

*Rex putat reginam se laudare:*

vg  
selected\_reading:puto\_think\_that  
polarity:pos  
cat:vg  
pos:v  
lex:putare  
voice:act  
tense:present  
mood:indicative  
number:sing  
person:3  
subject  
number:sing  
gender:masc  
**index:i(p(0,1))**  
cat:np  
sem:[hum]  
person:3  
lex:rex  
case:nom  
object  
cat:pred  
mood:infinitive  
tense:present  
number:sing  
gender:neuter  
case:or([nom,acc])  
person:3  
polarity:pos  
argbound:no  
flagint:no  
c\_str  
vg  
selected\_reading:laudo\_praise  
polarity:pos  
cat:vg  
gender:fem  
pos:v  
lex:laudare  
voice:act  
tense:present  
mood:infinitive  
number:sing  
subject  
number:sing  
gender:fem  
index:i(p(2,3))  
cat:np  
sem:[hum]  
person:3  
lex:regina  
case:acc  
object  
**index:i(p(0,1))**  
cat:np  
sem:[hum]  
lex:pp3refl  
number:or([sing,pl])  
person:3  
gender:or([masc,fem])  
case:acc

In the case of

*rex putat reginam se ipsam laudare*

we get the following subject-object pairing

subject

number:sing  
**gender:fem**  
**index:i(p(2,3))**  
cat:np  
sem:[hum]  
person:3  
lex:regina  
case:acc

object

**index:i(p(2,3))**  
cat:np  
sem:[hum]  
lex:pp3refl  
**emphasis:yes**  
number:sing  
person:3  
**gender:fem**  
case:acc

where the emphasis due to the *ipsam* prevented the binding with masculine *rex*.

## Weighting

We wish to stress that a weighting process is absolutely necessary for parsers. The parser is first and foremost a tool that yields structural descriptions of strings on the basis of what the linguist has specified as grammatical. If we do not build a weighting procedure on top, we will be presented with multiple parses of strings, all of them grammatical with respect to our grammar, although some of them may look very far-fetched, and, to put it bluntly, totally unacceptable as plausible readings of the string submitted to parsing. Consider the following example : *Amo magistros cupidos legendae historiae*, *I like teachers who are eager to read history (books)*.

The parse which gets the highest ranking in ALP is the natural one, in fact the only one that comes to mind when we read the Latin sentence (and the only one we expect the learner to work out): the sentence is made up of a predicate, *amo*, with first-person subject immediately derivable from the verb form; the predicate is transitive *amo*, which in the sentence has as object the noun phrase *magistros cupidos legendae historiae*, which is made up of a head, *magistros*, in the accusative as it should be, and an adjective phrase attached to it, namely *cupidos legendae historiae*, whose head, *cupidos*, is in its turn in the right case, gender and number. *Cupidus* is an argument-bearing adjective, its argument being a genitive phrase, noun phrase or gerund(ive) clause, as is the case here, the gerundive clause being *legendae historiae*, made up of a predicate, the gerundive *legendae*, and its argument, a noun phrase in the genitive case, *historiae*, *lego* being transitive just like the *amo* of two minutes ago. We have reached the end of the gerundive clause, the end of the argument of the adjective, the end of the noun phrase of which the adjective phrase is a part, the end of the argument of the main verb, the end of the sentence, the predicate having the two arguments it needs, a subject hidden in the verb form, and an object covering all the words except the predicate itself. Nothing could be simpler, there is no way of getting it wrong, and ALP certainly does not.

```
vg
  selected_reading:amo_love
  polarity:pos
  cat:vg
  pos:v
  lex:amare
  voice:act
  tense:present
  mood:indicative
  number:sing
  person:1
subject
  source:context_retrievable
  number:sing
  gender:or([masc,fem])
  person:1
  cat:np
  index:i(0,0)
  constraints_to_be_met:[sem:[hum]]
  case:nom
object
  index:i(p(1,2))
  cat:np
  sem:[hum]
  number:pl
  person:3
  gender:masc
  lex:magister
  case:acc
  c_str
  head
    pos:noun
    lex:magister
    case:acc
    gender:masc
    number:pl
```

```

sem:[hum]
adjp
  cat:adjp
  number:pl
  gender:masc
  lex:cupidus
  case:acc
  c_str
    cupidus
  object
    cat:pred
    subtype:gerundive
    mood:gerund
    local_case:gen
    number:sing
    person:3
    gender:neuter
    c_str
      vg
        selected_reading:lego_read
        pos:gdiv
        case:gen
        gender:fem
        number:sing
        lex:legere
        mood:gerund
        person:3
    object
      index:i(p(4,5))
      cat:np
      sem:[abstract]
      number:sing
      person:3
      gender:fem
      lex:historia
      case:gen

```

End of story? Well, there is Livy with *Pacis petendae oratores ad consulem miserunt* and Tacitus with *Germanicus Aegyptum proficiscitur cognoscendae antiquitatis*. And if we wish to account for the usage of our two historians, we need to make room for an adjunct of purpose built around a gerundive clause. And we run the risk of parsing our very simple sentence as meaning something along the lines of *I love greedy teachers in order to read history*.

In fact, there is no way of preventing the 'wrong' parse to come out, in so far as it is not a wrong parse at all – it is correct with respect to a grammar that is itself correct. What we can do to avoid the parse coming up to the surface is to deprioritize it, rank it down, or, what amounts to the same, prioritize what we regard as the natural parse, the one we have just shown to be the top choice of ALP.

The weighting procedure in ALP is based on two principles:

- 1) prefer tight links (such as that between a predicate and its arguments) over loose ones (an adjunct at clause level)
- 2) assign penalties to distortions of the underlying word order (where the subject precedes the object) and, first and foremost, to strains due to the distance separating elements which are naturally found together, such as an adjective or genitive np and the noun functioning as head of the resulting noun phrase.

The above strategies need to be put to work with a certain amount of care, so that they should cooperate rather than compete. We have also seen that we need the path algorithms studied above to put the second of them into practice.



# Test File

% 1 Terentianus Maurus De Syll 1286.

*Habent sua fata libelli.*

% 2 Cicero Ad Fam VIII 16 4.

*Me secum in Hispaniam ducit.*

% 3 Tacitus Agricola 2 1 1.

*Dedimus profecto grande patientiae documentum.*

% 4 Cicero Pro Murena VI 13.

*Saltatorem appellat Murenam Cato.*

% 5 adapted from Cicero Pro Murena VI 13.

*Non debes temere consulem populi Romani saltatorem vocare.*

% 6 Terentius Heauton Timorumenos I 77.

*Humani nil a me alienum puto.*

% 7 Tacitus Hist Lib I LXXVI 9.

*Carthaginem ceterae civitates secutae.*

% 8 Tacitus Hist Lib I 19 2.

*Censuerant patres mittendos ad Germanicum exercitum legatos.*

% 9 Ovidius Ars Amatoria 1 42.

*Elige cui dicas : tu mihi sola places.*

% 10 Horatius Carmina 1 5 1 - notice that the whole thing does not sport a single toolword.

*Me tabula sacer votiva paries indicat uvida suspendisse potenti vestimenta maris deo.*

% 11 Martial, 2 78.

*Aestivo serves ubi piscem tempore quaeris?*

% 12 Caesar De Bello Gallico I 9 3 4.

*Cupiditate regni adductus nouis rebus studebat.*

% 13 Tacitus Hist Lib I XXX 3.

*Falluntur quibus luxuria specie liberalitatis imponit.*

% 14 'dubitare quin' in Non-affirmative Contexts (from Cicero, Paradoxa, 6 48 1).

*Quis igitur dubitet quin in virtute divitiae sint?*

% 15 Cicero Att 4 12 1 5 fragmented main clause.

*Id tu quoniam Macronem tanti facis ignoscas mihi uelim.*

% 16 adapted from Tacitus Hist I 12 2.

*Maturavit ea res consilium Galbae iampridem de adoptione cum proximis agitantis.*

% 17 adapted from Tacitus Hist Lib I 74 3.

*Otho revocatis quos Galba miserat legatis ad utrumque Germanicum exercitum copias misit.*

% 17 bis.

*Revocatis quos Galba miserat legatos.*

% 18 adapted from Cicero Epistulae ad Familiares 9 14 1 1.

*Non dubito quin tu meis praeceptis et consiliis obtemperans praestantissimum te civem et singularem consulem praebeas.*

% 19 adapted from Cicero Pro Murena XXIII.

*Nostrae artes admirabilem utilitatem possident quae nobis studia populi Romani conciliat.*

% 20 adapted from Titus Livius Ab Urbe Condita Liber XXI 8.

*Quies inter labores renovavit corpora animosque ad omnia de integro patienda.*

% 21 Ovidius Ars Amatoria 1 307.

*Crede tamen speculo quod te negat esse iuencam.*

% 22 adapted from Cicero Phil I I 1.

*Graecum etiam verbum usurpavi quo in sedandis discordiis utitur civitas illa.*

% 23 Tacitus Hist Lib I 39 2.

*Agitasse Laco ignaro Galba de occidendo Vinio dicitur.*

% 24 Tacitus Agricola 2 4 1.

*Memoriam quoque ipsam cum voce perdidissemus si tam in nostra potestate esset oblivisci quam tacere.*

% 25 Cicero De Amicitia 10 1.

*Suis autem incommodis graviter angere non amicum sed se ipsum amantis est.*

% 26 Cicero Pro Murena XV.

*Omnibus regibus quibuscumque populus Romanus bellum gessit hunc regem nimirum antepones.*

% 27 Cicero Pro Murena XVII 35.

*Totam opinionem parva nonnumquam commutat aura rumoris.*

% 28 Catullus XIII 13-14.

*Quod tu cum olfacies, deos rogabis totum ut te faciant nasum.*

% 29 Cicero Pro Murena XXXIV 85.

*Versabitur in rostris furor, in curia timor, in foro coniuratio, in Campo exercitus, in agris vastitas.*

% 30 Cicero Pro Murena III 7.

