Two-level probabilistic grammars for natural language parsing
Infante Lopez, G.G.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 7

Sequences as Features

7.1 Introduction

In the parsing activities and methods discussed so far in this thesis, we set out to discover syntactic structure, and in particular word dependents, using only sequences of POS tags. In contrast, nearly all other parsing approaches discussed in the literature use both sequences of POS tags and sequences of grammatical relations (GRs). Grammatical relations are labels describing the relation between the main verb and its dependents and they can be viewed as a kind of non-terminal labels. This observation suggests an obvious research question: which of the two types of information helps more for the discovery of word dependents, sequences of POS tags or sequences of GRs? This is our main research question in this chapter. Let us make more precise what it means.

In order to obtain phrase structures like the ones retrieved in (Collins, 1999), the dependents of a POS tag should consist of pairs of POS tags and non-terminal labels instead of sequences of POS tags alone (Markovian rules capture such pairs; see Chapter 4 for details on how PCW-grammars capture them). Like sequences of POS tags, sequences of pairs of POS tags and non-terminal labels can be viewed as instances of a regular language: a regular language whose alphabet is the product of the set of possible POS tags and the set of possible non-terminal labels. Moreover, they can be viewed as instances of the combination of two different regular languages: one regular language modeling sequences of POS tags, and another regular language modeling sequences of non-terminal labels. Under this perspective, it is clear that Chapters 5 and 6 only use the first regular language, while non-lexicalized approaches use the second regular language, and Markovian rules use a combination of the two.

From the literature, it is clear that combining the regular language of POS tags and the regular language of non-terminal labels boosts parsing performance, but it is not clear why this is the case. Chapter 5 suggests that lexicalization improves the quality
Chapter 7. Sequences as Features

of the automata modeling sequences of POS tags, but it does not provide any insight about the differences or the interplay between these two regular languages.

In this chapter we design and implement experiments for exploring the differences between the regular language of POS tags and the regular language of non-terminal labels in a parsing setup. Our research aims at quantifying the difference between the two and at understanding their contribution to parsing performance. In order to clearly assess the contribution of these two features, we need to carry out an evaluation in terms of a task that clearly isolates the two regular languages. We decided to use the task of detecting and labeling dependents of the main verb of a sentence. Labels describing the relation between the main verb and its dependents are what we call grammatical relations (GRs), and they can be viewed as a kind of non-terminal labels.

We present two different approaches for dealing with the task of finding grammatical relations. In the first approach, we develop two grammars: one for detecting dependents and another for labeling them. The first grammar uses sequences of POS tags as the main feature for detecting dependents, and the second grammar uses sequences of GRs as the main feature for labeling the dependents found by the first grammar. The task of detecting and labeling dependents as a whole is done by cascading these two grammars. In the second approach, we build a single grammar that uses sequences of GRs as the main feature for detecting dependents and for labeling them. The task of detecting and labeling dependents as a whole is done in one go by this grammar. The two approaches differ in that the first one uses sequences of GRs and sequences of POS tags, while the second only uses sequences of GRs.

We show that English GRs follow a very strict sequential order, but not as strict as POS tags of verbal dependents. We show that the latter is more effective for detecting and labeling dependents, and, hence, it provides a more reliable instrument for detecting them. We argue that this feature is responsible for boosting parsing performance.

The rest of the chapter is organized as follows. Section 7.2 details the task we use for testing the two features. Section 7.3 explains how to encode this task as parsing. Section 7.4 discusses how the training material used in our experiments is obtained. Section 7.5 explains how the grammars we use are built. Section 7.6 discusses the optimization phase for those grammars. Section 7.7 shows the experimental results; Section 7.8 discusses related work and, Section 7.9 concludes the chapter.

### 7.2 Detecting and Labeling Main Verb Dependents

The task we use for our experiments is to find main verbs dependents and to determine their GR. In this section we describe the selected task as a black-box procedure. We specify its input and its output. The input of the task consists of the following items:
7.3. PCW-Grammars for Detecting and Labeling Arguments

1. the main verb of the sentence.
2. the head word for each of the chunks into which the sentence has been split, and
3. the POS tags for the heads of the chunks.

The definition of chunks becomes clearer in the next section. For now it is enough to know that the sentences is chunked and that not all the words are used. Figure 7.1 shows an example of the input data.

<table>
<thead>
<tr>
<th>Pierre Vinken</th>
<th>61 years</th>
<th>old</th>
<th>joined</th>
<th>[the board]</th>
<th>[as]</th>
<th>[a nonexecutive director]</th>
<th>[Nov. 29].</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP</td>
<td>NNS</td>
<td>JJ</td>
<td>VBD*</td>
<td>NN</td>
<td>PP</td>
<td>NN</td>
<td>CD</td>
</tr>
</tbody>
</table>

Figure 7.1: Example of the information to be parsed by the grammars we build.

The output consists of a yes/no tag for each element in the input string. A POS tag marked yes implies that the tag depends on the main verb. If a POS tag is marked yes, the outputs has to specify the GR between the POS tags and the main verb.

