Abstract

Pro3Gres is a robust, hybrid, deep-syntatic dependency-based parser. The architecture and the implementation are carefully designed to keep search-spaces small without compromising much on the linguistic performance or adequacy. The resulting parser is deep-syntactic like a formal grammar-based parser but at the same time mostly context-free and fast enough for large-scale application to unrestricted texts. It combines a number of successful current approaches into a hybrid, comparatively simple, modular and open model.

This technical documentation gives an overview of the code. Due to the complexity of the program, many simplifications in the description were unavoidable.

PRO3GRES TECHNICAL DOCUMENTATION

Gerold Schneider

August 11, 2011

Contents

Chapter 1: Overview of the Parser and the Architecture

In this chapter we first give an overview of the parser with an example. Each module is introduced and briefly discussed. Second, we give a brief overview of the structure of the code.

1.1 The Modular Architecture

Pro3Gres is a modular system. Each problem is broken down into sub-problems, and a simplified solution is used. As much of the processing and disambiguating as reasonably possible is done before and after the costly parsing stage. The parsing itself is also least resource-intense thanks to reasonably reducing search spaces and restricting to mostly CFG parsing. Fig. 1.1, a flowchart, gives an overview of the parsing architecture.

We will now illustrate the information flow by sketching the on-line processing of the following example sentence, respectively the off-line processing of Treebank training sentences.

(1) What could rescue the bill would be some quick progress on a bill amending the National Defense Education Act of 1958

Its top-ranked parser output can be seen in figure 1.2.

Tagging The raw text is part-of-speech tagged using a state-of-the-art tagger. We have used LTPos (Mikheev, 1997) and, alternatively, Ratnaparkhi's Maximum Entropy tagger (Ratnaparkhi, 1996) in the beginning. Later, the TreeTagger and also LingPipe were used. Currently, the simplest pipeline uses LT-TTT2, which allows the XML output of LT-TTT2 to be fed directly into the parser.

Almost all current taggers reach accuracy of 95-97% for tagging the Penn Treebank with the Penn Treebank tagset.Only an evaluation on unknown words is reported in Mikheev (1997), but LTPos and its successor LT-TTT2 are popular taggers because they support XML and integrate a chunker, which makes it particularly easy to integrate into a pipeline. Ratnaparkhi (1996) report an accuracy of 96.6%. The tagged text of our example sentence is shown in (2).

(2)
```
What_WP could_MD rescue_VB the_DT bill_NN
would_MD be_VB some_DT quick_JJ progress_NN on_IN
a_DT bill_NN amending_VBG the_DT National_NNP
Defense_NNP Education_NNP Act_NNP of_IN 1958_CD
```

Chunking The tagged text is chunked for verb groups, indicated by parentheses and base NPs indicated by square brackets with LTChunk (Mikheev, 1997). The resulting text is shown in (3).

(3)
```
What_WP (could_MD rescue_VB) [the_DT bill_NN]
(would_MD be_VB) [some_DT quick_JJ progress_NN] on_IN
[a_DT bill_NN] amending_VBG [the_DT National_NNP
Defense_NNP Education_NNP Act_NNP] of_IN 1958_CD
```

Because LTChunk was not robust enough for chunking the whole British National Corpus, we have used Carafe, a conditional random field chunker for that task[1]. Currently, the simplest option is to use LT-TTT2, which integrates tagging and chunking, and whose XML output can be fed directly into the parser.

[1] Carafe is a programming project available at `http://sourceforge.net/projects/carafe/`

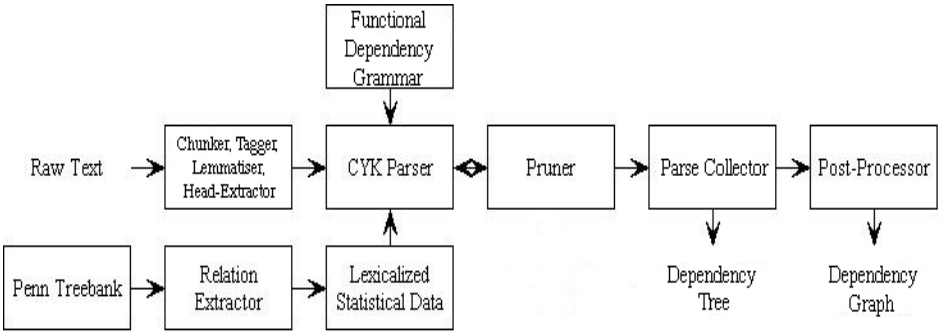


Figure 1.1: Pro3Gres flowchart

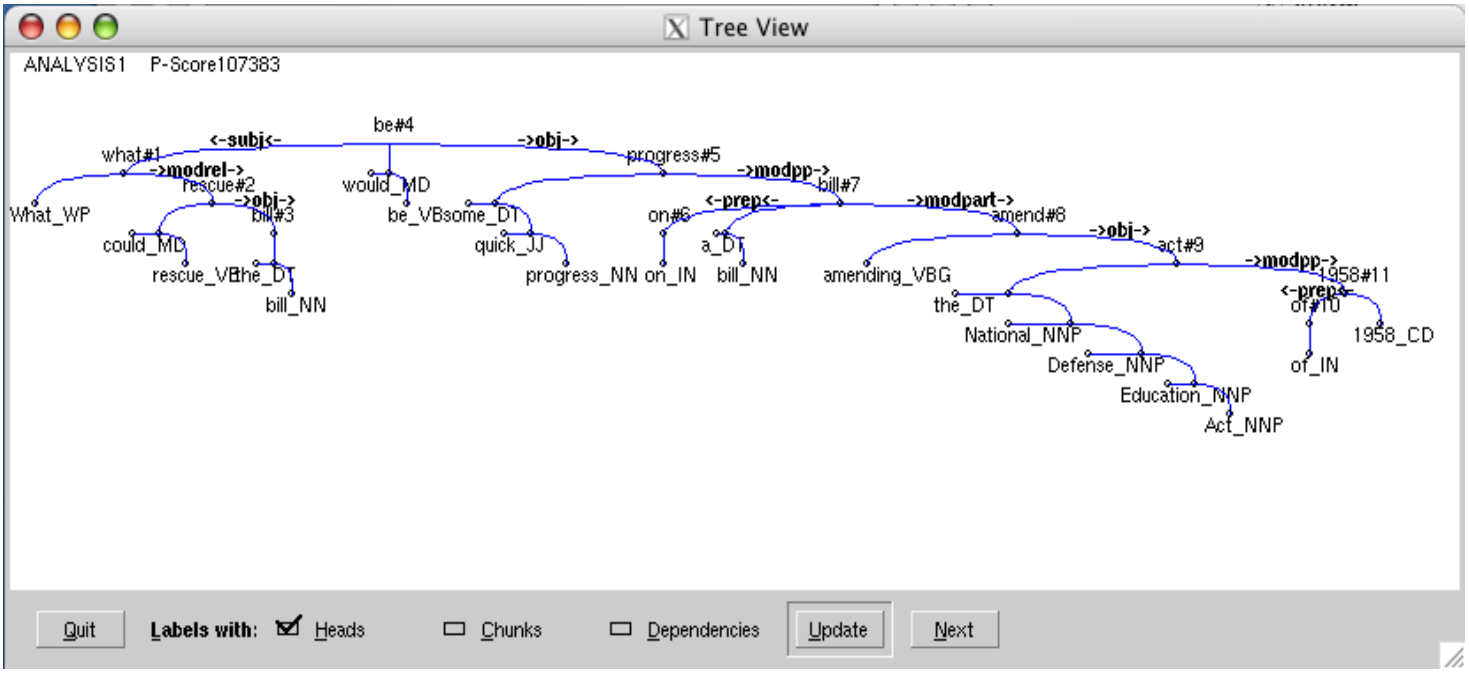


Figure 1.2: Parser output of the sample sentence *What could rescue the bill would be some quick progress on a bill amending the National Defense Education Act of 1958* in stemma notation

Head Extraction The linguistic heads are extracted using our own implementation of Magerman rules (Magerman, 1995). The original chunk and the tag are associated to the head. The linguistic head is the semantic and syntactic core of the chunk. It has frequently been suggested (see e.g. (Abney, 1995; Collins, 1996)) that it is sufficient to parse between heads of chunks. The sentence reduced to (head,tag) pairs is shown in (4).

(4) `What_WP rescue_VB bill_NN be_VB progress_NN on_IN bill_NN amending_VBG Act_NNP of_IN 1958_CD`

Lemmatizing The extracted heads are lemmatised. We use Morpha (Minnen, Carroll, and Pearce, 2000). In LT-TTT2, morpha is integrated.

Tagging, chunking and head-extraction issues are described in the chapter 2.

Functional Dependency Grammar The hand-written grammar is based on part-of-speech tags and on few closed-classed words. For example, the leftmost parsing step, combining `what` and `rescue` with a *relative modification* dependency is licensed by a rule involving closed class words, because it is restricted to only a subset of all WH-pronouns. First, only a subset of WH-pronouns can generally serve as relative pronoun, secondly, the relative dependency here is special since *what* is both the modified noun and the relative pronoun, short for `that which`. This rule is restricted to the `what` WH-pronoun only. The second left parsing step which leads to the analysis in fig. 1.2, the rule that licenses the *object* dependency, is very general and applies to any verb tag followed by any noun tag. We describe our Dependency Grammar in detail in chapter 5.

Parsing A CYK parser (Younger, 1967) is used. CYK is a generalised, all path version of chart-based shift-reduce parsing for CNF grammars. CYK is a bottom-up algorithm, the structure is built up in a breadth-first fashion from the lexical item, level by level.

The algorithm in pseudo-code is as follows (N is the number of chunks in the sentence):

```
for j = 2 to N              # length of span
  for i = 1 to N−j+1        # beginning of span
    for k = i+1 to i+j−1    # separator position
      if Z → XY and X in chart[i to k], Y in chart[k+1 to j]
      and Z not in chart[i to j]
      then insert Z into chart[i to j]
```

At the first level, where the span length j is 2, all combinations between adjacent words are built, as far as they are licensed by the grammar. The combination of A and B, the span from A to B, can either have A as head, (which can be noted as $A(B)$) or B as head (which can be noted as $B(A)$). The following graph illustrates the build-up at level 2 for the first 6 words in the sentence.

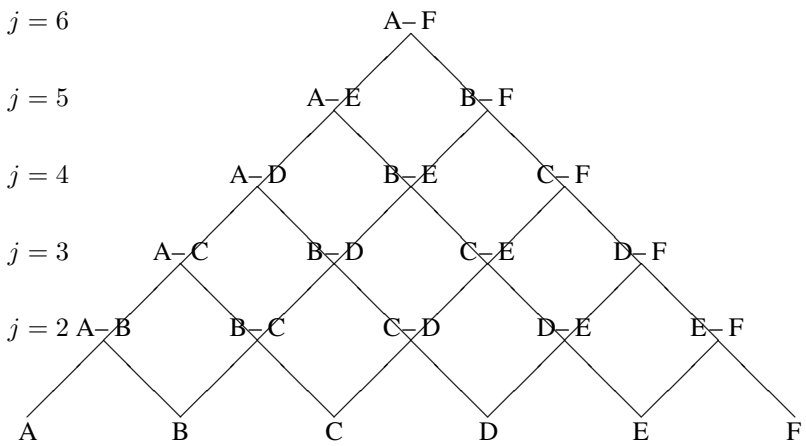
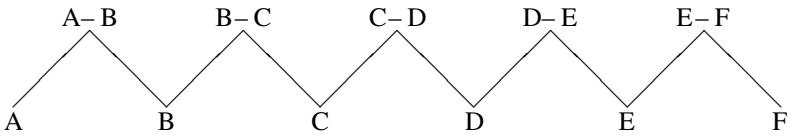


Figure 1.3: Complete CYK chart for an example sentence with 6 terminals



In our example, $A - B$ is $What - rescue$ which can be licensed by a *subject* relation, in which B is the head, $B(A)$ or by a *modification by relative clause* relation, in which A is the head, $A(B)$.

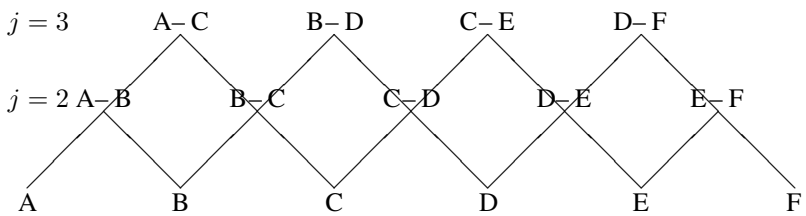
$B - C$ is $rescue - bill$ which can be be licensed by an *object* or an *adjunct* or a *subject* relation, in all cases $rescue$ being the head. The *object* relation rule succeeds and creates a chart entry. Since the inverted *subject* relation rule is restricted to a closed class of verbs, it fails. The adjunct relation rule is restricted to a closed class of dependents, namely temporal expressions, and thus fails.

$C - D$ is $bill - is$ which licenses a *subject* relation. This relation will not lead to a globally connected parse due to the parsing context, but it is possible locally, the rule succeeds and a chart entry is created.

Analogously, structures of span length 2 are constructed for the rest of the sentence.

At the next level, j is 3, structures with span length 3 are constructed. $A - C$ is $What(rescue(bill))$ with a *modification by relative clause* relation, or
$rescue(what)(bill) = rescue(what, bill)$ with a *subject* relation. Analogously, the whole structure is built up, in exactly N levels, where N is the number of words per longest parse span. All the theoretically possible chart entries for the cell $A - C$ are listed in the following illustration.

$$A + (B(C)) = A(B(C)) or B(A, C)$$
$$A + (C(B)) = A(C(B)) or C(A, B)$$
$$C + (A(B)) = C(A(B)) or A(B, C)$$
$$C + (B(A)) = C(B(A)) or B(A, C)$$



The algorithm continues analogously until one or several spans covering the entire sentence are found. In our example, that is at at $j = 6$, because this example has 6 terminals. Figure 1.3 shows the complete chart for this example.

In order to alleviate the overhead of keeping loop variables or executing loops that do not find results, a data-driven, fully declarative version of the CYK algorithm has been implemented, which we desribe in chapter 3.

Relation Extraction Each chart entry is weighted by a lexicalized probability. The frequency counts for the lexicalized statistics are obtained from the Gold Standard, which is the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993; Bies et al., 1995), at an off-line stage, before the parsing starts. Structural patterns expressing grammatical relations are applied to the Gold Standard. For local

relations the patterns are relatively simple, for long-distance dependencies they can be quite complex (see section **??**), spanning a large number of tree generations. Since the lexical heads need to be extracted, all patterns comprise more than one generation.

Let us consider a simple example of a *subject relation*. The first sentence in the Penn Treebank is annotated as follows.

(5)

```
( (S
    (NP-SBJ (NNP Mr.) (NNP Vinken) )
    (VP (VBZ is)
      (NP-PRD
        (NP (NN chairman) )
        (PP (IN of)
          (NP
            (NP (NNP Elsevier) (NNP N.V.) )
            (, ,)
            (NP (DT the) (NNP Dutch) (VBG publishing) (NN group) )))))
    (. .) ))
```

A subject relation holds between an NP with a functional label *SBJ* and its VP sister. In order to extract the lexical head, the extraction pattern needs to comprise the possible subtrees of the NP and the VP. They can both be arbitrarily nested, in our simple example that is not the case, both NP and VP immediately dominate their heads. The subject NP immediately dominates two nouns, however. The structural pattern selects the rightmost noun as the head. It thus reports a count for the (*noun,verb*) pair (*Vinken_NNP, is_VBZ*). The counts are lemmatized after the extraction, and back-off counts with semantic classes, tags, and only a subset of the heads specified are also calculated. We describe relation extraction in section 6.3.

Lexicalized Statistical Data from the Treebank During the build-up of the parse, three sources contribute to the disambiguation: first, the parsing context only allows a small subset of local structures to be combined into a full parse. Second, the analyses are ranked by the product of the probabilities of the parsing decisions. Third, analyses whose products of probabilities fall below a certain threshold are abandoned.

A Maximum Likelihood Estimation (MLE) probability model is used for the second and third source. We use the hash symbol (#) to symbolise frequency. The general Pro3Gres MLE estimation is as follows. We estimate the probability of dependency relation R at distance (in chunks) $dist$, given the lexical head a of the governor and the lexical head b of the dependent. By application of the chain rule we get:

$$P(R, dist|a, b) = P(R|a, b) \cdot P(dist|R, a, b) \cong \frac{\#(R, a, b)}{\#(a, b)} \cdot \frac{\#(R, dist, a, b)}{\#R, a, b} \tag{1.1}$$

We then take the assumption that the distance depends only on the relation type, but not on the lexical items. We have observed that some relations, for example the subordinating clause relation *sentobj* or the PP-attachment relations *modpp* and *pobj* can span many chunks, while for example in the object relation *obj* the object noun is almost always immediately adjacent to its governing verb chunk. We have also observed that there is only little variation based on lexical differences, so that including them would considerably increase the sparseness of the data at probably very little benefit.

$$P(R, dist|a, b) \cong P(R|a, b) \cdot P(dist|R) \cong \frac{\#(R, a, b)}{\#(a, b)} \cdot \frac{\#(R, dist)}{\#R} \tag{1.2}$$

Some relations use variations of this general estimation rule. Let us look at the ending of our example sentence, ... *a bill amending the National Defence Education Act of 1958*. The final PP, which is introduced by *of*, can syntactically be attached to 5 positions: *be*, *progress*, *bill*, *amend*, or *act*. For PP-attachment, a variation on the above MLE estimation, following Collins and Brooks (1995), is used. It includes the PP-internal noun (we will refer to it as description noun). a is the head (a verb or a subject), b the preposition and c the description noun, R is labelled $modpp$ for noun-attachment and $pobj$ for verb-attachment.

$$P(R, dist|a, b) \cong P(R|a, b) \cdot P(dist|R) \cong \frac{\#(R, a, b, c)}{\#(a, b, c)} \cdot \frac{\#(R, dist)}{\#R} \tag{1.3}$$

The co-occurrence count in the denominator expresses the sum of attachment cases and attachment candidates (i.e. syntactically licensed, potential attachments). Because potential attachment interacts with the rest of the parsing progress in complex ways it needs to be approximated. Following (Collins and Brooks, 1995) we model PP-attachment as 2-way ambiguous between noun- and verb-attachment[2]. As regards our above example, where attachment is 5-way ambiguous, that means that all 5 attachment possibilities are never compared directly. The denominator generally expresses the sum of competing relations. The MLE estimations for $pobj$ and $modpp$ are thus the distance factor times all attachment cases divided by all attachment cases plus all cases where the attachment went to any competing relation.

$$P(pobj, dist|verb, prep, desc.noun) \cong \frac{\#(pobj, verb, prep, desc.noun)}{\#(verb, prep, desc.noun)} \cdot \frac{\#(pobj, dist)}{\#pobj} \tag{1.4}$$

$$P(modpp, dist|noun, prep, desc.noun) \cong \frac{\#(modpp, noun, prep, desc.noun)}{\#(noun, prep, desc.noun)} \cdot \frac{\#(modpp, dist)}{\#modpp} \tag{1.5}$$

[2]There are other approximations, e.g. (Collins, 1996) uses the entire sentence as a window. Merlo, Crocker, and Berthouzoz (1997) model 3-way ambiguous situations

Since we model PP-attachment as ambiguous between verbal and nominal attachment (discussed in more detail in section 6), we make the approximation that the competing relations are only *pobj* and *modpp*, hence that the denominator is the sum of all nominal plus all verbal attachments given the lexical items. This approximation is appropriate, because PPs can usually only attach to nouns (*modpp*), to verbs (*pobj*) or to predicative adjectives (which is subsumed under the *pobj* relation). If we based our counts on all 2-way ambiguous verb-noun-pp sequences (Collins and Brooks, 1995) we could get the following probabilities.

$$P(pobj, dist|verb, noun, prep, desc.noun) \cong \frac{\#(pobj, verb, noun, prep, desc.noun)}{\#(pobj, verb, noun, prep, desc.noun) + \#(modpp, verb, noun, prep, desc.noun)} \cdot \frac{\#(pobj, dist)}{\#pobj}$$

$$P(modpp, dist|verb, noun, prep, desc.noun) \cong \frac{\#(modpp, verb, noun, prep, desc.noun)}{\#(modpp, verb, noun, prep, desc.noun) + \#(pobj, verb, noun, prep, desc.noun)} \cdot \frac{\#(modpp, dist)}{\#modpp}$$

We need a slightly different model for three reasons. First, this model only counts occurrences where a PP appears in an ambiguous position in the text. We would like to profit from the many cases in which informative PP-attachments in the Gold Standard do not appear in ambiguous positions, for example a sentence initial noun attaching an immediately following PP, or a verb attaching the immediately following PP. The latter case includes intransitive verbs, which are only in an ambiguous position if they are followed by an adjunct noun. Second, the parser needs attachment probabilities, for 2-way ambiguous, unambiguous, or multi-way ambiguous attachments alike. Third, when an attachment probability during parsing needs to be assigned, the analysis we have of the sentence is very incomplete and does not know which other attachment possibilities there will be. Let us look at the parsing situation where *act* and *of 1958* are about to be combined, when the probability of the noun attachment with *act* as governor and *of 1958* as dependent needs to be assigned. At this stage, there is no certain way of knowing that *act* will in fact be in competition with *amend*, *bill* and *progress* over attaching *of 1958*, nor that it will not be in competition with *what* and the first occurrence of the word *bill* in the sentence. An object dependency from *rescue* to the first occurrence of *bill* rendering it inaccessible as potential governor for *of 1958* will exist in the chart at this stage, but we cannot know if it will feature in the highest ranked analysis. These intricate interdependencies arising from the parsing context are very hard to estimate.

