A French Grammar Fragment

Hausarbeit for Computational Linguistics I
Lecturer: Tom Cornell
SfS · University of Tübingen

Jan-Philipp Söhn

WS 1999/2000

Contents

1

The Task - An Introduction

This is my project for "Computational Linguistics I". I chose to design a grammar fragment for French which is able to deal with sentences in the *passé composé*. This tense form in French must be used carefully because there are some sophisticated agreement rules. I developed a grammar in DCG-format which can handle such sentences and a small program to test it.

My program written in PROLOG reads a sentence typed in by the user, converts it to a list of PROLOG-Atoms, then starts a parser which uses my grammar and a lexicon that is also provided. If the sentence is grammatical, it writes the tree-structure for it on the screen and confirms with "yes". If the sentence is incorrect, it simply says "no".

In the following I explain the grammar phenomenon, describe how I completed this project and how I dealt with problems that occurred. The reader will also learn how to use my program. You can find a list of examples, the lexicon and the grammar in the appendix.

In this documentation, French expressions are printed in *italics* and, if considered necessary, followed by an English translation in parentheses.

2

Passé Composé - The Facts Explained

The *passé composé* (compound past) is a past tense used for describing single events or chains of events in a story. In contrast to this, backround information such as time or weather, would have to be told in the *imparfait*.

The *passé composé* consists of an inflected auxiliary, either *être* (be) or *avoir* (have), and the past participle of the main verb.

If the past participle selects *être* as its auxiliary, it has to agree with the subject of the sentence. For example:[1]

> *Les filles sont déjà allées en Allemagne.*
> the girls(fem/pl) are already gone(fem/pl) to Germany
> (The girls have already been to Germany.)

If it selects *avoir*, it agrees with the direct object, but only if the object precedes it. This could be the case either in sentences with an extraction, like

> *Je les ai trouvés.*
> I them(mask/pl) have found(mask/pl)
> (I found them.)

or in relative clauses, like

> *Montre-moi le dessert que tu as fait.*
> show me the dessert(mask/sg) which you have made(mask/sg)
> (Show me the dessert that you made!)

[1] Most sentences are from [CI1]

If there is no extraction or relative construction there is no agreement with the object.

Furthermore, there are reflexive verbs which always go with *être*, but they also agree with the direct object preceding it. This can be a reflexive pronoun (*me, te, se, nous, vous, se*), if there is no other direct object, or it can be an extracted direct object of the verb in form of a clitic or a relative pronoun (cf. Appendix A).

It proved to be very easy to implement simple examples of the passive voice. The construction is similar to the *passé composé*, but there is always an inflected form of *être*, no matter what kind of auxiliary the participle selects. So there is subject-agreement, like in the *être*-case of the *passé composé*:

> *Les bouteilles sont ouvertes.*
> the bottles(fem/pl) are opened(fem/pl)
> (The bottles are opened.)

There are two other tenses, that look exactly like *passé composé*: *futur antérieur* and *conditional passé*. Only the tense of the auxiliary is different.
Futur antérieur expresses the future past like "Jim will have gone to the playground by that time tomorrow". The tense of the auxiliary is *futur simple*.
Conditional passé expresses the conditional in the past like "Jim would have gone to the playground, if he had not been ill." The tense of the auxiliary is *conditional présent*.
I decided to add these two tenses to the lexicon, in order to increase the variety of sentences.

3

Implementing This

3.1 The Lexicon

At first, I created the lexicon, which you can find in Appendix B. The arguments of the lexical entries represent their features, like agreement information (number, case, gender and person), tense, category, labels for the tree and some special ones, which are explained below.

Firstly, proper names and common nouns have to be distinguished. The simple reason is that proper names must not occur with an article, whereas other nouns must have one. So I declared features “name” and “noname”.
Secondly, auxiliaries are either *être* (feature “e”) or *avoir* (feature “a”). As I explained in section two, participles select their auxiliaries.
Thirdly, the main verbs are either transitive (feature “trans”) or intransitive (feature “intr”). These features are important for the grammar in order not to license a sentence with an intransitive verb having a direct object.