*Huic ego satis facere cupio uosque adhibere arbitros.*

% 31 Cicero Pro Murena XVII 36.

*Nihil est incertius uolgo, nihil obscurius uoluntate hominum.*

% 32 Cicero Pro Murena XVIII 37.

*Munus amplissimum quod petitio praeturae desiderarat praetura restituit.*

% 33 adapted from Cicero Pro Murena XXXVIII 16.

*Cogita nonnullorum amicorum studia minui in eos a quibus provincias contemni intellegunt.*

% 34 Cic Ad Fam 1 1 2 10.

*Is ceteris in rebus se acerrimum tui defensorem fore ostendit.*

% 35 Tacitus Ann 12 8 1.

*Die nuptiarum Silanus mortem sibi conscivit delecto die augendam ad invidiam.*

% 36 adapted from Cicero Cat I 10.

*Exclusi eos quos tu salutatum miseras.*

% 37 adapted from Cicero Cat I 10.

*Illi venerunt quos ad me venturos esse praedixeram.*

% 38 adapted from Tacitus Hist Lib I 16 3.

*Mihi ac tibi providendum est ne etiam a bonis Nero desideretur.*

% 39 adapted from Tacitus Hist Lib I 44 1.

*Nullam caedem Otho maiore laetitia excepiisse dicitur.*

% 40 adapted from Tacitus Hist Lib I 79 1.

*Conversis ad civile bellum animis externa sine cura habebantur.*

% 41 Tacitus Ann 16 34 1 a.

*Arriam tentantem exemplum matris sequi monet retinere vitam.*

% 41 Tacitus Ann 16 34 1 b.

*Arriam monet filiae communi subsidium unicum non adimere.*

% 42 Tacitus Hist Lib I 85 9.

*Vitellianos milites venisse in urbem ad studia partium noscenda plerique credebant.*

% 43 Tacitus Hist Lib I 46 1.

*Omnia deinde arbitrio militum acta.*

% 44 Tacitus Hist Lib I 46 1.

*Praetorii praefectos sibi ipsi legere.*

% 45 Martialis Epigrammata 3 9 2.

*Non scribit cuius carmina nemo legit.*

% 46 Ovidius Ars Amatoria 2 658.

*Nominibus mollire licet mala.*

% 47 Tacitus Ann 14 53 4.

*Una defensio occurrit quod muneribus tuis obniti non debui.*

% 48 Tacitus Ann 14 10 1.

*A Caesare perfecto demum scelere magnitudo eius intellecta est.*

% 49 Horatius Ep 1 xiii 16.

*Ne volgo narres te sudavisse ferendo carmina.*

% 50 Cicero Ad fam xiv 4 5.

*Non vitium nostrum sed virtus nostra nos afflixit.*

% 51 Horatius Sat I 9 59-60.

*Nil sine magno vita labore dedit mortalibus.*

% 52 Horatius Sat II 6 79-81.

*Olim rusticus urbanum murem mus paupere fertur accepisse cavo.*

% 53 Tacitus Ann 11 22 2.

*Dolabella censuit spectaculum gladiatorum celebrandum pecunia eorum qui quaesturam adipiscerentur.*

% 54 adapted from Horatius Ep 2 ii 65-66.

*Romaene me poemata censes scribere posse inter tot curas?*

% 55 Hieronymus Ep LX 19.

*Cotidie morimur et aeternos nos esse credimus.*

% 56 Hieronymus Ep LX 19.

*Cum quo loqui non possumus de eo loqui numquam desinamus.*

% 57 Iuuenalis Sat 8 244.

*Roma patrem patriae Ciceronem libera dixit.*

% 58 Vergilius Georgica 1 463.

*Solem quis dicere falsum audeat?*

% 59 Ovidius Amores I 6 34.

*Solus eram si non saevus adesset Amor.*

% 60 Horatius Carm II 3 1-2.

*Aequam memento rebus in arduis servare mentem.*

% 61 Ovidius Ars Am 1 132.

*Haec mihi si dederis commoda miles ero.*

% 62 Peruigilium Veneris 1.

*Cras amet qui numquam amavit.*

% 63 Sallustius Bellum Iugurthinum V 1.

*Superbiae nobilitatis obviam itum est.*

% 64 Sallustius Bellum Iugurthinum XX 5.

*Legatos ad Iugurtham de iniuriis questum misit.*

% 65 Sallustius Bellum Iugurthinum XIV 8.

*Ego eis finibus eiectus sum quos maioribus meis populus Romanus dedit.*

% 66 adapted from Sallustius Bellum Iugurthinum V 1.

*Bellum scripsi quod populus Romanus cum rege Numidarum gessit.*

% 67 Sallustius Bellum Iugurthinum LXXXV 31.

*Ipsa se uirtus satis ostendit.*

% 68 Sallustius Bellum Iugurthinum LXXXV 31.

*Illis artificio opus est ut turpia facta oratione tegant.*

% 69 Sallustius Bellum Iugurthinum LXXXV 14.

*Contemnunt novitatem meam, ego illorum ignaviam.*

% 70 Sallustius Bellum Iugurthinum LXXXV 14.

*Mihi fortuna, illis probra obiectantur.*

% 71 Cicero Pro Murena 84 1.

*Mihi credite, iudices.*

% 72 Vergilius Eclogae 1 46.

*Fortunate senex, ergo tua rura manebunt.*

% 73 Vergilius Eclogae 1 6.

*O Meliboee, deus nobis haec otia fecit.*

% 74 Vergilius Eclogae 1 13.

*Hanc etiam vix, Tityre, duco.*

% 75 Vergilius Eclogae 4 60.

*Incipe, parve puer, risu cognoscere matrem.*

% 76 Vergilius Eclogae 9 43.

*Insani feriant sine litora fluctus.*

% 77 Cicero De Legibus II 1 3.

*Ille sapientissimus vir Ithacam ut videret immortalitatem scribitur repudiasse.*

% 78 Seneca Dialogi 6 26 3 4.

*Cur in domo nostra diutissime lugetur qui felicissime moritur?*

% 79 adapted from Cicero De Legibus II 4 6.

*Vestigia adsunt eorum quos diligimus.*

% 80 Horatius Ep II 2 102.

*Multa fero ut placem genus irritabile vatum.*

% 81 Tacitus Ann XIII 5.

*Ita specie pietatis obviam itum dedecori.*

% 82 Tacitus Ann XIII 15.

*Nero intellecta invidia odium intendit pararique venenum iubet.*

% 83 Tacitus Ann XV 52 1.

*Coniuratis tamen metu permotis placitum maturare caedem.*

% 84 Sallustius Catilinae Coniuratio 1 1 2.

*Omnes homines niti decet ne uitam silentio transeant.*

% 85 Tacitus Hist Lib I 62 1.

*Mira inter exercitum imperatoremque diversitas.*

% 86 Tacitus Hist Lib I 10 1.

*Oriens adhuc immotus.*

% 87 Tacitus Hist Lib I 9 7.

*In Britannico exercitu nihil irarum.*

% 88 Tacitus Hist Lib I 7 13.

*Venalia cuncta.*

% 89 Tacitus Hist Lib I 7 15.

*Eadem novae aulae mala.*

% 90 Tacitus Hist Lib I 2 13.

*Plenum exiliis mare.*

% 91 PS 27 1.

*Dominus illuminatio mea.*

% 92 Vergilius Georgica 2 458.

*Fortunatos nimium sua si bona norint agricolae.*

% 93 Catullus Carmina 61 184.

*Iam licet venias, marite.*

% 94 Catullus Carmina 61 94-95.

*Vide ut faces aureas quatiunt comas.*

% 95 Catullus Carmina 72 1.

*Dicebas quondam solum te nosse Catullum.*

% 96 Catullus Carmina 46 6.

*Ad claras Asiae volemus urbes.*

% 97 adapted from Cicero In Cat 1 17 2.

*Servi mei si me metuerent ut te metuunt cives tui domum meam relinquendam putarem.*

% 98 Cicero Att VII 3 4.

*De sua potentia dimicant homines hoc tempore periculo civitatis.*

% 99 Ovidius Tristia 1 9 5.

*Donec eris sospes multos numerabis amicos.*

% 100 Ovidius Tristia 1 9 6.

*Tempora si fuerint nubila solus eris.*

## Additional Test Sentences

% 1 Seneca Epistulae Morales ad Lucilium 7 1 4.

*Mane leonibus et ursis homines, meridie spectatoribus suis obiiciuntur.*

% 2 Cicero Pro Rege Deiotaro 30 14-15.

*Nulli parietes nostram salutem, nullae leges, nulla iura custodient.*

% 3 Tacitus Ann 16 34 1.

*Tum ad Thraseam in hortis agentem quaestor consulis missus vesperscente iam die.*

% 4 Tacitus Hist I 75 1.

*Insidiatores ab Othone in Germaniam, a Vitellio in urbem missi.*

% 5 Tacitus Hist I 11 1.

*Aegyptum copiasque quibus coerceretur equites Romani obtinent loco regum.*

% 6 Martialis Epigr Lib I 3 1-2.

*Argiletanas mauis habitare tabernas cum tibi, parue liber, scrinia nostra vacent.*

% 7 Martialis Epigr Lib I 117 8.

*Quod quaeris propius petas licebit.*

% 8 Propertius El III 25 17.

*Has tibi fatalis cecinit mea pagina diras.*

% 9 Propertius El III 25 18.

*Euentum formae disce timere tuae.*

% 10 adapted from Propertius El III 20 1-2.

*Credis iam tuae meminisse figurae quem uidisti a lecto dare uela tuo.*

% 11 Tacitus Hist IV 1 1.

*Interfecto Vitellio bellum magis desierat quam pax coeperat.*

% 12 Tacitus Ann 13 21 20.

*Vivere ego Britannico potiente rerum poteram?*

% 13 Tacitus Hist I 75 1.