The desired output for the input in Figure 7.1 is shown in Figure 7.2. Tags labeled yes have been replaced by links between the POS tags and the main verb.

![Figure 7.2: Information we use from each tree in the PTB.](image)

Note that not all POS tags in our example sentence bare a relation to the main verb. More generally, there might be POS tags that depend on the main verb but whose relation cannot be labeled by any of the labels we define later in this chapter. These links receive the NO-FUNC label. It is important to distinguish between the POS tags that do not have a relation to the main verb and those that depend syntactically on the main verb but whose relation cannot be labeled. The former are marked with the no tag, while the latter are marked with the yes tag and the GR is NO-FUNC. See Figure 7.2 for an example.

7.3 PCW-Grammars for Detecting and Labeling Arguments

In order to determine the contribution of the two kinds of information (sequences of POS tags and sequences of GRs), we set up the task of detecting and labeling as a
combination of two independent tasks. The first one is to find the dependents of the main verb, and the second to label them.

In order to try to use sequences of POS tags and sequences of GRs as features, we codify GRs in pre-terminal symbols. Figure 7.3 shows an example. It shows the verb dependents from Figure 7.1: nnp nn pp, and cd, with labels as pictured, while nns jj, and nn do not hold any relation to the main verb and, consequently, they are not linked or labeled and not shown in Figure 7.3.

In Figure 7.3, we can clearly distinguish the two regular languages that can be used for detecting dependents of verbs: the sequences NP-SBJ and NP-OBJ PP-CLR NP-TMP are instances of the regular languages whose alphabet is the set of possible GRs, while the sequences nnp and nn pp cd are instances of the regular language whose alphabet is the set of possible POS tags.

We build 3 different grammars:

1. A grammar $G_D$ that aims at detecting main verb dependents. This grammar uses automata that model sequences of POS tags. The parser that uses this grammar is fed as with all the POS tags.

2. A grammar $G_L$ that aims at labeling dependents. This grammar uses automata that model sequences of GRs. The parser that uses this grammar is fed with the POS tags that are believed to depend on the main verb. The result is a GR name for each POS tag in the input sentence.

3. A grammar $G$ that aims at detecting and labeling main dependents. This grammar uses automata that model sequences of GRs together with automata that models sequences of POS tags. The input and output of parsing with this grammar is as described in Section 7.2.
With these three grammars we can achieve the task described in Section 7.2 in two different ways:

1. We use $G_D$ for detecting dependents, and $G_L$ for labeling the dependents that $G_D$ outputs.

2. We use $G$ for detecting and labeling the main dependents.

Each of these three grammars are PCW-grammars (see Section 6.3), and they all are built using automata, just like the grammars we built in Chapters 5 and 6. In order to build them we have to carry out the following three steps:

1. generate the training material for training the automata,

2. optimize the automata, and

3. build the grammar.

Sections 7.5.1, 7.5.3 and 7.5.2 describe steps 1 and 3 for grammars for detecting dependents, labeling dependents and detecting and labeling dependents respectively. Section 7.6 explains which automata are optimized, and in which way.

For all grammars, Step 1 uses training material extracted from a labeled dependency grammar version of the PTB. The following section gives an overview of how we transform the PTB into labeled dependency trees.

### 7.4 Transforming the Penn Treebank to Labeled Dependency Structures

The set of GRs we aim to capture is a fixed set that is defined by the annotation schema followed in the Penn Treebank (PTB). We transformed the PTB into labeled dependency structures from which we induced our grammars.

All the training material we used comes from the PTB, hence the grammatical relations we are able to retrieve are those that are marked in the training material. We used chunklink.pl for transforming the PTB to labeled dependency structures and for marking all the information we need in the PTB (Buchholz, 2002). For detailed information on chunklink.pl, the reader is referred to the latter publication. In order to better understand the nature of the GRs to be found, we briefly describe how GRs are marked by chunklink.pl in the PTB.

The procedure for identifying links, chunks, and labels consists of four steps:

1. detecting words that may be heads of chunks,
2. drawing dependency relations (links) between these words,

3. assigning labels to these relations and

4. detecting chunks.

As to step (1) `chunklink.pl` detects heads using a head table, pretty much as explained in Section 2.1.1. Still, the application of tables described in Section 2.1.1 is different from the strategy used by `chunklink.pl`. In the latter, the head is either the right-most pre-terminal child that matches the (regular expression) POS tags list in the table, or all non-terminal children that match the (regular expression) constituent list. Consequently, there is a preference for lexical over non-lexical head children, but no preference within these groups. In the approach presented in Section 2.1.1, the list is ordered by preference and it also has an associated direction (starting left or right depending on the index of the table). The head detection algorithm first tries to find a child of the kind indicated by the first element of the list in the indicated direction, and stops as soon as it finds one. If no child of this kind can be found, the algorithm next looks for a child of the kind of the second element of the list, and so on, down to the last. If even the last kind of child cannot be found, the algorithm takes the left/rightmost child (of any kind) to be the head. The two different approaches, according to (Buchholz, 2002), differ mainly with respect to coordinated structures. For us, this difference is irrelevant, because we compare our approaches in each of the cases using the corresponding training material. Buchholz (2002) has an extensive description of the strategy followed by `chunklink.pl` for handling special cases that cannot be described in the table. Clearly, the result of both heuristics may be “unknown.” For these cases `chunklink.pl` returns an unknown head. In our experiments we discarded all sentences that contain at least one “unknown” head.