3.2 The Grammar

After that, I developed a context free grammar in definite-clause-notation (DCG). I must admit that my first version was quite a hack. It worked, but both node names and structure were not at all conform to the current work of syntacticians.
For example, the relative clause had a flat structure! I did it this way, because there are so many interrelations between the elements: the auxiliary has to agree with the subject of the relative clause, but the participle has to agree with the head of the clause (the extracted object of the verb).

This solution was certainly not acceptable. So I consulted Radford [Rad] and Ruwet [Ruw] to get some inspiration for the grammar. You can find a version which is not so confusing as the actual code in Appendix C.[1]

I opted for the labels of the node to be classical: S, $\bar{\text{S}}$, $\overline{\text{NP}}$, NP, $\overline{\text{VP}}$ and VP. There is no strong evidence for french to have DP instead of NP, for example. Only for relative clauses I introduced the label Srel below $\bar{\text{S}}$, in order to avoid indirect recursion with the same sentence[2].
The labels used in GB or Minimalism like CP, TP, Agr, AgrP, ... implicitly presuppose that there is almost always a kind of movement involved. I follow HPSG [PS94] tradition in the respect that there is no movement in declarative sentences. Besides, Agr and AgrP nodes are not needed in my grammar, for the features of the constituents are checked directly in the grammar rules.

The lexicon also looks a bit like a lexicon for a HPSG grammar: An entry specifies a word with all its grammatical features like number, tense and so on. However, there are differences, too. An important one is, that the lexicon does not specify the subcategorisation requirements of a word. Although verbs have features "trans"/"intr", the main work is done by the grammar rules. Different rules are applied for NPs in situ or extracted ones (not different lexical entries or lexical rules) and there is no lexical entry for a trace.
Nevertheless, my theory is a bit more lexical than Generative Grammar. For example, Ruwet splits the verb in a verbal root and a temporal affix and assumes them to be distinct terminal nodes. To illustrate this, see the following table:

verbal root	temp.affix	verbal root	temp.affix
mange-	*-rait*	*au-*	*-rait*
infinitive: *manger*		infinitive: *avoir*	

In my grammar, the inflected verbform as a whole is an entry in the lexicon. For regular verbs like *manger* splitting up the verb is simple, but there are problems with irregular verbs like *avoir*. There the root changes, too. In consequence, the affix has to select its root. That has to be specified somewhere... So putting everything in the lexicon seems to be a simple and appropriate solution.

Back to our flat relative clause. I solved the problem by introducing a new

[1] This version omitts the trees printed on the screen and cannot be used with the lexicon of Appendix B.

[2] e.g.: **Montre-moi le dessert que montre-moi le dessert que tu as fait.*

channel in the $\overline{\text{VP}}$, passing down information through several nodes. There are now five agreement arguments: person of subject, number of subject and object and gender of subject and object.
Now the grammar is like you can see in Appendix C. Still, a few amendments were necessary:
Firstly, I had to insert the tree structure which is to be printed on the screen. So the user may see how the grammar works.
Secondly, while writing this documentation, I noticed that I had forgotten to implement extracted and relative constructions for reflexive verbs (Appendix A.5).

The first one was quite straightforward, but for the second one, I had to insert the reflexive pronoun, which is not at all involved in agreement into the Srel-rule. So for a relative clause containing a reflexive verb, again there is a sort of flat structure. There might be nicer ways to solve this.