*Ita promissis simul ac minis tentabantur.*

% 14 Tacitus Hist IV 16 1.

*Civilis dolo grassandum ratus incusavit ultro praefectos quod castella deseruissent.*

% 15 Tacitus Hist V 2 1.

*Quoniam famosae urbis supremum diem tradituri sumus congruens videtur primordia eius aperire.*

% 16 Tacitus Hist II 2 1.

*Fuerunt qui accensum desiderio reginae vertisse iter crederent.*

% 17 Tacitus Hist IV 58 2.

*Bellum cum populo Romano vestris se manibus gesturum Classicus sperat.*

% 18 Tacitus Dialogus 27 3 2.

*Cum de antiquis loquaris utere antiqua libertate a qua vel magis degenerauimus quam ab eloquentia.*

% 19 Lucius Annaeus Seneca senior Controversiae 8 1 10.

*I ad illum quem magis amas quam patrem.*

% 20 (20-27 build up Tacitus Hist IV 64 3-4).

*ut amicitia societasque nostra in aeternum rata sint /*

% 21.

*postulamus a uobis muros coloniae munimenta servitii detrahatis.*

% 22.

*etiam fera animalia si clausa teneas virtutis obliviscuntur.*

% 23.

*Romanos omnis in finibus uestris trucidetis.*

% 24.

*haud facile libertas et domini miscentur.*

% 25.

*bona interfectorum in medium cedant.*

% 26.

*ne quis occulere quicquam aut segregare causam suam possit /*

% 27.

*liceat nobis vobisque utramque ripam colere.*



# A Few Example Parses

[0/carthagem,1/ceterae,2/ciuitates,3/secutae,endpos(4)]

```
illocutionary_force:statement
vg
  selected_reading:sequor_follow
  polarity:pos
  cat:vg
  * secutae *
  lex:sequi_2
  person:3
  mood:indicative
  tense:perfect
  voice:act
  number:pl
  gender:fem
  subject
    number:pl
    gender:fem
    * ceterae * * ciuitates *
    index:i(p(2,3))
    cat:np
    sem:[hum,loc]
    person:3
    lex:ciuitas
    case:nom
    c_str
      head
        * ciuitates *
        pos:noun
        lex:ciuitas
        case:nom
        gender:fem
        number:pl
        sem:[hum,loc]
      adjp
        cat:adjp
        * ceterae *
        case:nom
        number:pl
        gender:fem
        lex:ceterus
  object
    * carthagem *
    index:i(p(0,1))
    cat:np
    sem:[city,thing,abstract]
    number:sing
    person:3
    gender:fem
    lex:carthago
    case:acc
    c_str
      carthagem
```

[0/id,1/tu,2/quoniam,3/macronem,4/tanti,5/facis,6/ignoscas,7/mihi,8/uelim,endpos(9)]

subordinator  
lex:quoniam  
pos:sub  
argbound:no  
mood:indicative  
value:reason  
subordinate\_clause  
illocutionary\_force:statement  
vg  
selected\_reading:mwu\_tanti\_facio\_appreciate  
polarity:pos  
cat:vg  
\* facis \*  
pos:v  
lex:facere  
voice:act  
tense:present  
mood:indicative  
number:sing  
person:2  
subject  
source:context\_retrievable  
number:sing  
gender:or([masc,fem])  
person:2  
cat:np  
index:i(0,0)  
constraints\_to\_be\_met:[sem:[hum]]  
case:nom  
object  
\* macronem \*  
index:i(p(3,4))  
cat:np  
sem:[hum]  
lex:macro  
number:sing  
person:3  
gender:masc  
case:acc  
main\_clause  
illocutionary\_force:statement  
vg  
selected\_reading:uolo\_want\_sbd\_to  
polarity:pos  
cat:vg  
\* uelim \*  
pos:v  
lex:uelle  
voice:act  
tense:present  
mood:subjunctive  
number:sing  
person:1  
subject  
source:context\_retrievable  
number:sing  
gender:or([masc,fem])  
person:1  
cat:np  
index:i(0,0)  
constraints\_to\_be\_met:[sem:[hum]]  
case:nom  
object  
cat:pred  
\* id \* \* tu \* \* ignoscas \* \* mihi \*  
illocutionary\_force:statement  
number:sing  
person:3  
vgperson:2  
gender:neuter  
mood:subjunctive  
tense:present  
polarity:pos  
argbound:no

add:no  
c\_str  
illocutionary\_force:statement  
vg  
selected\_reading:ignosco\_pardon  
polarity:pos  
cat:vg  
\* ignoscas \*  
pos:v  
lex:ignoscere  
voice:act  
tense:present  
mood:subjunctive  
number:sing  
person:2  
subject  
number:sing  
gender:or([masc,fem])  
\* tu \*  
index:i(p(1,2))  
cat:np  
sem:[hum]  
lex:pp2sg  
person:2  
case:nom  
object  
\* id \*  
index:i(p(0,1))  
cat:np  
lex:prpersnomaccneutersing  
number:sing  
person:3  
gender:neuter  
case:or([nom,acc])  
i\_object  
\* mihi \*  
index:i(p(7,8))  
cat:np  
sem:[hum]  
lex:pp1sg  
number:sing  
person:1  
gender:or([masc,fem])  
case:dat

[0/otho,1/reuocatis,2/quos,3/galba,4/miserat,5/legatis,  
6/ad,7/utrumque,8/germanicum,9/exercitum,10/copias,11/misit,endpos(12)]

```
illocutionary_force:statement
vg
  selected_reading:mitto_send
  polarity:pos
  cat:vg
  * misit *
  pos:v
  lex:mittere
  voice:act
  tense:perfect
  mood:indicative
  number:sing
  person:3
  subject
    number:sing
    gender:masc
    * otho *
    index:i(p(0,1))
    cat:np
    sem:[hum]
    lex:otho
    person:3
    case:nom
  object
    * copias *
    index:i(p(10,11))
    cat:np
    sem:[hum]
    number:pl
    person:3
    gender:fem
    lex:copiae
    case:acc
    c_str
    copias
  prep_cplt
    * ad * * utrumque * * germanicum * * exercitum *
    index:i(p(9,10))
    case:acc
    prep:ad
    sem:[thing,hum,loc]
    lex:exercitus
    cat:pp
    c_str
    prep:ad
    head
      * utrumque * * germanicum * * exercitum *
      index:i(p(9,10))
      cat:np
      sem:[thing,hum,loc]
      number:sing
      person:3
      gender:masc
      lex:exercitus
      case:acc
      c_str
      head
        * exercitum *
        pos:noun
        lex:exercitus
        case:acc
        gender:masc
        number:sing
        sem:[thing,hum,loc]
      adjp
        cat:adjp
        * utrumque * * germanicum *
        case:acc
        number:sing
        gender:masc
        lex:germanicus
        c_str
        uterque
```

germanicus  
 ablative\_absolute  
 \* reuocatis \* \* quos \* \* galba \* \* miserat \* \* legatis \*  
 c\_str  
 lex:reuoco\_call\_back  
 pos:p\_p  
 case:abl  
 gender:masc  
 number:pl  
 lex:reuocare  
 mood:participle  
 person:3  
 object  
 \* quos \* \* galba \* \* miserat \* \* legatis \*  
 cat:np  
 index:i(p(5,6))  
 number:pl  
 gender:masc  
 sem:[hum]  
 person:3  
 case:abl  
 lex:legatus  
 c\_str  
 head  
 legatis  
 rel\_clause  
 \* quos \* \* galba \* \* miserat \*  
 index:i(p(5,6))  
 number:pl  
 gender:masc  
 case:acc  
 reltype:restrictive  
 mood:indicative  
 tense:pluperfect  
 c\_str  
 illocutionary\_force:statement  
 vg  
 selected\_reading:mitto\_send  
 polarity:pos  
 cat:vg  
 \* miserat \*  
 pos:v  
 lex:mittere  
 voice:act  
 tense:pluperfect  
 mood:indicative  
 number:sing  
 person:3  
 subject  
 number:sing  
 gender:masc  
 \* galba \*  
 index:i(p(3,4))  
 cat:np  
 sem:[hum]  
 lex:galba  
 person:3  
 case:nom  
 object  
 e:i(p(5,6))

[0/reuocatis,1/quos,2/galba,3/miserat,4/legatos,endpos(5)]

illocutionary\_force:statement

vg

selected\_reading:reuoco\_call\_back

polarity:pos

cat:vg

\* reuocatis \*

pos:v

lex:reuocare

voice:act

tense:present

mood:indicative

number:pl

person:2

subject

source:context\_retrievable

number:pl

gender:or([masc,fem])

person:2

cat:np

index:i(0,0)

constraints\_to\_be\_met:[sem:[hum]]

case:nom

object

\* quos \* \* galba \* \* miserat \* \* legatos \*

cat:np

index:i(p(4,5))

number:pl

gender:masc

sem:[hum]

person:3

case:acc

lex:legatus

c\_str

head

legatos

rel\_clause

\* quos \* \* galba \* \* miserat \*

index:i(p(4,5))

number:pl

gender:masc

case:acc

reltype:restrictive

mood:indicative

tense:pluperfect

c\_str

illocutionary\_force:statement

vg

selected\_reading:mitto\_send

polarity:pos

cat:vg

\* miserat \*

pos:v

lex:mittere

voice:act

tense:pluperfect

mood:indicative

number:sing

person:3

subject

number:sing

gender:masc

\* galba \*

index:i(p(2,3))

cat:np

sem:[hum]

lex:galba

person:3

case:nom

object

e:i(p(4,5))









## **Appendix I ALP tackles coordination ... during a quick ... coffee break**

AUGUSTINE GOES TO CHURCH TO BUY THE SERVICES OF A PROSTITUTE

Conf. 3, III,5, 6-9 Budé 1933 de Labriolle

*"Ausus sum etiam in celebritate sollemnitatum tuarum intra parietes ecclesiae tuae concupiscere et agere negotium procurandi fructus mortis"*

De Labriolle: "N'ai-je pas osé, en pleine célébration de vos solennités, dans l'enceinte de votre église, convoiter des fruits de mort et négocier le moyen de me les procurer?" (qu'en termes élégants...)