We have therefore decided to model PP-attachment as generally ambiguous between noun attachment and verb attachment (the latter including adjective attachment), that is to use the putative parsing context of Collins and Brooks (1995) as an approximation, where every verb is in competition with one noun, and every noun is in competition with one verb. The actual competitions during parse time are never in direct comparison, but indirectly via the comparison of the putative parsing context.

Generally, an MLE probability is the result of the positive counts divided by the candidate counts. For our PP-attachment model, positive counts are cases that do attach in the Gold Standard, and candidate counts are cases that do attach in the Gold Standard *plus* cases that could attach but that do not, according to the putative parsing context. For verb attachment, then, candidate cases are all cases where attachment as *pobj* occurs, *plus* all cases where in the ambiguous context of a verb-noun-PP sequence the PP attaches to the noun (the label will be *modpp*).

$$P(pobj, dist|verb, prep, desc.noun) \cong \frac{\#(pobj, verb, prep, desc.noun)}{\#(pobj, verb, prep, desc.noun) + \#(modpp, verb, \sum(noun), prep, desc.noun)} \cdot \frac{\#(pobj, dist)}{\#pobj} \tag{1.6}$$

$$P(modpp, dist|noun, prep, desc.noun) \cong \frac{\#(modpp, noun, prep, desc.noun)}{\#(modpp, noun, prep, desc.noun) + \#(pobj, \sum(verb), noun, prep, desc.noun)} \cdot \frac{\#(modpp, dist)}{\#modpp} \tag{1.7}$$

The counts are backed off across several levels, following Merlo and Esteve Ferrer (2006) by including semantic classes, the WordNet lexicographer file ID for nouns and the Levin top class for verbs. The probabilities used for the attachment of the PP *of 1958* in the example sentence are as follows.

Probability of attachment of the PP *of 1958* to the noun *bill*

$$P(modpp, dist|noun, prep, desc.noun) \cong \frac{\#(modpp, bill, of, Class18) = 8}{\#(modpp, bill, of, Class18) + \#(pobj, \sum(verb), bill, of, Class18) = 8} \cdot \frac{\#(modpp, dist) = 677}{\#modpp = 53591} = 0.0126$$

Probability of attachment of the PP *of 1958* to the verb *amend*

$$P(pobj, dist|verb, prep, desc.noun) \cong \frac{\#(pobj, \sum(verb), of, \sum(noun)) = 420}{\#(pobj, \sum(verb), of, \sum(noun)) + \#(modpp, verb, \sum(noun), of, desc.noun) = 5586} \cdot \frac{\#(pobj, dist) = 25650}{\#pobj = 46124} = 0.0418$$

Probability of attachment of the PP *of 1958* to the noun *act*

$$P(modpp|noun, prep, desc.noun) \cong$$
$$\frac{\#(modpp, act, of, Class18) = 14}{\#(modpp, act, of, Class18) + \#(pobj, \sum(verb), act, of, Class18) = 14} \cdot \frac{\#(modpp, dist) = 11918}{\#modpp = 53591} = 0.2224$$

As we have mentioned, there is no direct competition between different possible noun-PP attachments in this model. For example *bill* and *act* are not in direct competition in this model, their probabilities differ due to the distance and their relative probabilities when in competition to verb attachment.

We have now explained the probability for a single parsing decision, equalling a single attachment. The probability of a tree for a given sentence is the product of all the single decisions that build up the tree. As can be seen in the triangular chart that is built up by the CYK algorithm (see figure 1.3), the number of chart entries m is related to the number of terminals n as follows.

$$m = \frac{(n-1)^2 + (n-1)}{2} \tag{1.8}$$

The probability of a tree T given a sentence S with n terminals is therefore:

$$P(T|S) = \prod_{i=1}^{m} P(R_i, dist_i|a_i, b_i) \tag{1.9}$$

We estimate the probability of a tree given a sentence. We thus use a discriminative, not a generative model (Johnson, 2001).

MLE is not the statistical method leading to the best possible results. However, Bikel (2004, 109 ff) points out that using advanced statistical methods does often not improve performance considerably. He only reports a 1 % increase in parsing performance on a Collins (1999) parser when replacing the MLE model by a Maximum Entropy model. We have therefore decided to investigate into a sophisticated back-off model rather than into alternative statistical methods. We describe the statistical disambiguation in more detail in chapter 6.

Pruner In complex real-world sentences, constructing all possible chart entries can become very time-consuming. It has been shown (see e.g. Brants and Crocker (2000)) that discarding locally very improbable partial analyses hardly affects a parser's performance, because the chance that locally very improbable analyses become parts of the most probable analysis later is very small. Pruning happens during parsing, as indicated by the double arrow in figure 1.1. Pruning is a standard procedure, used in all beam parsers (Ratnaparkhi, 1997; Henderson, 2003).

We use the following three pruning methods: hard local cut, fixed beam pruning, and large chart panic mode. The hard local cut method rules out local attachments that are very improbable. As a default, all attachments that have a probability below 1 % are immediately cut. The fixed beam pruning method restrict the number of chart entries for each span. As a default value for robust, large-scale parsing we use 5. The large chart panic mode is used to cope with very complex real-world sentences. It only has an effect in a small minority of real-world sentences. When the total number of chart entries exceeds a certain threshold (we use 1000 as a default) the value used for the hard local cut is increased according to a heuristic function which takes the span length of the chart entries and the total number of chart entries into account. The large beam panic mode entails that in very complex sentences some permissible spans are never found, but it allows the parser to deliver a set of long partial analyses for every sentence. The largest sentence we have encountered in the British National Corpus consists of over 200 chunks. We describe pruning in chapter 4.

Parse Collection In case no complete span for an input sentence can be found, the parser needs to resorts to collecting partial parses. Starting from the most probable longest span, recursively the most probable longest span to left and right is collected. There are two reasons why no complete span can be found in some sentences. Either it contains a (rare, highly marked or potentially unacceptable) construction that is not covered by our grammar, or the pruner prevents a possible complete structure from being created. We describe parse collection in chapter 4.

Post-Processing After parsing, a post-processing module converts the dependency tree into a graph structure which contains additional, deep-syntactic relations, so-called long-distance dependencies. We give a short overview of long-distance dependencies in section **??**.

1.2 Structure of the Code

This section describes how the individual predicates call each other. It is an extremely simplified version of a trace. Only high-level predicates are described.

xml_to_p3g_parse(XMLFile) :: TOP CALL for XML format input. See chapter 2.
+ xml_to_prolog(XML) :: converts to Prolog format. See chapter 2.
+ collect_sents :: treats each input sentence individually. Almost the TOP CALL for Prolog-formatted input. See chapter 2.
++ sparse/6 :: initialises the chart, creating a chart entry for each chunk. See chapter 3.
+ + + intrachunk0/4 :: outputs chunk-internal relations. Described in chapter 7

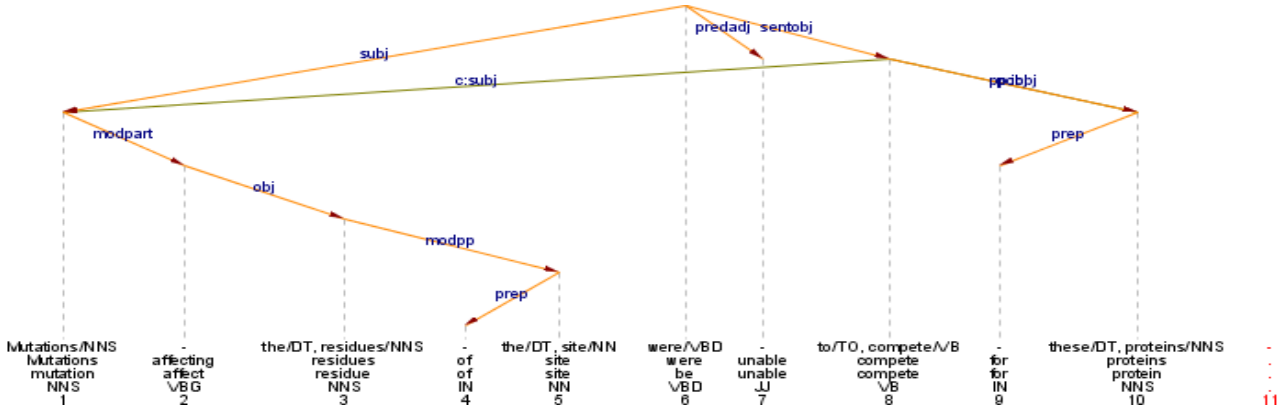


Figure 1.4: Parser output

+ + + nextlevel(1,MAX) :: recurses the CYK levels, until there is nothing left to be reduced. Described in chapter 3.
+ + ++ sparse/9 :: the main parsing predicate: foreach two adjacent chart entries, checks if there are head/7 rules and creates new chart entries if rules match. Described in chapter 3.
+ + + + + head/7 :: the grammar rules, described in chapter 5.
+ + + + + stats/12 :: lexical probabilistic statistics, described in chapter 6.
+ + ++ prune/3 :: pruning, described in chapter 4.
++ pathfinder/0 :: collect and output partial parses. Described in chapter 4.

This technical report should enable readers to get a rough understanding of the Prolog code when browsing it. Given the brevity of the report, more than a shallow understanding is beyond reach. The most important files of the parser are

- Saar/6566_stanford.pl %% current version, 0.6566 of the main program. Currently about 4,300 lines. The word after the underscore is a mnemonic
- Saar/grammar_v1_6.pl %% current version, 1.6 of the hand-written grammar. Currently about 1,800 lines.
- intrachunk0.2.pl %% current version, 0.2 of the iuntrachunk rules. Currently 350 lines.
- Pro3Gres_parameters.pl %% current parameter settings, e.g. for the CYK algorithm. pruning, statistical backoffs.
- results_from_WSJ_morph_nov03/* %% statistical data from the Penn Treebank. 102 MB, ca. 2,240,000 lines
- results_from_BNC/* %% statistical data from the BNC, mainly used for chunk-internal relations. 211 MB, ca. 4,380,000 lines
- NomBank/* %% Nominalisation lexicon
- GENIA2/* %% N-gram counts form the Genia corpus. Used for relational nouns
- TTT2/* %% Development data and code for the integration of LT-TTT2
- results_from_wordnet/* %% development of statistical data from Wordnet, used for the semantic statistical backoffs.
- perl/* Various conversion scripts
- EVAL-Carroll/* %% The GREVAL evaluation corpus and evaluation routines

Chapter 2: Pre-Processing

The Pro3Gres parser needs tagged, lemmatized, chunked, and head-extracted input in Prolog format, or in XML.

2.1 Prolog input format

An example of the Prolog input format, for the sentence , is:

```
w('failure' ,'NN',['Failure_NN'],'Failure').
w('do' ,'VB',['to_TO','do_VB'],'do').
w('this' ,'RB',['this_RB'],_).
w('continue' ,'VB',['will_MD','continue_VB'],'continue').
w('place' ,'VB',['to_TO','place_VB'],'place').
w('burden' ,'NN',['a_DT','disproportionate_JJ','burden_NN'],'burden').
w('on' ,'IN',['on_IN'],'on').
w('taxpayer' ,'NNS',['Fulton_NNP','taxpayers_NNS'],'Fulton taxpayers').
w('s' ,'S',['S_S'],_).
```

The Prolog input format can be created in several ways. We have generated it using LTPos, morpha, and LTChunk in early pipelines. The input is a sequence of $w/4$ predicates, sentence boundaries need to be indicated as shown in the last w predicate in the example. Each $w/4$ predicate contains one chunk, in the order of the original text. The four arguments need to contain the followowing:

1. The lemma of the chunk head.
2. The tag of the chunk head.
3. the chunk, as a list of word_tag pairs.
4. The content of the fourth argument is typically ignored, although versions in which it contains technical information, for example word number, head position number, non-head lemmas, morphosyntactic features, etc. exist, for example the Prolog format which is generated automatically from the LT-TTT2 XML input which we will discuss in the following.

2.2 LT-TTT2 XML input format

An example of the XML input format, for the sentence *Since 1958, 13 labour life peers and peeresses have been created*, is:

```
<p><s id="s9"><pg><w pws="yes" id="w772" p="IN">Since</w></pg>
<ng><w pws="yes" id="w778" p="CD" headn="yes">1958</w></ng>
<w pws="no" id="w782" p=",">,</w> <ng><w pws="yes" id="w784" p="CD">13</w>
 <w pws="yes" id="w787" p="JJ">labour</w>
 <w l="life" pws="yes" id="w794" p="NN" headn="yes">life</w>
 <w l="peer" pws="yes" id="w799" p="NNS" headn="yes">peers</w></ng>
 <w pws="yes" id="w805" p="CC">and</w> <ng>
 <w l="peeress" pws="yes" id="w809" p="NNS" headn="yes">peeresses</w></ng>
 <vg tense="presorbase" voice="pass" asp="perf" modal="no">
 <w l="have" pws="yes" id="w819" p="VBP">have</w>
 <w l="be" pws="yes" id="w824" p="VBN">been</w>
 <w l="create" pws="yes" id="w829" p="VBN" headv="yes">created</w></vg>
 <w sb="true" pws="no" id="w836" p=".">.</w></s></p>
```

The XML format is most easily created by applying LT-TTT2 to raw text, without running the named-entity recognition module. The XML needs to be in the format which LT-TTT2 delivers. A version of the LT-TTT2 script which delivers the correct format is e.g. (ttt.sh, see chapter 8):

```
#!/bin/bash

#Tokenise and POS tag

/usr/local/TTT2/scripts/preparetxt < $1 > $2.prep
/usr/local/TTT2/scripts/tokenise < $2.prep > $2.tok

/usr/local/TTT2/bin/lxconvert -w -q s -s /usr/local/TTT2/lib/postag/pos.cnv $2.tok |
  /usr/local/candc-1.00/bin/pos -model /usr/local/TTT2/models/pos 2>/dev/null |
  /usr/local/TTT2/bin/lxconvert -r -q s -s /usr/local/TTT2/lib/postag/pos.cnv -x $2.tok > $2.pos

# Lemmatize
/usr/local/TTT2/bin/lxconvert -w -q w -s /usr/local/TTT2/lib/lemmatise/lemmatise.cnv $2.pos |
  /usr/local/morph/morpha.ppc_darwin -f /usr/local/TTT2/lib/lemmatise/verbstem.list |
  /usr/local/TTT2/bin/lxconvert -r -q w -s /usr/local/TTT2/lib/lemmatise/lemmatise.cnv -x $2.pos > $2.lem

# chunk
/usr/local/TTT2/scripts/chunk -s nested -f inline < $2.lem > $2.chunk
```

Pro3Gres first converts the XML input into the Prolog input format previously described, and then parses it. The predicate that converts is `xml_to_p3g/1`, the predicate which parses and then converts is `xml_to_p3g_parse/1`, the argument is the XML LT-TTT2 chunked file. The Prolog input format which is produced automatically is e.g.

```
sid(s9).
w('Since','IN',['Since_IN'],[1,0,w772,[pws=yes,id=w772,p='IN']]).
w('1958','CD',['1958_CD'],[2,0,w778,['1958_CD'='1958']]).
w(c,'COMMA',[c_COMMA],[3,0,w782,[pws=no,id=w782,p= (',')]]).
w(peer,'NNS',['13_CD',labour_JJ,life_NN,peers_NNS],[4,3,w799,[peers_NNS=peer,life_NN=life,labour_JJ=labour,'13_CD'='13']]).
w(and,'CC',[and_CC],[5,0,w805,[pws=yes,id=w805,p='CC']]).
w(peeress,'NNS',[peeresses_NNS],[6,0,w809,[peeresses_NNS=peeress]]).
w(create,'VBN',[have_VBP,been_VBN,created_VBN],[7,2,w829,[created_VBN=create,been_VBN=be,have_VBP=have,tense=presorbase,voice=pass,asp=perf,modal=no]]).
w('.','.',['._.'],[8,0,w836,[sb=true,pws=no,id=w836,p='.']]).
w(s,'S',['S_S'],[9,0,s9,[id=s9]]).
```

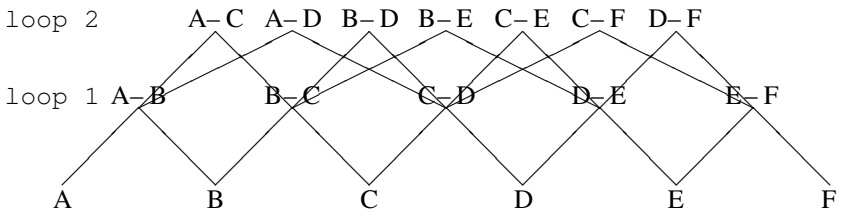
It is a version of the Prolog input format in which the fourth argument contains technical information: chunk number, head position number, non-head lemmas and morphosyntactic features. These features are pipe through and output during the parsing.