At the end of this section, let us have a closer look at the label names. Terminal symbols are: determiners (det), nouns (noun), clitics (clit), auxiliaries (aux), participles (part), relative pronouns (relpron) and pronouns (pron), which can be either personal or reflexive.
In my grammar the lowest node in a tree above the terminals is VP or VPr. A VP consists of a participle, an object NP in situ and possibly an adjunct phrase. A VPr is basically the same as a VP, but contains a participle of a reflexive verb. This distinction in the program is necessary, because a reflexive verb always has a preceeding reflexive pronoun whereas a nonreflexive verb is not allowed to have one.
Furthermore, an NP is made up of a noun and a possible determiner or it is a personal pronoun. Other nodes at that level are called PP, consisting of a pronoun (p) and a noun (pnoun)[3], and Imperative, being the mother of a verb in imperative form (imp_v) and a special pronoun (imp_pron).
At the next level there are $\overline{\text{VP}}$s and $\overline{\text{VPr}}$s. Here the auxiliary is adjoined. When there is extraction, another $\overline{\text{VP}}$ at one higher level contains the clitics and pronouns and, optionally, an adverb. The VPpassive is needed for the different auxiliary selection.
An Srel is the mother of an NP, a $\overline{\text{VP}}$ and - depending on the verb - a reflexive pronoun. It is the sister node of the relative pronoun and the daughter of an $\bar{\text{S}}$. An $\bar{\text{S}}$ is the sister of an NP and the daughter of a $\overline{\text{NP}}$. Finally, S is the uppermost node.
For the confused reader I recommend consulting Appendix C.

[3]cf. section 6.1

4

How To Use The Program - The User Interface

In order to use the program to test my grammar fragment, the reader has to start a Prolog environment, in my system SICStus-Prolog. The main file is called “fparse.pl”, which can be consulted by typing [fparse]. This in turn consults all other program files which are required: the parser, the grammar and the lexicon.
The main predicate is “french.” When called, there appears a short instruction and then a new prompt “*Proposition*>”(phrase). Now the program waits for an input in natural spelling. This of course has to be translated into a format understandable for the grammar and the lexicon. This is done by the user interface (see below). If the input is grammatical, the user is given back the structure of the sentence indicated by the bracketing. The confirmation “yes” concludes the output. If the input is ungrammatical, the simple answer is “no”. Not very helpful...

Making it possible for the user to type in natural spelling requires a user interface. This also is contained in “fparse.pl”. I do not provide the full code for it in this documentation. For the interested reader, it is part of the directory containing all the .pl-files of the testing program. Basically I use a program which was part of an exercise of a lab session in our course. The version at hand is the solution proposed by Tom Cornell, but it had to be adapted. The interface works in the following way: At first, it reads in the ASCII-code of the input and creates a list. A period, the end of the sentence, marks the end of that list. This list is stored in the temporary memory and so it is independent from the read-in and ready for further processing. This procedure is necessary, because there occurred some problems with the backtracking during the expansion (see below). The list now contains the codes of the letters typed in by the user. From this row of numbers, words have to be separated from spaces and other symbols like hyphens, quotes or periods. This is done by the next predicate. It converts that list of characters into a list of PROLOG-atoms and changes all upper case letters to lower case. Still there are words which are no lexical entries like *montre-moi* or *s'est*. The combined words have to be split up and the contracted forms have to be expanded into lexical items. Fig. 4.1 and Fig. 4.2 show the most important relations for this procedure.

At the end of this process, the list of atoms contains only words which are in the lexicon. For example: the input sentence *"Marie s'est lavée"* looks now like this: [marie, se, est, lavée].

After converting the input, the parser is started.

```
%%% the sentence will be a list of atoms
%%% without spaces, hyphens, single quotes and periods
get_sent( CharList0, AtomList ):-
        get_word( CharList0, CharList, Word0, Terminator ),
        downcase( Word0, Word ),      %%% no upper case letters
        (   Terminator = period ->    %%% end of sentence
            name( Atom, Word ),       %%% -> end of list
            AtomList = [Atom]
        ;
            Terminator = space ->     %%% separating the words
            name( Atom, Word ),
            AtomList = [Atom|Rest],
            get_sent( CharList, Rest )
        ;
            Terminator = quote ->     %%% contracted form
            convert(Atom, Word),      %%% -> call to convert
            AtomList = [Atom|Rest],
            get_sent( CharList, Rest )
        ).
```