Arnaud d'Andilly: "Mon impudence passa même jusqu'à ce point, qu'en l'une de vos fêtes les plus solennelles, et dans votre propre église, j'osai concevoir un désir damnable et ménager un accord funeste qui ne pouvoit produire que des fruits de mort."  
(no marks for concision...)

William Watts (Loeb Classical Library): "I was so bold one day, as thy solemnities were a celebrating, even within the walls of thy Church, to desire and to execute a business, enough to purchase me the very fruits of death."  
(not very clear...)

De Labriolle (rightly, it seems to me) parses 'fructus mortis' as the object of both 'concupiscere' and 'procurandi' - that's well beyond the power of ALP !!!

We use a somewhat simpler version:

*Ausus sum in ecclesia tua concupiscere fructus mortis et agere negotium procurandi eos.*

[0/ausus,1/sum,2/in,3/ecclesia,4/tua,5/concupiscere,6/fructus,7/mortis,8/et,9/agere,10/negotium,11/procurandi,12/eos,endpos(13)]  
cputime : cputime : 217.53125 sec i.e. **more than three minutes !!!**

illocutionary\_force:statement

vg

selected\_reading:audeo\_dare

polarity:pos

cat:vg

\* ausus \* \* sum \*

lex:audere

person:1

mood:indicative

tense:perfect

voice:act

number:sing

gender:masc

subject

source:context\_retrievable

number:sing

gender:or([masc,fem])

person:1

cat:np

index:i(0,0)

constraints\_to\_be\_met:[case:or([nom,acc]),sem:[hum]]

case:nom

object

cat:pred

mood:infinitive

\* in \* \* ecclesia \* \* tua \* \* concupiscere \* \* fructus \* \* mortis \* \* et \* \* agere \* \* negotium \* \* procurandi \* \* eos \*

case:or([nom,acc])

c\_str

head

vg

selected\_reading:concupisco\_desire

polarity:pos

cat:vg

\* concupiscere \*

pos:v

lex:concupiscere

voice:act

tense:present

mood:infinitive

object

\* fructus \* \* mortis \*

index:i([p(6,7)])

sem:[thing,abstract]

number:pl

person:3

cat:np

lex:fructus

gender:masc

case:acc

c\_str

head

\* fructus \*

index:i(p(6,7))

cat:np

sem:[thing,abstract]

number:pl

person:3

gender:masc

lex:fructus

case:acc

c\_str

fructus

noun\_cplt

\* mortis \*

index:i(p(7,8))

cat:np

sem:[abstract,thing,hum]

number:sing

person:3

gender:fem

lex:mors

case:gen

c\_str

mortis

prep\_phrase\_adjunct\_3

\* in \* \* ecclesia \* \* tua \*

index:i(p(3,4))

case:abl

prep:in

sem:[hum,thing,loc]

lex:ecclesia

cat:pp

c\_str

prep:in

head

```

* ecclesia * * tua *
index:i(p(3,4))
cat:np
sem:[hum,thing,loc]
number:sing
person:3
gender:fem
lex:ecclesia
case:abl
c_str
  head
    * ecclesia *
    pos:noun
    lex:ecclesia
    case:abl
    gender:fem
    number:sing
    sem:[hum,thing,loc]
  adjp
    cat:adjp
    * tua *
    case:abl
    number:sing
    gender:fem
    lex:tuus
coord:et
head
vg
  selected_reading:negotium_ago_take_care_of
  polarity:pos
  cat:vg
  * agere *
  pos:v
  lex:agere
  voice:act
  tense:present
  mood:infinitive
object
  * negotium *
  index:i(p(10,11))
  cat:np
  sem:[abstract]
  number:sing
  person:3
  gender:neuter
  lex:negotium
  case:acc
  c_str
  negotium
object_cplt
  cat:pred
  mood:gerund
  local_case:gen
  number:sing
  person:3
  gender:neuter
  * procurandi * * eos *
  c_str
  vg
    selected_reading:procuro_procurare
    pos:v
    lex:procurare
    mood:gerund
    person:3
    case:gen
  object
    * eos *
    index:i(p(12,13))
    cat:np
    lex:prpersacemasepl
    number:pl
    person:3
    gender:masc
    case:acc

```

Such a result is to be seen in an optimistic light – ALP plods on, but comes up with a correct parse. Much better than just giving up, or, worse, turning out rubbish.

It should be kept in mind that the most time-consuming phrases (in terms of parsing, of course) are the ones that are not bound to any argument but function as clause-level adjuncts. As a general rule, a fifteen-word limit imposed on strings to be parsed seems reasonable. Consider the following data based on Livius, *Ab Urbe Condita*, Liber XXI, 8:

*Per totum tempus hiemis quies inter labores iam exhaustos aut mox exhauriendos renovavit corpora animosque ad omnia de integro patienda.*

*Quies renovavit corpora animosque.*

5 words

cputime : 0.15

*Quies inter labores renovavit corpora animosque.*

7 words

cputime : 1.14

*Quies inter labores renovavit corpora animosque ad omnia patienda.*

10 words

cputime : 18.37

*Quies inter labores renovavit corpora animosque ad omnia de integro patienda.*

12 words

cputime : 21.65

*Quies inter labores exhaustos aut exhauriendos renovavit corpora animosque ad omnia de integro patienda.*

15 words

cputime : 150.00

The parses produced I believe to be correct. Here is the one associated with the 15-word string:

```
illocutionary_force:question
vg
  selected_reading:renouo_renew
  polarity:pos
  cat:vg
  ** renouavit **
  pos:v
  lex:renouare
  voice:act
  tense:perfect
  mood:indicative
  number:sing
  person:3
subject
  number:sing
  gender:fem
  ** quies **
  index:i(p(0,1))
  cat:np
  sem:[abstract]
  person:3
  lex:quies
  case:nom
  c_str
  quies
object
  ** corpora ** ** que ** ** animos **
  index:i([p(7,8),p(9,10)])
  sem:[thing]
  lex:corpus
  number:pl
  person:3
  gender:masc
  case:acc
  coord:yes
  c_str
  head
  corpora
```



case:acc  
gender:masc  
number:pl  
sem:[abstract]  
adjp  
cat:adjp  
\*\* exhaustos \*\* \*\* aut \*\* \*\* exhauriendos \*\*  
case:acc  
number:pl  
gender:masc  
lex:exhaustos  
c\_str  
  head  
  exhaustos  
  head  
  exhauriendos

## Appendix II How to Use ALP

ALP is made up of three SWI-Prolog programs :

*alp.pl* : main program

*template.pl* : lexical entries for *argument bearers*, mainly verbs

*makelex.pl* : program building 'vocfile', i.e. the morphological variants needed by ALP

Once SWI-Prolog has been properly installed, ALP is ready for use.

Both *alp* and *makelex* are executables, with a GO step. To execute the programs, simply type in : *go.* (don't forget the final dot). In the case of *makelex*, press the *Enter key* once the system has replied with '*true*'.

On execution, ALP will load both the *vocfile* produced by *makelex.pl* and the *template.pl* file, provided the files are reachable by ALP.

The easiest way to use ALP, therefore, is to have a single directory containing both the executable files (*alp.pl* and *makelex.pl*), the *template.pl* file, the *vocfile* and the test files that one wishes to submit to the parsing process.

If no testfile is available, *stdin* can be used as input stream. The sentences one wishes to parse are then to be entered directly from the keyboard, and should end with either a *dot*, a *question mark*, or a *slash* (see below).

The test file names should preferably be made up of a single atom, such as *test*, *testfile*, *mytest*, *alptest*, etc.

The test files must end on a line that is made up of the single word *stop* followed by a dot, i.e. *stop.*

The result file name (i.e. the name of the file containing the parses) should also be a single atom, to which the system will add a *.lst* extension. The filename selected by the user for output can therefore be the same as for input (*test* → *test.lst*).

Each sentence in a test file should end with a final punctuation sign, either dot (*.*), question mark (*?*), or a slash (*/*). The question mark is not used by the parser as an indication that it should parse the string as a question. The slash is used for partial structures, as in :

% 20.

*Ut amicitia societasque nostra in aeternum rata sint /*

% 21.

*postulamus a uobis muros coloniae munimenta servitii detrahatis.*

Commas are best left out, unless they are used to isolate an *addressee* (vocative), as in :

*Argiletanas mauis habitare tabernas cum tibi, parue liber, scrinia nostra vacent.*

*Incipe, parve puer, risu cognoscere matrem.*

*Hanc etiam vix, Tityre, duco.*

Or to guide *distributed* structures as in:



*Contemnunt novitatem meam, ego illorum ignaviam.*  
*Mihi fortuna, illis probra obiectantur.*  
*Nulli parietes nostram salutem, nullae leges, nulla iura custodient.*  
*Insidiatores ab Othone in Germaniam, a Vitellio in urbem missi.*  
*Versabitur in rostris furor, in curia timor, in foro coniuratio, in Campo exercitus, in agris vastitas.*

Comment lines open with the percentage sign (%) and close with a dot (.). No dot can be used within a comment line. See the test file for examples.

The following settings are made available in the ALP settings menu:

*Depth Level for Finite Pass*

- 1. Shallow % default
- 2. Intermediate
- 3. Deep

*Type of Parses Displayed*

- 1. Whole Sentences Only % default
- 2. Phrases and Clauses

*Number of Parses Displayed*

- 0. one parse only
- 1. one parse only, or two if the second parse has the same ranking as the first % default
- 2. two parses, whatever the ranking of the second
- 3. all available parses

The Prolog programs are *ascii* files, and can therefore be explored with a large number of tools. The editor within SWI-Prolog is a wise choice (type in *edit.* once the program has loaded).

The *vocfile* produced by *makelex* is also an *ascii* file. It can be sorted and explored outside of Prolog.

