A language−independent shallow−parser Compiler

Alexandra Kinyon
CIS Dpt. .
University of Pennsylvania
kinyon@linc.cis.upenn.edu
http://www.cis.upenn.edu/~kinyon

Abstract

We present a rule−based shallow−parser compiler, which allows to generate a robust shallow−parser for any language, even in the absence of training data, by resorting to a very limited number of rules which aim at identifying constituent boundaries. We contrast our approach to other approaches used for shallow−parsing (i.e. finite−state and probabilistic methods). We present an evaluation of our tool for English (Penn Treebank) and for French (newspaper corpus "LeMonde") for several tasks (NP−chunking & "deeper" parsing) .

1 Introduction

Full syntactic parsers of unrestricted text are costly to develop, costly to run and often yield errors, because of lack of robustness of wide−coverage grammars and problems of attachment. This has led, as early as 1958 (Joshi & Hopely 97), to the development of shallow−parsers, which aim at identifying as quickly and accurately as possible, main constituents (and possibly syntactic functions) in an input, without dealing with the most difficult problems encountered with "full−parsing". Hence, shallow−parsers are very practical tools. There are two main techniques used to develop shallow−parsers:

1− Probabilistic techniques (e.g. Magerman 94, Ratnaparkhi 97, Daelmans & al. 99)

2− Finite−state techniques (e.g. Grefenstette 96)

Probabilistic techniques require large amounts of syntactically−annotated training data[1], which makes them very unsuitable for languages for which no such data is available (i.e. most languages except English) and also, they are not domain−independent nor "style−independent" (e.g. they do not allow to successfully shallow−parse speech, if no annotated data is available for that "style"). Finally, a shallow−parser developed using these techniques will have to mirror the information contained in the training data. For instance, if one trains such a tool on data were only non recursive NP chunks are marked[2], then one will not be able to obtain richer information such as chunks of other categories, embeddings, syntactic functions...

On the other hand, finite−state techniques rely on the development of a large set of rules (often based on regular expressions) to capture all the ways a constituent can expend. So for example for detecting English NPs, one could write the following rules :

NP → Det adj* noun adj*

NP → Det adj (for noun ellipsis)

NP → ProperNoun etc ....

But this is time consuming and difficult since one needs to foresee all possible rewriting cases, and if some rule is forgotten, or if too many POS errors are left, robustness and/or accuracy will suffer.

Then these regular expressions have to be manipulated i.e. transformed into automata, which will be determinized and minimized (both being costly operations). And even though determinization and minimization must be done only once (in theory) for a given set of rules, it is still costly to port such tools to a new set of rules (e.g. for a new language, a new domain) or to change some existing rules.

In this paper, we argue that in order to accomplish the same task, it is unnecessary to develop full sets of regular expression : instead

[1] We are leaving aside unsupervised learning techniques here, since to our knowledge they have not proved a successful for developing practical shallow−parsers.

[2] See (Abney 91) for the definition of a chunk.

of specifying all the ways a constituent can be rewritten, it is sufficient to express how it begins and/or ends. This allows to achieve similar results but with far fewer rules, and without a need for determinization or minimization because rules which are written that way are de−facto deterministic. So in a sense, our approach bears some similarities with the constraint−based formalism because we resort to "local rules" (Karlsson & al. 95), but we focus on identifying constituent boundaries (and not syntactic functions), and allow any level of embedding thanks to the use of a stack.

In the first part of this paper, we present our tool: a shallow−parser compiler. In a second part, we present output samples as well as several evaluations for French and for English, where the tool has been used to develop both an NP−chunker and a richer shallow−parser. We also explain why our approach is more tolerant to POS−tagging errors. Finally, we discuss some other practical uses which are made of this shallow−parser compiler.

2 Presentation of the compiler

Our tool has been developed using JavaCC (a compiler compiler similar to Lex & Yacc, but for java). The program takes as input a file containing rules. These rules aim at identifying constituent boundaries for a given language. (For example for English, one such rule could say "When encountering a preposition, start a PP"), either by relying on function words, or on morphological information (e.g. gender) if it is appropriate for the language which is being considered.