Once heads have been marked, the algorithm proceeds with step (2). Links (dependency relations) are drawn in a bottom-up fashion. In the process of drawing links, the tree ends up in an intermediate structure, which is then used in the third step. We describe here how the tree is transformed into a directed graph; the resulting graphs contain a partial dependency structure like the one used in previous chapters, plus links between non-terminals in the original tree and words in their yield. Step (3) uses the latter links for labeling relations in the dependency tree and completing the dependency tree; it finally eliminates them from the the graph producing a labeled dependency tree.

Step (2) starts by adding a link for each word in the yield of the tree that links the word to itself. Recall that, for each non-terminal, the algorithm knows which of its child nodes contains the head word. Step (2) traverses the tree in a bottom-up fashion: it adds a link between the current non-terminal and whatever the head child is pointing at. The result is a link between the non-terminal and a word in the yield. Step
(2) also redirects all links outgoing from the current non-terminal child to the same word the current non-terminal is pointing to. Note that step (2) outputs an incomplete dependency structure at word level, together with a graph where all non-terminals point to a word in the yield.

Step (3) uses the structure that step (2) outputs to add labeled links to words that remain to be linked in the dependency structure output by (3). A link between a non-terminal \( T \) and a word \( w \) in the yield encodes a dependency between \( w \) and the word that is found by descending in the tree from \( T \) to the yield, always following head children. These dependencies are the ones that step (3) draws. Recall that each non-terminal points to a word \( w \) in the yield. Step (3) adds a link for each non-terminal \( NT \) in the tree. The link goes from the word in the yield resulting from going down the non-terminal, always following the head children and the word \( w \) in the yield.

The pointer introduced by step (3) indicates dependency relations between syntactic constituents and head words. We are interested in relations between constituents and words, but to establish relations between pairs of words, step (3) traverses again the tree in a top-down fashion, and pushes the labeled pointer of the parent to the head child. If the syntactical part of the pointer label and of the non-terminal head children are identical, the pointer label stays the same. However, if the syntactic part is different, the head child label is prefixed to the pointer label, separated by the "\(/") symbol. When "pushing down" the function pointers, we lose the information about the level at which they were originally attached. In most cases this information is not relevant, as the function is defined by the combination of syntactic category and function tag. However, in the case of NPs without function tag, the level of attachment makes the difference between a complement (object) and an adjunct. In order to preserve this distinction, Buchholz (2002) adds the new function tag -OBJ to the following constituents if they occur without function tags and as siblings of lexical heads: NP, VP, ADJP, S, SBAR, SINV, SQ, SBARQ. These is done during phase (2).

Step (4) uses the links between words that were already present at step (2) for finding chunks. Buchholz (2002) defines a chunk to consist of a head, i.e., any word that has a labeled pointer, plus the continuous sequence of all words around it that have an unlabeled pointer to this head. Since labeled links between words were introduced by step (3), chunks are defined by all links between words that appear up to step (2) in the algorithm. This chunk correspond to the projection of the pre-terminal level in the original tree.

Since our experiments use the transformed version of the PTB and try to compare two different aspects of syntax, there are two issues we should discuss. The first is related to the nature of the transformation defined by chunklink.pl. Clearly, chunklink.pl takes a phrase structure as input and returns a dependency tree: it might be that this transformation discards some information from the PTB and that this
loss of information produces misleading results in our experiments. It seems to us that chunklink.pl does not define an invertible procedure, i.e., the dependency trees returned by it can not be transformed back to the original phase structure tree, because labels of some of the intermediate constituents are deleted during pruning (Buchholz, 2002, page 60). Buchholz (2002, page 59) also mentions loss of information regarding the original attachment position of grammatical functions. Despite all this, we think that chunklink.pl does not discard too much information and that the structures it produces are still meaningful.

The second issue to discuss is that, in theory, the transformation might be more beneficial for one of our experiments than for the other. It is not clear to us that this is indeed the case. All of our experiments are close to each other in that they use the same type of information and that the transformation does not favor a particular experiment.

7.5 Building the Grammars

For each of the tree grammars we build, we have to follow the same 3 steps:

1. extract the training material,
2. find the best automata, and
3. use the automata to build the grammar.

The optimization procedure we use for selecting the best automata, step (2), is the same for all grammars, while steps (1) and (3) are different for each particular grammar. Sections 7.5.1, 7.5.2, and 7.5.3 describe steps (1) and (3) for each of the grammars, while Section 7.6 describes step (2).