To be able to add vocabulary to ALP one should make a careful study of the lexical items that ALP already features. We should emphasize that there is no point in adding a verb to *makelex* without giving it an argument structure in *template*. Only then can it be used in a sentence to be parsed.

## Appendix III Parsing and Word Sense Assignment

On top of revealing syntactic structure a parser should attempt to assign word senses to the lexical items that make up its leaves. It should not stop at the lemmatisation stage, in so far as word sense recognition ties up with structural assignment.

Wordsenses do not exist out there : their status as lexicographical constructs is not in doubt. But recognizing their true nature as constructs should not detract from acknowledging their usefulness in the accomplishment of a number of NLP-oriented tasks, translation being an outstanding example.

As a matter of fact, any description of sense is a construct. There is a lot to be said in favour of keeping the metalinguistic description to a minimum, so that belonging to a WordNet *synset* may be the most adequate way of specifying a wordsense. Each word sense of a lexical item should give rise to assignment to a different synset. The synset glosses should serve for orientation only, and are not to be confused with definitions. Definitions are part of the exploitable lexicographical description, as will be shown below.

A good starting point is an examination of the wordsenses assigned by the body of lexicographical work for a given language, in a monolingual perspective first, but without rejecting a bilingual or multilingual framework (translation-oriented dictionaries).

The question of granularity is likely to be settled by the range and depth of the analysis tools that we are able to devise.

For the items that the parser recognizes as *argument bearers*, we shall need to specify word sense descriptions for the arguments as well as for the arg bearer itself. We shall have to proceed step by step. In a first stage, we suggest that an argument can be described by a *lexical world*, i.e. a set of lemmas gleaned mainly from the examples offered by the lexicographical resources. See below for two wordsenses of *colo* as described in Forcellini.

A *lexical world* should exclude *toolwords* which we can store in a *stoplist*, in order to be able to concentrate on full lexical items. In specific cases, we may wish to register wordforms instead of lemmas, and we should be ready to house descriptions of any degree of specificity in the case of *multi-word units* (phraseological component).

Once our tools are powerful enough, we can use them in a bootstrapping procedure by applying them to the dictionary-registered citations, attempting to bring the granularity level of our analysis down to word senses.

In the parsing process, to work out whether an item in the string to be parsed potentially filling in an arg position is to be parsed as filling that position indeed, we have to measure the lexical distance between the item in text and the lemmas belonging to the lexical world assigned to that arg position in the description the parser makes use of.

In the standard case, where the arg bearer has a number of word senses, each word sense will entail a specific description for the args, and the measure of lexical distance should lead to the selection of the appropriate word senses, both at the level of the arg and at that of the arg bearer.

Lexical distance can be equated with the length of the path that needs to be covered for two items to meet along synset and synset-extending relations (hyponymy, etc). In our case, the two items should

be pairs resulting from the pairing of the item found in the text to be parsed and each member of the lexical world associated with the arg under a given wordsense. The lexical distance to be used for decision taking with respect to wordsense selection would be based on a mean value computed on all the pairs.

We would need to specify limits to the exploration of lexical distance. If the two items do not meet on a path of length  $l$  they will be deemed to be unrelated (although they may of course be related at a higher level).

## Two word senses of COLO in Forcellini

*Colo, colare vs colo, colere* : mostly decided on the basis of lemmatisation ; if necessary (parsing the the very form *colo*), on the basis of the object arg.

*Colo, colere.*

We should exploit, not only the examples, but also the definitions and semantic characterization with respect to *habitare, observare, venerari*.

The *lexical worlds* can be hand-crafted or, more roughly, computer-derived. As a matter of fact, if we insert the missing headword in the examples where it does not occur, and decorate it with the appropriate wordsense number, or simply add the wordsense number where the headword occurs, and then build up a concordance excluding stopwords, we would have a most useful tool for specifying the lexical worlds that we have in mind.

Definition : **studium, operam, laborem** pono in re aliqua **perficienda, assiduus** sum circa rem aliquem, **exerceo, excolo, curo**

(It. *lavorare, coltivare, esercitare*; Fr. *travailler, cultiver, soigner*; Hisp. *labrar, cultivar*; Germ. *etwas treiben, betreiben, um etwas herum sein Geschäft treiben*; Angl. *to exercise, labour upon, cultivate*).

We concentrate on the object arg for the two wordsenses that ALP has so far been concerned with :

```
lexarg(colere,
  arglist:[ws(colo_inhabit,tr_cod,clause:[],mwuw:0,
    args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
      object:[type:np,oblig:yes,constraints:[case:acc, sem:[loc]]]),
    ws(colo_practise,tr_cod,clause:[],mwuw:0,
      args:[subject:[type:np,oblig:yes,constraints:[sem:[hum]]],
        object:[type:np,oblig:yes,constraints:[case:acc, sem:[abstract]]])])].
```

¶ 2. Quoniam qui agros colunt, ibidem **permanere** et **habitare** necessario debent, (qua in re populi agricolae a pastoribus distinguuntur); vel etiam quoniam loco, in quo habitamus, curam impendimus: factum est, ut colere ponatur saepissime pro habitare, quamvis aliquantulum ab ipso differat: *V. Homonym. in fin. Occurrit autem* -

*a) Cum Accusativo loci. Plaut. Aulul. prol. 3. Hanc domum jam multos annos est cum possideo et colo. Id. Bacch 2. 2. 21. Colere regiones Acberunticas. Cic. 2. Fam. 12. 2. Urbem, urbem, mi Rufe, cole et in ista luce vive. Forcellinus recte hinc accepit colere pro frequentem et assiduum esse in loco aliquo; quod tamen idem est atque habitare. Id. 2. Nat. D. 66. 164. Homines, qui has nobiscum*

terras ab oriente ad occidentem colunt. Adde *Tac.* 2. *Ann.* 60. *Lucret.* 5. 953. Colere nemora, montes silvasque. *Catull.* 63. 70. Idae loca. *Virg.* 4. *AEn.* 343. urbem Trojanam. *Id.* 5. *ibid.* 735. Elysium. *Ovid.* 4. *Fast.* 487. Unaque, pastorem vidisset an arva colentem, Vox erat. *Id.* 11. *Met.* 146. Colere silvas et rura. *Curt.* 4. 8. 6. Urbem advenarum frequentia cultura. *Id.* 7. 7. 4. Colere regionem. *Id.* 9. 2. 3.; et *Tac.* 1. *Ann.* 59. et *Germ.* 28. ripam. - Et de Poetis *Propert.* 3. 5. 41. Me juvat in primâ coluisse Heliconâ juventâ. - De brutis vero *Virg.* 3. *G.* 430. Anguis stagna colat. *Ovid.* 2. *Met.* 380. Quae colat elegit (*cygnus*) contraria flumina flammis. -

*Lexical world* : {domus, regio, urbs, terra, nemus, mons, silva, locus, Elysium, arvum, rus, ripa, Helicon, stagnum, flumen}

¶ 2. Saepissime, morali ratione, de rebus ponitur, et est **curam** habere, **fovere**, **plurimi facere**, **honorare**. *Cic.* 1. *Off.* 2. 4. In officio colendo sita est vitae honestas omnis, in negligendo turpitudine. *Ovid.* 2. *Art. am.* 121. ingenuas pectus coluisse per artes Cura sit. - Hac significatione jungitur pluribus Substantivis, quorum praecipua, servato litterarum ordine, exhibentur. *Plaut.* *Men.* 4. 2. 10. aequum et bonum. *Lucret.* 5. 1144. aevum. *h. e.* aetatem agere. *Cic.* 3. *Herenn.* 3. 4. affinitates. *Plaut.* *Cist.* 1. 1. 23. et *Cic.* 1. *Off.* 2. 5. amicitiam. *Sall.* *Jug.* 8. amicitiam populi Romani. *Cic.* 2. *Fin.* 26. 83. et *Amic.* 22. 35. amicitias. *Curt.* 8. 2. 32. amicitiam cum fide. *Virg.* 11. *AEn.* 584. et *Tibull.* 2. 4. 52. amorem. *Cic.* 3. *Fam.* 13.; *Ovid.* 2. *Fast.* 508. et *Sueton.* *Tib.* 60. artes. *Plaut.* *Trin.* 2. 2. 16. artes malas. *Propert.* 3. 13. 48. aurum victâ pietate. *Liv.* 7. 30. beneficium acceptum. *Cic.* 3. *Herenn.* 3. 4. clientelas, cognationes. *Id.* 1. *Off.* 41. 149. conciliationem et consociationem communem totius generis hominum. *Id.* 2. *Fin.* 31. 102. diem natalem alicujus. *Id.* 2. *Orat.* 35. 148. diligentiam. *Id.* *Brut.* 31. 117. disciplinam aliquam. *Plaut.* *Capt.* 2. 1. 26. dolos astu. *Cic.* 1. *Invent.* 2. 3. et *Curt.* 7. 8. fidem. *Ovid.* 1. *Met.* 90. fidem rectumque. *Curt.* 10. 3. 9. fortitudinem. *Cic.* 1. *Off.* 1. 3. genus orationis aequabile et temperatum. *Id.* 3. *Herenn.* 3. 4. hospitium. *Id.* *ibid.* et 1. *Legg.* 12. 33. jus. *Liv.* 27. 17. jus et fas. *Cic.* 1. *Off.* 2. 5. justitiam et liberalitatem. Adde *eumd.* 2. *Tusc.* 13. 31. et 3. *Fin.* 22. 71. *Id.* *Parad.* 5. 1. . 34. leges sequi et colere. *Sil. It.* 8. 464. Martem. *h. e.* bellum. *Cic.* 2. *Fin.* 31. 101. memoriam defuncti. *Id.* *post red. ad Quirit.* 10. 24. memoriam beneficii colere benevolentia sempiterna. *Ovid.* 2. *Fast.* 518. militiam. *Plin. Paneg.* 1. morem. *h. e.* sequi. *Plaut.* *Cas.* 5. 4. 1. mores Massilienses. *Sall.* *Cat.* 9. mores bonos. *Martial.* 9. 12. Musas severiores. *Cic.* 1. *Legg.* 5. 16. munus aliquod colere et efficere. *Plaut.* *Stich.* 1. 1. 35. officium suum. *Liv.* 45. 8. et *Ovid.* 11. *Met.* 297. pacem. *Plaut.* *Asin.* 3. 1. 6.; *Ter. Hecyr.* 3. 4. 33.; et *Cic.* 3. *Off.* 21. 82. pietatem. *Cic.* 5. *Verr.* 3. 6. pudorem et pudicitiam. *Plaut.* *Poen.* 5. 2. 137. quaestum suum. *Cic.* *Flacc.* 4. 9. religionem et fidem testimoniorum. Cf. *eumd.* *Fontej.* 10. 21. *Plaut.* *Poen.* 4. 2. 7. servitum colere apud aliquem. *h. e.* servire. *Cic.* 3. *Herenn.* 3. 4. societates. *Id.* 5. *Fin.* 19. 50. et 3. *Fam.* 13. studia. *Id.* *Brut.* 91. 315. studium philosophiae. *Id.* 12. *Att.* 28. nec victum, nec vitam. *Id.* *fragm.* apud *Non.* 4. 280. *edit. vet.* victum lautum et elegantem. *Cic.* *Arch.* 7. 16. virtutem percipere et colere. *Plaut.* *Rud.* 1. 5. 25. vitam. *h. e.* agere, seu vivere. Adde *eumd.* *Trin.* 3. 2. 74. Sic *Ter. Heaut.* 1. 1. 84. vitam inopem colere. Adde *Cic.* 4. *Herenn.* 14. 21. et 12. *Att.* 28.; et *Virg.* 3. *G.* 532.