These rule files specify :

- A mapping between the "abstract" morpho−syntactic tags, used in the rules, and "real" morpho−syntactic tags as they will appear in the input.
- A declaration of the syntactic constituents which will be detected (e.g. NP, VP, PP ...)
- A set of unordered rules

From this rule file, the compiler generates a java program, which is a shallow−parser based on the rule file. One can then run this shallow−parser on an input to obtain a shallow−parsed text[3].

The compiler itself is quite simple, but we have decided to compile the rules rather than interpret them essentially for efficiency reasons. Also, it is language independent since a rule file may be written for any given language, and compiled into a shallow−parser for that language.

Each rule is of the form:

{*Preamble****}*** *disjunction of patterns* ***then*** *actions*

2.1 A concrete example : compiling a simple NP−chunker for English

In this section we present a very simple "toy" example which aims at identifying some NPs in the Penn Treebank[4] (Marcus & al 93).

In order to do so, we write a rule file, shown on figure 1. The top of the file declares a mapping between the abstract tagset we use in our rules, and the tagset of the PennTreebank. For example *commonN* corresponds to the 3 tags *NN, NNS, NNPS* in the PennTreebank. It then declares the labels of the constituents which will be detected (here there is only one: NP). Finally, it declares 3 rules.

```
%% A small NP−chunker for the Penn−treebank
tagmap <QuantityAdv:any,some,many>;
tagmap<ProperN:NNP>;
tagmap<det:DT,PDT>;
tagmap<commonN:NN,NNS,NNPS>;
tagmap<DemPro:D*>;
tagmap<Adj:JJ*>;
tagmap<OtherTags:V*,P,C*,RB*.,:,,>;
label NP;
%% rule 1
{} (:$det) | ($QuantityAdv:) | (:$DemPro) then close(),open(NP);
%% rule 2
{!NP} (:$commonN) | (:$Adj) | (:$ProperN) then close(),open(NP);
%% rule 3
{} (:$OtherTags) then close();
```

FIGURE 1 : An example of a rule−file

Rule 1 says that when a determiner, a quantity adverb or a demonstrative pronoun is encountered, the current constituent must be closed, and an NP must be opened. Rule 2 says that, when not inside an NP, if a common noun, an adjective or a proper noun is encountered, then the current constituent should be closed and an NP should be opened. Finally, Rule 3 says that when some other tag is encountered (i.e. a verb, a preposition, a punctuation, a conjunction

[3] The input is generally POS−tagged, although this is not an intrinsic requirement of the compiler.

[4] This example is kept very simple for sake of clarity. It does not aim at yielding a very accurate result.

or an adverb) then the current constituent should be closed.

This rule file is then compiled into an NP−chunker. If one inputs (a) to the NP−chunker, it will then output (b)

(a) The/DT cat/NNS eats/VBZ the/DT mouse/NNS ./.

(b) <NP> *The/DT cat/NNS </NP> eats/VBZ <NP> the/DT mouse/NNS </NP> ./.*

In our compiler, rules access a limited context :

- The constituent(s) being built
- The previous form and POS
- The current form and POS
- The next form and POS

So contrary to standard finite−state techniques, only **constituent boundaries** are explicited, and it is not necessary (or even possible) to specify all the possible ways a constituent may be realized .

As shown in section 3, this reduces greatly the number of rules in the system (from several dozens to less than 60 for a wide−coverage shallow−parser). Also, focussing only on constituent boundaries ensures determinism : there is no need for determinizing nor minimizing the automata we obtain from our rules.

Our tool is robust : it never fails to provide an output and can be used to create a parser for any text from any domain in any language.