Figure 4.1: Getting a sentence from the `CharList`

```
%%% single quotes and hyphens are treated as spaces
get_word( [C|L1], L, Word, Terminator ):-
        (   C = 46 ->   %%% '.'
            Word = [], Terminator = period,
            L1 = L
        ;
            C = 32 ->   %%% ' '
            Word = [], Terminator = space,
            L1 = L
        ;
            C = 45 ->   %%% '-'                 %%% that's for "montre-moi"
            Word = [], Terminator = space,   %%% the imperative is split
            L1 = L                           %%% in two separate words
        ;
            C = 39 ->   %%% '''                 %%% that's for s'est or t'es
            Word = [], Terminator = quote,   %%% -->  two separate words
            L1 = L
        ;
            Word = [C|Word1],
            get_word( L1, L, Word1, Terminator )
        ).
%%% converts  j' -> je; t' -> te;
%%% s' -> se; l' -> le or la and qu' -> que;
convert(Pron, Contracted):-
        (   Contracted = [0'j] ->      Pron = je
        ;
            Contracted = [0'q, 0'u] -> Pron = que
        ;
            Contracted = [0't] ->      Pron = te
        ;
            Contracted = [0's] ->      Pron = se
        ;
            Contracted = [0'l] -> expansion(Pron)
        ).
%%% auxiliary predicate for expanding the l'
expansion(le).
expansion(la).
```

Figure 4.2: Predicates for conversion called by `get_sent`

5

The Parser

Since the focus of this project is not the parser, I did not write one myself, but used a version programmed by Tom Cornell in January 2000. The only adaptation I made was preventing printing of debugging information.
It is an Earley-Parser for definite clause grammars based on the algorithm by Jay Earley, developed in 1970 [Mel, p. 149]. This parsing strategy generalises the CYK algorithm to work with grammars having more than two symbols on the right hand side of their rules. That is the case with my grammar (see above). Where there is a well-formed substring table in the CYK parser, there is now a so-called active chart which maintains entries for substring analyses (edges) that can be either complete or partial. Here, the notion of a "dotted rule" must be introduced. That is a an entry in the chart comprising a rule of the grammar with a dot between the symbols matching the input and those that have yet to be matched. For example:

$NP \rightarrow Det \bullet N$ (the Det is found, but not yet the N)

Another data structure used in the parsing strategy is the "agenda", where edges are entered before they are added to the chart. Otherwise the edges entered directly into the chart would have a kind of left-to-right order which would entail a rigidness which is not desired.
How the actual algorithm works is specified in the comments of "earley-dcg.pl".
That is just a first glimpse of the parser. I don't want to go any deeper into details now. See [Mel, Chapter 12] or [Cor, "First Look at Chart Parsing"] for further explanations.
Earley's active chart parsing algorithm is top down and the most efficient general CFG parsing strategy. Chart parsing has become the standard framework for grammars describing natural languages.

6

Future Research

Two issues, among others, might have to be handled differently: adjuncts and what I call the transitive-*être*-problem. You might recall that the structure of the relative clause containing a reflexive verb is a problem, too. However, I won't describe that more detailed in this section.

6.1 Adjuncts

I paid little attention to adjuncts or indirect objects in my grammar. Just in order to get a little more natural or interesting sentences, I added the adjunct "*en Allemagne*" (in/to Germany). In French, there are different pronouns, each selected by a different kind of noun. For example: "*<u>en</u> Allemagne*", but "*<u>à</u> Portugal*". Implementing this would exceed the amount of allocated time to this Hausarbeit. Major changes would have to be made in the grammar to account for all possibilities of placing an adjunct in a sentence.

6.2 The Transitive-*Être*-Problem

As explained, verbs select their auxiliary in the *passé composé*-construction. But is there a relation between the kind of auxiliary selected and the property of being transitive or intransitive? Well, that is not so clear.
There are transitive and intransitive verbs selecting *avoir*, but I found no example of an *être*-verb being transitive. More mysteriously, many intransitive verbs selecting *être* can also be used transitively, then they go with *avoir*! For example: *Il est descendu du train* (intransitive, He got off the train.), but *Il a descendu le Mount Everest* (transitive, He descended the Mount Everest).
As I have not found any counterexamples, I omitted two grammar-rules for

transitive verbs going with *être* where there would be subject agreement. However, there is no rule stating that all transitive verbs must select *avoir*. So maybe counterexamples exist, then my grammar would have to be amended.

