Internal Use Only

TR-IT-0143

005

Towards a Long Sentences Preprocessor for the ATR English Grammar

Hassan EL NAHAS

December 22, 1995

Abstract

In general, natural-language parsers are extremely inaccurate on very long sentences. In this report, a new approach will be outlined to the problem of insuring accurate parses of long sentences by broad-coverage grammar/parsers, or at least by the ATR English Grammar parser. In addition to an explanation of the overall strategy being pursued, this report will include an overview on preliminary experimental results within a crucial area of this approach: reliably identifying grossly-categorized sentence types as a first, preprocessing step in the parsing of long sentences.

Contents

1 Introduction

In this report, I will try to explain why it was so important to have a preprocessor of long sentences for the ATR English project. I will give first an overview of the ATR main grammar, then I will describe this preprocessor and the grammar that it involves in Section 2. I will describe the steps taken to provide the Long Sentences Grammar with the necessary training data and questions to build the decision trees in Section 3. Experimental results and conclusions obtained using this grammar follow in Section 4.

2 The Grammar

For many decades, resarchers have been dreaming of a parser for everyday English sentences. You input any sentence from today's newspaper, from your computer manual, from the latest business letter you have received. The ouput is this sentence correctly analysed. I will explain later what I mean exactly by an analysis of a sentence, when I will describe the grammar. We can understand easily that with such an analysis, it would be easier to do automatic translation from one language to another. And there are many other applications of parsing.

2.1 A View Of The ATR English Grammar

The statistical grammar used at ATR is a feature-based context-free phrase structure grammar employing traditional syntactic categories. It consists of the following items:

- Features - The values of these features represent the syntactic and semantic interpretations of a sentence. They give information about the usage of words and phrases in the sentence.
- Feature bundles - These are collections of feature-variables and their values.
- Rules - A rule consists of a parent feature bundle, and a sequence of one or more child feature bundles. In a parsed tree, each node of the tree corresponds to one rule, and the leaves of this tree represent the tags. Therefore, each leaf corresponds to one word.

To illustrate the rules, here is one example:

$$\begin{pmatrix} f1:a \\ f2:V1 \\ f3:V2 \end{pmatrix} \rightarrow \begin{pmatrix} f1:b \\ f2:V1 \\ f3:V3 \end{pmatrix} \begin{pmatrix} f1:c \\ f2:V1 \\ f3:V2 \end{pmatrix}$$

Where f1, f2, and f3 are features; a, b, and c are feature values; and V1, V2, and V3 are variables over the feature values. This rule can easily be extended to a certain number of rules with no variables. In each of the three feature bundles specified in this rule, only three features figure, that means that all the others take default values.

From now on, I will refer to the number of words in a sentence by n and to the number of rules in the grammar by G.

For more details of the grammar see [3].

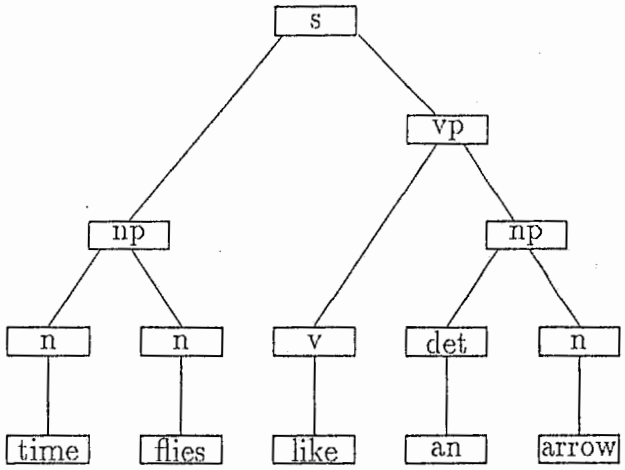
2.2 What Is A Correct Analysis?

A correct analysis of a sentence consists of the two following steps:

- Tagging: assigning to each word in the sentence the correct tag.
- Parsing: defining the constituents of the sentence in order to build the parsed tree.

To understand what is exactly a parsed tree, here is an example[1]:

[1]The parse shown is this example is in fact false, but shows how difficult the problem of ambiguity in parsing is.



Our problem here is not tagging, since it does not take a long time to tag a sentence compare to parse it. The algorithm of parsing is very expensive and it is aproximately the same in any parser, its compexity is: $O(G^2.n^3)$. In addition to that, generally parsers do poorly on long sentences.

The idea was then to preprocess the tagged sentence with a small grammar in order to find the high level structure of this sentence. Then, it is possible to parse each splitted part of the sentence independantly with the main grammar. After all, we can get the whole parse by reattaching these parts together.