It is also important to note that the parsing is done incrementally : the input is scanned strictly from left to right, in one single pass. And for each pattern matched, the associated actions are taken (i.e. constituent boundaries are added). Since there is no backtracking, this allows an output in linear time. If several patterns match, the longest one is applied. Hence our rules are declarative and unordered. Although in theory conflicts could appear between 2 patterns of same length (as shown in (c1) and (c2)), this has never happened in practice. Of course the case is nonetheless dealt with in the implementation, and a warning is then issued to the user.

(c1) {} (:a) (:b) then close();
(c2) {} (:a) (:b) then open(X);

As is seen on figure 1, one can write disjunctions of patterns for a given rule.

In this very simple example, only non recursive NP−chunks are marked, by choice. But it is not an intrinsic limitation of the tool, since any amount of embedding can be obtained (as shown in section 3 below), through the use of a Stack. From a formal point of view, our tool has the power of a deterministic push−down automaton.

When there is a match between the input and the pattern in a rule, the following actions may be taken :

- close(): closes the constituent last opened by inserting </X> in the output, were X is the syntactic label at the top of the Stack.
- open(X): opens a new constituent by inserting label <X> in the output
- closeWhenOpen(X,Y): delays the closing of constituent labeled X, until constituent labeled Y is opened.
- closeWhenClose(X,Y): delays the closing of constituent labeled X, until constituent labeled Y is closed.
- doNothing(): used to "neutralize" a shorter match.

Examples for the actions *open()* and *close()* were provided on figure 1. The actions *closeWhenOpen(X,Y)* and *closeWhenClose(X,Y)* allow to perform some attachments. For example a rule for English could say :

{NP} (:$conjCoord) then close(), open(NPcoord), closeWhenClose(NPcoord,NP);

This rule says that when inside an NP, a coordinating conjunction is encountered, a NPcoord should be opened, and should be closed only when the next NP to the right will be closed. This allows, for example, to obtain output (d) for a coordination of NPs[5].

(d) John likes
<NP>Apples</NP>
<NPcoord> and
<NP> green beans </NP>
</NPcoord>

An example of the action *doNothing()* for English could be:

{} (:$prep) then open(PP);
{} P(:$prep) (:$prep) then doNothing() ;

The first rule says that when a preposition is encountered, a PP should be opened. The second rule says that when a preposition is encountered, if the previous tag was also a preposition, nothing should be done. Since the pattern for

[5] This is shown as an example as to how this action can be used, it does not aim at imposing this structure to coordinations, which could be dealt with differently using other rules.

rule 2 is longer than the pattern for *rule 1*, it will apply when the second preposition in a row is encountered, hence "neutralizing" *rule 1*. This allows to obtain "flatter" structures for PPs, such as the one in (e1). Without this rule, one would obtain the structure in (e2) for the same input.

(e1) This costs <PP> up to 1000 $ </PP>

(e2) This costs
<PP> up
<PP> to 1000 $ </PP>
</PP>

3. Some "real world" applications

In this section, we present some uses which have been made of this Shallow−Parser compiler. First we explain how the tool has been used to develop a 1 million word Treebank for French, along with an evaluation. Then we present an evaluation for English.

It is well known that evaluating a Parser is a difficult task, and this is even more true for Shallow−Parsers, because there is no real standard task (some Shallow−parsers have embedded constituents, some encode syntactic functions, some encode constituent information, some others dependencies or even a mixture of the 2) There also isn't standard evaluation measures for such tools. To perform evaluation, one can compare the output of the parser to a well−established Treebank developed independently (assuming one is available for the language considered), but the result is unfair to the parser because generally in Treebanks all constituents are attached. One can also compare the output of the parser to a piece of text which has been manually annotated just for the purpose of the evaluation. But then it is difficult to ensure an objective measure (esp. if the person developing the parser and the person doing the annotation are the same). Finally, one can automatically extract, from a well−established Treebank, information that is relevant to a given , widely agreed on, non ambiguous task such as identifying bare non−recursive NP−chunks, and compare the output of the parser for that task to the extracted information. But this yields an evaluation that is valid only for this particular task and may not well reflect the overall performance of the parser. In what follows, in order to be as objective as possible, we use these 3 types of evaluation, both for French and for English[6], and use standard measures of recall and precision. Please bear in mind though that these metric measures, although very fashionable, have their limits[7]. Our goal is not to show that our tool is the one which provides the best results when compared to other shallow−parsers, but rather to show that it obtains similar results, although in a much simpler way, with a limited number of rules compared to finite−state techniques and more tolerance to POS errors, and even in the absence of available training data (i.e. cases were probabilistic techniques could not be used). To achieve this goal, we also present samples of parsed outputs we obtain, so that the reader may judge for himself/herself.