While working on that project, I realised, that no matter how restricted a grammar fragment is, one easily gets into trouble and there is always a lot of work left.

Appendix A

Some Examples

1. Subject-agreement of verbs with *être*
   (Words in parentheses are optional!)
   Jean est (déjà) allé (en Allemagne).
   Marie est (déjà) allée (en Allemagne).
   Les gens sont (déjà) allés (en Allemagne).
   Les filles sont (déjà) allées (en Allemagne).

2. Direct-object-agreement of verbs with *avoir*
   Je l'ai fait. (= le dessert)
   Tu l'as mise. (= la table)
   Il les a trouvés. (= les verres(m))
   Ils les ont ouvertes. (= les bouteilles(f))

3. Direct-object-agreement of verbs with *avoir* in relative clauses
   Montre-moi le dessert que tu as fait.
   Montre-moi la table que tu as mise.
   Montre-moi les verres que tu as trouvés.
   Montre-moi les bouteilles que tu as ouvertes.

4. Direct-object-agreement of reflexive verbs (with *être*)
   Marie s'est lavée. (se = direct object)
   Marie s'est lavé les mains. (se = indirect object)
   Les jeunes se sont lavés. (se = direct object)
   Ils se sont acheté des vêtements (en Allemagne). (se = indirect obj.)

5. Direct-object-agreement of reflexive verbs (with *être*)
   - extraction and relative construction
   Marie se les est lavées. (se = indirect object, les = direct object)
   Ils se les sont achetés (en Allemagne).
   (se = indirect object, les = direct object)
   Montre-moi les mains que tu t'es lavées.
   Montre-moi les vêtements que Marie s'est achetés.

6. No agreement
   Marie a pensé.
   Les filles ont mis la table.
   Les gens ont ouvert les bouteilles.

7. Passive Voice
   Le dessert est fait.
   La table est mise.
   Les verres sont trouvés.
   Les bouteilles sont ouvertes.

8. Other tenses
   Jean sera allé en Allemagne. (Future II)
   J'aurai mis la table.
   Les gens seraient allés en Allemagne. (Conditional II)
   Elle aurait trouvé les verres.