*Lexical world 1*: {officium, aequum, bonum, affinitas, amicitia, amor, ars, aurum, beneficium, clientela, cognatio, fides, fortitudo, hospitium, ius, fas, lex, Mars, bellum, memoria, militia, mos, Musa, munus, pax, pietas, pudor, pudicitia, quaestus, religio, servitus, societas, studium}

*Lexical world 2*: {aevum, aetas, vita, militia, victus} : basis for a semi-open mwu : vitam colere/agere ? The decision depends on whether militiam colere is ambiguous with respect to militia as military service vs military spirit, with colo as spend/pass vs cultivate/cherish (see Lewis and Short)

# Word sense assignment in ALP

*Liceat nobis vobisque utramque ripam colere.*

```
illocutionary_force:statement
vg
  selected_reading:licet_permission_is_given_to
  polarity:pos
  cat:vg
  * liceat *
  pos:v
  lex:licet
  voice:act
  tense:present
  mood:subjunctive
subject
  cat:pred
  mood:infinitive
  tense:present
  * nobis * * que * * uobis * * utramque * * ripam * * colere *
  case:or([nom,acc])
  polarity:pos
  c_str
    vg
      selected_reading:colo_inhabit
      polarity:pos
      cat:vg
      * colere *
      pos:v
      lex:colere
      voice:act
      tense:present
      mood:infinitive
    object
      * utramque * * ripam *
      index:i(p(5,6))
      cat:np
      sem:[thing,loc]
      number:sing
      person:3
      gender:fem
      lex:ripa
      case:acc
      c_str
        head
          * ripam *
          pos:noun
          lex:ripa
          case:acc
          gender:fem
          number:sing
          sem:[thing,loc]
        adjp
          cat:adjp
          * utramque *
          case:acc
          number:sing
          gender:fem
          lex:uterque
          c_str
            uterque
    dative_of_interest
      * nobis * * que * * uobis *
      index:i([p(1,2)])
      sem:[hum]
      lex:pp1pl
      number:pl
      person:1
      gender:or([masc,fem])
      case:dat
      coord:yes
      c_str
        head
          pp1pl
          coord:que
        head
          pp2pl
```

Missi sunt cultum virtutem.

```
illocutionary_force:statement
vg
  selected_reading:mitto_send
  polarity:pos
  cat:vg
  * missi * * sunt *
  lex:mittere
  person:3
  mood:indicative
  tense:perfect
  voice:pass
  number:pl
  gender:masc
subject
source:context_retrievable
number:pl
person:3
cat:np
index:i(0,0)
constraints_to_be_met:[case:nom]
case:nom
cplt
cat:pred
mood:supine
tense:present
* cultum * * uirtutem *
number:sing
gender:neuter
case:or([nom,acc])
person:3
c_str
  vg
    selected_reading:colo_practise
    polarity:pos
    cat:vg
    * cultum *
    pos:v
    lex:colere
    mood:supine
    person:3
  object
    * uirtutem *
    index:i(p(3,4))
    cat:np
    sem:[abstract,hum]
    number:sing
    person:3
    gender:fem
    lex:uirtus
    case:acc
    c_str
      uirtutem
```

In the case of *colo* the specifications so far are in terms of **semantic features**, whose hierarchy is explored by the feature unification algorithm, so that hyponymy is taken into account (we are asking for a dog, we are given a poodle, OK ; we are asking for a poodle, we are given a dog, noway).

In a sentence such as *Ille sapientissimus vir Ithacam ut videret immortalitatem scribitur repudiasse*, we should realize that inserting commas around *Ithacam ut videret* amounts to guiding the parsing; as a matter of fact, it can be seen as a first, important step in parsing the sentence. But we should try to do without such commas, as they do not belong to the text but were introduced at various points in time by editors, sometimes in a way that reflects usage in one of our modern languages, such as the comma we find following *vereor* and preceding *ne* in the Budé edition of the *De Amicitia*, 14, which is more appropriate to German than to French, the language of Combès himself:

*Quocirca maerere hoc eius eventu uereor, ne inuidi magis quam amici sit.* (edited by Robert Combès, Paris, 1971)

Here too selectional restrictions are essential. Not only to discriminate between *repudiare* as *reject* and *repudiare* as *repudiate*, but to help the parser assign *ithacam* as object of *videret* and *immortalitatem* as object of *repudiasse*.

The two readings of *repudiare* place different selectional restrictions on the object:

```
% REPUDIARE
lexarg(repudiare,
      arglist:[ws(repudio1_reject, tr_cod, clause:[], mwuw:0,
                  args:[subject:[type:np, oblig:yes, constraints:[sem:[hum]]],
                        object:[type:np, oblig:yes, constraints:[case:acc, sem:[abstract] ]]])]).

lexarg(repudiare,
      arglist:[ws(repudio2_repudiate, tr_cod, clause:[], mwuw:0,
                  args:[subject:[type:np, oblig:yes, constraints:[sem:[hum]]],
                        object:[type:np, oblig:yes, constraints:[case:acc, sem:[hum]]]])]).
```

Our test sentence, with no commas added:

```
% 77 Cicero De Legibus II 1 3.
```

```
Ille sapientissimus vir Ithacam ut videret immortalitatem scribitur repudiasse.
```

is assigned the following parse:

```

subordinator
lex:ut
pos:sub
mood:subjunctive
value:or([purpose,consequence])
subordinate_clause
ut
  illocutionary_force:statement
  vg
    selected_reading:uideo_see
    polarity:pos
    cat:vg
    * uideret *
    pos:v
    lex:uidere
    voice:act
    tense:imperfect
    mood:subjunctive
    number:sing
    person:3
  subject
    source:context_retrievable
    number:sing
    person:3
    cat:np
    index:i(0,0)
    case:nom
  object
    * ithacam *
    index:i(p(3,4))
    cat:np
    sem:[loc]
    number:sing
    person:3
    gender:fem
    lex:ithaca
    case:acc
    c_str
    ithacam
main_clause
vg
  polarity:pos
  cat:vg
  * scribitur *
  pos:v
  lex:scribere
  voice:pass
  tense:present
  mood:indicative
  number:sing
  person:3
pred
subject
  * ille * * sapientissimus * * uir *
  index:i(p(2,3))
  cat:np
  sem:[hum]
  number:sing
  person:3
  gender:masc
  lex:uir
  case:nom
  c_str
  _head
    * uir *
    pos:noun
    lex:uir
    case:nom
    gender:masc
    number:sing
    sem:[hum]
  adjp
    cat:adjp
    * ille * * sapientissimus *
    case:nom
    number:sing
    gender:masc
    lex:sapiens
    c_str
    ille
    sapiens
pred
vg
  selected_reading:repudiol_reject
  polarity:pos
  cat:vg
  * repudiauise *
  pos:v
  lex:repudiare
  voice:act
  tense:past
  mood:infinitive
  object
    * immortalitatem *
    index:i(p(6,7))
    cat:np
    sem:[abstract]
    number:sing
    person:3
    gender:fem
    lex:immortalitas
    case:acc
    c_str
    immortalitatem

```



## Appendix IV Syncrétisme grammatical ou ambiguïté de construction ?

Considérons Tacite Hist. I 79 1 :

*conuersis ad ciuile bellum animis externa sine cura habebantur*

La ponctuation que nous allons introduire n'est pas innocente :

*Conuersis ad ciuile bellum animis, externa sine cura habebantur.*

Avec le marquage des limites de l'énoncé et sa division à l'aide de la virgule, l'interprétation de *conuersis ad ciuile bellum animis* comme ablatif absolu est fortement suggérée. C'est le texte proposé entre autres par Bornecque, et sa traduction (inspirée de celle de Burnouf) est conforme à cette interprétation qui scinde la phrase en deux :

*Les esprits tournés à la guerre civile, les questions extérieures étaient négligées.*

Le lien entre les deux parties de la phrase est laissé à l'inférence, procédé d'interprétation que l'on sait universel autant qu'automatique. Comme le dit très bien Lavency dans VSVS, l'ablatif absolu *décrit la situation concomitante au procès signifié dans la proposition principale*. C'est l'inférence qui donne à cette concomitance la possibilité de se teinter de la valeur sémantique et discursive qui complète son insertion dans l'énoncé.