3.1. A shallow−parser for French

We used our compiler to create a shallow−parser for French. Contrary to English, very few shallow−parsers exist for French, and no Treebank actually exist to train a probabilistic parser (although one is currently being built using our tool c.f. (Abeillé & al. 00)). Concerning shallow−parsers, one can mention (Bourigault 92) who aims at isolating NPs representing technical terms, whereas we wish to have information on other constituents as well, and (Ait−Moktar & Chanod 97) whose tool is not publicly available. One can also mention (Vergne 99), who developed a parser for French which also successfully relies on function words to identify constituent boundaries. But contrary to us, his tool does not embed constituents[8]. And it is also not publicly available.

In order to develop a set of rules for French, we had to examine the linguistic characteristics of this language. It turns out that although French has a richer morphology than English (e.g. gender for nouns, marked tense for verbs), most constituents are nonetheless triggered by the occurrence of a function word. Following the linguistic tradition, we consider as function words all words associated to a POS which labels a closed−class i.e. : determiners, prepositions, clitics, auxiliaries, pronouns (relative, demonstrative), conjunctions

[6] Of course, manual annotation was done by a different person than the one who developed the rules.

[7] For instance in a rule−based system, performance may often be increased by adding more rules.

[8] Instead, it identifies chunks and then assigns some syntactic functions to these chunks.

(subordinating, coordinating), auxiliaries, punctutation marks and adverbs belonging to a closed class (e.g. negation adverbs "ne" "pas")[9].
The presence of function words makes the detection of the beginning of a constituent rather easy. For instance, contrary to English, subordinating conjunctions (*que/that*) are never omitted when a subordinating clause starts. Similarly, determiners are rarely omitted at the beginning of an NP.
Our aim was to develop a shallow−parser which dealt with some embedding, but did not commit to attach potentially ambiguous phrases such as PPs and verb complements. We wanted to identify the following constituents : NP, PP, VN (verbal nucleus), VNinf (infinitivals introduced by a preposition), COORD (for coordination), SUB (sentential complements), REL (relative clauses), SENT (sentence boundaries), INC (for constituents of unknown category), AdvP (adverbial phrases).
We wanted NPs to include all adjectives but not other postnominal modifiers (i.e. postposed relative clauses and PPs), in order to obtain a structure similar to (f).

(f) <NP> Le beau livre bleu </NP>
<PP> de <NP>ma cousine</NP> </PP> …
(my cousin's beautiful blue book)

Relative clauses also proved easy to identify since they begin when a relative pronoun is encountered. The ending of clauses occurs essentially when a punctuation mark or a conjunction of coordination is encountered or when another clause begins, or when a sentence ends (g1) . These rules for closing clauses work fairly well in practice (see evaluation below) but could be further refined, since they will yield a wrong closing boundary for the relative in a sentence like (g2)

(g1) <SENT> <NP> Jean </NP>
<VN> voit</VN>
<NP>la femme </NP>
<REL> qui
<VN> pense </VN>
<SUB> que
<NP> Paul </NP>
<VN> viendra </VN>
</SUB>
</REL> . </SENT>
(John sees the woman who thinks that Paul will come)

(g2) * <SENT> <NP> Jean </NP>
<VN> pense</VN>
<SUB> que
<NP> la femme </NP>
<REL> que
<NP> Pierre </NP>
<VN> voit</VN>
<VN> aime </VN>
<NP> Donald </NP>
</REL>
</SUB> . </SENT>
*(*John thinks that the woman [REL that Peter sees likes Donald])*

Concerning clitics, we have decided to group them with the verb (h1) even when dealing with subject clitics (h2). One motivation is the possible inversion of the subject clitic (h3).