Appendix B

The Lexicon

```
%%% determiners for subjects or objects
word( le,       det( sg, _Case, mask, def, det )).
word( la,       det( sg, _Case, fem, def, det )).
word( les,      det( pl, _Case, _Gend, def, det )).

word( un,       det( sg, _Case, mask, indef, det )).
word( une,      det( pl, _Case, fem, indef, det )).
word( des,      det( pl, _Case, _Gend, indef, det )).


%%% subjects and objects
%%% distinction between proper names and nouns is necessary
word( jean,     noun( sg, _Case, mask, name, noun )).
word( marie,    noun( sg, _Case, fem, name, noun )).

word( gens,     noun( pl, _Case, mask, noname, noun )).
word( jeunes,   noun( pl, _Case, mask, noname, noun )).
word( fille,    noun( sg, _Case, fem, noname, noun )).
word( filles,   noun( pl, _Case, fem, noname, noun )).
word( dessert,  noun( sg, _Case, mask, noname, noun )).
word( desserts, noun( pl, _Case, mask, noname, noun )).
word( table,    noun( sg, _Case, fem, noname, noun )).
word( tables,   noun( pl, _Case, fem, noname, noun )).
word( verre,    noun( sg, _Case, mask, noname, noun )).
word( verres,   noun( pl, _Case, mask, noname, noun )).
word( bouteille,noun( sg, _Case, fem, noname, noun )).
word( bouteilles, noun( pl, _Case, fem, noname, noun )).
```

```
word( main,      noun( sg, _Case, fem, noname, noun )).
word( mains,     noun( pl, _Case, fem, noname, noun )).
word( vêtement, noun( sg, _Case, mask, noname, noun )).
word( vêtements,noun( pl, _Case, mask, noname, noun )).


%%% personal pronouns
word( je,        perspron( sg, nom, _Gend, first, pron )).
word( tu,        perspron( sg, nom, _Gend, second, pron )).
word( il,        perspron( sg, nom, mask, third, pron )).
word( elle,      perspron( sg, nom, fem,  third, pron )).
word( ils,       perspron( pl, nom, mask, third, pron )).
word( elles,     perspron( pl, nom, fem,  third, pron )).


%%% pro-clitics
word( le,        clitic( sg, akk, mask, clit )).
word( la,        clitic( sg, akk, fem, clit )).
word( les,       clitic( pl, akk, _Gend, clit )).


%%% reflexive pronoun
word( me,        reflpron( first, sg, pron )).
word( te,        reflpron( second, sg, pron )).
word( se,        reflpron( third, _N, pron )).


%%% relative pronoun
word( que,       relpron(akk, relpron )).


%%% imperative "show me"
%%% needs no agreement features in this grammar fragment
%%% here the speaker is first person singular,
%%% the hearer is second person singular
word( montre,    imp_v(verb) ).
word( moi,       imp_pron(pron) ).


%%% auxiliaries
%%% special feature for avoir (a) or être (e)
```

```
word( suis,     aux( e, first, sg, pres, aux )).
word( ai,       aux( a, first, sg, pres, aux )).
word( es,       aux( e, second, sg, pres, aux )).
word( as,       aux( a, second, sg, pres, aux )).
word( est,      aux( e, third, sg, pres, aux )).
word( a,        aux( a, third, sg, pres, aux )).
word( sont,     aux( e, third, pl, pres, aux )).
word( ont,      aux( a, third, pl, pres, aux )).

word( serai,    aux( e, first, sg, fut, aux )).
word( aurai,    aux( a, first, sg, fut, aux )).
word( seras,    aux( e, second, sg, fut, aux )).
word( auras,    aux( a, second, sg, fut, aux )).
word( sera,     aux( e, third, sg, fut, aux )).
word( aura,     aux( a, third, sg, fut, aux )).
word( seront,   aux( e, third, pl, fut, aux )).
word( auront,   aux( a, third, pl, fut, aux )).

word( serais,   aux( e, first, sg, cond, aux )).
word( aurais,   aux( a, first, sg, cond, aux )).
word( serais,   aux( e, second, sg, cond, aux )).
word( aurais,   aux( a, second, sg, cond, aux )).
word( serait,   aux( e, third, sg, cond, aux )).
word( aurait,   aux( a, third, sg, cond, aux )).
word( seraient, aux( e, third, pl, cond, aux )).
word( auraient, aux( a, third, pl, cond, aux )).


%%% past participles select either avoir (a) or être (e)
%%% verbs are either transitive or intransitive
word( allé,     pastpart( e, sg, mask, intr, part )).
word( allée,    pastpart( e, sg, fem, intr, part )).
word( allés,    pastpart( e, pl, mask, intr, part )).
word( allées,   pastpart( e, pl, fem, intr, part )).

word( fait,     pastpart( a, sg, mask, trans, part )).
word( faite,    pastpart( a, sg, fem, trans, part )).
word( faits,    pastpart( a, pl, mask, trans, part )).
word( faites,   pastpart( a, pl, fem, trans, part )).

word( mis,      pastpart( a, _Num, mask, trans, part )).
```

```
word( mise,     pastpart( a, sg, fem, trans, part )).
word( mises,    pastpart( a, pl, fem, trans, part )).

word( trouvé,   pastpart( a, sg, mask, trans, part )).
word( trouvée,  pastpart( a, sg, fem, trans, part )).
word( trouvés,  pastpart( a, pl, mask, trans, part )).
word( trouvées, pastpart( a, pl, fem, trans, part )).

word( ouvert,   pastpart( a, sg, mask, trans, part )).
word( ouverte,  pastpart( a, sg, fem, trans, part )).
word( ouverts,  pastpart( a, pl, mask, trans, part )).
word( ouvertes, pastpart( a, pl, fem, trans, part )).

word( pensé,    pastpart( a, sg, mask, intr, part )).
word( pensée,   pastpart( a, sg, fem, intr, part )).
word( pensés,   pastpart( a, pl, mask, intr, part )).
word( pensées,  pastpart( a, pl, fem, intr, part )).

%%% the reflexives "se laver" and "s'acheter"
%%% they are always transitive, but the relative pronoun
%%% can be either direct or indirect
word( lavé,     pastpartrefl( e, sg, mask, part )).
word( lavée,    pastpartrefl( e, sg, fem, part )).
word( lavés,    pastpartrefl( e, pl, mask, part )).
word( lavées,   pastpartrefl( e, pl, fem, part )).

word( acheté,   pastpartrefl( e, sg, mask, part )).
word( achetée,  pastpartrefl( e, sg, fem, part )).
word( achetés,  pastpartrefl( e, pl, mask, part )).
word( achetées, pastpartrefl( e, pl, fem, part )).


%%% optional fillers
word( déjà,     adv(adv) ).
word( en,       p(pron) ).
word( allemagne,pnoun(noun) ).
```