2.3 Long Sentences Grammar

Long sentences splitting grammar like ATR grammar consists of the following items:

- 34 features, compare to 67 in the main grammar.
- 31 rules, compare to 1075 in the man grammar.
- 2625 tags, compare to 2625 in the main grammar.

We chose an initial definition of 25 words of length to characterize the notion *long sentence*. This grammar is expected to reduce considerably the amount

of calculus needed to train the model and parse the sentences, since the number of rules has been considerably reduced. Besides, among the 34 features that remain, only 6 are relevants to long sentences splitting; the others have been kept to define all the tags.

What is important to know about the rules in the long sentences splitting grammar is that it must include one particular rule which is the rule *no split*. This rule states that a sentence can not be splitted and that it must be parsed entirely. The *no split* rule works like the other rule and there is no need to consider it except when generating it for the training data. In total, there is only 20 non-terminal rules.

2.4 One Example

Here is a parse using long sentences grammar:

```
[start [ibbar_sd_period [ibbar [wordstring1 For_IIFOR [wordstring1
 all_DB [wordstring1 the_AT [wordstring1 furor_NN1INTER-ACT
[wordstring ,_, wordstring] wordstring1] wordstring1] wordstring1]
 wordstring1] ibbar] [sd [wordstring1 there_EX [wordstring1 is_VBZ
[wordstring1 nothing_PN1 [wordstring1 particularly_RRDEGREE
[wordstring1 complex_JJCOMP-B [wordstring1 about_IIABOUT [wordstring1
the_AT [wordstring1 concept_NN1MEANING [wordstring1 of_IIOF
[wordstring1 stock-index_JJDOCUMENT [wordstring1 arbitrage_NN1COMP-B
[wordstring1 ,_, [wordstring1 the_AT [wordstring1 most_RGT
[wordstring1 controversial_JJPROBLEM [wordstring1 type_NN1CLASS
[wordstring1 of_IIOF [wordstring1 computer-assisted_JJVVNHELP
[wordstring1 program_NN1SYSTEM [wordstring trading_NVVGINTER-ACT
wordstring] wordstring1] wordstring1] wordstring1] wordstring1]
wordstring1] wordstring1] wordstring1] wordstring1] wordstring1]
wordstring1] wordstring1] wordstring1] wordstring1] wordstring1]
wordstring1] wordstring1] wordstring1] wordstring1] wordstring1]
sd] ._. ibbar_sd_period] start]
```

And here the parse of the same sentence using the main grammar:

```
[start [sprpd1 [sprime2 [ibbar1 [i1e [p1 For_IIFOR [nbar4 [d10 all_DB
the_AT d10] [n1a furor_NN1INTER-ACT n1a] nbar4] p1] i1e] ,_, ibbar1]
```

```
[sd3 there_EX [vbar1 [v2 is_VBZ [nbarq25a [nbarq21 [nbar6 nothing_PN1
nbar6] [j5a [r1 particularly_RRDEGREE r1] complex_JJCOMP-B j5a]
nbarq21] [i1e [p1 about_IIABOUT [nbarq10 [nbarq4 [nbar4 [d1 the_AT d1]
[n1a concept_NN1MEANING n1a] nbar4] [i1e [p1 of_IIOF [nbar1 [n4 [n2a
stock-index_JJDOCUMENT n2a] [n1a arbitrage_NN1COMP-B n1a] n4] nbar1]
p1] i1e] nbarq4] ,_, [nbarq4 [nbar28 [d1 the_AT d1] [j6 [r1 most_RGT
r1] controversial_JJPROBLEM j6] [n1a type_NN1CLASS n1a] nbar28] [i1e
[p1 of_IIOF [nbar12 [j1 computer-assisted_JJVVNHELP j1] [n4 [n1a
program_NN1SYSTEM n1a] [n11 trading_NVVGINTER-ACT n11] n4] nbar12] p1]
i1e] nbarq4] nbarq10] p1] i1e] nbarq25a] v2] vbar1] sd3] sprime2]
._. sprpd1] start]
```

This was the correct parse for the 26 words following sentence:

```
For all the furor, there is nothing particularly complex about the
concept of stock-index arbitrage, the most controversial type
of computer-assisted program trading.
```

What follows each word by an underscore is the tag assigned to this word. And it is clear that tags are the same in both cases. We can see that this long sentences splitting grammar parse gives only the following high level structure of the sentence:

$$\text{ibbar\_sd\_period} \rightarrow \text{ibbar sd period}$$

All constituents labels like *wordstring* or *wordstring1* are temporary and will be changed when the splitted parts will be parsed using the main grammar.

3 Experimental Method

As described in Section 2, the long sentences splitting grammar is similar formally to the usual grammar. Accordingly the model is trained like before and the same code can be used. The grammar has been written using the grammarian's workbench tool[2], which provides the project with an effective environment for editing and testing rules on example sentences. One is only required to check that all the feature bundles have been specified and that

[2]This tool was developped by David M. Magerman.

nothing has been forgotten[3].