(h1) <SENT><NP> JEAN </NP>
<VN> le lui donne</VN> . </SENT>
(J. gives it to him).

(h2) <SENT> <VN> Il le voit </VN> . </SENT>
(He sees it)

(h3) <SENT><VN> L'as tu vu </VN> ? </SENT>
(Him did you see ?).

Sentences are given a flat structure, that is complements are not included in a verbal phrase[10] (i). From a practical point of view this eases our task. From a theoretical point of view, the traditional VP (with complements) is subject to much linguistic debate and is often discontinuous in French as is shown in (j1) and (j2): In (j1) the NP subject (IBM) is postverbal and precedes the locative complement (*sur le marché*). In (j2), the adverb *certainement* is also postverbal and precedes the NP object (*une augmentation de capital*).

(i) <SENT><NP> JEAN </NP>
<VN> donne</VN>
<NP>une pomme</NP>
<PP> à <NP> Marie </NP> </PP> . </SENT>
(John gives an apple to Mary)

(j1) *les actions **qu'a mises** IBM **sur le marché***
(the shares that IBM put on the market)

(j2) *Les actionnaires **décideront** certainement **une augmentation de capital** (the stock holders will certainly decide on a raise of capital)*

3.1.1 Evaluation for French

[9] Considering punctuation marks as function words may be "extending" the linguistic tradition. Nonetheless, it is a closed class, since there is a small finite number of punctuation marks.

[10] Hence the use of VN(for verbal nucleus) instead of VP.

```
<SENT>  <NP> La:Dfs proportion:NC </NP>
          <PP> d':P <NP> étudiants:NC </NP> </PP>
          <PP> par_rapport_à:P
           <NP> la:Ddef population:NC</NP> </PP>
          <PONCT> ,:PONCT </PONCT>
          <PP> dans:P <NP> notre:Dposs
pays:NC</NP> </PP>
         <PONCT> ,:PONCT</PONCT>
         <VN> est:VP inférieure:Aqual </VN>
        <PP> à:P <NP> ce:PROdem</NP> </PP>
         <REL> qu:PROR3ms
                 <VN> elle:CL est:VP </VN>
           <COORD> <PP> à:P <NP> les:Ddef Etats-
Unis:NP </NP> </PP>   ou:CC
            <PP> à:P <NP> le:Ddef
           Japon:NP</NP></PP> </COORD>
         </REL> <PONCT> .:PONCT</PONCT>
</SENT>
<SENT>  <NP> Les:Dmp pays:NC</NP>
          <NP> les:Ddef plus:ADV efficaces:Aqual
          économiquement:ADV</NP>
           <VN> sont:VP</VN>
           <NP> ceux:PROdem</NP>
           <REL> qui:PROR
               <VN> ont:VP</VN>
          <NP> les:Ddef travailleurs:NC les:Ddef
          mieux:ADV</NP>
               <VN> formés:VK</VN>
           </REL> <PONCT> .:PONCT</PONCT>
</SENT>
<SENT>  <ADVP> D'autre_part:ADV</ADVP>
          <PONCT> ,:PONCT </PONCT>
         <SUB> si:CS
       <VN> nous:CL voulons:VP demeurer:VW
</VN>
        <NP> une:Dind grande_puissance:NC</NP>
         </SUB> <PONCT> ,:PONCT</PONCT>
          <VN> nous:CL devons:VP rester:VW</VN>
           <NP> un:Dind pays:NC</NP>
        <REL> qui:PROR
                 <VN> crée:VP</VN>
                <NP> le:Ddef savoir:NC</NP>
      </REL><PONCT> .:PONCT</PONCT>
</SENT>
<SENT>  <COORD> Et:CC
          <PP> pour:P <NP> cela:PROdem</NP>
          </PP> </COORD>
          <PONCT> ,:PONCT </PONCT>
         <VN> il:CL faut:VP</VN>
          <NP> un:Dind
          enseignement_supérieur:NC fort:Aqual</NP>
          <PONCT> .:PONCT</PONCT> </SENT>
<SENT>  <COORD> Mais:CC
            <PP> en_dehors_de:P
                    <NP> ces:Ddem raisons:NC
          économiques:Aqual ou:CC
          philosophiques:Aqual </NP> </PP>
          </COORD>
         <PONCT> ,:PONCT </PONCT>
         <VN> il:CL y:CL a:VP </VN>
         <NP> la:Ddef réalité:NC </NP>
          <NP> les:Ddef étudiants:NC</NP>
         <VN> sont:VP</VN>
         <PP> à:P <NP> notre:Dposs porte:NC</NP>
</PP>  <PONCT> .:PONCT</PONCT> </SENT>
```