Chapter 3: The CYK Parser

3.1 The Pro3Gres chart-driven CYK implementation

We have discussed the standard CYK algorithm in chapter 1. We use a version of the algorithm that is largely driven by the chart data. The gist of the algorithm is as follows: Foreach 2 adjacent chart entries, combine them to a new chart entry if any grammar rule allows that. Repeat this until you cannot create new chart entries any more. The pseudo-code of the "chartdata-driven" CYK implementation is as follows:

1. Add all terminals to chart
2. Loop: `foreach chart entry` $X\lambda i.\lambda k.[i-k]$
   `foreach chart entry` $Y\lambda j.[k-j]$ `# adjacent`
   `if` $\neg$ `tried(`X,Y`)`
   `foreach` $Z \rightarrow X,Y$ `assert` $Z[i-j]$ `to chart (for next Loop)`
   `assert tried(`X,Y`)`
3. Loop: for each pair of adjacent but untried chart entries, check if there is a grammar rule. Add to table of tried chart pairs. If successful, create new chart entry for next loop.
4. If any rule was successful, prune and then Loop again, else terminate.



3.2 Simplified CYK Parser

A simple Prolog version of the actual code, from an early version of the parser, is:

```
nextlevel(L) :-
  chart( FID,[FPos,Ffrom-Fto,FScore],[[F,Ftag,FType]],FuncF),
  % foreach chart entry (by backtrack)
  Gto is Ffrom-1,
  chart( GID,[GPos,Gfrom-Gto,GScore],[[G,Gtag,GType]],FuncG),
  % find adjacent chart entries
  bagof(_,sparse(FID,[FPos,Ffrom-Fto,FScore],[[F,Ftag,FType]],FuncF,
                 GID,[GPos,Gfrom-Gto,GScore],[[G,Gtag,GType]],FuncG),_),
                 % parse (non-recursive, chart version)
  write('End of Level '), write(L), write(' reduced items : '),
  perlevel(X), write(X), X > 0, % only recurse if successful
  prune(L,X),
  L1 is L+1,
  retractall(perlevel(_)), assert(perlevel(0)),
  nextlevel(L1).

nextlevel(_).
```

The predicate nextlevel/1 recurses until no new chart entries can be created. and then succeeds (last line, second version of nextlevel/1). Chart entries in this example consist of 4 arguments, e.g.

`chart( FID,[FPos,Ffrom-Fto,FScore],[[F,Ftag,FType]],FuncF)` where

FID: the unique chart ID number, simply incrtemented for each chart entry.
Fpos: the position of the head word in the sentence.
Ffrom-Fto: The span from which position to which position in the sentence. Positions are measured in chunks, not in words. In the initialised chart only consisting of chunks, i.e. before parsing has started, Fpos=Ffrom=Fto.
FScore: The probabiity score of this chart entry.
F: the head word.
Ftag: The part-of-speech tag of the head word.
FType: The dependency relation label.
FuncF: the syntactic structure being built.

A simplified version of the sparse predicate is:

```
sparse(FID,[FPos,Ffrom-Fto,FScore],[[F,Ftag,FType]],FuncF,
       GID,[GPos,Gfrom-Gto,GScore],[[G,Gtag,GType]],FuncG) :-
  (tried(FID,GID) -> !, fail; assert(tried(FID,GID))), % already tried
  ... % get various context info for the grammar rules:
  head(Ftag,Gtag,1,Type,Transtag,[FChunk,GChunk,FF,FG,OF,OG],FPos-GPos),
  % rule for LEFT reduction (l)
```

```
  (statschart(Ftag,FF,SF,Gtag,FG,SG,Type,Prob,Percent,Dist,FChunk,OG) -> true ;
    (stats(Ftag,FF,SF,Gtag,FG,SG,Type,Prob,Percent,Dist,FChunk,OG),
     assert(statschart(Ftag,FF,SF,Gtag,FG,SG,Type,Prob,Percent,Dist,FChunk,OG)))),
  (Prob < 0.01 -> fail; true), %% early exclusion
  %% Build Func-Struc:
  ...
  asserta(chart(ID,[[FF,Ftag,FChunk,FID,FScore],[FG,Gtag,GChunk,GID,GScore]],
    [FPos,GPos,Gfrom-Fto,PScore,DLen],[[FF,Transtag,Type,ID]],FuncFTRes,Level)),
  retract(perlevel(X)), X1 is X+1, assert(perlevel(X1)),
  (Prob > 0.98 -> !; true), %% early commitment
  fail.
```

The sparse predicate has several versions. The version shown here is for head/7 rules to the the left (3rd argument), i.e. where the dependent is to the left of the governor. There is also a version to the right, and there are many additional version for treating long-distance dependencies and sentence-initial special constituents.

3.3 Format of current chart entries

The chart entires are more complex in current versions of the parser, than in the simple early example shown above. The chart entries of a parse of the simple sentence *East Providence now has no civil defense program* illustrate the current chart entry format:

```
sent([[providence,'NNP',['East_NNP','Providence_NNP'],'East Providence'],
    [have,'VBZ',[now_RB,has_VBZ],has],
    [program,'NN',[no_DT,civil_JJ,defense_NN,program_NN],program]]).
%% East Providence now has no civil defense program
<PROLOG_INTRACHUNK>
ncmod1('providence#1','east#1',_G1167,'(<-)').
%% stanf [east#1,providence#1,<icdep,_G1248]
ncmod1('have#2','now#2',_G1496,'(<-)').
%% stanf [now#2,have#2,<icdep,_G1541]
detmod1('program#3','no#3',_L1187,'(<-)').
%% stanf [no#3,program#3,<icdep,_G1876]
ncmod1('program#3','civil#3',_G1980,'(<-)').
%% stanf [civil#3,program#3,<icdep,_G2043]
ncmod1('program#3','defense#3',_G2162,'(<-)').
%% stanf [defense#3,program#3,<icdep,_G2225]
</PROLOG_INTRACHUNK>

   Call: (21) spy_me2 ? leap
   Exit: (21) spy_me2 ? leap
analyses(1,24.124396740045043,[],1-3,['have#2'('providence#1'(['East_NNP','Providence_NNP']),'<-subj<-',[now_RB,has_VBZ],'->obj->','program#3'
have#2(providence#1([East_NNP,Providence_NNP]),<-subj<-,[now_RB,has_VBZ],->obj->,program#3([no_DT,civil_JJ,defense_NN,program_NN]))
<PROLOGPREDS 1>
subj('have#2','providence#1',_G1195,'(<-)').
obj('have#2','program#3',_G1361,'(->)').
</PROLOGPREDS>
1 +> 24.124396740045043 :: []
                      have#2
        ________________|________________
       /                |       |       \
 providence#1    <-subj<-   .   ->obj-> program#3
       |                  _|_              |
       |                /   \             |
       .              now_RB .            .
    ___|___                  |        ____|___
   /       \                 |       /        \
East_NNP     .              has_VBZ  no_DT      .
           |                                 ____|____
           |                                /         \
     Providence_NNP                      civil_JJ      .
                                                   ___|__
                                                  /      \
                                              defense_NN   .
                                                           |
                                                           |
                                                      program_NN

========================================================================
   Call: (31) spy_me2 ? abort

[debug]  ?- listing(chart).
:- dynamic chart/8.

chart(6, 1, 3, [
      ['have#2', 'VBZ', [now_RB, has_VBZ], 4, 7.727071383281478],
      ['providence#1', 'NNP', ['East_NNP', 'Providence_NNP'], 1, 1]],
      [2, 1, 1-3, 24.124396740045043, 2],
      [['have#2', 'VBZ', subj, 6]],
      'have#2'('providence#1'(['East_NNP', 'Providence_NNP']), '<-subj<-', [now_RB, has_VBZ], '->obj->', 'program#3'([no_DT, civil_JJ, defense
chart(5, 1, 2, [
      ['have#2', 'VBZ', [now_RB, has_VBZ], 2, 1],
      ['providence#1', 'NNP', ['East_NNP', 'Providence_NNP'], 1, 1]],
      [2, 1, 1-2, 3.122062103922232, 1],
      [['have#2', 'VBZ', subj, 5]],
      'have#2'('providence#1'(['East_NNP', 'Providence_NNP']), '<-subj<-', [now_RB, has_VBZ]), 1).
chart(4, 2, 3, [
      ['program#3', 'NN', [no_DT, civil_JJ, defense_NN, program_NN], 3, 1],
      ['have#2', 'VBZ', [now_RB, has_VBZ], 2, 1]],
```

```
      [2, 3, 2-3, 7.727071383281478, 1],
      [['have#2', 'VBZ', obj, 4]],
      'have#2'([now_RB, has_VBZ], '->obj->', 'program#3'([no_DT, civil_JJ, defense_NN, program_NN])), 1).
chart(3, 3, 3, [
      [program, 'NN', [no_DT, civil_JJ, defense_NN, program_NN], program, 1]
      ],
      [3, 3, 3-3, 1, 1],
      [[program, 'NN', w, 3]],
      'program#3'([no_DT, civil_JJ, defense_NN, program_NN]), 0).
chart(2, 2, 2, [
      [have, 'VBZ', [now_RB, has_VBZ], have, 1]
      ],
      [2, 2, 2-2, 1, 1],
      [[have, 'VBZ', w, 2]],
      'have#2'([now_RB, has_VBZ]), 0).
chart(1, 1, 1, [
       [providence, 'NNP', ['East_NNP', 'Providence_NNP'], providence, 1]
       ],
       [1, 1, 1-1, 1, 1],
       [[providence, 'NNP', w, 1]],
       'providence#1'(['East_NNP', 'Providence_NNP']), 0).

[debug]  ?- displaychart(4).
4
         have#2
     ________|______
    /        |      \
    .   ->obj-> program#3
   _|              |
  /  \             |
now_RB .            .
       |        ____|___
       |       /        \
   has_VBZ   no_DT       .
                    ____|____
                   /         \
                civil_JJ       .
                           ___|__
                          /      \
                      defense_NN   .
                                   |
                                   |
                               program_NN
```

The command displaychart/1allows one to see a tree representation of the syntactic parse of the chart whose ID is given as argument. The structure of chart ID 4 is commented in detail in the following. *Left-combine* is the left chunk, *right-combine* is the right chunk that are combined into *combination*.

```
chart(4, 2, 3, [
      # combination ID, left-combine ID, right-combine ID,
      ['program#3', 'NN', [no_DT, civil_JJ, defense_NN, program_NN], 3, 1], # right-combine
      # [word, tag, chunk, position, score]
      ['have#2', 'VBZ', [now_RB, has_VBZ], 2, 1]], #left-combine
      #[word, tag, chunk, position, score]
      [2, 3, 2-3, 7.727071383281478, 1],
      #[governor-combine ID, dependent-combine ID, span, score, CYK level]
      [['have#2', 'VBZ', obj, 4]], # combination
      #[word,tag,relation,combination ID]
      'have#2'([now_RB, has_VBZ], '->obj->', 'program#3'([no_DT, civil_JJ, defense_NN, program_NN])), 1).
```

3.4 CYK Levels

In its original conception, the parser recurses the nextlevel/1 predicate until no new chart entries can be created. In its current form, it first parses everything between commas (except for enumerations, i.e. commmas that are conjunctions or appositions), then it parses again overstepping commas. The rules between commas have the predicate *head* in the grammar, the rules overstepping commas have the predicate *headc2*.

Chapter 4: Pruning and Partial Parses

In order to keep the search space manageable, pruning is essential. Pruning, but also incomplete grammar or grammatically not well-formed input entail that the longest parse does not necessarily span the entire input sentence. In such cases, the partial parses need to be selected. They are combined with the ad-hoc relation *bridge*.

4.1 Pruning

4.1.1 Hard Local Cut

- Very unlikely local structures rarely form part of the most likely global structure
- If a very unlikely local structure forms part of the correct global structure, it chances of getting among the most likely parses are very low
- Very simple to implement
- Biggest gain in complexity reduction, no partial structures at all are built

```
sparse(FID,[FPos,Ffrom-Fto,FScore],[[F,Ftag,FType]],FuncF,
       GID,[GPos,Gfrom-Gto,GScore],[[G,Gtag,GType]],FuncG) :-
  (tried(FID,GID) -> !, fail; assert(tried(FID,GID))), % already tried
  ... head(Ftag,Gtag,1,Type,Transtag,[FChunk,GChunk,FF,FG,OF,OG],FPos-GPos),
  ... % stats
  (Prob < 0.01 -> fail; true), %% early exclusion
  ... asserta(chart(ID,[[FF,Ftag,FChunk,FID,FScore],[FG,Gtag,GChunk,GID,GScore]],
    [FPos,GPos,Gfrom-Fto,PScore,DLen],[[FF,Transtag,Type,ID]],FuncFTRes,Level)),
  retract(perlevel(X)), X1 is X+1, assert(perlevel(X1)),
  (Prob > 0.98 -> !; true), %% early commitment
  fail.
```

4.1.2 Fixed Beam Pruning

- Keep a maximum amount of readings for every span
- Only chart entries with equal span can be compared

```
prune(L,XFact) :- XFact > 3,
    Beam is 3,
    %% foreach stretch A-Z : only keep 3 most likely spans, if there are at least 3 possibilities
    chart(_,_,[_,_,Ffrom-Fto,_Score,_],_,_,_),
    findall((Score,ID), chart(ID,_,[_,_,Ffrom-Fto,Score,_],_,_,_),List),
    len(List,Len),
    (Len < 4 -> fail ;
      (sort(List,SList),
       Till is Len-Beam,
       prunechart(0,Till,SList),
       fail)).

prune(_,_).
```

4.1.3 Complexity-Dependent Pruning

- Pruning is only useful from a certain level of ambiguity on. Prune more and more strongly
- Again, only chart entries with equal span can be compared

```
prune(L,XFact) :- XFact > 3,
    Div is (XFact/3),
    %% foreach stretch A-Z : discard lowest prob part, if there are at least 3 possibilities
    chart(_,_,[_,_,Ffrom-Fto,_Score,_],_,_,_),
    findall((Score,ID), chart(ID,_,[_,_,Ffrom-Fto,Score,_],_,_,_),List),
    len(List,Len),
    (Len < 4 -> fail ;
      (sort(List,SList),
       Till is Len-(Len/Div),
       prunechart(0,Till,SList),
       fail)).

prune(_,_).
```

Prunechart discards the first 'Till' chart entries

```
prunechart(C,Till,[(_Score,ID)|RList]) :-
    C < Till, !,
    %displaychart(ID),
    retract(chart(ID,_,_,_,_,_)),
    %spy_me2,
```

```
    C1 is C+1,
    prunechart(C1,Till,RList).

prunechart(_,_,_). %eorec
```

Fixed beam pruning and complexitiy-dependent pruning are used as alternatives. Fixed beam pruning performs slightly better and is thus used in the current version.

4.1.4 Large Beam Panic Mode

When there are more than say 1000 chart entries, only promising paths are pursued.

- it is accepted that some permissible spans will never be found
- increasing severity based on span length and beam size

```
...,
(ID>1000 -> (((OPScore / ((Len+(Len**sqrt(2)))+(ID/2))) < 0.01)
    -> (write(' TOO LOW!'),nl,fail) ;
    ... ) % else continue
```

4.2 Partial Parses

Many sentences do not get a full parse, but the chart contains hundreds of partial structures (PS). The following heuristic is used to collect partial parses:
loop {
  take longest PS
  to left loop
  to right loop
  add up scores
}
reverse sort added up scores

The few most complex sentences in BNC consist of 90 chunks and more, typically lists of names. They report so many partial parses that the collection of partial parses can run out of memory or take several days. We therefore call the predicate *pathfinder* with a timeout.

Chapter 5: The Grammar Rules

Pro3Gres is a Formal Grammar parser in many senses, because it follows a formalized and established grammar theory, because it treats the phenomena for which formal grammars were invented, for example long-distance dependencies, and because it analyzes for the entities defined in Formal Grammars such as LFG: deep-syntactic functions. At the same time, it is a robust parser, and it integrates lexicalized statistics obtained from the Penn Treebank. One of the aspects that Pro3Gres shares with many systems based on formal grammars is its use of a hand-written grammar, which we explore in this chapter.

We describe the design principles and the hand-written grammar in detail. It is explained why we have decided to use a hand-written grammar and which governing principles we have followed during the development. The individual rule types are then presented and discussed in detail.

5.1 Introduction

Before the success of probabilistic parsers such as Collins (1999) and Charniak (2000) the use of hand-written grammars was commonplace, in formal grammar based systems it still is. Grammar writing and grammar engineering proved to be a feasible, but very labour-intense and complex task.

> *Grammar writing is much more difficult than rule writing. The intricate interrelations of the individual rules of a grammar make grammar writing a complex and error-prone process, much like computer programming.* (Friedman, 1989, 254)

The majority of classical probabilistic approaches learns the grammar from the corpus, obviating the need for a cumbersome hand-written grammar, but the amount of manual work for annotating a large corpus manually is considerable. Recently, hand-written formal grammars are combined with statistical data (Riezler et al., 2002) or formal grammars are learnt from syntactically annotated corpora (Hockenmaier and Steedman, 2002; Burke et al., 2004; Hockenmaier and Steedman, 2002). Miyao, Ninomiya, and Tsujii (2005) develop an interesting semi-automatic grammar acquisition algorithm. They state the traditional wisdom on hand-written grammars as follows: *"Although a few studies could apply a hand-crafted grammar to a real-world corpus, (Riezler et al., 2002), these required considerable effort that lasted over a decade."* (Miyao, Ninomiya, and Tsujii, 2005, 684). We follow the hand-written option and combine a hand-written rule-based grammar with lexicalized statistical data obtained from the Penn Treebank.

We have experienced that the amount of work needed to write a broad-coverage grammar manually is manageable, as we describe in subsection 5.1.2. The grammar, once written, can be ported without or with only small changes across most domains. What changes between domains is terminology and lexicalization probabilities. Before describing the grammar, we need to introduce the tagset on which it is based: the Penn Treebank tagset. We do so in subsection 5.1.1.

5.1.1 The Penn Treebank

The Penn Treebank is a large collection of syntactically annotated sentences (Marcus, Santorini, and Marcinkiewicz, 1993; Bies et al., 1995). It is annotated with morphosyntactic part-of-speech information, and with syntactic constituency information. The tagset for the part-of-speech annotation is very small, it only uses 36 tags. They are summarised in figure 5.1. Notable idiosyncrasies of this tagset are that the word *to* is never disambiguiated (it can e.g. be a preposition or an infinitive marker), and that no distinction is made between complementizer and preposition. The latter requires disambiguation for parsing.

Functional roles known form LFG and functional DG are only partially annotated. Specifically, subjects are annotated, objects are not, some PPs are functionally or semantically annotated. Functional and semantic tags are summarised in table 5.2. Structurally, all PPs modifying verbs are attached in under the VP, thus appearing as arguments, while all PPs modifying nouns are Chomsky-adjoined. Also the functional annotations do not always deliver argument or adjunct status in a consistent way. For example, the functional label *LOC* is both used for location adjuncts (e.g. *sit on a bench*) and adjuncts (e.g. *a rise in interest rates*). Careful use of empty categories and co-indexation expressing long-distance dependencies is made in the Penn Treebank.