Appendix C

The Grammar

```
%%% normal declarative with or without reflexive verb
s --->  [ np( NSubj, nom, GSubj, P ),
          vp_bar( P, NSubj, GSubj )].

%%% passive voice
s --->  [ np( NSubj, nom, GSubj, P ),
          vp_passive( P, NSubj, GSubj )].

%%% imperative sentence with a relative clause
s --->  [ imperative,
          np_bar ].

%%% relative clause
s_bar( NObj, GObj) --->
        [ relpron( akk ),
          srel( NObj, GObj )].          %%% could be s, but srel avoids
                                        %%% indirect recursion
%%% NPbar is NP plus relative clause
np_bar --->
        [ np( NObj, akk, GObj, third ),
          s_bar( NObj, GObj )].

%%% NPs
np( N, C, G, third ) --->
        [ det( N, C, G, _Def ),         %%% article noun
          noun( N, C, G, noname )].     %%% noun is not a proper name
np( N, C, G, third ) --->
        [ noun( N, C, G, name )].       %%% proper names cannot occur
```

```
                                        %%% with an article
np( N, C, G, P ) --->
        [ perspron( N, C, G, P )].      %%% personal pronouns


%%% a relative clause has a gap
%%% the predicate agrees with the np
%%% but the participle agrees with the head of the srel
srel( NObj, GObj ) --->
        [ np( NSubj, nom, _GSubj, P ),
          vp_bar( P, NSubj, NObj, GObj )].

%%% a relative clause with a reflexive verb
%%% the predicate agrees with the np
%%% but the participle agrees with the head of the srel
srel( NObj, GObj ) --->
        [ np( NSubj, nom, _GSubj, P ),
          reflpron( _, _ ),             %%% this reflpron has no influence
          vp_r_bar( P, NSubj, NObj, GObj )]. %%% to the agreement

vp_bar( P, N, G ) --->
        [ aux( AE, P, N, _T ),
          vp( AE, N, G, _ )].
vp_bar( P, N, G ) --->
        [ aux( AE, P, N, _T ),
          adv,                          %%% optional adverb like déjà
          vp( AE, N, G, _ )].
vp_bar( P, NSubj, _GSubj ) --->
        [ clitic( NObj, akk, GObj ),    %%% the participle must agree
          vp_bar( P, NSubj, NObj, GObj )]. %%% with clitic or subject

vp_bar( P, N, G ) --->
        [ reflpron( P, N ),             %%% the dir obj is the reflpron
          vp_r_bar( P, N, G )].         %%% -> agreement

vp_bar( P, NSubj, _GSubj ) --->
        [ reflpron( P, NSubj ),         %%% the reflpron is the ind obj
          clitic( NObj, akk, GObj ),    %%% there is agreement
          vp_r_bar( P, NSubj, NObj, GObj )]. %%% with the direct object

%%% overloaded vp_bar for object agreement
```

```
vp_bar( P, NSubj, NObj, GObj ) --->
        [ aux( AE, P, NSubj, _T ),
          vp( AE, NObj, GObj, extract )].

%%% distinction between vp_bar and vp_r_bar is necessary
%%% nonreflexives with a reflexive pronoun are not allowed
vp_r_bar( P, N, G ) --->
        [ aux( e, P, N, _T ),           %%% reflexives always go
          vp_r( e, N, G, insitu )].     %%% with être