FIGURE 2 : Sample output for French

When we began our task, we had at our disposal a 1 million word POS tagged and hand−corrected corpus (Abeillé & Clément 98). The corpus was meant to be syntactically annotated for constituency. To achieve this, precise annotation guidelines for constituency had been written and a portion of the corpus had been hand−annotated (independently of the development of the shallow−parser) to test the guidelines (approx. 25 000 words) .

To evaluate the shallow parser, we performed as described at the beginning of section 3 : We parsed the 1 million words. We set aside 500 sentences (approx. 15 000 words) for quickly tuning our rules. We also set aside the 25 000 words that had been independently annotated in order to compare the output of the parser to a portion of the final Treebank. In addition, an annotator hand−corrected the output of the shallow−parser on 1000 new randomly chosen sentences (approx. 30 000 words). Contrary to the 25 000 words which constituted the beginning of the Treebank, for these 30 000 words verb arguments, PPs and modifiers were not attached. Finally, we extracted bare non−recursive NPs from the 25 000 words, in order to evaluate how the parser did on this particular task.

When compared to the hand−corrected parser's output, for opening brackets we obtain a recall of 94.3 % and a precision of 95.2%. For closing brackets, we obtain a precision of 92.2 % and a recall of 91.4 %. Moreover, 95.6 % of the correctly placed brackets are labeled correctly, the remaining 4.4% are not strictly speaking labeled incorrectly, since they are labeled INC (i.e. unknown) These unknown constituents, rather then errors, constitute a mechanism of underspecification (the idea being to assign as little wrong information as possible)[11].

When compared to the 25 000 words of the Treebank, For opening brackets, the recall is 92.9% and the precision is 94%. For closing brackets, the recall is 62,8% and the precision is 65%. These lower results are normal, since the Treebank contains attachments that the parser is not supposed to make.

Finally, on the specific task of identifying non−recursive NP−chunks, we obtain a recall of 96.6 % and a precision of 95.8 %. for opening

[11] These underspecified label can be removed at a deeper parsing stage, or one can add a guesser .

brackets, and a recall and precision of resp. 94.3% and 92.9 % for closing brackets.

To give an idea about the coverage of the parser, sentences are on average 30 words long and comprise 20.6 opening brackets (and thus as many closing brackets). Errors difficult to correct with access to a limited context involve mainly "missing" brackets (e.g. "comptez vous * ne pas le traiter" (do you expect not to treat him) appears as single constituent, while there should be 2) , while "spurious" brackets can often be eliminated by adding more rules (e.g. for multiple prepositions : "de chez"). Most errors for closing brackets are due to clause boundaries(i.e. SUB, COORD and REL).

To obtain these results, we had to write only 48 rules.