7.5.1 Grammars for Detecting Main Dependents

The grammar for detecting dependents is very similar to the grammars we built in Chapters 5 and 6. For each sentence parsed with this grammar, the parser outputs a dependency structure; the main verb dependents are found in this dependency structure.

Extracting Training Material

In order to obtain training material we transformed the PTB, sections 11–19, as explained in Section 7.4. For each dependency tree in the transformed TB, we extracted a sample set of right and left sequences of dependents. Figure 7.4 shows an example of a dependency tree, and Table 7.1 shows the sample sets of right and left dependents we extracted from it. We built two different sample bags per POS tag, one containing
7.5. Building the Grammars

all instances of left dependents and one containing all instances of right dependents. For each of the bags we built an automaton. The description of how to build an automaton from a bag of samples and the steps we follow for optimizing all automata are discussed in Section 7.6.

<table>
<thead>
<tr>
<th>POS</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP</td>
<td>NNP COMMA COMMA</td>
<td>NNP</td>
</tr>
<tr>
<td>COMMA</td>
<td>NNP</td>
<td>NNP</td>
</tr>
<tr>
<td>NNS</td>
<td>NNS</td>
<td>NNS</td>
</tr>
<tr>
<td>JJ</td>
<td>JJ NNS</td>
<td>JJ</td>
</tr>
<tr>
<td>COMMA</td>
<td>COMMA</td>
<td>COMMA</td>
</tr>
<tr>
<td>VBD</td>
<td>VBD NNP</td>
<td>VBD NNP PP CD DOT</td>
</tr>
<tr>
<td>NN</td>
<td>NN</td>
<td>NN</td>
</tr>
<tr>
<td>PP</td>
<td>PP</td>
<td>PP NN</td>
</tr>
<tr>
<td>NN</td>
<td>NN</td>
<td>NN</td>
</tr>
<tr>
<td>CD</td>
<td>CD</td>
<td>CD</td>
</tr>
<tr>
<td>DOT</td>
<td>DOT</td>
<td>DOT</td>
</tr>
</tbody>
</table>

Table 7.1: Instances of left and right dependents extracted from the tree in Figure 7.4.

Building the Grammar

Once the training material has been extracted, we build two different automata per POS tag, one modeling left dependents and one modeling right dependents. Let \( POS \) be the set of possible POS tags, and let \( w \) be an element in \( POS \); let \( A_L^w \) and \( A_R^w \) be the two automata associated to it. Let \( G_L^w \) and \( G_R^w \) be the PCFGs equivalent to \( A_L^w \) and \( A_R^w \), respectively, following (Abney et al., 1999), and let \( S_L^w \) and \( S_R^w \) be the start symbols of \( G_L^w \) and \( G_R^w \), respectively. We build a grammar \( G_D \) with start symbol \( S \), by defining its meta-rules as the disjoint union of all rules in \( G_L^w \) and \( G_R^w \) (for all POS \( w \)), its set of pseudo-rules as the union of the sets \( \{ S \xrightarrow{s} S_L^w v^* S_R^w : v \in \{ VB, VBD, VBG, VBN, VBP, VBZ \} \} \). The grammar is designed in such a way that the grammar’s start symbol \( S \) only yields the head words of the sentences which are marked with the * symbol. The main difference between the grammar we built in this section and the grammars we built in
Chapters 5 and 6 is that the sentences that are parsed with this grammar have the main head verb marked. We design the grammar to take advantage of this information.

To understand our experiments, we need to take a closer at the probabilities, and specially assigned by the grammars to the tree languages involved. Here is an example. Figure 7.5 shows a tree generated by \( G_L \) together with its probability. Here, \( p(w_h w_1 w_2) p(w_h w_3 w_4) \) is the probability assigned by the automata to \( w_h w_1 w_2 \) and \( w_h w_3 w_4 \). In fact, this is a simplification of the probability; it does not affect the analysis we carry out later but it makes the analysis clearer.

\[
p_D(t) = p(w_h w_1 w_2) p(w_h w_3 w_4) \times p(t_1) \ldots p(t_4)
\]

(a) \( S \)

(b)

Figure 7.5: (a) An example of a structure retrieved by the grammar \( G_D \), and (b) the probability value \( G_D \) assigns to it.

### 7.5.2 Grammars for Labeling Main Dependents

The second grammar we build is for labeling dependents. The sentences this grammar process are supposed to be only the dependents of a verb. In other words, the grammar assumes that somehow the right dependents have been identified, the task of this grammar is to assign the correct label to the dependents. It assigns a label to all elements in the input string. The grammar built in the previous sections selects a set of candidate dependents, and the selected dependents are fed to grammars described in the present section.

Sequences of GRs are modeled as instances of regular languages. Every verb tag has two automata associated to it: one modeling the sequence of left GRs and one modeling sequences of right GRs. A sequence of left (right) GRs is then an instance of the left (right) automata.