Si l'on va voir chez Burnouf, on constate que sa traduction a bel et bien été remaniée par Bornecque. Burnouf traduisait en effet :

*Les esprits tournés à la guerre civile ne songeaient plus aux dangers du dehors.*

Ici l'énoncé est un seul bloc et le lien entre les deux parties n'est pas laissé à l'inférence mais matérialisé par le rapport sujet-verbe.

Goelzer va plus loin encore dans l'intégration des deux parties. Il interprète *externa* comme adjectif épithète, le *bella* qu'il modifie se déduisant sans peine du *bellum* de la première partie<sup>3</sup> :

*Les esprits tournés à la guerre civile n'avaient plus souci des guerres étrangères ;*

Notons bien le point virgule en clôture d'énoncé. Ce qui suit est en effet la relation d'une attaque sarmate en Mésie, un acte de guerre :

*conuersis ad ciuile bellum animis externa sine cura habebantur eo audentius rhoxolani sarmatica gens priore hieme caesis duabus cohortibus magna spe moesiam inruperant ad nouem milia equitum ex ferocia et successu praedae magis quam pugnae intenta*

Si le *conuersis ad ciuile bellum animis* n'est pas un ablatif absolu, comment faut-il l'analyser ? On peut en faire un complément d'agent de *sine cura habebantur*, *animis* ne nécessitant pas de préposition pour jouer ce rôle.

3 ALP n'a pas la sophistication nécessaire pour prendre en considération l'analyse de *externa* comme adjectif modifiant un *bella* sous-entendu. Il fait d'*externis* l'ablatif d'un adjectif substantivé neutre pluriel, *externa*, que l'on rencontre chez Tacite au même titre que l'adjectif *externus* non substantivé. Il me semble préférable de considérer *externa* comme substantivé avec le sens de 'ce qui concerne l'extérieur', les 'affaires extérieures'. En effet la guerre n'est pas négligée, la réponse à l'attaque des Sarmates est décisive et musclée, mais c'est le climat d'indifférence envers ce qui n'est pas Rome ou l'Italie qui a offert l'occasion de la guerre aux Sarmates. Je propose : *Les esprits préoccupés par la guerre civile n'avaient cure des affaires extérieures.*

Tacite ne se proposait pas d'accompagner son texte d'arbres syntaxiques qui nous indiqueraient la lecture qu'il avait en tête au moment de l'écriture. On connaît son goût pour l'ablatif absolu. Il peut très bien avoir commencé en ablatif absolu et continué en complément d'agent, se rendant coupable (ou enrichissant son texte) de syncrétisme grammatical.

Il est intéressant de constater que ALP retient les deux analyses, les jugeant donc conformes à la grammaire sur laquelle se base son algorithme de passage. Conformément au système de pondération qui favorise les arguments au détriment des satellites (tels les circonstanciels et les ablatifs absolus), l'analyse de *conuersis animis* comme complément d'agent est préférée à l'analyse qui en fait un ablatif absolu.

[0/conuersis,1/ad,2/ciuile,3/bellum,4/animis,5/externa,6/sine,7/cura,8/habebantur,endpos(9)]

7.47--->

```
illocutionary_force:statement
vg
  selected_reading:habeo_have
  polarity:pos
  cat:vg
  * habebantur *
  pos:v
  lex:habere
  voice:pass
  tense:imperfect
  mood:indicative
  number:pl
  person:3
subject
  number:pl
  gender:neuter
  * externa *
  index:i(p(5,6))
  cat:np
  sem:[abstract,thing]
  person:3
  lex:externus
  case:nom
  c_str
  externa
agent
* conuersis * * ad * * ciuile * * bellum * * animis *
index:i(p(4,5))
cat:np
sem:[hum,thing,abstract]
number:pl
person:3
gender:fem
lex:anima
case:abl
c_str
  head
    * animis *
    index:i(p(4,5))
    cat:np
    sem:[hum,thing,abstract]
    number:pl
    person:3
    gender:fem
    lex:anima
    case:abl
    c_str
    animis
  participle_clause
    * conuersis * * ad * * ciuile * * bellum *
    cat:ppclause
    number:pl
    gender:fem
    lex:conuerto_turn
    case:abl
    c_str
      past_participle
        conuerto_turn
          prep_cplt
            * ad * * ciuile * * bellum *
            index:i(p(3,4))
            case:acc
            prep:ad
            sem:[thing]
            lex:bellum
            cat:pp
            c_str
              prep:ad
              head
                * ciuile * * bellum *
                index:i(p(3,4))
                cat:np
                sem:[thing]
                number:sing
                person:3
                gender:neuter
                lex:bellum
                case:acc
                c_str
                  head
                    * bellum *
                    pos:noun
                    lex:bellum
                    case:acc
                    gender:neuter
                    number:sing
                    sem:[thing]
                  adjp
```

```
prep_phrase_adjunct_2
* sine * * cura *
index:i(p(7,8))
case:abl
prep:sine
sem:[abstract]
lex:cura
cat:pp
c_str
  prep:sine
  head
    * cura *
    index:i(p(7,8))
    cat:np
    sem:[abstract]
    number:sing
    person:3
    gender:fem
    lex:cura
    case:abl
    c_str
    cura
```

```
cat:adjp
* ciuile *
case:acc
number:sing
gender:neuter
lex:ciuilis
c_str
ciuilis
```

0.5--->

```
illocutionary_force:statement
vg
  selected_reading:habeo_have
  polarity:pos
  cat:vg
  * habebantur *
  pos:v
  lex:habere
  voice:pass
  tense:imperfect
  mood:indicative
  number:pl
  person:3
subject
  number:pl
  gender:neuter
  * externa *
  index:i(p(5,6))
  cat:np
  sem:[abstract,thing]
  person:3
  lex:externus
  case:nom
  c_str
  externa
prep_phrase_adjunct_2
  * sine * * cura *
  index:i(p(7,8))
  case:abl
  prep:sine
  sem:[abstract]
  lex:cura
  cat:pp
  c_str
  prep:sine
  head
    * cura *
    index:i(p(7,8))
    cat:np
    sem:[abstract]
    number:sing
    person:3
    gender:fem
    lex:cura
    case:abl
    c_str
    cura
ablative absolute
  * conuersis * * ad * * ciuile * * bellum * * animis *
  c_str
  lex:conuerto_turn
  pos:p_p
  case:abl
  gender:fem
  number:pl
  lex:conuertere
  mood:participle
  person:3
  prep:cplt
  * ad * * ciuile * * bellum *
  index:i(p(3,4))
  case:acc
  prep:ad
  sem:[thing]
  lex:bellum
  cat:pp
  c_str
  prep:ad
  head
    * ciuile * * bellum *
    index:i(p(3,4))
    cat:np
    sem:[thing]
    number:sing
    person:3
    gender:neuter
    lex:bellum
    case:acc
    c_str
    head
      * bellum *
      pos:noun
      lex:bellum
      case:acc
      gender:neuter
      number:sing
      sem:[thing]
  adjp
  cat:adjp
  * ciuile *
```

```

case:acc
number:sing
gender:neuter
lex:ciuilis
c_str
ciuilis

object
* animis *
index:i (p(4,5))
cat:np
sem:[hum,thing,abstract]
number:pl
person:3
gender:fem
lex:anima
case:abl
c_str
animis

```

Considérons un second exemple, où l'interprétation de la chaîne en tant qu'ablatif absolu distend les liens de manière plus nette encore, bien que l'inférence s'avère assez puissante pour resserrer les liens que l'analyse grammaticale a relâchés. L'exemple est ici aussi tiré des Histoires de Tacite,

Tacite Hist. III 41

*missis ad vitellium litteris auxilium postulat*

Ici aussi Bornecque scinde l'énoncé en deux, et lit la première partie comme ablatif absolu :

*Missis ad Vitellium litteris, auxilium postulat.*

Et ici aussi il traduit (ou plutôt reprend textuellement la traduction de Burnouf) par un énoncé en deux parties, jointes par la conjonction de coordination *et* :

*Il écrit à Vitellius et lui demande du secours.*

L'inférence fait en sorte que le lecteur ne s'imagine pas que l'écriture de la lettre et la demande de secours sont deux actions différentes, la coordination étant à interpréter comme marquant la succession chronologique (une inférence tout à fait puissante en d'autres contextes).

Cela donne bien sûr une traduction assez relâchée, semblable à celle de Goelzer, qui néanmoins propose un texte sans virgule :

*Missis ad Vitellium litteris auxilium postulat.*

*Il envoie un message à Vitellius et lui réclame du renfort.*

Si on se libère de la lecture de *missis litteris* comme ablatif absolu, on aura moins de scrupule à proposer une traduction plus resserrée :

*Il écrit à Vitellius pour lui réclamer du renfort.*

*Par lettre il réclame à Vitellius des renforts.*

C'est ce type de traduction qu'on rencontre en anglais :

*He wrote to Vitellius asking for aid.*

(Complete Works of Tacitus. Tr. Alfred John Church and William Jackson Brodribb 1873)

Ici, l'analyse alternative fait de *missis litteris* un ablatif de moyen. Le *missis* peut apparaître alors comme assez superflu, peu conforme à la recherche de la brièveté qui caractérise notre auteur. Encore un cas de syncrétisme ? ALP donne les deux analyses, auxquelles il confère le même poids, puisqu'elles font toutes deux de *missis litteris* un satellite et non un argument (circonstanciel de

## moyen ou ablatif absolu).

[0/missis,1/ad,2/uitellium,3/litteris,4/auxilium,5/postulat,endpos(6)]