5.1.2 The Difficulty of Writing Grammars

While automatically or semi-automatically acquiring grammars is a very promising approach, we maintain that the expense needed for writing a large-scale dependency grammar over Penn tags is sufficiently small and that grammars are quite domain-independent. The broad-coverage English Functional Dependency grammar described in this chapter was developed in about a person-month.

The Penn tagset consists of 36 tags (plus some punctuation tags). Considering that dependency rules are always binary, that there are about 20 rule types and 2 possible dependency directions, the upper bound – both on rule writing and parsing search space complexity – is about 50,000 rules. In a realistic scenario, assuming on average one direction and relation type per possible tag combination (36^2) we get about 1000 rules. We only need grammar rules containing lexical information for closed classes. For example, a small class of gerunds, such as *including* and *excluding* can serve as a preposition, or a closed list of temporal expressions serving as adjuncts is used.

Count	Tag	Legend
1.	CC	Coordinating conjunction
2.	CD	Cardinal number (N.B. ordinal numbers are adjectives)
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal verb
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer (*such* a good time, *both* the girls)
17.	POS	Possessive ending (John *'s* idea)
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Verbal particle (give *up*)
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner (*which*, relative pronouns)
34.	WP	Wh-pronoun (*what, who, whom*)
35.	WP$	Possessive wh-pronoun (*whose*)
36.	WRB	Wh-adverb (*how, where, why*)

Table 5.1: The Penn Treebank Tagset

Label	Legend	Example or Explanation
Grammatical Functions		
-CLF	true cleft	[S-CLF it was Casey who ...]
-NOM	non NP in NP function	heard of [S-NOM asbestos being dangerous]
-ADV	clausal and NP adverbial	reaches 10,000 barrels [NP-ADV a day]
-LGS	logical subject in passive	done by [NP-LGS the president]
-PRD	non VP predicate	is [NP-PRD a producer]
-SBJ	surface subject	[NP-SBJ Peter] walks.
-TPC	topicalised constituent	[S-TPC-1 I agree, he said [SBAR [S-1]]
-CLR	closely related	“open class of other cases"
Semantic Roles		
-VOC	vocative	Close the door, [NP-VOC John]!
-DIR	direction & trajectory	attention [PP-DIR to the problem]
-LOC	location	declines [PP-LOC in interest rates]
-MNR	manner	happy [PP-MNR like a kid]
-PRP	purpose and reason	[PP-PRP (in order) to ...]
-TMP	temporal phrase	shares rose [NP-TMP yesterday]

Table 5.2: Penn Treebank Functional Labels

For all open word classes, the Penn tags provide enough generalisation to deliver syntactically correct analyses in the vast majority of cases. The task of the lexicalized disambiguation is to select the semantically most convincing among all syntactically possible analyses.

The current Pro3Gres grammar has about 1200 rules. The number of rules may seem high because of tag combinatorics leading to many almost identical rules. A subject relations is e.g. possible between the 6 verb tags and the 4 noun tags. The supplied Pro3Gres grammar does not aim at covering all phenomena of the English language. The decision of which phenomena to exclude depends on "armchair linguistics" intuition, followed by many test cycles to supplement the necessary empirical evidence. This decision also depends on the amount of ambiguity and errors rare rules introduce. The grammar rules contain the dependent's and the head's tag, the direction of the dependency, lexical information for closed class words, and context restrictions. The rules are described in detail in the following section 5.2.

The task of grammar writing is further simplified because we follow the Dependency-based broad architecture suggested by (Abney, 1995), which uses chunking for the base phrase level, thus making it unnecessary to write base NP grammars. This approach naturally integrates chunking and dependency parsing and has proven to be practical, fast and robust (Collins, 1996; Basili and Zanzotto, 2002). Tagging and chunking are robust, finite-state approaches, parsing only occurs between heads of chunks.

The most time-consuming aspect of manual grammar writing is the practice of incremental engineering across a large number of test cycles. We first consulted Quirk et al. (1985), a comprehensive and very carefully written grammar of English. A small set of simple rules is tested first, the missing rules are then added in a large number of development cycles. There is a constant quality feedback from the development set corpus at each development cycle. Rules that are found to create more errors than correct analyses are refined or eventually discarded. We have trained over Penn section 2-22 and used section 0 as development corpus. Evaluations (see Doctoral thesis) have been made on John carroll's 500 sentence test corpus, and 100 random sentences form the GENIA corpus.

5.1.3 Benefits of a Hand-Written Grammar

Writing a grammar manually is more time-consuming than applying a machine learning algorithm to learn the rules from a corpus. Still, there may be potential benefits from writing a grammar manually.

Parsing Questions Sentence types that are underrepresented in a training corpus are difficult to learn. In the Penn Treebank, this is the case for questions, which are crucial for Question Answering applications. (Hermjakob, 2001) shows that question parsing considerably improves if a grammar is enriched with additional question parsing knowledge. In section **??** we show that Pro3Gres is highly capable of parsing both simple and complex questions. Pro3Gres has been employed for question parsing at a TREC conference (Burger and Bayer, 2005).

Correcting Tagging Errors In a hand-written grammar, some typical parsing errors can be corrected by the grammar engineer, or rules can explicitly ignore particularly error-prone distinctions. Examples of rules that can correct tagging errors without introducing many new errors are allowing *VBD* to act as a participle, or the possible translation of *VBG* to an adjective. Most taggers perform poorly in the distinction between verb past tense *VBD* and participle *VBN*. But the distinction can usually be made in the parsing process. Therefore, the grammar leaves this tag distinction underspecified for a number of constructions, for example the *modpart* relation. A second example of ignoring error-prone distinctions is the distinction between prepositions and verbal particles, which are known to be particularly unreliable – and also human inter-annotator agreement is quite low. The grammar has therefore been designed to make no distinction between verbal particles and prepositions.

The Power of Linguistic Constraints Linguistic knowledge allows us to place strong non-local restrictions on the co-occurrence of different relation types. Verbs that have attached adjuncts cannot attach complements, since this would violate X-bar constraints. Verbs that have no object cannot attach secondary objects. The application of dependency rules can often be lexically restricted: for example, only temporal expressions occur as NP adjuncts. We have noticed during the development that these restrictions play a crucial role for the improvement of the parser's performance. History-based parsers learning a sufficiently large number of subtree generations, such as in data-oriented parsing (Bod, Scha, and Sima'an, 2003), inherently learn these linguistic constraints, while we concisely express them in the grammar rules. The fact that the majority of these non-local constraints become local in DG makes it easy to express them in a DG grammar, and reduces search spaces.

Almost Complete Grammar A practical system, such as the one devised in this work, can choose to rule out linguistic constructions that are possible, but very marked and rare, and that often introduce more errors than improving the coverage. For example, while it is generally possible for nouns to be modified by more than one PP, only nouns seen in the Treebank with several PPs are allowed to have several PPs in the best-performing grammars used in Pro3Gres. Or, while it is generally possible for a subject to occur to the immediate right of a verb (*said she*), this is only allowed for verbs seen with a subject to the right in the training corpus, typically verbs of utterance, and only in a comma-delimited or sentence-final context.

Grammar Teaching and Learning A hand-written grammar offers the possibility to the user to inspect, edit and experiment with the grammar. Although this possibility does not give any scientific advantage, it is useful for learning and teaching purposes. On a personal level, manually writing and testing a grammar has considerably improved our knowledge of English grammar.

RELATION	LABEL	EXAMPLE
verb–subject	$subj$	*he sleeps*
verb–direct object	obj	*sees it*
verb–second object	$obj2$	*gave (her) kisses*
verb–adjunct	adj	*ate yesterday*
verb–subord. clause	$sentobj$	*saw (they) came*
verb–pred. adjective	$predadj$	*is ready*
verb–prep. phrase	$pobj$	*slept in bed*
noun–prep. phrase	$modpp$	*draft of paper*
noun–participle	$modpart$	*report written*
noun–preposition	$prep$	*to the house*
verb–complementizer	$compl$	*to eat apples*
noun–gerund-adjective	$adjtrans$	*developing countries*

Table 5.3: The major Pro3Gres dependency types

Structures constrain each other While many non-parsing approaches to grammatical role discovery (e.g. Buchholz (2002)) have proven successful, we wanted to profit from the fact that structures constrain each other. The majority of locally possible structures cannot be combined into a globally possible structure, they can be disambiguated by the parsing context. Linguistic knowledge allows us to place strong restrictions on the co-occurrence of different relation types in our hand-written rules. Verbs that have attached adjuncts cannot attach complements, since this would violate X-bar constraints. Verbs that have no object cannot attach secondary objects. The application of dependency rules can often be lexically restricted: for example, only temporal expressions occur as NP adjuncts. We have noticed during the development of the grammar that these restrictions play a crucial role for the improvement of the parser's performance. Formulating constraints has a huge impact both on the search space and on the accuracy of the parser. Refer to the evaluation section of the doctoral thesis for more details.

5.2 The Rules in Detail

We now describe the rules in detail. Each rule has four arguments and a restriction part. The arguments are the tag of the head, the tag of the dependent, the tag of the projection, and the direction. The tag of the projection is usually identical to the head tag, according to the endocentricity principle, which most head-driven formalisms, for example HPSG, X-bar or Dependency Grammar use. There are a few exceptions, for example a preposition and a noun project into a prepositional phrase, the projection tag is PP. The direction can be left underspecified, like in an ID/LP grammar. The restriction part allows one to place arbitrary restrictions on the application of a rule. Some rules are lexically restricted. For example, there is a postposition rule forming a PP from a preposition and a preceding noun (i.e. the direction of the dependency is to the right), but this rule is restricted to a closed class of words like *ago*. Other rules express non-local restrictions on the co-occurrence of different dependency types. For example, a verb that already has attached an adjunct cannot attach a complement, since this would violate X-bar constraints.

We distinguish between the following three classes of dependency types:

1. Major types: these are the classical dependency types like subject and object. For each of them, statistical data has been extracted from the Penn Treebank, and an appropriate version of our probability model is used. This class consists of the following dependency types: $subj$, obj, $obj2$, adj, $sentobj$, $pobj$, $modpp$, $modpart$, $prep$, $compl$ and $adjtrans$. Examples of these types are found in table 5.3.
2. Minor types: minor types are not probabilistic. They have auxiliary functions, such as building up noun chunks that the chunker has missed. Not all of them can be semantically interpreted. The adverb relation attaches to any noun or verb without aiming at a meaningful interpretation. Some relations are in this class because they give rise to very little ambiguity and thus do not warrant a probabilistic treatment: the $modrel$ relation that attaches relative pronouns and the $predadj$ relation that attaches predicative adjectives.
3. Unconventional types: there are a number of types that are each very different from any other type. Conjunctions, a classical DG problem, are split into two binary rules and leave ambiguities underspecified. WH-rules receive a simple treatment as real long-distance dependencies (see section **??**). Commas have an important function in structuring the sentence and are treated in a special way, as will be discussed in subsection 5.2.4.

5.2.1 Rule Format

```
% left: F is head
head(Ftag,Gtag,l,Type,Transtag,[FChunk,GChunk,FF,FG,OF,OG],FPos-GPos) :-
    restrictions.
% right: G is head
head(Gtag,Ftag,r,Type,Transtag,[FChunk,GChunk,FF,FG,OF,OG],FPos-GPos) :-
    restrictions.
```

```
%% F is always before G
%% Type      relation (subj, obj, etc.)
%% Transtag  =head tag, except PP or translations
%% FChunk    Chunk of F
%% GChunk    Chunk of G
%% FF        lexical head of F
%% FG        lexical head of G

%% OF        dependents of F until now
%% OG        dependents of G until now
%% OF is always FURTHER FRONT : if "l" --> of HEAD, if "r" --> of DEP
%% OG is always FURTHER BACK  : if "l" --> of DEP,  if "r" --> of HEAD
%% FPos-GPos positions
```

5.2.2 Major Types of Grammar Rules

All major verb types are probabilistic. They comprise the relations given in table 5.3.

Subject

There are 145 subject rules, the high number is due to tag combinations: Any verb tag, corresponding to the regular expression *VB[ZPGDN]?* can combine with any noun tag, corresponding to the regular expression *(NN[P]?[S]? or EX or IN or WP or DT or CD or PRP or RB or VBG)*. Some possible combinations are ruled out in order to express agreement constraints: for example, a verb tagged *VBZ* cannot govern a noun tagged *NNS*. Each rule exists in a version with and without a comma between the verb and the subject. The subject relation is the only relation to have two probability models: one for active verbs and one for passive verbs.

General restrictions:

- Maximally one $subj$: The subject candidate is only allowed to attach if the verb does not already have a subject.
- Verb-chunk is not *to* followed by an infinitive: infinitival verbs are assumed not to take subjects. This entails a GPSG-style treatment of control, where the matrix verb takes both a nominal and a clausal object.

Special restrictions:

- Plural nouns: verb chunk does not contain *VBZ*. This is a simple method to ensure agreement.
- if the verb tag is *VBG*: verb chunk length>1. Present participles (gerunds) are not allowed to take a subject.
- if the noun tag is *VBG*: noun chunk length=1, only with copula verb. Gerunds acting as nouns (e.g. *Swimming_VBG is fun*) are only allowed to attach to copular verbs (**Swimming_VBG likes fun)*. The length restriction on gerund chunks is redundant for the current parser.
- Relation to right: only verbs seen in training corpus (in assertive sentences) are allowed to have verb-subject inversion (e.g. *says she* The restriction to seen material is an intermediate step between a rule- and probability-based model. It restricts the application of rare rules to attested cases. This measure can reduce search spaces considerably. The chunk distance is restricted to one, which means that no intervening elements are permitted (**says quickly she*).
- if the noun tag is *EX*: no noun conjunction is allowed. Expletives are not allowed to participate in conjunctions (**there and it seems to be a problem*).
- Noun tag *RB* for pronouns: The pronouns *this, that, it* are tagged *RB*. Only this closed lexical class is permitted to be subject.

Example:

```
head('VBG','NNS',1,subj,'VBG',[FC,_,_,_,UG,OG],_) :-
   \+ member('<-subj<-',UG), \+ member('->subj->',UG),
   \+ ending_member(FC,'_VBZ'), \+len(FC,1), \+ first(FC,'to_TO').
```

This example rule expresses that a verb head in *-ing*-form (*VBG*) can take a plural noun (*NNS*) to its left

- if the verb does not have a subject yet (subcategorization),
- if the verb group does not contain a third person singular verb (agreement),
- if the length of the verb group is longer than 1 (not a lonely participle),
- and if it does not start with the infinitive marker.

Object

There are 78 object rules, the high number is due to tag combinations. Any verb tag, corresponding to the regular expression *VB[ZPGDN]?* can combine with with any noun tag, including lonely determiners, numbers, or WH-words, pronouns, gerunds and symbols, corresponding to the regular expression *NN[P]?[S]? or IN or WP or DT or CD or PRP or VBG or SYM or RB*. Verb-object dependencies are not allowed to overstep commas.

General restrictions:

- Maximally one obj: the 2nd, ditransitive object is labeled $obj2$. This ensures proper subcategorization.
- Verb has no adjunct to the right yet (all objects are closer to verb than adjuncts to the right). According to X-bar theory, case and θ-role assignment, complements need to be closer than adjuncts. In the CYK derivation history, this means that a verb can only attach complements before it attaches adjuncts. There can be rare and marked violations of X-bar theory in real-world language (*?they will ask today John*), which we explicitly intend not to cover in our parsing approach.
- Verb has no $sentobj$ (all objects are closer to verb than subordinate clauses): this assumption is based on the closeness of the object to the verb, and on the observation that considerably longer constituents are usually placed further than short constituents.
- Verb has no $pobj$ (almost all objects are closer to verb than verb-PPs). This assumption, based on the same observations, is occasionally flouted *?they prescribed to John this dangerous stuff that had relieved generations of previous death candidates from unbearable pain*. While missing some readings, the restriction disambiguates for instance *she donated to the poor every Sunday* to an adjunct reading for *every Sunday*
- Verb has no $predadj$ (predicate adjective): verbs cannot have a predicative adjective and an object simultaneously.

Special restriction: noun tag RB for pronouns: The pronouns *this, that, it* are tagged *RB*. Only this closed lexical class is permitted to be object.

Assumptions: In order to facilitate parsing, two assumptions are made:

- The predicate of a copular verb is considered to be obj (also in the lexicalized probability model). This does not mean that the complement of a copular verb (e.g. *president* in *Mary became president*) is really an object, but it allows a uniform syntactic treatment of verbs. Since copular verbs are a closed and unambiguous class, a simple mapping to e.g. a *complement* label would always be possible.
- In a ditransitive verb, obj is the indirect object (also in the lexicalized probability model).

Example:

```
head('VBD','NN',r,obj,'VBD',[_,_,_,_,_,OG],_)  :-
   \+ member('->adj->',OG), \+ member('->sentobj->',OG), \+ member('->obj->',OG),
   \+  member('->pobj->',OG), \+ member('->predadj->',OG).
```

This example rule expresses that a past verb (*VBD*) can take a singular noun *NN* object to its right

- if no adjunct is attached yet (X-bar)
- if no $sentobj$ is attached yet (objects are closer)
- if no object is attached yet (subcategorization)
- if no PP is attached yet (closeness constraint, rarely flouted)
- and if no predicative adjective is attached yet. See the $adjective$ rule.

Object2

There are 24 second object rules. Any verb tag, corresponding to the regular expression *VB[ZPGDN]?* can combine with any noun tag, corresponding to the regular expression *NN[P]?[S]?*. Other $obj2$ tags are rare but possible, the grammar is thus potentially still incomplete. The tags *CD* and *WP* seem possibple, *DT* and *PRP* seem unlikely, as the following examples illustrate.

(6) She gave him 23_CD.

(7) ? She gave him what_WP ?

(8) ?? She gave him this_DT.

(9) * She gave him it_PRP.

The indirect objects in these examples tend to be expressed rather by means of a *to*-PP.

General restrictions:

- Maximally one $obj2$. The second, ditransitive object is labeled $obj2$. This ensures proper subcategorization.
- Verb already has an object.
- Only verbs seen in the training corpus or from a licensing list: this restricts $obj2$ to verbs attested as ditransitive.
- No temporal expression noun: temporal expressions are excluded. This can lead to a few errors but reduces the search space.
- Verb has no adjunct to the right yet (all objects are closer to verb than adjuncts to the right): according to X-bar theory, case and θ-role assignment, complements need to be closer than adjuncts.

- Verb has no $sentobj$ (all objects are closer to verb than subordinate clauses), similar to the object relation.
- Noun has no $modpart$: this heuristic constraint is prone to introduce errors, for instance in *He gave her the roses bought in the shop*, which is rare, but not impossible. It was introduced to correct frequent misanalyses of sentences such as *He reported her the roses sold in the shop*. In other words, a structural constraint that prefers a superordinate zero-complementizer to a subordinate zero-relative has been implemented. While structural soft constraints are not in the spirit of a statistical system, the very successful constraint grammar (CG) approaches rely on elaborate versions of such constraints. Hybrid combinations of such different approaches are a promising field of research.