The grammar we build in this chapter is similar to the grammars built in Chapters 5 and 6 in that automat are used for building meta-rules. In contrast, automat are used to model sequences of GRs instead of sequences of POS tags. Figure 7.6 shows an example of a possible tree. From the figure, it is clear that GRs are encoded in preterminal symbols. All trees in the tree language defined by this grammar are flat trees of depth two. GRs are at depth one and they are modeled with automat and meta-rules. The yield of the tree is at depth two and it is modeled using pseudo-rules.
7.5. **Building the Grammars**

pseudo-rules rewrite GR names into a POS tag and they are read from the tree-bank; their probabilities are computed using the maximum likelihood estimator (Manning and Schütze, 1999).

Observe that these are w-trees, and not CFG trees; all meta-derivations that took place to produce nodes at depth 1 remain hidden. Hence, the sequence of GRs to the right and to the left of the main verb are instances of the regular languages modeling right or left GRs, respectively.

Summing up, we build a grammar for labeling GRs by combining two techniques for estimating probabilities for rules: the techniques presented in Chapters 5 and 6 for estimating probabilities for meta-rules, and maximum likelihood estimators for estimating probabilities for pseudo-rules.

**Extracting Training Material**

For this grammar we build two automata per verb POS tag, one modeling left GRs and one modeling right GRs. In order to extract the training material required for this grammar, we discarded all information not related to GRs from the transformed PTB. Figure 7.6 shows an example of the information we kept from the tree in Figure 7.2.

Figure 7.6: The tree representation we use, extracted from tree in Figure 7.2.

From the tree in Figure 7.6 we extract two kinds of information. The first kind is used to model meta-rules yielding GRs, i.e., the first level of the output trees, while the second kind of information is used to model pseudo-rules that rewrite names of GRs into POS tags, i.e., the third level of the output tree.

We first discuss the extraction of the material to build the automata. For this purpose we build two training bags, one containing right GRs and the other containing left GRs. Table 7.2 shows all instances to be added to the training material extracted from the tree in Figure 7.2.

We build two sample bags per verb POS tag, one containing all instances of left sequences of GRs and one containing all instances of right sequences of GRs. For each of the resulting bags we build one automaton. The description of how to build an automaton from a bag of samples, and the steps we follow for optimizing all automata, are discussed in Section 7.6.
Chapter 7. Sequences as Features

<table>
<thead>
<tr>
<th>VBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
</tr>
<tr>
<td>NP-SBJ VBD</td>
</tr>
</tbody>
</table>

Table 7.2: Data extracted from the tree in Figure 7.2. Left dependents should be read from right to left.

In contrast to Chapters 5 and 6, where probabilities for pseudo-rules were hand-coded, for this grammar probabilities for pseudo-rules have to be estimated from the training material. This is the case because pseudo-rules in Chapters 5 and 6 could be rewritten in only one way. For the present grammar, this is no longer the case. Left hand symbols of pseudo-rules are GRs, and these names can yield different POS tags. In order to estimate probabilities, we extracted all pairs of (GR, POS) from the training material and put them aside in only one bag. Table 7.3 shows the instances of pairs extracted from the tree in Figure 7.2.

<table>
<thead>
<tr>
<th>GR</th>
<th>POS tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP-SBJ</td>
<td>npn</td>
</tr>
<tr>
<td>NP-OBJ</td>
<td>nn</td>
</tr>
<tr>
<td>PP-CLR</td>
<td>pp</td>
</tr>
<tr>
<td>NP-TMP</td>
<td>cd</td>
</tr>
<tr>
<td>NO-FUNC</td>
<td>dot</td>
</tr>
</tbody>
</table>

Table 7.3: Pairs of GRs and POS tags extracted from tree in Figure 7.2.

Building a Grammar for Labeling

For building a grammar for labeling, we have to estimate two sets of probabilities and rules. On the one hand, we have to estimate meta-rules and their probabilities, and on the other hand we have to estimate the probabilities for pseudo-rules. The estimation of meta-rules and their probabilities is done by inducing an automaton for each of the sample sets of sequences of GRs.

Once the training material for meta-rules has been extracted, we build two automata per POS tag, one modeling left sequences of GRs and one modeling right sequences of GRs. Let $VS$ be the set of possible verb tags, let $v$ an element in $VS$; and $A_L^v$ and $A_R^v$ the two automata associated with it. Let $G_L^v$ and $G_R^v$ be the PCFGs equivalent to $A_L^v$ and $A_R^v$, respectively, and let $S_L^v$ and $S_R^v$ be the start symbols of $G_L^v$ and $G_R^v$, respectively. We build a grammar $G_L$ with start symbol $S$, by defining its meta-rules as the disjoint union of all rules in $G_L^v$ and $G_R^v$ (for all verb POS tags $v$), and its set of
pseudo-rules as the union of the two sets. One set, given by

\[ \{ S \xrightarrow{s} S_L^v v^* S_R^v : v \in VS \}, \]

is in charge of connecting the automata modeling left sequences of GRs with the automata modeling right sequences of GRs. The second set, given by

\[ \{ GR \xrightarrow{s} w : w \in POS \}, \]

where GR is the name of a GR, w is a POS tag, and p the probability associated to the rule, is computed using the pairs of (GR, POS) extracted from the training material, using the maximum likelihood estimator.

\[
p_L(t) = p(GR_1 \ldots GR_4) \times p(GR_1 \xrightarrow{s} w_1) \times \ldots \times p(GR_4 \xrightarrow{s} w_4)
\]

Figure 7.7: (a) An example of a structure retrieved by the grammar G_L, and (b) its probability value.