Concerning speed, as argued in (Tapanainen & Järvinen, 94), we found that rule−based systems are not necessarily slow, since the 1 million words are parsed in 3mn 8 seconds.

One can compare this to (Ait−Moktar & Chanod 97), who, in order to shallow−parse French resort to 14 networks and parse 150words /sec (Which amounts to approx. 111 minutes for one million words)[12]. It is difficult to compare our result to other results, since most Shallow−parsers pursue different tasks, and use different evaluation metrics. However to give an idea, standard techniques typically produce an output for one million words in 20 mn and report a precision and a recall ranging from 70% to 95% depending on the language, kind of text and task. Again, we are not saying that our technique obtains best results, but simply that it is fast and easy to use for unrestricted text for any language. To give a better idea to the reader, we provide an output of the Shallow−parser for French on figure 2.

In order to improve our tool and our rules, a demo is available online on the author's homepage.

3.2 A Shallow−Parser for English

We wanted to evaluate our compiler on more than one language, to make sure that our results were easily replicable. So we wrote a new set of rules for English using the PennTreebank tagset, both for POS and for constituent labels.

We sat aside sections 00 and 01 of the WSJ for evaluation (i.e. approx. 3900 sentences), and used other sections of the WSJ for tuning our rules.

Contrary to the French Treebank, the Penn Treebank contains non−surfastic constructions such as empty nodes, and constituents that are not triggered by a lexical items.

Therefore, before evaluating our new shallow−parser, we automatically removed from the test sentences all opening brackets that were not immediately followed by a lexical item, with their corresponding closing brackets, as well as all the constituents which contained an empty element. We also removed all information on pseudo−attachment. We then evaluated the output of the shallow−parser to the test sentences. For bare NPs, we compared our output to the POS tagged version of the test sentences (since bare−NPs are marked there).

For the shallow−parsing task, we obtain a precision of 90.8% and a recall of 91% for opening brackets, a precision of 65.7% and recall of 66.1% for closing brackets. For the NP−chunking task, we obtain a precision of 91% and recall of 93.2%, using an "exact match" measure (i.e. both the opening and closing boundaries of an NP must match to be counted as correct).

The results, were as satisfactory as for French. Concerning linguistic choices when writing the rules, we didn't really make any, and simply followed closely those of the Penn Treebank syntactic annotation guidelines (modulo the embeddings, the empty categories and pseudo−attachments mentioned above).

Concerning the number of rules, we used 54 of them in order to detect all constituents, and 27 rules for NP−chunks identification. . In sections 00 and 01 of the wsj there were 24553 NPs, realized as 1200 different POS patterns (ex : CD NN, DT $ JJ NN, DT NN…). Even though these 1200 patterns corresponded to a lower number of regular expressions, a standard finite−state approach would have to resort to more than 27 rules. One can also compare this result to the one reported in (Ramshaw & Marcus 95) who, obtain up to 93.5% recall and 93.1% precision on the same task, but using between 500 and 2000 rules.

3.3 Tolerance to POS errors

To test the tolerance to POS tagging errors, we have extracted the raw text from the English

[12] They report a recall ranging from 82.6% and 92.6% depending on the type of texts, and a precision of 98% for subject recognition, but their results are not directly comparable to ours, since the task is different.

corpus from section 3.2., and retagged it using the publicly available tagger TreeTagger (Schmid, 94). without retraining it. The authors of the tagger advertise an error−rate between 3 and 4%. We then ran the NP−chunker on the output of the tagger, and still obtain a precision of 90.2% and a recall of 92% on the "exact match" NP identification task: the fact that our tool does not rely on regular expressions describing "full constituent patterns" allows to ignore some POS errors since mistagged words which do not appear at constituent boundaries (i.e. essentially lexical words) have no influence on the output. This improves accuracy and robustness. For example, if "first" has been mistagged *noun* instead of *adjective* in *[NP the first man] on the moon* ..., it won't prevent detecting the NP, as long as the determiner has been tagged correctly.