The main tool to attribute a tag to each word, and to compute the probability of each parse found is the statistical binary decision tree. These decision trees ask binary questions about the sentence, then in accordance to the answers, we get through the trees and at the leaf, we read a probability distribution. I will not describe decision trees more than that because it is not the topic of this report here; for more information see [1].

To build a decision tree we do the supervised training, so we need:

1. A large number of correctly parsed sentences to provide the system with the statistics.

2. Relevant questions written by a grammarian. We can imagine here any kind of questions that might be relevant to the fact that we have to split the sentence or not.

3.1 Generating The Training Data

The training data represent all the data necessary to provide the system with statistics. Such statistics are used to build the decision trees. It is very important to represent all of the rules in the three types of data files: train, heldout, and test. The proportions that have been used are respectively 80%, 10% and 10%. The way to generate this data from the usual data is easy but takes a long time. The number of sentences that train the model must be more than 1500 sentences. The results I have with 1000 sentences only were bad. This number is purely experimental. This number may depend on the number of rules in the grammar. The more the rules are, the more sentences may be needed to train the model.

Since it takes a long time to parse sentences by hand, it was advisable to translate the sentences already parsed with the main grammar format to the long sentences splitting grammar format. In the following are the steps taken

[3]I have noticed, while running experiments, that some feature bundles were missed; in this case the system cannot recognize the parse of the sentence, and therefore neglects the sentence from the training data.

to generate the sentences according to a certain number of rules including the rule *no split*. The input is a large corpus of sentences parsed correctly according to the main grammar[4].

- Filter this corpus so that only sentences with more than 25 words remain.
- For each rule of the long sentences splitting grammar, modify, keep or delete the constituent labels of the sentence that correspond to this rule so that they correspond to our grammar.
- Make all the sentences that remain, *no split* sentences[5].
- Append 80% of each rule file to the training data, 10% to the smoothing data[6] and 10% to the test sentences.

3.2 Writing Questions

Once the training data are ready, questions must be correctly written in the machine language. The questions are usually written by a grammarian. For the purpose of writing any possible questions, a question language has been provided to the project which contains a large number of primitives. Questions are written as combination of these primitives as shown in the examples below. It is interesting to add more and more primitives to the question language.

There are two different kinds of questions:

- Binary questions: return *yes* or *no* answer.
- Bit questions: have several possible answers; each answer is represented by a bit string. Usually a file is associated to such a question containing the bit string assigned to each possible answer.

[4]What I mean by main grammar is the grammar that will parse the splitted parts of a long sentence.

[5]The percentage of *no split* sentences must be less than 30%; if this percentage exceed 30% the grammar will perform poorly. In our first experiments this percentage was about 25%, which also can be lowered by adding new relevant rules to the grammar.

[6]The smoothing data are used to make the decision trees less representative of the specific training data.

A binary question returns a *yes* or *no* answer, while a bit question returns an answer represented by a bit string. Usually a file is associated to each bit question and contains the bit string assigned to all possible answers of the question. The number of questions does not mean anything: what is important is the fact that these questions are relevant. To check that the questions have worked correctly, it is useful to view the decision tree through a tree editor, as described in Appendix A. Viewing the decision tree gives a lot of information about the efficiency of the questions[7].

Some examples:

- myquestion Bit(IsEven(SentLength()))
- comma_first Bit(Member(WordSet(','),ConsWord(PickNode(0,0),0)))
- numcomma Bits("num.bits",NumWord(','))

"num.bits" is:

0	1000
1	0100
2	0010
3	0001

All the functions that figure in the examples are primitives. I have written the first question myself; it returns *yes*, if and only the length of the sentences is an even number. It is evident that this question is not relevant, but it is reassuring write such a question and see that decision tree will simply ignore it.
The second question returns *yes* if and only if the first word in the active node is a comma.
The third question is a bit question; it returns the number of commas in the sentence. For three and more it will return the same bit string.

The way to improve the system is to write more and more questions, a task undertaken by the ATR grammarians. For more details about the question language see Appendix B.

[7] I have noticed that some questions were not expected to be important compared to another, and vice versa. Such remarks could be very crucial to improve the system.

4 Results

The characteristics of the first experiments were the following:

- 2115 long sentences were used.
- The average length of these sentences was 34.3 words.
- The percentage of the rule *no split* was about 25%.
- 49 questions were written; they return in total 212 bits.

I suppose that the percentage of *no split* sentences is still big, but it can be reduced by adding more relevant rules to the actual version of long sentences splitting grammar. We believe that most long sentences in English can be splitted.