Again, the key to understanding our experiments lies in the way G_L assigns probability mass to its grammatical trees. Figure 7.7 shows a tree generated by G_L together with the probability assigned to it. In the figure, \( p(GR_i \xrightarrow{s} w_i) \) refers to the probability assigned to the rule \( GR_i \xrightarrow{s} w_i \) and \( p(GR_1 \ldots GR_4) \) is a simplification of the probability associated to the string \( GR_1 \ldots GR_4 \) by the automata modeling sequences of GRs.

If the grammar for labeling dependents is fed with the dependents found by the grammar for detecting dependents, the probability associated to a tree like the one pictured in Figure 7.8 is

\[
p_{cascading}(t) = p_P(t) \times p_L(t) \]

\[
= p(GR_1 \ldots GR_4) \times p(GR_1 \xrightarrow{s} w_1) \ldots p(GR_4 \xrightarrow{s} w_4) \times p(w_1 w_2) p(w_3 w_4) \times p(t_1) \ldots p(t_4)
\]
That is, \( p_{\text{cascading}} \) is the product of the probability assigned by \( G_L \) and \( G_D \), the main difference between this probability and \( p_{\text{one-go}} \) (the one assigned by the grammar \( G \) defined in Section 7.5.3 below), is that \( p_{\text{cascading}} \) uses the probability of the sequences \( w_h w_1 w_2 \) and \( w_h w_3 w_4 \) for detecting and labeling dependents.

Summing up, we have two probability distributions for the very same task, one of the distributions uses one more feature. An empirical comparison of these two distributions would provide us with information about the value of the extra feature; this is what we turn to in Section 7.7.

![Diagram](image)

Figure 7.8: The result of cascading the grammars for detecting and labeling dependents.

### 7.5.3 Grammars for Detecting and Labeling Main Dependents

This grammar to be defined in this section does in one go what the two grammars in Section 7.5.1 and 7.5.2 do in two steps.

**Extracting Training Material**

The training material we used for building this grammar is the union of the training materials we used for building the two previous grammars.

**Building the Grammar**

The automata we used for building this grammar are the same as the automata used in the previous two grammars, but the set of rules is different. Let \( \text{POS} \) be the set of possible POS tags, let \( w \) be a an element in \( \text{POS} \); let \( A^w_L \) and \( A^w_R \) be the two automata built for each POS tag in Section 7.5.1. Let \( \text{VS} \) be the set of possible verb tags, \( v \) an element in \( \text{VS} \); let \( A^v_L \) and \( A^v_R \) the two automata built for verb tags in Section 7.5.2. Let \( G^w_L, G^w_R, G^w_L, \text{ and } G^w_R \) be the PCFGs equivalent to \( A^w_L, A^w_R, A^v_L \) and \( A^v_R \), respectively, and let \( S^w_L, S^w_R, S^v_L \) and \( S^v_R \) be the start symbols of \( G^w_L \) and \( G^w_R \), respectively. We build a grammar \( G \) with start symbol \( S \), by defining its meta-rules as the disjoint union of
all rules in $G_L^v$, $G_R^v$, $G_L^w$ and $G_R^w$, for all POS tags and all verbs tags, while its set of pseudo-rules is the union of the following sets:

\[ \{ S \xrightarrow{s} S_L^v w S_R^w : v \in VS \}, \{ W \xrightarrow{s} S_L^w w S_R^v : w \in POS \}, \text{ and} \]

\[ \{ GR \xrightarrow{s} p S_L^w w S_R^v : w \in POS \}, \]

where $p$ is the probability assigned to the rule $\{ GR \xrightarrow{s} w : w \in POS \}$ in Section 7.5.2.

The difference between this grammar and the grammar for detecting dependents is that this grammar uses sequences of GRs for detecting the main dependents, while the grammar for detecting dependents uses sequences of POS tags.

Figure 7.9 shows an example of a tree accepted by $G$ together with the probability $G$ assigns to it.

Figure 7.9: (a) An example of a structure retrieved by the grammar $G$, and (b) its probability value.

Now that we have the tree probability distributions we can establish the relation between the two. Let $p_{\text{cascading}}$ be the probability distribution generated over trees by cascading the two first grammars, and let $p_{\text{one-go}}$ be exactly the probability distribution generated by $G$. The probability distributions $p_{\text{one-go}}$ and $p_{\text{cascading}}$ assign probabilities to the same set of trees, and the two are related as follows

\[ p_{\text{cascading}}(t) = p_{\text{one-go}}(t) \times p(w_1 w_2 w_3 w_4) \]  

(7.2)

As is clear from Equation 7.2, the difference between the two distributions is the probability of the sequence of POS tags $w_1 \ldots w_4$.