Conclusion

We have presented a tool which allows to generate a shallow−parser for unrestricted text in any language. This tool is based on the use of a imited number of rules which aim at identifying constituent boundaries. We then presented evaluations on French and on English, and concluded that our tools obtains results similar to other shallow−parsing techniques, but in a much simpler and economical way.

We are interested in developing new sets of rules for new languages (e.g. Portuguese and German) and new style (e.g. French oral texts). It would also be interesting to test the tool on inflectional languages.

The shallow−parser for French is also being used in the SynSem project which aims at syntactically and semantically annotating several millions words of French texts distributed by ELRA[13]. Future improvements of the tool will consist in adding a module to annotate syntactic functions, and complete valency information for verbs, with the help of a lexicon (Kinyon, 00).

Finally, from a theoretical point of view, it may be interesting to see if our rules could be acquired automatically from raw text (although this might not be worth it in practice, considering the small number of rules we use, and the fact that acquiring the rules in such a way would most likely introduce errors).

Acknowledgement We especially thank F. Toussenel, who has performed most of the evaluation for French presented in section 3.1.1.

References

Abeillé A. Clément L. 1999 : A tagged reference corpus for French. Proc. LINC−EACL'99. Bergen

Abeillé A., Clément L., Kinyon A., Toussenel F. 2001 Building a Treebank for French. In Treebanks (A Abeillé ed.). Kluwer academic publishers.

Abney S. 1991. Parsing by chunks. In Principle−based Parsing. (R. Berwick, S. Abney and C. Tenny eds), Kluwer academic publishers.

Aït−Mokhtar S. & Chanod J.P. 1997. Incremental Finite−State Parsing. Proc. ANLP'97, Washington,

Bourigault 1992 : Surface Grammatical analysis for the extraction of terminological noun phrases. Proc. COLING'92. Vol 3, pp. 977−981

Brants T., Skut W., Uszkoreit H., 1999. Syntactic Annotation of a German Newspaper Corpus. Proc. ATALA Treebank Workshop. Paris, France.

Daelemans W., Buchholz S., Veenstra J.. Memory−Based Shallow Parsing.Proc.CoNLL−EACL'99

Grefenstette G.. 1996. Light Parsing as Finite−State Filtering. Proc. ECAI '96 workshop on "Extended finite state models of language".

Joshi A.K. Hopely P. 1997. A parser from antiquity. In Extended Finite State Models of Language. (A. Kornai ed.). University Press.

Karlsson F., Voutilainen A., Heikkil J., Antilla A. (eds.) 1995. Constraint Grammar: a language−independent system for parsing unrestricted text. Mouton de Gruyer.

Kinyon A. 2000. Hypertags. Proc. COLING'00. Sarrebrucken.

Magerman D.M., 1994 Natural language parsing as statistical pattern recognition. PhD Dissertation, Stanford University.

Marcus M., Santorini B., and Marcinkiewicz M.A. 1993. Building a large annotated corpus of english: The penn treebank. Computational Linguistics, 19:313−−330.

Ramshaw, L.A. & Marcus, M.P., 1995. Text Chunking using Transformation−Based Learning, ACL Third Workshop on Very Large Corpora, pp.82−94, 1995.

Ratnaparkhi A. 1997. linear observed time statistical parser based on maximum entropy models. Technical Report cmp-lg/9706014.

Tapanainen P. and Järvinen T., 1994, Syntactic Analysis of a Natural Language Using Linguistic Rules and Corpus−Based Patterns. Proc. COLING'94. Vol I, pp 629−634. Kyoto.

Schmid H. 1994 Probabilistic Part−Of−Speech Tagging Using Decision Trees. Proc. NEMLAP'94.

Vergne J. 1999. Etude et modélisation de la syntaxe des langues à l'aide de l'ordinateur. Analyse syntaxique automatique non combinatoire. Dossier d'habilitation à diriger des recherches. Univ. de Caen.

[13] European Language Ressources Association