Example:

```
head('VBN','NNS',r,obj2,'VBN',[_,_,N,V,_,OG],_) :-
   (vpnpnpo_bak_verb(V,_);obj2except(V)), \+tmp(N), \+ member('->adj->',OG),
   \+ member('->sentobj->',OG), members('->obj->',OG,1), \+  member('->pobj->',OG),
   \+  member('->obj2->',OG), \+  member('->modpart->',OG).
```

Adjunct

There are 50 adjunct rules. Any verb tag, corresponding to the regular expression *VB[ZPGDN]?* can combine with any noun tag, corresponding to the regular expression *NN[P]?[S]?*. Adjunct relations are allowed to overstep commas.

General restrictions:

- Only nouns seen in the training corpus or from a list of temporal expressions. Temporal expressions are a closed class. This step greatly reduces the search space.
- Noun has no $modpart$ relation.
- Verb has no $sentobj$: this heuristic constraint forces a low attachment in (*stocks were said [to rise] Friday*).

Special restriction: The direction of the adjunct relation is possible to the left only if the verb has subject (*Friday they said ...*)

Example:

```
head('VBD','NNS',r,adj,'VBD',[_,_,N,_,UG,OG],_) :-
   tmp(N), \+ member('->sentobj->',OG), \+ member('->modpart->',UG).
```

Sentobj

There are 130 sentential object rules. Any verb or noun tag, corresponding to the regular expression *VB[ZPGDN]? or NN[P]?[S]?* can combine with any verb tag, corresponding to the regular expression *VB[ZPGDN]?*. $sentobj$ relations are allowed to overstep commas.

General restrictions:

- Maximally one $sentobj$: this ensures proper subcategorization.
- The subordinate clause is generally required to have a subject. This constraint is however subject to the following special restrictions.

Special restrictions:

- Infinite subordinate clauses with *to* are allowed not to have a subject: these are typically control structures, which get their object at the post-parsing predicate-argument stage.
- Comma-involving $sentobj$ requires the subordinate verb to have a complementizer or the superordinate verb to be a verb of utterance: this helps us to restrict ambiguities arising from zero-complementizers. Zero-complementizers are rare if there is a comma between the matrix and the subordinate clause, except when the matrix verb is a verb of utterance.

  (10) She said, the winners have arrived.

  (11) ?She believed, the winners have arrived.
- Nouns can have subordinate clauses, but only subjectless ones, and only if the noun was seen in the training corpus. These are mostly relational nouns, e.g. *tendency to go*. The fact that they need to be attested greatly reduces the search space and eliminates many incorrect analyses.

Pobj

There are 30 verb-PP attachment rules. Any verb or adjective tag, corresponding to the regular expression *JJ or VB[ZPGDN]?* can combine with any prepositional phrase, corresponding to the regular expression *PP*. $pobj$ is allowed to overstep commas. Attachment to left is allowed if a PP is fronted to the beginning of a sentence.

General restrictions:

- No verb-PP attachment if there is a predicative adjective (**she grew [tired] from too much walking, she grew [tired from too much walking]*)
- Unlimited number of PPs possible: no distinction between PP-arguments and adjuncts is made, an unlimited number of PP-attachments is thus allowed.

Special restrictions:

- *According* as verb is disallowed, because it is analysed as a preposition.
- Adjective-PP attachment is restricted to small distances only.
- The verb *be* is not allowed to attach PPs unless it has no object: this heuristic was introduced to correct a large number of incorrect analyses.

Example:

```
head('VBN','PP',r,pobj,'VBN',[_,_,_,V,_,OG],_)  :-
   \+ member('->predadj->',OG), (V='be' -> \+ member('->obj->',OG) ; true).
```

Modpp

There are 10 noun-PP attachment rules. Any noun tag, corresponding to the regular expression *NN[P]?[S]?* can combine with any prepositional phrase, corresponding to the regular expression *PP*. $modpp$ is allowed to overstep commas.

General restrictions:

- Noun has no $modrel$: this restriction does not allow a full analysis of *she has a problem [which annoys her] with computers*, but it eliminates attachment ambiguities in *she has a problem [which annoys people with computers]*.
- Only relational nouns are allowed to have more than one PP (relational noun recognition is approximated by Wordnet). This heuristic is a rather crude approximation. It leads to errors with nouns that have several adjunct PPs. But we have empirically seen on the development corpus that the error rate increases considerably if we eliminate this restriction.

Example:

```
head('NNP','PP',r,modpp,'NNP',[FC,_,_,N,UG,OG],_) :-
   (\+member('->modpp->',OG);nountoclass(N,_,'04'), \+ member('->modrel->',OG)),
   \+member('->adjective->',OG), (FC=as([as_IN]) -> (member(UG,'->modpart->'),
   fail); true).
```

The last constraint in this example excludes noun-PP attachment for the preposition *as* if the description noun is modified by a participle – thus giving preference to verb attachment in these cases.

They [consider the president as a man acting_VBG in his own interest only].
They [consider the president as a man lost_VBN in the sea].
They consider [the person as a whole] when judging. While this piece of crude heuristics improved performance on the development corpus, its status is not beyond question, and it may well be removed again.

Modpart

There are 30 modification by participle rules. Any noun tag, corresponding to the regular expression *NN[P]?[S]?* can combine with gerunds, corresponding to the regular expression *VB[NDG]*. $modpart$ is allowed to overstep commas. VBD is included (because this is a very frequent tagging error), in order to exclude at least the perfect tense under the restriction that it is the first element in the verb chunk (unless an adverb precedes).

General restrictions:

- Maximally one $modpart$ per noun.
- Verb has no subject.
- Noun has no apposition ($appos$): a heuristic that rules out possible but rare readings: *the report, 60 pages, issued yesterday, shows that ...* but considerably increases performance on the development corpus.
- Verb has no object, unless it is an *appoint class* verb.
- Noun is no temporal expression.
- The distance is very short.

Special restrictions:

- If the tag is VBD then the verb chunk is checked in order to exclude verbs that are not participles.

Example:

```
head('NN','VBD',r,modpart,'NN',[FC,_,W,_,UG,OG],F-G)   :-
  \+ W='be', D is F-G, D=<3, \+ ending_member(FC,'_VBZ'),
  (first(FC,'_VBD');first(FC,'_RB')),(\+ member('->obj->',UG);obj2except(W)),
  \+ member('->modpart->',UG), \+ member('->modpart->',OG),
  \+ member('->appos->',OG), \+ member('<-subj<-',UG), \+ member('->subj->',UG).
```

Prep

There are 72 preposition rules. Any noun tag, including lonely determiners, adjectives, numbers, symbols, or PPs, corresponding to the regular expression *NN[P]?[S]? or PP or DT or CD or SYM or JJ[RS]?* can combine with preposition tags and verbal particle tags, corresponding to the regular expression *IN or TO or RP*. PPs can attach prepositions to treat multi-word prepositions, for example in *from under the bed*. Some rules need access to lexical items. For example,

- *including_VBG* is analyzed as a preposition contrary to its verb tag.
- The adverbs *ago, later, before* are considered to be real postpositions.
- *that_IN* and *because_IN*, if allowed as prepositions, lead to many incorrect analyses.

General restrictions:

- The distance needs to be very short distance. Adjacency is thus enforced.
- *that_IN* is not a preposition.
- *because_IN* is not a preposition.

Special restrictions:

- The direction allowed to the right only for the English postpositions *ago, later, before*.
- *Including_VBG, involving_VBG* can be prepositions.
- *Because of, according to* is a multi-word preposition. The attachment of *because* to an *of*-PP is thus allowed, although *because* is otherwise no preposition. The present participle *according* is allowed to attach an *to*-PP.

Example:

```
head('NN', 'IN',r,prep,'PP',[_,_,ago,_,_,_],F-G) :- D is F-G, D is 1.
```

Compl

There are 24 complementizer rules. Any verb tag, corresponding to the regular expression *VB[ZPGDN]?* can combine with tags and closed class words expressing complementizers, corresponding to the regular expression *IN or WRB or whether_CC or but_CC)*. An example for *WRB* is the following.

(12) While many of the risks were anticipated *when_WRB* Minneapolis-based Cray Research first announced the spinoff ...

General restriction:

- The subordinated clause verb needs to have a canonical subject.

Example:

```
head('VBD','CC',1,compl,'VBD',[_,_,_,whether,UG,_],_) :- members('<-subj<-',UG,1).
```

Adjtrans

There are 12 adjective translation rules. Any noun tag, corresponding to the regular expression *NN[P]?[S]?* can combine with verb gerund tags, corresponding to the regular expression *VB[DNG]*, as shown in the following example.

(13) Longer maturities are thought to indicate *declining_VBG* interest rates

Gerunds that can act as verbs or adjectives are a major source of ambiguity. The chunker does not include them in the noun chunk, which means that the parser has to decide. The gerund cannot be an adjective if the noun chunk contains a determiner. The local ambiguity created by gerunds often survives up to the sentence level: *He likes developing countries.*

Restrictions:

- Short distance. Adjacency is enforced.
- The noun chunk does not contain a determiner.
- No verb of utterance is allowed: in order to avoid conflicts with verb-subject inversion, verbs licensing this inversion cannot act as adjectives.

  (14) Share prices will fall, stock brokers kept quoting.

  (15) ? Share prices will fall, kept quoting stock brokers.

Example:

```
head('NN','VBG',1,adjtrans,'NN',[FC,[_],_,V,OF,[]],F-G) :-
   D is F-G, D is 1, \+ ending_member(FC,'_DT'), \+ vpnpsubj_bak_verb(V,_).
```

NB. A probabilized version has been implemented after the completion of the thesis.

5.2.3 Minor Types

All minor types do not have a probability model.

Nountrans

"Lonely" adjectives, i.e. adjectives outside noun chunks (for example *the same*, *the poor*) can have noun function. There are 3 noun translation rules.

Restriction: Only an adjacent determiner can trigger the translation.

Example:

```
head('JJ',  'DT',1,nountrans,'NN',[_,[_],_,_,_,_],F-G) :- D is F-G, D is 1.
```

Adv

There are 51 adverb rules. Any verb or noun tag, corresponding to the regular expression *VB[ZPGDN]? or NN[P]?[S]?* can combine with any adverb tag, corresponding to the regular expression *RB[RS]?*. adv is allowed to overstep commas. Proper adverb-attachment is often not ensured. There is no probability model, the main purpose of the adverb-relation is to attach adverbs somewhere, so that an adverb does not fragment the parse into two parts. This is an area where Pro3Gres can be improved in the future.

Restrictions:

- Short distance
- The preposition *about* can be analysed as an adverb (*about 200 people came.*)

Modrel

There are 65 modification by relative clause rules. Any noun tag, including WH-words, corresponding to the regular expression *(NN[P]?[S]? or WP* can combine with any verb tag, corresponding to the regular expression *VB[ZPGDN]?*. $modrel$ is allowed to overstep commas. No distinction between restrictive and non-restrictive relative clauses is made, although that would be easy to integrate.

General Restrictions:

- The relative clause has a subject
- This subject is a pronoun: the rel. pronoun or a personal pronoun
- The relativized noun does not have a modpart

Special restriction: if the subject of the relative clause is a personal pronoun, the relative pronoun (zero-relative object) may be absent. The restriction of zero-relatives to the pronoun case is a pragmatic simplification.

Example:

```
head('NN','VBZ',r,modrel,'NN',[_,FD,_,_,UG,OG],_) :-
   member('<-subj<-',UG), (member('who',UG) ; member('that',UG);
     member('which',UG); ending_member(FD,'_PRP')),
   \+ member('->modpart->',OG).
```

Adjective

This auxiliary relation covers a number of noun-adjective modification including an adjective equivalent to $modpart$. There are 2 rules. Examples for the $adjective$ relation are the following.

(16) ... for the purpose of keeping *[the_DT prices_NNS] reasonable_JJ*

(17) We have demonstrated that triggering delivers *[signals_NN] capable_JJ* of activating the NF-AT transcription factor

Predadj

This rule is used to attach predicative adjectives to the verb. There are 18 rules. Any verb tag, corresponding to the regular expression *VB[ZPGDN]?* can combine with any adjective tag, corresponding to the regular expression *JJ[RS]*.

Restrictions:

- Short distance
- Verb has no object, except if it is a verb of the *elect class*: (*consider them incompetent*). This is a consequence of our treatment of copular and *elect* verb complements as objects.

Example:

```
head('VBD','JJ',r,predadj,'VBD',[_,_,_,BE,UG,OG],F-G) :-
  (\+member('->obj->',OG) ; obj2except(BE)), D is F-G, D=<3.
```

Comp

$comp$ is used to build up comparison constructions involving an adverb *RBR* or an adjective *JJR* in the comparative.

Example:

```
head('RBR','COMP',r,comp,'RBR',[_,_,than,_,_,_],F-G) :- D is F-G, D=<3.
head('JJR','COMP',r,comp,'JJR',[_,_,than,_,_,_],F-G) :- D is F-G, D=<3.
head('IN','NN',r,comp,'COMP',[_,_,_,than,_,_],F-G) :- D is F-G, D=<3.
```

Gen, Pos

gen and pos are used to build up the Saxon genitive. The gen relation attaches the *'s_POS* or *'_APOSTR* marker to the noun, which can then be attached to its head noun. The head noun cannot be a proper noun.

Example:

```
head('NNP','POS',r,gen,'NNP$',_,A-Z) :-  D is A - Z, D < 3.
head('NNPS','APOSTR',r,gen,'NNP$',_,A-Z) :-  D is A - Z, D=1.
head('NN','NNP$',l,pos,'NN',[_,_,_,_,UG,OG],A-Z) :-  D is A - Z, D < 4.
head('NNS','NNP$',l,pos,'NNS',[_,_,_,_,_,_],A-Z) :-  D is A - Z, D < 4.
```

5.2.4 Unconventional Types

This class contains a number of relation types which are each quite different from all other relations.

Conj

Conjunctions have always been a problem for DG. At least three elements, two conjoined elements and the conjunction, need to be combined into a structure. This is difficult in a grammar that knows binary rules only. Also, on a semantic level, it cannot be said that any of the conjoined elements governs the other. In order to treat conjoined and non-conjoined phrases on a par, it is also not desirable to have the conjuction as the governor. As a pragmatic solution, we use binarized conjunction rules. In a first step, a word of a given word class can govern a preceding conjunction. In a second step, that word (it is checked that it governs a conjunction) can be governed by a preceding word of the same part-of-speech tag. As a consequence, the following simplification is made for lexicalization: only the lexical item of the first conjoined element is respected.

```
head('NN','CC',l,conj,'NN',_,_).
head('NN','NN',r,conj,'NNS',  [_,_,_,_,UG,OG],_) :- member('<-conj<-',UG).
```

Enumerations: A comma is allowed to be a conjunction if the enumeration is terminated by a conjunction. Enumerations are a case where commas are easily disambiguated.

```
head('NN', 'COMMA',l,conj,'NN', [_,_,_,_,UG,OG],F-G) :-
   D is F-G, D=<2,  member('->conj->',UG).
```

Apposition and other relations stepping across commas

Commas are generally ambiguous. Three cases are distinguished: Enumerations (see above), appositions and boundaries. Appositions are (often) identifiable as starting and ending with a comma, the head of the apposition is a noun or an adjective, and appositions modify nouns.

Boundary commas are commas which have structuring information. They do not alter relation types but mark a separation a high level in the syntax tree. In a first parsing step, only the apposition and conjunction relations are allowed to span across commas. When parsing finishes, i.e. all possible reductions have been made, all other syntactic relations that can span across commas (e.g. subject, PP-attachment, but not object) are allowed to do so and parsing continues, finding new reductions that overstep commas. This procedure implements the intuition that commas are a strong boundary.

High separation assumption (works relatively well):

1. parse without commas, except for enumeration until all partial analyses are found.
2. then, allow comma-involving relations to transgress commas.

Nchunk

$nchunk$ corrects some chunking insufficiencies involving $adjtrans$, currencies, percent signs, numbers, etc. For example, in *the corresponding_VBG solution* the chunker does not deliver a chunk. The $adjtrans$ relation attaches the present participle to the noun, then an $nchunk$ relation can attach the determiner. There are 51 rules of this type.

Aux and wh-preparsing

The aux relation is needed to attach auxiliary verbs in questions. Since DG does not distinguish internal and external arguments, the auxiliary verb can attach locally to the matrix verb. WH-relations the are only real long-distance dependencies.

Chapter 6: Statistical Disambiguation

6.1 Decision-Based Parsing

We have given an introduction to the probabilistic disambiguation that we use in chapter 1. The probabilities that we use are not generation probabilities, but decision probabilitities: given a certain parser configuration, which attachment decision is how probable? Not the sum of p of possible parses, but the sum of p of possible decisions at a decision point add to 1. Whether to attach or not (in shift/reduce parlance: to reduce or to shift) is e.g. a decision.

The probability-based score of a parse is the product of the (normalized) decisions taken during parsing.

6.2 Backoffs

Sparse data is a serious problem, most decisions need to be taken based on less information than is desirable.

6.2.1 The $pobj$ relation backoff as an example

We give the example of the current back-off hierarchy of the $pobj$ relation (verb-PP attachment) in Prolog code.

```
%%%% PP-Attachment
%% v<pp
stats2(_HTag,FH,_SH,_DTag,DescN,Prep,pobj,P,NP,D,_HC,_OG) :-
    ((vppp(FH,Prep,DescN,Count), coocvppp(FH,Prep,DescN,CoCount),baklev(vppp,0));  % full, backoffs below
     ((nountoclass(DescN,_,DescNClass) -> true;DClass=18),
       vppp_bak_nclass(FH,Prep,DescNClass,Count), coocvppp_bak_nclass(FH,Prep,DescNClass,CoCount), baklev(vppp,1)
     ); % verb & prep & nounclass
     (vppp_bak_verb_prep(FH,Prep,Count), coocvppp_bak_verb_prep(FH,Prep,CoCount), baklev(vppp,2)
     ); % verb & prep
     (verbtoclass(FH,HClass),
       vppp_bak_vclass(HClass,Prep,DescN,Count), coocvppp_bak_vclass(HClass,Prep,DescN,CoCount), baklev(vppp,3)
     ); % verbclass & prep & noun
     ((nountoclass(DescN,_,DescNClass) -> true;DClass=18),verbtoclass(FH,HClass),
       vppp_bak_class(HClass,Prep,DescNClass,Count), coocvppp_bak_class(HClass,Prep,DescNClass,CoCount), baklev(vppp,4)
     ); % verbclass & prep & nounclass
     (vppp_bak_prep_descnoun(Prep,DescN,Count), coocvppp_bak_prep_descnoun(Prep,DescN,CoCount), baklev(vppp,5)
     ); % prep & descnoun
     (vppp_bak_prep(Prep,Count), coocvppp_bak_prep(Prep,CoCount),baklev(vppp,6)
     ); % prep only
     (Count is 0.05, CoCount is 1, baklev(vppp,7)) %% Smoother
    ),
    dist(vppp,D,C,_,TotC), DP is 0.8 + ((C/TotC)*2),
    P is (Count/CoCount),
    NP is ((Count/CoCount)*2)*DP. %% PROB: ATTACHMENT / COOCCURRENCE * # of poss. attachments
```

The predicate vppp/4 contains the counts (4th argument) of verb-PP attachment involving the governor verb FH, the preposition $prep$ and the description noun $DescN$. The predicate coocvppp/4 contains the counts (4th argument) of candidate configurations, i.e. configurations that *could* be verb-PP attachments, including those where the Penn Treebank has a noun-PP attachment.