Here are the results:

Statistics	Percentage
Correct in top 0	00.00
Correct in top 1	26.06
Correct in top 5	76.36
Correct in top 10	80.61
Correct in top 15	83.03
$p^c > p^b/10$	03.03
$p^c > p^b/100$	09.09

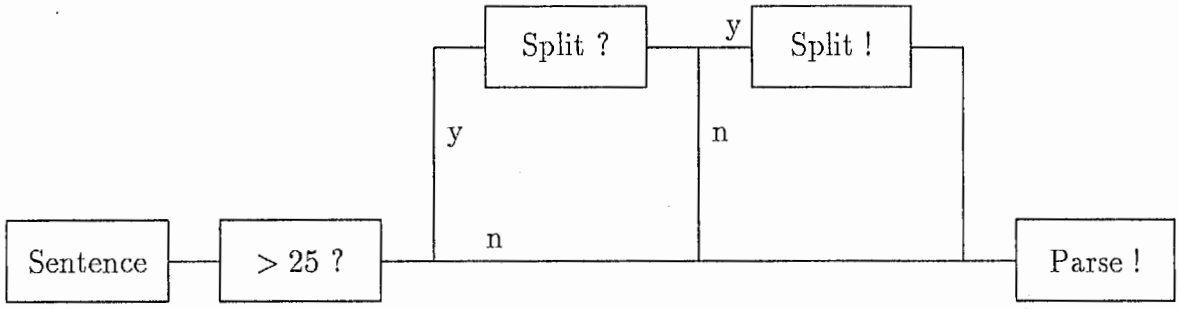
Where p^c and p^b refer respectively to the probabilities of the correct parse and the best parse. The aim is that they refer to the same parse.
The results appeared to be bad, but when I looked through the parses found I noticed that the most probable parse was always the *no split* parse. That meant that the rule *no split* was too powerful. Actually, it is easy to see that splitting a sentence is more difficult than assigning it one of the splitting rules described in the grammar. The idea is to build a split/no split model before trying to split the sentence.

I can say that these results are promising for the following reasons:

- We have succeeded in identifying a very small set of parses containing the correct parse, a good percentage of the time, choosing from a very large group of parses.
- The parses that have a probability better than the correct might be filtered later when processed with the main grammar.

The conclusion then is to add a new grammar before the long sentences splitting grammar, whose task is to predict if a sentence is to be split or not. With such a grammar, the rule *no split* can be removed without problems from long sentences splitting grammar and can avoid affecting the grammar. Consequently, the long sentences splitting grammar is able to deal exclusively with splittable sentences. The *no split* parse will not be in the parses found. The 2nd statistics show the correct results.

A summary of what precedes can be shown in the following figure:



The grammar which must be added is represented in the box *Split ?*. I don't expect that this grammar will be difficult to write and most questions asked will be taken from the actual long sentences splitting grammar. In addition to that, the actual version of long sentences splitting grammar and the set questions are preliminary versions only; now that they are written it is easy for the grammarian to improve them. Of course, the main part of the project remains in the box "*Parse !*" which represents what the main grammar does.

Appendices

A Viewing Decision Trees

The idea of viewing decision trees is not new, since parsed trees are always seen by a tree editor. A tree editor uses an explicit representation of node and subtree contours. The editor that we use is the OO-Browser graphical interface [4]. This program demonstrates a dynamic tree-drawing.

The decision trees are generally very big. The depth of the tagger tree for example can be more than 15. The number of leaves in this case is about 2^{15}, so it is easy to understand that it is quite unfeasible to get through the tree manually.

To view a decision tree, the format of the tree must be changed so that it can be read by the OO-Browser. Some functions have been added by Stephan Eubank and I for that purpose.

B Question Language

Questions must be written for building decision trees. It is then useful to provide the project with some primitives that can be used in a certain combination to write the questions. This is what actually does the question language described in [2].

The question language functions are divided into five different categories, based on the data type they return: node functions, bit-array functions, list functions, boolean functions, and value functions. Each question language function extracts some information from the current parse state. The parse state consists of a sequence of parse nodes spanning the entire sentence with an indication of which parse node is active.

Actually, a large number of functions are already written, but it would be nice to add some more functions. The way to add a function is not difficult. When writing a new function, we must distinguish between the question

that return always the same answer whenever asked to one sentence and the question that returns some information about the current active node.

References

[1] David M. Magerman. 1994. Natural Language Parsing as Statistical Pattern Recognition. Ph. D. Dissertation. Standford University. Standford, Ca., USA.

[2] David M. Magerman. 1995. A Question Language for the ATR Statistical Parser. Manuscript, ATR ITL. Kyoto, Japan.

[3] David M. Magerman. 1995. A Statistical Grammar Development Environment. Manuscript, ATR ITL. Kyoto, Japan.

[4] IEEE Software. July 1990, pp.21-28.