2--->

```
illocutionary_force:statement
vg
  selected_reading:postulo_request
  polarity:pos
  cat:vg
  * postulat *
  pos:v
  lex:postulare
  voice:act
  tense:present
  mood:indicative
  number:sing
  person:3
subject
  source:context_retrievable
  number:sing
  person:3
  cat:np
  index:i(0,0)
  constraints_to_be_met:[sem:[hum]]
  case:nom
object
  * auxilium *
  index:i(p(4,5))
  cat:np
  sem:[thing,abstract]
  number:sing
  person:3
  gender:neuter
  lex:auxilium
  case:acc
  c_str
  auxilium
adjunct
  * missis * * ad * * uitellium * * litteris *
  value:manner_means
  c_str
  head
    * litteris *
    index:i(p(3,4))
    cat:np
    sem:[thing,abstract]
    number:pl
    person:3
    gender:fem
    lex:litterae
    case:abl
    c_str
    litteris
  participle_clause
    * missis * * ad * * uitellium *
    cat:ppclause
    number:pl
    gender:fem
    lex:mitto_send
    case:abl
    c_str
    past_participle
    mitto_send
    prep_cplt
      * ad * * uitellium *
      index:i(p(2,3))
      case:acc
      prep:ad
      sem:[male]
      lex:uitellius
      cat:pp
      c_str
      prep:ad
      head
        * uitellium *
        index:i(p(2,3))
        cat:np
        sem:[male]
        lex:uitellius
        number:sing
        person:3
        gender:masc
        case:acc
        c_str
        uitellius
```

2--->

```
illocutionary_force:statement
vg
  selected_reading:postulo_request
  polarity:pos
  cat:vg
  * postulat *
  pos:v
  lex:postulare
  voice:act
  tense:present
  mood:indicative
  number:sing
  person:3
subject
  source:context_retrievable
  number:sing
  person:3
  cat:np
  index:i(0,0)
  constraints_to_be_met:[sem:[hum]]
  case:nom
object
  * auxilium *
  index:i(p(4,5))
  cat:np
  sem:[thing,abstract]
  number:sing
  person:3
  gender:neuter
  lex:auxilium
  case:acc
  c_str
  auxilium
ablative_absolute
* missis * * ad * * uitellium * * litteris *
c_str
  lex:mitto_send
  pos:p_p
  case:abl
  gender:fem
  number:pl
  lex:mittere
  mood:participle
  person:3
  prep_cplt
  * ad * * uitellium *
  index:i(p(2,3))
  case:acc
  prep:ad
  sem:[male]
  lex:uitellius
  cat:pp
  c_str
  prep:ad
  head
  * uitellium *
  index:i(p(2,3))
  cat:np
  sem:[male]
  lex:uitellius
  number:sing
  person:3
  gender:masc
  case:acc
  c_str
  uitellius
object
  * litteris *
  index:i(p(3,4))
  cat:np
  sem:[thing,abstract]
  number:pl
  person:3
  gender:fem
  lex:litterae
  case:abl
  c_str
  litteris
```



Il est à noter que l'on trouve dans les *Histoires* des exemples clairs de *litteris* comme ablatif de moyen :

*Flaccus omisit inceptum aliisque litteris Gallum monuit ne terreret abeuntis* 4,19  
*profectus eadem nocte Flavianus obuiis Vespasiani litteris discrimini exemptus est* 3,10

On trouve de même *missis ... litteris* comme ablatif absolu :  
*missis sane ad eum Primi Antonii litteris* 4,13 (le *ad eum* est déterminant dans l'analyse comme ablatif absolu).

La voie est ainsi ouverte à une analyse de nos énoncés qui penche d'un côté ou de l'autre, ou se maintient au beau milieu. Nous ne pensons pas que cette dernière analyse, ouverte au syncrétisme syntaxique, soit à rejeter d'emblée. Un parseur tel que ALP a le mérite de nous faire voir où une grammaire donnée nous conduit.

Il convient de dire clairement que ce syncrétisme est un produit de l'analyse, et non un donné linguistique. Il n'existe que des textes et des interprétations, les deux intimement liés, puisque l'interprétation se fait sur la base des textes, et les textes s'établissent conformément à des interprétations, qui à leur tour se matérialisent par le biais de traductions, lesquelles sont à leur tour soumises à une procédure d'interprétation : on ne peut s'échapper de la prison de la langue.

Lavency 2002, aux prises avec le phénomène que nous examinons ici, parle de 'doubles analyses' et 'd'incertitudes d'analyse'. C'est le point de vue analytique. L'option syncrétique est celle de l'interprétation.

Notons en passant qu'un des exemples donnés par Lavency pour illustrer cette incertitude est assez curieux :

*Id Helvetii ratibus ac lintribus iunctis transibant.* (Caesar, B.G. I, 12).

Lavency écrit :

L'expression *flumen ratibus ac lintribus iunctis transire* (cfr Caes., G.II, 12, 1) est lue comme Ablatif de modalité (*hoc modo*) dans la mesure où *iunctis*, jugé omissible, est interprété comme participe épithète. Lu comme élément nécessaire, on considère l'ensemble comme Ablatif absolu.

Il semblerait toutefois que *iunctis* ne peut être associé qu'à *lintribus* (il s'agit d'un pont de bateaux), et non pas à l'ensemble *ratibus ac lintribus*<sup>4</sup>. L'omissibilité de *iunctis* est dès lors de nature purement grammaticale et non sémantique. Certes, pour qu'il y ait ablatif absolu, le *iunctis* est nécessaire. Mais cet ablatif absolu serait coordonné à un ablatif complément de moyen, ce qui est extrêmement peu probable sous la plume de César. Je crois que ce qui a joué ici c'est l'association quasi automatique chez l'analyste du couple **GN+pp à l'ablatif** avec la possibilité de l'attribution de la construction syntaxique **ablatif absolu**.

Or, dès lors qu'un participe passé entre en jeu, il est susceptible d'avoir sous sa dépendance tout le système actanciel de son prédicat, quel que soit le rôle joué localement par le participe. Dans le second exemple d'incertitude analytique impliquant un ablatif absolu proposé par Lavency, à savoir

---

4 Les traductions sont conformes : Les Helvétiens la passaient sur des radeaux et sur des barques jointes ensemble. (Artaud)  
Les Helvètes étaient en train de la franchir à l'aide de radeaux et de barques assemblées. (Constans)  
Les Helvètes la traversaient sur des radeaux et sur un **pont de bateaux**. (Maissiat)

*Cuius aduentu spe inlata militibus ac redintegrato animo cum pro se quisque in conspectu imperatoris etiam extremis suis rebus operam nauare cuperet paulum hostium impetus tardatus est.*  
(Caesar, B.G. II, 25, 3),

à propos duquel Lavency écrit :

Lors de l'épisode critique de la bataille contre les Nerviens, César s'avance en première ligne : *cuius aduentu spe inlata militibus... paulum hostium impetus tardatus est* (Caes., G. II, 25, 3). L'arrivée de César peut être tenue comme le moment du rétablissement du moral (paradigme *hoc tempore*) ; un admirateur de César peut choisir une autre lecture (paradigme *ea re + passif*) et y voir la source du sursaut romain. Avec ses deux constituants jugés solidaires, *spe inlata* est sans doute spontanément interprété comme Ablatif absolu et comme désignant le cadre dans lequel se relâche la pression ennemie.

je crois que le même automatisme d'analyse (GN ablatif + pp -> Abl. Absolu) s'est déclenché. Une analyse où les liens grammaticaux sont plus étroits (un complément d'agent de *tardatus est*, à deux volets, *spe inlata* et *redintegrato animo*, les participes *inlata* et *redintegrato* ayant leur propre agent, *cuius aduentu*) me semble plus plausible. Mais dans l'optique syncrétique que je tente de développer ici, la lecture de *spe inlata* et *redintegrato animo* (*inlata* et *redintegrato* se partageant l'agent *cuius aduentu*) comme ablatif absolu coordonné et explicité par la subordonnée en *cum* (qui en développe la conséquence), est *concurrente* au sens étymologique du mot.

## References

### Dictionary

LASLA Frequency Dictionary = DICTIONNAIRE FRÉQUENTIEL ET INDEX INVERSE DE LA LANGUE LATINE, L. Delatte ,Et. Evrard, S. Govaerts, J. Denooz, L.A.S.L.A.,1981

Forcellini = Forcellini et al. LEXICON TOTIUS LATINITATIS

Lewis and Short = Lewis, C.T. and Short,C. A LATIN DICTIONARY, Oxford, 1879

### Grammars

Michel = Jacques Michel, GRAMMAIRE DE BASE DU LATIN, 2nd edition, De Sikkel, 1962

VSVS = Marius Lavency, USUS: GRAMMAIRE LATINE: DESCRIPTION DU LATIN CLASSIQUE EN VUE DE LA LECTURE DES AUTEURS, Duculot, 1985

Ernout, A. and Thomas, F., SYNTAXE LATINE, 2nd edition, Klincksieck, 1964

### Other References

Covington 2003 : Michael A. Covington, *A Free-Word-Order Dependency Parser in Prolog*, Artificial Intelligence Center,The University of Georgia, 2003

Gal et al. 1991 : Gal, A., Lapalme, G., Saint-Dizier, P. & Somers, H., PROLOG FOR NATURAL LANGUAGE PROCESSING, Wiley, Chichester, 1991

Koster 2005 : C.H.A. Koster, *Constructing a Parser for Latin*, in A. Gelbukh (ed.): CICLing 2005, LNCS 3406, pp. 48–59, Springer-Verlag, Berlin Heidelberg, 2005

Lavency 2002 : Marius Lavency, *Pour une taxinomie des syntagmes à l'Ablatif en latin classique*, Folia Electronica Classica, 4, Louvain-la-Neuve, 2002

Michiels 2016 : Archibald Michiels, *VERBA, a Multi-word-unit-oriented Feature-unification-based Parser*, unpublished paper, University of Liège, 2016

Wielemaker 2003 : Jan Wielemaker, *An overview of the SWI-Prolog Programming Environment*, in Mesnard, F. and Serebenik, A., eds, Proceedings of the 13th International Workshop on Logic Programming Environments, Katholieke Universiteit Leuven, Heverlee, Belgium, 2003