The top counts of vppp/4 and coocvppp/4 are:

```
cincano:results_from_WSJ_morph_nov03 gschneid$ head -20 vp_pp_morpha.pl
vppp('close','in','trading', 66).
vppp('file','in','court', 51).
vppp('reach','for','comment', 49).
vppp('rise','to','yen', 45).
vppp('contribute','to','article', 44).
vppp('account','for','%', 42).
vppp('file','for','protection', 40).
vppp('rise','to','%', 38).
vppp('say','in','interview', 36).
vppp('be','for','year', 36).
vppp('be','in','market', 35).
vppp('buy','at','price', 33).
vppp('rise','from','yen', 32).
vppp('say','in','statement', 31).
vppp('raise','to','%', 31).
vppp('rise','in','september', 29).
vppp('price','at','par', 29).
vppp('file','with','commission', 29).
vppp('fall','to','%', 29).
vppp('rise','in','quarter', 26).

cincano:results_from_WSJ_morph_nov03 gschneid$ head -20 vp_pp_cooc_morpha.pl
coocvppp('close','in','trading', 66).
coocvppp('report','for','quarter', 54).
coocvppp('file','in','court', 51).
coocvppp('reach','for','comment', 49).
coocvppp('rise','to','yen', 45).
coocvppp('contribute','to','article', 44).
coocvppp('be','in','market', 42).
coocvppp('account','for','%', 42).
coocvppp('file','for','protection', 40).
coocvppp('rise','to','%', 38).
coocvppp('be','for','year', 38).
coocvppp('say','in','interview', 36).
coocvppp('buy','at','price', 33).
coocvppp('rise','from','yen', 32).
coocvppp('say','in','statement', 31).
coocvppp('raise','to','%', 31).
coocvppp('rise','in','september', 29).
coocvppp('price','at','par', 29).
coocvppp('file','with','commission', 29).
coocvppp('fall','to','%', 29).
```

For example, all 66 cases where the configuration 'close in trading appears' is seen in the Penn Treebank all verb-PP attacment, while most (or all) configurations 'report for quarter' (e.g. *the company reported a loss for the third quarter*) are not verb-PP, but (probably) noun-PP attachment.

The counts when backing off to verb+prep only are:

```
cincano:results_from_WSJ_morph_nov03 gschneid$ cat vp_pp_morpha.pl | grep "vppp_bak_verb_prep('close'"
vppp_bak_verb_prep('close','across',1).
vppp_bak_verb_prep('close','after',2).
vppp_bak_verb_prep('close','against',1).
vppp_bak_verb_prep('close','around',1).
vppp_bak_verb_prep('close','at',73).
vppp_bak_verb_prep('close','because',6).
vppp_bak_verb_prep('close','before',2).
vppp_bak_verb_prep('close','below',1).
vppp_bak_verb_prep('close','by',6).
vppp_bak_verb_prep('close','despite',1).
vppp_bak_verb_prep('close','for',16).
vppp_bak_verb_prep('close','from',1).
vppp_bak_verb_prep('close','in',109).
vppp_bak_verb_prep('close','near',1).
vppp_bak_verb_prep('close','of',2).
vppp_bak_verb_prep('close','on',26).
vppp_bak_verb_prep('close','to',11).
vppp_bak_verb_prep('close','with',3).
vppp_bak_verb_prep('close','within',2).

cincano:results_from_WSJ_morph_nov03 gschneid$ cat vp_pp_cooc_morpha.pl | grep "coocvppp_bak_verb_prep('close'"
coocvppp_bak_verb_prep('close','across',1).
coocvppp_bak_verb_prep('close','after',2).
coocvppp_bak_verb_prep('close','against',1).
coocvppp_bak_verb_prep('close','around',1).
coocvppp_bak_verb_prep('close','at',73).
coocvppp_bak_verb_prep('close','because',6).
coocvppp_bak_verb_prep('close','before',2).
coocvppp_bak_verb_prep('close','below',1).
coocvppp_bak_verb_prep('close','by',6).
coocvppp_bak_verb_prep('close','despite',1).
coocvppp_bak_verb_prep('close','for',16).
coocvppp_bak_verb_prep('close','from',2).
coocvppp_bak_verb_prep('close','in',116).
coocvppp_bak_verb_prep('close','near',1).
coocvppp_bak_verb_prep('close','of',12).
coocvppp_bak_verb_prep('close','on',26).
coocvppp_bak_verb_prep('close','since',1).
coocvppp_bak_verb_prep('close','to',12).
coocvppp_bak_verb_prep('close','with',4).
coocvppp_bak_verb_prep('close','within',2).
```

We see e.g. that the probability for verb-PP attachment with *close+in* is high, while *close+of* is low.

6.2.2 Mapping Grammatical Relations for the Probability Estimation

We integrate a wide selection of knowledge sources in our system, and a variety of techniques. Integrating grammatical knowledge does not only have an impact on the grammar rules. We now give an example of how grammatical knowledge can be used for the probability estimation.

We have discussed the MLE estimation in general in section 1.1, and the case of PP-attachment relations. Some relations have models that slightly differ from the general estimation. For example, the noun-participle relation integrates linguistic knowledge about similar grammatical relations.

The noun-participle relation is also known as reduced relative clause. Examples are *the report written* or *the soldiers returned home were greeted*. Reduced relative clauses are frequent enough to warrant a probabilistic treatment, but considerably sparser than verb-subject or verb–object relations. They are in direct competition with the subject-verb relation (as discussed in section **??**), because both are licensed by a NP followed by a VP. We have a subject-verb relation in *the report announced the deal* and a noun-participle relation in *the report announced yesterday*. The majority of modification by participle relations, if the participle is a past participle, functionally correspond to passive constructions (*the report written* $\cong$ *the report which has been written*). In order to reduce data sparseness, we have added the verb–passive-subject counts ($psubj$) to the noun–participle counts. Some past participles also express adjunct readings (*the week ended Friday*); therefore the converse, i.e. adding noun–participle counts to verb–passive-subject counts, cannot be recommended.

The probability estimation for the $modpart$ relations across all backoff levels is as follows (we map the noun a to its Wordnet-class $\mathring{a}$ and the verb b to its Levin-class $\mathring{b}$).

$$P(modpart, dist|a,b) = \left\{ \begin{array}{ll} \frac{\#(modpart,right,a,b)+\#(psubj,left,a,b)}{\#(modpart,right,a,b)+\#(psubj,left,a,b)+\#(asubj,left,a,b)} & if>0,else \\ \frac{\#(modpart,right,\mathring{a},\mathring{b})+\#(psubj,left,\mathring{a},\mathring{b})}{\#(modpart,right,\mathring{a},\mathring{b})+\#(psubj,left,\mathring{a},\mathring{b})+\#(asubj,left,\mathring{a},\mathring{b})} & if>0,else \\ \frac{\#(modpart,right,b)+\#(psubj,left,b)}{\#(modpart,right,b)+\#(psubj,left,b)+\#(asubj,left,b)} & if>0,else \\ \frac{\#(modpart,right,a)+\#(psubj,left,a)}{\#(modpart,right,a)+\#(psubj,left,a)+\#(asubj,left,a)} & \end{array} \right\} \cdot \frac{\#(modpart,dist)}{\#modpart}$$

As the last backoff, a low non-zero probability is assigned for most relations. Again, based on grammatical knowledge, there are exceptions to this: the verb–adjunct relation can only occur with a closed class of nouns, mostly with adverbial expressions of time. In order reduce parsing complexity, only the backoff levels that include the adjunct noun or its class are used. The backoff hierarchy is also changed. Before using a semantic class on the dependent and keeping the lexical head of the governor, the adjunct relation keeps the head of the dependent and uses the semantic class of the verb. While predicates place selectional restrictions on their arguments, adjuncts are themselves a restricted class, a class that can, however, be selected by almost any predicate (Merlo and Esteve Ferrer, 2006).

Verb-PP Backoff decision points			
pobj	0	full	124
	1	verb & prep & nounclass	2624
	2	verb & prep	2631
	3	verbclass & prep & noun	337
	4	verbclass & prep & nounclass	5004
	5	prep & noun	995
	6	prep	4762
	7	NONE	4747

Table 6.1: Verb-PP Backoff decision points for the Fully Lexicalized, Backed-Off System on Carroll's test suite

"Successful" Verb/Noun-PP Backoff decisions		*pobj*	*modpp*
0	full	26	25
1	verb/noun & prep & nounclass	0	0
2	verb/noun & prep	292	260
3	verb/nounclass & prep & noun	19	20
4	verb/nounclass & prep & nounclass	83	106
5	prep & noun	3	7
6	prep	20	23
7	NONE	0	0

Table 6.2: Current Verb/Noun-PP Backoff "successful" decision points for the aggressively pruned System on Carroll's test suite

6.2.3 Sparseness and Quality

In order to assess the impact of sparseness, we illiustrate at which backoff level decisions are taken, i.e. at which level in the backoff the first non-zero count occurs, from which the probbilites are then used. We will refer to them as them decision points.

We can see in table 6.1 that late backoff decisions are disappointingly frequent. But which decisions are successful in the sense that they actually feature in the globally first reading? These "successful" decision points are given in table 6.2.

Decisions that can be taken early in the backoff chain, i.e. better informed decisions, are expected to better performing. This is indeed the case, as figure 6.1 illustrates.

6.3 Obtaining lexical statistics

There are a number of ways in which the MLE statistics can be obtained. The logic is always the same: Obtain all the counts for the counter, and all the candidate counts for the denominator of the probability calculation.

6.3.1 With TGrep from Penn Treebank

TGrep is, as its name suggests, a *grep* untility for trees. The original TGrep was written byRichard Pito and is no longer maintained. Its successor, TGrep2, is written by Douglas Rohde and available for download [1].

The rule file which extracts the lexical information, `XTRACT_RELATIONS.tcsh`, is written for the original TGrep. It is not fully compatible with tgrep2, due to a different semantics of the flag $-a$ which delivers all solutions – a necessary requirement for statistics. When recreating the statistics this way, it is vital to make sure that counts are similar to the original ones.

We give an introduction and a few examples in the following.

Introduction to TGrep

"tgrep patterns are composed of a node followed by the relationships which that node participates in. For example,

```
1)       S < NP << S
```

will match an S node which immediately dominates an NP *and* which dominates some other S node. Note that the second relationship "<< S" refers to the first S and not to the NP. This syntax has been adopted to avoid using clumsy AND statements. You can use parenthesis to group relationships so that,

```
2)       S < (NP << S)
```

[1] http://tedlab.mit.edu/ dr/Tgrep2/

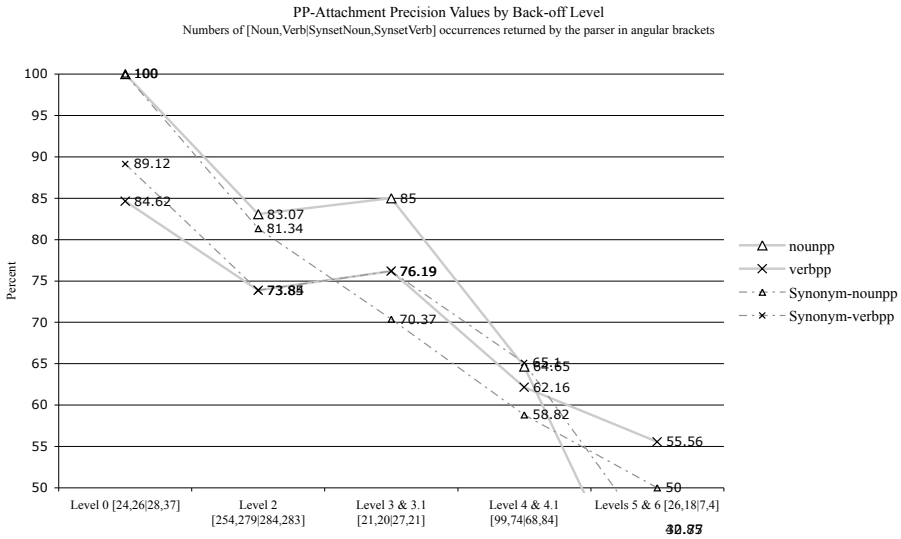


Figure 6.1: Evaluation Preview: Quality of Backoff

will match an S node which immediately dominates an NP node which in turn dominates some S node." (Rohde 1993, tgrep manpage) There are several basic kinds of relationships between nodes. They are:

```
A < B       A immediately dominates B
A <X B      B is the Xth child of A (i.e. first child
            is A <1 B)
A <-X B     B is the Xth to last child of A
            (i.e. last child is A <-1 B)
A <- B      the last child of A is B
            (same as A <-1 B)
A << B      A dominates B
A >> B      A is dominated by B
A > B       A is immediately dominated by B
A <<, B     B is a leftmost descendant of A
A <<' B     B is a rightmost descendant of A
A . B       A immediately precedes B
A .. B      A precedes B
A $ B       A and B are sisters (note that A $ A
            is FALSE)
A $. B      A and B are sisters and A immediately
            precedes B
A $.. B     A and B are sisters and A precedes B
```

Their negations are also available, e.g. `A !< B`, which expresses that A does not immediately dominate B

Example queries

Verb-Object Relations In a verb-object relation, an NP is a daughter of a VP. This is expressed in a tgrep query as `VP < NP`. The full command is e.g.

```
tgrep 'VP < NP' | more
```

.

If one wants to extract the verb and noun heads, things get more complicated, mainly because one needs to traverse the NP and the VP down to the lexical head. The tgrep backtick operator (‘) only outputs the constituent in the query after the operator, in the following examples the lexical heads. These are thecurrently used verb-object patterns:

```
[1]tgrep -alfs " " -n 'VP << `/VB/ < (/NP$/|NP-PRD <-
`(/NN/|PRP|WDT|WP|CD))' > OBJECT_ADJUNCT/VP\{NPO_1.res
[2]tgrep -alfs " " -n 'VP << `/VB/ < (/NP$/|NP-PRD <1 (NP <-
`(/NN/|PRP|WDT|WP|CD)))' > OBJECT_ADJUNCT/VP\{NPO_2.res
[3]tgrep -alfs " " -n 'VP << `/VB/ < (/NP$/|NP-PRD <1 (NP<1(NP <-
`(/NN/|PRP|WDT|WP|CD))))' > OBJECT_ADJUNCT/VP\{NPO_3.res
[4]tgrep -alfs " " -n 'VP << `/VB/ < (/NP$/|NP-PRD <1 (NP<(NP<1(NP <-
`(/NN/|PRP|WDT|WP|CD)))))' > OBJECT_ADJUNCT/VP\{NPO_4.res
[5]tgrep -alfs " " -n 'VP << `/VB/ < (/NP$/|NP-PRD <1 (NP<(NP<(NP<1(NP
<- `(/NN/|PRP|WDT|WP|CD))))))' > OBJECT_ADJUNCT/VP\{NPO_5.res
[6]tgrep -alfs " " -n 'VP << `/VB/ < (/NP$/|NP-PRD <1
(NP<(NP<(NP<(NP<1(NP <- `(/NN/|PRP|WDT|WP|CD)))))))' > OBJECT_ADJUNCT/VP\{NPO_6.res
```

Without the flag $-a$ tgrep only finds one occurrence per sentence. This entails that counts of frequent relations will be far too low and inacurrate. If one uses the flag $-a$, like in Prolog backtracking, many often seemingly incorrect solutions are reported. It is vital to formulate constrained and precise queries to reduce overgeneration. The complexity arising from this has led me to split every query into a set of simpler queries, in the above example 6 queries. The currently used number of queries in `XTRACT_RELATIONS.tcsh` is around 500.

PP-Attachment relations The basic queries for PP-attachment are

- for noun-attached PPs:
  tgrep '/NP/ < (/PP/ < (IN|TO) < NP)'
- for verb-attached PPs:
  tgrep 'VP < (/PP/ < (IN|TO) < NP)'

Candidate Counts Formulating queries for candidates, i.e. configurations that could e.g. be verb-PP attached, but which are not always, is more involved. A good approximation for verb-PP attachment is to take typical ambiguous configurations: a VP that has a NP which is itself modified by a PP is ambiguous, because the PP could be attached to the VP.

One of the tgrep patterns for verb-PP attachment is:

```
tgrep -afs " " -n 'VP << `/VB/ < (/PP/ < `/IN/ < (NP <- `/NN/ ))'
  > PP/VP\{PP_verbhead+prep+descNoun_s.res
```

The corresponding tgrep pattern for the opposite case, i.e. the ambigous case in which the PP is attached to the noun and not the verb, is:

```
grep -afs " " -n 'VP << `/VB/ < (/NP/ <- `/NN/) < (/PP/ < `/IN/ < (NP <- `/NN/))'
  > PP/VP\{NP_PP_11.res
```

Some of the candidate cases, we also call them co-occurrence cases, for other relations are:

- subjects: non-subject NPs that immediately precede a VP, like *the report which blames* **the major has been published** *now.*
- object: adjunct NPs like *we* **discussed today** *that ...* and subordinate subjects like we *showed our hypothesis* is correct.
- subordinating conjunctions vs. prepisitions: both have the same tag (IN) in the Penn treebank. Subordinating conjunctions have an S node as sister, prepositions an NP node.

6.3.2 With Perl script from CoNLL Output

For domain adaptation, it may be desirable to include statistical lexical data from other corpora. For adapting to the the Biomedical domain, an extraction of statistical data from the CoNLL format is currently being developped. The CoNLL-format originates from the CoNLL-XI shared task (Nivre et al., 2007). An example output is:

```
1       Abstract        abstract        NN      _       0       ROOT

1       Two     two     CD      _       3       NMOD
2       recent  recent  JJ      _       3       NMOD
3       papers  paper   NNS     _       4       SBJ
4       provide provide VBP     _       0       ROOT
5       new     new     JJ      _       6       NMOD
6       evidence        evidence        NN      _       4       OBJ
7       relevant        relevant        JJ      _       6       NMOD
8       to      to      IN      _       7       AMOD
9       the     the     DT      _       10      NMOD
10      role    role    NN      _       8       PMOD
11      of      of      IN      _       10      NMOD
12      the     the     DT      _       16      NMOD
13      breast  breast  NN      _       14      NMOD
14      cancer  cancer  NN      _       15      NMOD
15      susceptibility  susceptibility  NN      _       16      NMOD
16      gene    gene    NN      _       17      NMOD
17      BRCA2   brca0   NN      _       11      PMOD
18      in      in      IN      _       10      NMOD
19      DNA     dna     NN      _       20      NMOD
20      repair  repair  NN      _       18      PMOD
21      .       .       .       _       4       P
```

Comments on the complex mapping between the Pro3Gres format and the CoNLL format can be found in chapter 7 (evaluation) of the thesis. While the mapping problems forbid the use of CoNNL format for accurate evaluation, an extraction of relations with full precision at incomplete recall is possible.