%%% overloaded vp_r_bar for extraction of direct object
vp_r_bar( P, NSubj, NObj, GObj ) --->
        [ aux( e, P, NSubj, _T ),
          vp_r( e, NObj, GObj, extract )].

vp_passive( P, N, G ) --->
        [ aux( e, P, N, _T ),           %%% Passive: inflected être and
          vp( _, N, G, extract )].      %%% past participle

%%% verbs with être -> subject agreement
vp( e, N, G, intr ) --->
        [ pastpart( e, N, G, intr )].   %%% no objects or adjuncts

%%% verbs with avoir -> no agreement
vp( a, _N, _G, intr ) --->
        [ pastpart( a, sg, mask, intr )]. %%% no objects or adjuncts

vp( e, N, G, intr ) --->
        [ pastpart( e, N, G, intr ),
          pp ].                         %%% optional adjunct pp
vp( a, _N, _G, intr ) --->
        [ pastpart( a, sg, mask, intr ),
          pp ].                         %%% optional adjunct pp

vp( a, N, G, extract ) --->
        [ pastpart( a, N, G, trans )].  %%% extraction
vp( a, N, G, extract ) --->
        [ pastpart( a, N, G, trans ),   %%% extraction
          pp ].                         %%% optional adjunct pp

%%% transitives with avoir, no extraction -> no agreement
```

```
vp( a, _N, _G, trans ) --->
        [ pastpart( a, sg, mask, trans ),
          np( _, akk, _, _ )].          %%% object np


vp( a, _N, _G, trans ) --->
        [ pastpart( a, sg, mask, trans ),
          np( _, akk, _, _ ),           %%% object np
          pp ].                         %%% optional adjunct pp

%%% VPr insitu or extract
vp_r( e, N, G, _ ) --->
        [ pastpartrefl( e, N, G )].
vp_r( e, N, G, _ ) --->
        [ pastpartrefl( e, N, G ),
          pp ].

%%% when there is a direct object np,
%%% the reflexive pronoun is indirect -> no agreement
vp_r( e, _N, _G, insitu ) --->
        [ pastpartrefl( e, sg, mask ),
          np( _, akk, _, _ )].
vp_r( e, _N, _G, insitu ) --->
        [ pastpartrefl( e, sg, mask ),
          np( _, akk, _, _ ),
          pp ].

pp --->
        [ p, pnoun ].

imperative --->
        [ imp_v, imp_pron ].
```

References

[CI1] GÜNTER FREITAG & ALFRED GÖLLER: *Etudes françaises - Echanges, Cours Intensif 1, Grammatisches Beiheft.* Ernst Klett Schulbuchverlag, Stuttgart (1989)

[Ruw] NICOLAS RUWET: *Introduction à la Grammaire Générative.* Paris, Librairie Plon (1968)

[Rad] ANDREW RADFORD: *Syntactic Theory And The Structure of English, A Minimalist Approach.* Cambridge Textbooks of Linguistics, Cambridge University Press (1997)

[PS94] CARL POLLARD & IVAN A. SAG: *Head-Driven Phrase Structure Grammar.* The University of Chigago Press, CSLI (1994)

[Mel] CHRIS MELLISH: *Techniques in Natural Language Processing 1, Module Workbook, Autumn Term 1994.* Dept. of Artificial Intelligence, University of Edinburgh, ©Mellish, Whitelock, Ritchie 1994

[Cor] TOM CORNELL: *Lecture Notes for Computational Linguistics I.* Seminar für Sprachwissenschaft, Universtität Tübingen (1999/2000)