7.6 Optimizing Automata

Let $T$ be a bag of training material extracted from the transformed tree-bank. The nature of $T$ depends on the grammar we are trying to induce. But since we use the
same technique for optimizing all automata, we describe the procedure for a general bag.

Recall from Section 2.2.2 that we use two different measures for evaluating the quality of automata. Let $Q$ be a test bag extracted as $T$. As before, we use perplexity (PP) and missed samples (MS) to evaluate the quality of a probabilistic automaton. A PP value close to 1 indicates that the automaton is almost certain about the next step while reading the string. MS counts the number of strings in the test sample $Q$ that the automaton failed to accept.

We search for the value of $\alpha$ that minimizes $q = \sqrt{PP^2 + MS^2}$ (see Chapter 6.4.2, page 103, for the motivation of this function). In Figures 7.10 and 7.11 we have plotted $\alpha$ vs. PP, MS and $q$ for all verb POS tags used in the grammar for detecting the main dependents of verbs. Table 7.4 shows the values of $\alpha$ that produce the minimum value of $q$.

<table>
<thead>
<tr>
<th>POS tag</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB</td>
<td>0.0004</td>
<td>0.0004</td>
</tr>
<tr>
<td>VBD</td>
<td>0.0004</td>
<td>0.0002</td>
</tr>
<tr>
<td>VBG</td>
<td>0.0002</td>
<td>0.0004</td>
</tr>
<tr>
<td>VBN</td>
<td>0.0005</td>
<td>0.0004</td>
</tr>
<tr>
<td>VBP</td>
<td>0.0004</td>
<td>0.0004</td>
</tr>
<tr>
<td>VBZ</td>
<td>0.0004</td>
<td>0.0004</td>
</tr>
</tbody>
</table>

Table 7.4: Optimal values of PP and MS for automata used for labeling dependents.

In Figures 7.12 and 7.13 we have plotted $\alpha$ vs. PP, MS and $q$ for all verb POS tags used in the grammar for labeling dependents, an instance of the regular languages these automata model are GRs sequences. Table 7.5 shows the values of $\alpha$ that produce the minimum value of $q$. Recall from Section 7.5.2 that we build one automaton per verb POS tag.

Note that, even though the PP values for automata modeling sequences of GRs and the PP values for automata modeling POS tags are close to each other, the difference between their MSs is remarkable. We think that data sparseness affects the modeling of GRs much more than the modeling of POS tags. This sparseness prevents the MDI algorithm from inducing a proper language for GRs.

### 7.7 Experiments

For our experiments we shuffle the PTB sections 10 to 19 into 10 different sets. We run the experiments using set 1 as the test set and sets 2 to 10 as training sets. The tuning
7.7. Experiments

Figure 7.10: Values of PP and MS for automata used for labeling.
Figure 7.11: Values of PP and MS for automata used for labeling.
7.7. **Experiments**

![Graphs showing the values of PP and MS for automata used for detecting dependents.](image)

**Figure 7.12**: Values of PP and MS for automata used for detecting dependents.
Figure 7.13: Values of PP and MS for automata used for detecting dependents.
7.7. Experiments

<table>
<thead>
<tr>
<th>POS tag</th>
<th>Alpha</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB</td>
<td>0.00015</td>
<td>0.00015</td>
<td></td>
</tr>
<tr>
<td>VBD</td>
<td>0.00020</td>
<td>0.00010</td>
<td></td>
</tr>
<tr>
<td>VBG</td>
<td>0.00030</td>
<td>0.00020</td>
<td></td>
</tr>
<tr>
<td>VBN</td>
<td>0.00020</td>
<td>0.00020</td>
<td></td>
</tr>
<tr>
<td>VBP</td>
<td>0.00050</td>
<td>0.00050</td>
<td></td>
</tr>
<tr>
<td>VBZ</td>
<td>0.00030</td>
<td>0.00015</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.5: Optimal values of alpha for automata used for detecting dependents.

samples were extracted from Section 00. All the sentences we fed to the parser have the main head marked; all sentences whose main head was not tagged as a verb were filtered out.

We start by performing the whole task (detecting dependents and labeling their relation with the main verb) by the two different approaches; results are shown in Table 7.7. These results, together with Equation 7.2, answer one of our main research questions, namely what is the importance of the sequences of POS tags for parsing.