A simple conversion routine fragment is e.g.:

```
foreach $word(@words) {
   ($id,$full,$lemma,$tag,$u,$tohead,$rel) = split(/\t/,$word) ;
    if ($rel eq "SBJ") {
        $gov = $words[$tohead-1]; print "% SBJ $gov --> $word\n";
        ($hid,$hfull,$hlemma,$htag,$hu,$htohead,$hrel) = split(/\t/,$gov) ;
        print "subj($hlemma,$lemma).\n";
    }
...
}
```

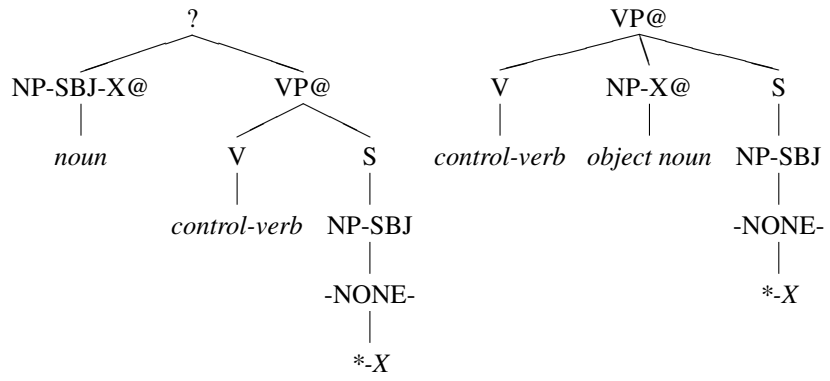


Figure 6.2: Extraction pattern for subject control (left) and object control (right)

6.3.3 With Prolog from Simon's format

6.3.4 Functional restrictions and long-distance dependencies from the Penn Treebank

Many additional lexical counts have also been collected form the Penn Treebank, there are various versions of `XTRACT_RELATIONS*`

For example, verb-locativePP Relations which extract heads can be formulated as follows:

```
tgrep -alfs " " -n 'VP << `/VB/ < (PP-LOC < `(IN|TO) < (NP <-
  `(/NN/|PRP|WDT|WP|CD) ))' > PP-FUNC/VP\{PP-LOC_verbhead+prep+descNoun_s.res
tgrep -alfs " " -n 'VP << `/VB/ < (PP-LOC < `(IN|TO) < (NP <1 (NP <-
  `(/NN/|PRP|WDT|WP|CD) )))' > PP-FUNC/VP\{PP-LOC_verbhead+prep+descNoun_1.res
tgrep -alfs " " -n 'VP << `/VB/ < (PP-LOC < `(IN|TO) < (NP <1 (NP < (NP <-
  `(/NN/|PRP|WDT|WP|CD) ))))' > PP-FUNC/VP\{PP-LOC_verbhead+prep+descNoun_2.res
tgrep -alfs " " -n 'VP << `/VB/ < (PP-LOC < `(IN|TO) < (NP <1 (NP < (NP <(NP <-
  `(/NN/|PRP|WDT|WP|CD) )))))' > PP-FUNC/VP\{PP-LOC_verbhead+prep+descNoun_3.res
tgrep -alfs " " -n 'VP << `/VB/ < (PP-LOC < `(IN|TO) < (NP <1 (NP < (NP <(NP<(NP <-
  `(/NN/|PRP|WDT|WP|CD) ))))))' > PP-FUNC/VP\{PP-LOC_verbhead+prep+descNoun_4.res
```

An important aspect of the Pro3Gres parser is that it treats long-distance dependencies by reducing them to local dependencies. Chapter 6 of my doctoral thesis explains the details. Here we just give an introduction and the example of the subject control relation.

Control Structures

Control is a course-book example of long-distance dependency. The extraction pattern for control is in fig. 6.2.

Control and raising coreferences can be reconstructed successfully from the context of the matrix and the subordinate clause. If the matrix clause contains a control verb, a raising verb, or a control adjective, and if the subordinate clause contains a verb in the infinitive with *to* and a corresponding unfilled argument position, then coreference is assumed. This is the assumption expressed by our fixed extraction pattern.

How many filler and gap indices coincide in our fixed pattern?

Filler and gap indices coincide in our fixed pattern to a high degree. Comparing the counts with enforced and relaxed filler identity we get 98 % identity for subject-control. In the 320 cases tested, 7 had no identity of gap and filler. An example of these 7 sentences where the subject-control pattern goes astray is:

```
(NP-SBJ-1 (DT Some)
          (NNP Golenbock)
          (NNS lawyers))
(VP (MD wo)
    (RB n't)
    (VP (VB be)
        (VP (VBN invited)
            (NP-2 (-NONE- *-1))
            (S (NP-SBJ (-NONE- *-2))
               (VP (TO to)
                   (VP (VB join)
                       (NP (NNP Whitman)
                           (CC &)
                           (NNP Ransom)))))
            (, ,)
            (PP (VBG according)
                (PP (TO to)
                    (NP (NP (NNS partners))
                        (PP-LOC (IN at)
                                (NP (DT both)
                                    (NNS firms)))))))))
```

Complex interaction between different types of long-distance dependencies, in this case between passive and control, means that the extraction pattern can make errors. While this affects the recall of the pattern on the Penn Treebank for obtaining lexical statistics, reconstruction of the control relation during parsing remains unaffected.

We obtain above 99 % identity for object-control. Of the 264 cases tested in the appendix, one failed to have identity.

How many fillers do not find their gap in the subordinate subjectless clause?

In this experiment, we tested in how many cases a filler occurs without a gap position in the position expected by our fixed pattern, i.e. the subordinate clause subject position. Our experiment, detailed in the appendix, shows that the control pattern occurs 37 times more often than an (otherwise identical) pattern explicitly requiring a no-trace subordinated subject.

The 8 tested subject-control cases where no gap occurs in the subordinate clause subject position include a conjunction that triggers a mismatch, an annotation error, a complex interaction between passive and control, and one case where the movement is longer than what the pattern matches, but not a typical case of control:

```
(NP-SBJ-1 (PRP we))
(VP (MD should)
   (VP (VB be)
       (VP (VBG helping)
           (S (NP-SBJ (NNP U.S.)
                      (NNS companies))
              (VP (VB improve)
                  (NP (VBG existing)
                      (NNS products))
                  (PP (RB rather)
                      (IN than)
                      (S-NOM (NP-SBJ (-NONE- *-1))
                             (ADVP-TMP (RB always))
                             (VP (VBG developing)
                                 (NP (JJ new)
                                     (NNS ones)))))))))))
```

For object control, the control pattern is only 22 times more frequent than an (otherwise identical) pattern explicitly requiring a no-trace subordinated subject. But the tested 15 object-control cases where no gap occurs in the subordinate clause subject position include 13 cases where the pattern erroneously matches a temporal expression in the object position.

Control is almost completely fixed. Although there are a few exceptions and fixed patterns fail when several movements interact, we only lose very few cases when extracting relations with a fixed pattern. While the extraction of long-distance dependencies may fail when several movements interact, the impact on lexical statistics used for parsing and thus the impact on parsing success is very small, and the post-processing step after parsing delivers the correct result unless there are intervening parsing errors.

The trace of both a passive and a control relation is expressed by an *NP** constituent in the Penn Treebank. Campbell (2004) uses a single, purely configurational rule to recover *NP** from Treebank trees where they have been removed. Campbell (2004) reports similar rules for other long-distance relations.

We have written structural patterns corresponding to such rules. Each long-distancce dependency type corresponds to a pattern or a set of patterns. Grammatical role labels, empty node labels and tree configurations spanning several local subtrees are used as integral part of long-distance dependency patterns. This leads to much flatter trees, as typical for DG, which has the advantages that (1) it helps to alleviate sparse data by mapping nested structures that express the same dependency relation[2], (2) fewer decisions are needed at parse-time, which reduces complexity and the risk of errors (Johnson, 2002), (3) the costly overhead for dealing with unbounded dependencies can be partly avoided. It is ensured that the lexical information that matters is available in one central place, allowing the parser to take one well-informed decision.

Verb-Subject-control relation extraction rules can be formulated as follows in tgrep:

```
tgrep -alfs " " -n '/NP-SBJ-/ <- `(/NN/|PRP|WDT|WP) $.. (VP < (`/VB/ \!$ VP $..
  (S < /NP-SBJ/)) ) ' > NPSUBJ_V/NP_VP_CONTROL11.res
tgrep -alfs " " -n '/NP-SBJ-/ <- `(/NN/|PRP|WDT|WP) $.. (VP < (VP <(`/VB/ \!$ VP $..
  (S < /NP-SBJ/))) ) ' > NPSUBJ_V/NP_VP_CONTROL12.res
tgrep -alfs " " -n '/NP-SBJ-/ <- `(/NN/|PRP|WDT|WP) $.. (VP < (VP <(VP <(`/VB/ \!$ VP $..
  (S < /NP-SBJ/)))) ) ' > NPSUBJ_V/NP_VP_CONTROL13.res
tgrep -alfs " " -n '/NP-SBJ-/ <- `(/NN/|PRP|WDT|WP) $.. (VP < (VP <(VP <(VP<(`/VB/ \!$ VP $..
  (S < /NP-SBJ/))))) ) ' > NPSUBJ_V/NP_VP_CONTROL14.res
tgrep -alfs " " -n '/NP-SBJ-/<1(NP <- `(/NN/|PRP|WDT|WP)) $.. (VP < (`/VB/ \!$ VP $..
  (S < /NP-SBJ/)) ) ' > NPSUBJ_V/NP_VP_CONTROL21.res
tgrep -alfs " " -n '/NP-SBJ-/<1(NP <- `(/NN/|PRP|WDT|WP)) $.. (VP < (VP <(`/VB/ \!$ VP $..
  (S < /NP-SBJ/))) ) ' > NPSUBJ_V/NP_VP_CONTROL22.res
tgrep -alfs " " -n '/NP-SBJ-/<1(NP <- `(/NN/|PRP|WDT|WP)) $.. (VP < (VP <(VP <(`/VB/ \!$ VP $..
  (S < /NP-SBJ/)))) ) ' > NPSUBJ_V/NP_VP_CONTROL23.res
tgrep -alfs " " -n '/NP-SBJ-/<1(NP <- `(/NN/|PRP|WDT|WP)) $.. (VP < (VP <(VP <(VP<(`/VB/ \!$ VP $..
  (S < /NP-SBJ/))))) ) ' > NPSUBJ_V/NP_VP_CONTROL24.res
tgrep -alfs " " -n '/NP-SBJ-/<1(NP<(NP <- `(/NN/|PRP|WDT|WP))) $.. (VP < (`/VB/ \!$ VP $..
  (S < /NP-SBJ/)) ) ' > NPSUBJ_V/NP_VP_CONTROL31.res
tgrep -alfs " " -n '/NP-SBJ-/<1(NP<(NP <- `(/NN/|PRP|WDT|WP))) $.. (VP < (VP <(`/VB/ \!$ VP $..
  (S < /NP-SBJ/))) ) ' > NPSUBJ_V/NP_VP_CONTROL32.res
tgrep -alfs " " -n '/NP-SBJ-/<1(NP<(NP <- `(/NN/|PRP|WDT|WP))) $.. (VP < (VP <(VP <(`/VB/ \!$ VP $..
  (S < /NP-SBJ/)))) ) ' > NPSUBJ_V/NP_VP_CONTROL33.res
tgrep -alfs " " -n '/NP-SBJ-/<1(NP<(NP<(NP <- `(/NN/|PRP|WDT|WP)))) $.. (VP < (`/VB/ \!$ VP $..
  (S < /NP-SBJ/)) ) ' > NPSUBJ_V/NP_VP_CONTROL41.res
```

I argue in my doctoral thesis that the vast majority of LDDs can be treated locally. Only relative pronouns and embedded WH-questions need to be treated non-locally. Refer to chapter 6 of my doctoral thesis for the theoretical discussion and the implementation of non-local parsing for these structures.

[2]Data sparseness also depends on the probability model. As an alternative to collapsing the structure in the annotation as we do, it could also be collapsed in the statistical model

	Antecedent	POS	Label	Count	Description/Example
1	NP	NP	*	22,734	NP trace *Sam* was seen *
2		NP	*	12,172	NP PRO * to sleep is nice
3	WHNP	NP	*T*	10,659	WH trace the woman *who* you saw *T*
(4)			*U*	9,202	Empty units $ 25 *U*
(5)			0	7,057	Empty complementizers Sam said 0 Sasha snores
(6)	S	S	*T*	5,035	Moved constituents *Sam had to go*, Sasha said *T*
7	WHADVP	ADVP	*T*	3,181	WH-trace Sam explained *how* to leave *T*
(8)		SBAR		2,513	Empty clauses *Sam had to go*, said Sasha (SBAR)
(9)		WHNP	0	2,139	Empty relative pronouns the woman 0 we saw
(10)		WHADVP	0	726	Empty relative pronouns the reason 0 to leave

Table 6.3: The distribution of the 10 most frequent types of empty nodes and their antecedents in the Penn Treebank (adapted from Johnson 2002). Row numbers in parentheses indicate cases that are inherently local in our functional DG

Chapter 7: Chunk-Internal Relations

We have implemented a pre-processing module into Pro3Gres that recovers relations inside chunks. This module is called between tagging and parsing. The module is rule-based, but has a statistical component. It is based on the following assumptions.

1. Generally, all non-head words in a chunk modify the head. There are four exceptions to this rule:
   - In verb chunks, every verb modifies the succeeding verb (example: *would have been going*).
   - If a noun chunk contains an adjective and more than one noun, the adjective modifies the succeeding non-head noun if it is seen more often before the non-head noun lemma than the head noun lemma in the British National Corpus (example: *conventional forces strengthening*).
   - If a noun chunk contains more than two nouns, the frontmost noun modifies the succeeding non-head noun if it is seen more often before the non-head noun lemma than the head noun lemma in the British National Corpus (example: *Eisenhower administration effort*).
   - If a noun chunk contains more than two proper names, every proper name modifies the succeeding proper name (example: *Fulton County Grand Jury*).
2. Generally, the modification type deterministically follows from the part-of-speech tag of the head and the modifier. There is one exception.
   - Words that can be determiners or adjectives, for example *several, many, few, one* are assigned a $detmod$ relation to the head noun if they occur as the frontmost word in the chunk, and a $ncmod$ relation otherwise.

Using the Ratnaparkhi tagger, we get the performance shown in figure 7.1. $ncmod$ performs relatively poorly. There are several important sources of errors. First, the statistical component often makes errors due to sparse data. For example, in *automobile title law*, the counts are too low. Second, adverbs are included in the verb chunk, although adverbs in copular verb chunks typically modify the verb complement. For example *generally* in *[was generally] favourable* is prevented from modifying the predicative adjective *favourable*. Third, very long base NPs contain very many $ncmod$ relations, but are also at highest risk of being chunked incorrectly. The $ncmod$ relation is thus much more affected by chunking errors than the $detmod$ relation.

Since Pro3Gres V 0.6566, chunk-internal relations are an external file, *intrachunk0.x*. The core of the chunk-internal relations is the intrachunk2/7 predicate. The intrachunk predicate iteratiively traverses the chunk, word by word; the intrachunk2/7 predicate . A version for the detection of the $ncmod1$ relation is commented in the following. An illustrating trace for the noun chunk *free enterprise track* is also given. bixxnn/5 reports the bigram counts from the BNC. *free enterprise* occurss 54 times, *free track* zero times (which is smoothed to 1). As *free enterprise* is more frequent than *free track*, it is assumed to be more likely that *free* modifies the non-head *enterprise* rather than the head

```
intrachunk2(Word,R,'JJ',FID,Ftag,_Lem,Dir) :-
%% args: Word: current word in the iteration, R: Rest of the chunk,
%% FID: head word, Ftag: tag of the head, _, Dir: direction
  intralex(yes), %% is intrachunk flag on?
  R=[F|RR], RR \= [],  %% is the adjective the non-last non-head word?
  wordtag(F,FW,_),  %% get FW=word after current
  lexic(FID,FID1,ID), %% get FID1= word part of current word
  (bixxnn(CN1,Word,_,FW,_)->true;CN1=1),  (bixxnn(CNH,Word,_,FID1,_)->true;CNH=1), CN1 > CNH,
  %% which modifier modifies which?
  !,
  writeintra(ncmod1,[FW-ID,Word-ID,Lem,Dir]).

   Call: (31) intrachunk2(free, [enterprise_NN, track_NN], 'JJ', 'track#7', 'NN', _G2743, '(<-)') ? creep
   Call: (32) intralex(yes) ? creep
   Exit: (32) intralex(yes) ? creep
   Call: (32) [enterprise_NN, track_NN]=[_G2734|_G2735] ? creep
   Exit: (32) [enterprise_NN, track_NN]=[enterprise_NN, track_NN] ? creep
   Call: (32) [track_NN]\=[] ? creep
   Exit: (32) [track_NN]\=[] ? creep
   Call: (32) wordtag(enterprise_NN, _G2742, _G2743) ? skip
   Exit: (32) wordtag(enterprise_NN, enterprise, 'NN') ? creep
   Call: (32) lexic('track#7', _G2745, _G2746) ? skip
   Exit: (32) lexic('track#7', track, 7) ? creep
   Call: (32) bixxnn(_G2780, free, _G2782, enterprise, _G2784) ? skip
   Exit: (32) bixxnn(54, free, 'JJ', enterprise, 'NN') ? creep
   Call: (32) true ? creep
   Exit: (32) true ? creep
   Call: (32) bixxnn(_G2780, free, _G2782, track, _G2784) ? skip
   Fail: (32) bixxnn(_G2780, free, _G2782, track, _G2784) ? creep
   Call: (32) _G2783=1 ? creep
   Exit: (32) 1=1 ? creep
^  Call: (32) 54>1 ? creep
^  Exit: (32) 54>1 ? creep
   Call: (32) writeintra(ncmod1, [enterprise-7, free-7, _G2788, '(<-)']) ? skip
ncmod1('enterprise#7','free#7',_G2788,'(<-)').
```

Relation	P%	R%	Counts Correct
detmod	93.5	90.6	1018
ncmod	83.1	67.9	1080
aux	94.2	89.8	342

Table 7.1: Percentage and absolute values for chunk-internal relations

Chapter 8: Installing and running your own copy of Pro3Gres

Ask the author for a license of the Pro3Gres parser if you are interested: gschneid@ifi.uzh.ch

Installation of the parser is trivial. Unzip with the password, untar and follow 8.1.1.

Installation of the preprocessing tools may be more cumbersome, depending on your choice. We describe the simplest option for pre-processing, using LT-TTT2, here, and two other options in section 8.2

8.1 Installing the parser Pro3gres

8.1.1 Installing Pro3gres

The parser requires SWI or Sisctus Prolog. SWI is recommended. It is available from http://www.swi-prolog.org

A source code distribution looks as follows, for example:

```
-rwxr-xr-x    1 gschneid  gschneid     10053 Jul 29  2005 000RELEASE_NOTES
-rwxr-xr-x    1 gschneid  gschneid     12541 Oct 11 16:20 000RELEASE_NOTES2007
-rw-------    1 gschneid  gschneid     10664 Sep 17 10:33 000RELEASE_NOTES2007.save
-rw-------    1 gschneid  gschneid     12131 Oct  2 11:08 000RELEASE_NOTES2007.save.1
drwxr-xr-x   14 gschneid  gschneid       476 Jul 29  2005 EVAL-Carroll
drwxr-xr-x    3 gschneid  gschneid       102 Jul 29  2005 PCFG
-rw-r--r--    1 gschneid  gschneid  15227661 Sep 17 09:52 Pro3Gres.tar.gz
-rwxr-xr-x    1 gschneid  gschneid      2100 Jul 29  2005 Pro3Gres_parameters_mid_all.pl
drwxr-xr-x   12 gschneid  gschneid       408 Jul 29  2005 Saar
drwxr-xr-x   64 gschneid  gschneid      2176 Jul 29  2005 perl
drwxr-xr-x    3 gschneid  gschneid       102 Jul 29  2005 results_from_WSJ_morph
drwxr-xr-x   55 gschneid  gschneid      1870 Jul 29  2005 results_from_WSJ_morph_nov03
-rwxr-xr-x    1 gschneid  gschneid        29 Jul 29  2005 tar_it
```

EVAL-Carroll/ contains preprocessed evaluation texts (the 500 sentence GREVAL corpus by John Carroll)

Saar/ contains versions of the parser main program in Prolog, e.g. 6514_STANDARD.pl

results_from*/ contains statistical data, mainly from the Penn treebank, but also from the BNC.

Pro3Gres runs under Sicstus Prolog and SWI-Prolog. It is optimized for the latter, where a graphical interface is also available, mainly suitable for demonstration purposes. Loading the statistics files in Sicstus Prolog may take very long. SWI Versions 5.2 to 5.6, and Sicstus Version 3.8.6 have been tested on Mac OS X, Linux, Windows 2000 and XP and Sun Solaris.

```
#%%%%%% THIS ONLY APPLIES IF YOU GET SOURCE AND YOU WANT TO USE SICSTUS
#Depending which Prolog you are using, set the fact whichprolog(swi) or whichprolog(sicstus).
#In the latter case ONE clause needs to be changed manually (uncomment, comment as appropriate)
#
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%SICSTUS VERSION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%findlongest(MaxLen,A-Z,A-I-J-Z,MIn-MOut,ScIn-ScOut,SIn-SOut) :-
#% whichprolog(sicstus),
#% if((chart(_,I,J,_,[_,_,I-J,CScore,MaxLen],_,CStruc,_), \+ I=J, I>=A, J=<Z, append(SIn,[CStruc],SOut), ScOut is ScIn*CScore, MOut=MIn),
#%    true,
#%    (NewMaxLen is MaxLen-1,findlongest(NewMaxLen,A-Z,A-I-J-Z,MIn-MOut,ScIn-ScOut,SIn-SOut))).
#
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%SWI VERSION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#findlongest(MaxLen,A-Z,A-I-J-Z,MIn-MOut,ScIn-ScOut,SIn-SOut) :-
#  whichprolog(swi),
#  (chart(_,I,J,_,[_,_,I-J,CScore,MaxLen],_,CStruc,_), \+ I=J, I>=A, J=<Z, append(SIn,[CStruc],SOut), ScOut is ScIn*CScore, MOut=MIn)
#  *-> true;
#     (NewMaxLen is MaxLen-1,findlongest(NewMaxLen,A-Z,A-I-J-Z,MIn-MOut,ScIn-ScOut,SIn-SOut)).
```

8.1.2 Installing LT-TTT2

If you want to see the parser at work immediately, you can also skip to chapter 2.2. and come back here later. But you will need to install LT-TTT2 or a preprocessing option descibed in chapter 3 to be able to run your own texts.

You can obtain LT-TTT2 from

http://www.ltg.ed.ac.uk/software/lt-ttt2

After you have pre-processed your texts with LT-TTT2 they should look like the provided file TTT2/ex100.chunk , which was created from the raw tex TTT2/ex100.txt

OBS.:!!!! We use a slightly different call to LT-TTT2 compared to the one that is included in the LT-TTT2 distribution. The script that we use is provided in TTT2/ttt.sh

Sample calls are: bash ttt.sh myinput.txt myoutputfiles bash ttt.sh ex100.txt ex100

In particular, we we skip named entity recognition (NER), as it interferes with chunking. Texts using LT-TTT2 NER will not parse!

As an alternative to TTT2/ttt.sh you can also you the script TTT2/scripts/run_noNER , which is also provided . It differs from the script 'run' provided in the LT-TTT2 distribution only by commenting the named-entity recognition step. As this script only ran on Linux but not on OS X for us, we wrote the script TTT2/ttt.sh

FAQ: Can I parse raw text in just 1 command including the calls to LT-TTT2 and the parser?

Not yet currently. Tell me if you have programmed a pipe. For the time being, first pre-process, then parse.

8.1.3 Running Pro3Gres

Processing your documents happens in two stages, as alluded to. First, your document is tagged, chunked and morphologically analyzed, secondly it is parsed.

Running Pro3Gres from the commandline

Go to the distribution root directory and enter the following command in the shell.

```
$ ./build.pl
```

If everything works ok, you should get many messages (some error messages, too) and a file called pro3gres.exe, with a size of about 40 to 50 MB, should appear in the directory.

You can now parse. The provided pre-tagged and chunked file EVAL-Carroll/suste-prologfacts_morphed5.pl, which is LTPos output converted to Prolog format, can be parsed as follows:

```
$ ./pro3gres.exe -file EVAL-Carroll/suste-prologfacts_morphed5.pl
-pfile Pro3Gres_parameters.pl
```

The provided pre-tagged and chunked file TTT2/ex100.chunk , which is LT-TTT2 output, can be parsed as follows:

```
$ ./pro3gres.exe -xmlfile TTT2/ex100.chunk
-pfile Pro3Gres_parameters.pl
```

OBS.! I have only tested a few thousand sentences with this method (-xmlfile, i.e. via TTT2), much fewer than via LTPos/LTCHunk or via Treetagger/Carafe. Please report bugs to me.

8.1.4 Running Pro3Gres in Prolog

If 8.1.3 fails, or if you want to see in more detail what happens, or if you plan to use Pro3Gres mainly from inside Prolog, proceed as follows.

Start Prolog, with large stack sizes. If you get memory allocation errors during loading or parsing, your stack sizes are probably too small.

Start swi Prolog, e.g. with

```
$ swipl -L16m -G16m -T8m -A8m
```

In the 64-bit versions of SWI, which run from OS X 10.6 on, stack sizes are large enough by default.

In Sicstus under Mac OS X, you need to the following to increase stack sizes: Issue the shell command

```
$ ulimit -d unlimited
```

before starting Sicstus Prolog. In case of problems, consult the Sicstus manual and online help.

IF YOU HAVE SOURCE CODE:

cd to the root ot the distribution, e.g.

```
?- cd('SOURCE-DISTRIBUTION').
```

Consult the parser (e.g. progres_6556.pl), for example as follows in the Prolog Shell:

```
?- consult('Saar/progres_6556.pl').
```

You will get one or two error messages and loads of singleton warnings, that is ok. Load the statistical files and the grammar by entering the following command:

```
?- start.
```

You will get many syntax error messages, since the data hasn't been fully cleaned yet. Note that loading in the Prolog facts may take very long in Sicstus.

If you want to create a saved state (aka image file or executable):

```
?- save_me.
```

will produce the image file (currently progres6556.exe) Image files are machine-dependent. They can be started from Prolog or directly from the command-line, making it easy to integrate pro3gres into your pipeline, and faster to start up next time, by either issuing the Prolog command

```
?- restore('progres6556.exe').
```

or equivalently, from the command line

```
$ ./progres6556.exe
```

You can parse your tagged and chunked files now.

If you have preprocessed with LT-TT2, and produced XML output that looks like the provided files, e.g. TTT2/ex100.chunk you can parse with

```
?- xml_to_p3g_parse('TTT2/ex100.chunk').
?- spy(spy_me2), xml_to_p3g_parse('TTT2/ex100.chunk').  %% sentence by sentence
```

You can convert the XML format to the Prolog-based input format (see details in chapter 3), by

```
?- tell('ex100_parserin.pl'), xml_to_p3g_parse('TTT2/ex100.chunk'), told.
```

If you have used LTPos and LTChunk, as described before, leading to a Prolog-based input format, you can parse as follows:

```
?- go_textual('MYFILE').
```

In the distribution, the Carroll tagged and chunked evaluation corpus is provided, in three version: One tagged with LTPos, and one

tagged with the Ratnaparkhi MaxEntropy tagger.

It can be parsed e.g. with

```
?- seen, go_textual('EVAL-Carroll/suste-prologfacts_morphed5.pl').  %% output to screen,
LTPos tagged input
?- tell('6556.OUT'), seen,
go_textual('EVAL-Carroll/suste-prologfacts_morphed5.pl'), told.  %% output to file
6566.OUT
?- seen, go_textual('EVAL-Carroll/suste-prologfacts_morphed5.pl').  %% output to screen,
use Ratnaparkhi-tagged input
?- spy(spy_me2), seen,
go_textual('EVAL-Carroll/suste-prologfacts_morphed5.pl').  %% sentence by sentence.
```

You can set various options by retracting and asserting Prolog facts. The most important ones are:

`?- ambi(X). % X-1 is the number of readings returned`

`?- modus(showtree).` OR `?- modus(fileout).` either shows an ASCII syntax tree or not. For batch processing large corpora, fileout is recommended.

`?- mlf(yes)` or `?- mlf(no).` should Minimal Logical Forms (MLF) be produced? ATTENTION: mlf(yes) still leads to errors!

`?- statmod(backoff).` OR `?- statmod(interpol).` uses backoff (recommended) or interpolation statistics

All parameters are in the file 'Pro3Gres_parameters.*'

You can also use the graphical mode of the parser, as follows:

Parse one or two sentences in non-graphical mode (yes, this is a bug) and ONLY THEN

`?- go_xpce.`

'Select a file' to be parsed (pre-tagged, chunked etc. for example EVAL-Carroll/suste-prologfacts_morphed5.pl), press 'start parsing' and then 'next'. Any 'next' will bring you one sentence further. You can play around with the different tree display formats.

If you like trees, but do not like graphics ;-) , you can also use ASCII tree output, which will perhaps conjure up a feeing of being in an old science fiction movie, but if you have a big screen this is one of the best ways to browse the parser output when debugging.

```
?- abolish(modus/1).
?- assert(modus(showparses)).
=====
```

8.2 Preprocessing options

8.2.1 More on LT-TTT2 for tagging, chunking and morphological analysis

LT-TTT2 is an XML-based suite of low-level processors doing tagging, morphological analysis, and tagging. They are a further development of LTPos and LTChunk. They support layered chunking, and named-entity recognition, but we do not support the latter. We plan to move our main pipeline to LT-TTT2, as these powerful tools also provide information on tense, aspect etc. and are maintained.

The main davantage of this option is that it is simple and powerful. The main disadvantage is that we have only tested a few thousand sentences, the conversion from XML to Prolog input format may still throw an error occasionally. Mail me if that happens.

8.2.2 LTPOS Tagger and Chunker, Morphological Analyzer morpha

This is historically the first option. It has been widely tested. LTChunk was found to crash occasionally on extremely long sentences.

Install the chunker LTChunk and the morphological analyzer morpha first. LTChunk may be obtained from

http://www.ltg.ed.ac.uk/software/chunk/index.html

**Update: LTChunk is no longer distributed. I can provide a binary for Solaris or Linux. ALTERNATIVES:

- Treetager: http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/ this is the tagger we currently use
- TNT: see e.g. http://mokk.bme.hu/resources/hunpos (Linux, Mac, Win) good and more performant than LTPos
- Ratnaparkhi: e.g. http://www.cogsci.ed.ac.uk/ jamesc/taggers/MXPOST.html good and more performant than LTPos

if you want to use a different tagger: ATTENTION: needs to produce Penn Treebank tags.

Morpha may be obtained from

http://www.cogs.susx.ac.uk/lab/nlp/carroll/morph.html

** Update: morpha is no longer distributed. I can provide a binary for Solaris or Linux. LT-TTT2 is the official successor.

Some scripts are provided for this option. You may have to adapt them.

Pipe your documents to the shell script PREPROCESS_XXXX.csh.

e.g.

```
$ cat /home/netapp/gschneid/LTPos/EVAL-Carroll/suste-raw | csh PREPROCESS_LTPos.csh |
more
$ cat /home/netapp/gschneid/LTPos/EVAL-Carroll/suste-raw | csh PREPROCESS_treetagger.csh
| more
```

STDOUT should output Prolog facts like the following:

```
w('jury' ,'NNP',['The_DT','Fulton_NNP','County_NNP','Grand_NNP','Jury_NNP'],'Fulton County Grand Jury').
w('say' ,'VBD',['said_VBD'],'said').
w('friday' ,'NNP',['Friday_NNP'],'Friday').
w('investigation' ,'NN',['an_DT','investigation_NN'],'investigation').
w('of' ,'IN',['of_IN'],'of').
w('election' ,'NN',['Atlanta_NNP','A_APOSTR','s_POS','recent_JJ','primary_JJ','election_NN'],'election').
w('produce' ,'VBD',['produced_VBD'],'produced').
w('evidence' ,'NN',['no_DT','evidence_NN'],'evidence').
w('that' ,'IN',['that_IN'],'that').
w('irregularity' ,'NNS',['any_DT','irregularities_NNS'],'irregularities').
w('take' ,'VBD',['took_VBD'],'took').
w('place' ,'NN',['place_NN'],'place').
w('s' ,'S',['S_S'],_).
```

!!!!!!!! If you create your own pipeline, CHECK CAREFULLY TO MAKE SURE YOUR PROLOG PREDICATES LOOK LIKE THIS !!!!!!!!! AS PRO3GRES IS A ROBUST SYSTEM, IT WILL ACCEPT SLIGHTLY MALFORMED INPUT, BUT ITS PERFORMANCE WILL BE MUCH LOWER!

If the Prolog facts you get do not look like Prolog code, you may need to correct the scripts to fit your input idiosyncracies. To test, consult(MYPREDICATES). If you get no error messages or only an error every 10'000 words or so, and the firsrt argument of w/4 corresponds to the following arguments, everything is ok (a possible source of error is that morpha needs to be started from its directory to work correctly). Direct the output of PREPROCESS.csh to a file. It will be the input for the parser.

8.2.3 Treetagger and Carafe Chunker

The main advantage of this option is that it certainly scales very well. We have parsed several billions of words like this on a Mac OS X server.

Carafe: http://sourceforge.net/projects/carafe. A binary distribution for Mac can be obtained from me. I have used this to parse the BNC and my other corpora, because LTChunk crashed. Recommendable.

The Treebank tagger has an option returning lemmas: recommended, because then you need no morphological analyzer (but the Perl scripts need to be manually changed)

OBS: Carafe is much greedier than the other chunkers, produces different output.

References

Abney, Steven. 1995. Chunks and dependencies: Bringing processing evidence to bear on syntax. In Jennifer Cole, Georgia Green, and Jerry Morgan, editors, *Computational Linguistics and the Foundations of Linguistic Theory*, pages 145–164. CSLI.

Basili, Roberto and Fabio Massimo Zanzotto. 2002. Parsing engineering and empirical robustness. *Journal of Natural Language Engineering*, 8/2-3.

Bies, Ann, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for Treebank II style Penn Treebank project. Technical report, University of Pennsylvania.

Bikel, Daniel. 2004. *On the Parameter Space of Lexicalized Statistical Parsing Models*. Ph.D. thesis, Department of Computer and Information Science.

Bod, Rens, Remko Scha, and Khalil Sima'an, editors. 2003. *Data-Oriented Parsing*. Center for the Study of Language and Information, Studies in Computational Linguistics (CSLI-SCL). Chicago University Press.

Brants, Thorsten and Matthew Crocker. 2000. Probabilistic parsing and psychological plausibility. In *Proceedings of 18th International Conference on Computational Linguistics COLING-2000*, Saarbrücken/Luxembourg/Nancy.

Buchholz, Sabine. 2002. *Memory-Based Grammatical Relation Finding*. Ph.D. thesis, University of Tilburg, Tilburg, Netherlands.

Burger, John D. and Sam Bayer. 2005. MITRE's Qanda at TREC-14. In E. M. Voorhees and Lori P. Buckland, editors, *The Fourteenth Text REtrieval Conference (TREC 2005) Notebook*.

Burke, M., A. Cahill, R. O'Donovan, J. van Genabith, and A. Way. 2004. Treebank-based acquisistion of wide-coverage, probabilistic LFG resources: Project overview, results and evaluation. In *The First International Joint Conference on Natural Language Processing (IJCNLP-04), Workshop "Beyond shallow analyses - Formalisms and statistical modeling for deep analyses"*, Sanya City, Hainan Island, China.

Campbell, Richard. 2004. Using linguistic principles to cover empty categories. In *Proceedings of ACL-2004*, pages 646–653, Barcelona, Spain.

Charniak, Eugene. 1996. Tree-bank grammar. Technical Report Technical Report CS-96-02, Department of Computer Science, Brown University.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the North American Chapter of the ACL*, pages 132–139.

Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 184–191, Philadelphia.

Collins, Michael. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Collins, Michael and James Brooks. 1995. Prepositional attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, Cambridge, MA.

Friedman, J. B. 1989. Computational testing of linguistic models in syntax and semantics. In I. S. Batori, W. Lenders, and W. Putschke, editors, *Computational Linguistics - Computerlinguistik*. de Gruyter, Berlin, pages 252–259.

Henderson, James. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of HLT-NAACL 2003*, Edmonton, Canada.

Hermjakob, Ulf. 2001. Parsing and question classification for question answering. In *Proceedings of the ACL 2001 Workshop on Open-Domain Question Answering*, Toulouse, France.

Hockenmaier, Julia and Mark Steedman. 2002. Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia.

Johnson, Mark. 2001. Joint and conditional estimation of tagging and parsing models. In *Proceedings of the 39th Meeting of the ACL*, pages 314–321, Toulouse, France.

Johnson, Mark. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Meeting of the ACL*, University of Pennsylvania, Philadelphia.

Klein, Dan and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan.

Magerman, David. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Meeting of the Association for Computational Linguistics*, Boston, MA.

Marcus, Mitch, Beatrice Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.

Merlo, Paola, Matthew Crocker, and Cathy Berthouzoz. 1997. Attaching multiple prepositional phrases: Generalized backed-off estimation. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, Providence, RI.

Merlo, Paola and Eva Esteve Ferrer. 2006. The notion of argument in PP attachment. *Computational Linguistics*, 32(2):341 – 378.

Mikheev, Andrei. 1997. Automatic rule induction for unknown word guessing. *Computational Linguistics*, 23(3):405–423.

Minnen, Guido, John Carroll, and Darren Pearce. 2000. Applied morphological generation. In *Proceedings of the 1st International Natural Language Generation Conference (INLG)*, Mitzpe Ramon, Israel.

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2005. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee, and Oi Yee Kwong, editors, *Natural Language Processing - IJCNLP 2004*, pages 684–693. Springer.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932.

Quirk, Randolph, Sidney Greenbaum, Geoffrey Leech, and Jan Svartvik. 1985. *A comprehensive grammar of the English language. 11th edn.* Longman, London.

Ratnaparkhi, Adwait. 1996. A Maximum Entropy Part-Of-Speech tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*, University of Pennsylvania.

Ratnaparkhi, Adwait. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Brown University, Providence, Rhode Island.

Riezler, Stefan, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, Philadephia, PA.

Tesnière, Lucien. 1959. *Eléments de Syntaxe Structurale*. Librairie Klincksieck, Paris.

Younger, D. H. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189-208.