Recall from Equation 7.2 that the only difference between the two probability distributions $p_{one-go}$ and $p_{cascading}$ is the probability that $p_{cascading}$ associates to sequences of POS tags. Note also that the grammar that does not take it into account, namely $G$, performs significantly worse than the one that does take this sequence into account. From this we can conclude that the 10% jump in performance is due to the use of this specific information. The grammar $G_L$ for labeling dependents allows us to quantify how effective are the sequences of GRs together with pseudo-rules $GR \xrightarrow{s} w$ for labeling GRs. To isolate these features, we used grammar $G_L$ for labeling dependents that are known to be the right dependents. We extracted the correct sequences of dependents from the gold standard and used grammar $G_L$ for labeling them. Table 7.7 shows the results of this experiment.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Precision</th>
<th>Recall</th>
<th>$f_{\beta=1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascading</td>
<td>0.73</td>
<td>0.73</td>
<td>0.73</td>
</tr>
<tr>
<td>One Go</td>
<td>0.65</td>
<td>0.67</td>
<td>0.66</td>
</tr>
</tbody>
</table>

Table 7.6: The results on detecting and labeling dependents of main verbs.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>$f_{\beta=1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.76</td>
<td>0.76</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Table 7.7: Results of the experiment on labeling gold standard dependents.
The experimental results show that labeling is not a trivial task. The score obtained is low, even more so if we take into account that the sentences we fed to the parser consisted only of correct dependents. We think that the poor performance of this grammar is due to the data sparseness problem mentioned above, because there is a high amount of MS in the automata that model GRs.

The two grammars in the first approach allow us to quantify how the errors percolated from detecting dependents to labeling them. Now, the only aspect of the task that is left is to study is the detection of dependents. Table 7.8 shows the results of the experiment to assess the goodness of $G_D$ for detecting dependents.

<table>
<thead>
<tr>
<th>Precision</th>
<th>Recall</th>
<th>$f_{\beta=1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.85</td>
<td>0.88</td>
<td>0.86</td>
</tr>
</tbody>
</table>

Table 7.8: Results of the experiment on detecting dependents.

We can see how sensitive the task of labeling dependents is to errors in its input. Table 7.7 states that the labeling precision drops from 0.76 to 0.73 when only the 85% of the arguments fed to the labeling grammar are correct.

### 7.8 Related Work

The task of finding GRs has usually been considered as a classification task (Buchholz, 2002). A classifier is trained to find relations and to decide the label of the relations that are found. The training material consists of sequences of 3-tuples (main verb, label, and context). In order to have a better impression of the difficulty of the task, Table 7.9 shows some baselines extracted from (Buchholz, 2002). To understand the table, it is important to note that ”no relations” refers to the absence of the predicted relation and that 0 divided by 0 is defined as 1. In contrast to approaches based on classifiers, we

<table>
<thead>
<tr>
<th>Description</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>always predict “no relation”</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>always predict NP-SBJ</td>
<td>6.85</td>
<td>30.73</td>
<td>11.20</td>
</tr>
<tr>
<td>most probable class for focus chunk type/POS</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>most probable class for focus word</td>
<td>31.21</td>
<td>1.07</td>
<td>2.07</td>
</tr>
<tr>
<td>most probable class for distance</td>
<td>49.43</td>
<td>37.30</td>
<td>42.51</td>
</tr>
</tbody>
</table>

Table 7.9: Some possible baselines. Results extracted from Table 3.2 in (Buchholz, 2002).

consider the task of finding GRs as a parsing task. We build grammars that specifically
try to find GRs. It is possible to find GRs as a side product of full parsing because full trees output by a parser can be transformed as we transformed PTB. In order to give an impression of state-of-the-art methods for finding and labeling main dependents, we compare experiments to the approach presented in (Buchholz, 2002).

<table>
<thead>
<tr>
<th>Approach</th>
<th>Precision</th>
<th>Recall</th>
<th>(F_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascading</td>
<td>0.73</td>
<td>0.73</td>
<td>0.73</td>
</tr>
<tr>
<td>One-go</td>
<td>0.65</td>
<td>0.67</td>
<td>0.66</td>
</tr>
<tr>
<td>Memory Based Approach</td>
<td>0.86</td>
<td>0.80</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Table 7.10: Comparison to state-of-the-art techniques for detecting and labeling main verb dependents.

The main difference in the scores obtained by Buchholz (2002) and our own approach is probably due to the little information we used for performing the task. In contrast to our approach, Buchholz (2002) uses all kinds of features for detecting and labeling dependents.

### 7.9 Conclusions and Future Work

In this chapter we designed and implemented experiments for exploring the differences between the regular language of POS tags and the regular language of non-terminal labels in a parsing setup. Our research aimed at quantifying the difference between the two and at understanding their contribution to parsing performance. In order to clearly assess the contribution of these two features, we needed to carry out an evaluation in terms of a task that clearly isolates the two regular languages. We used the task of detecting and labeling dependents of the main verb of a sentence.

We presented two different approaches for dealing with the task of finding grammatical relations. In the first approach, we developed two grammars: one for detecting dependents and another for labeling them. The first grammar used sequences of POS tags as the main feature for detecting dependents, and the second grammar used sequences of GRs as the main feature for labeling the dependents found by the first grammar. The task of detecting and labeling dependents as a whole was done by cascading these two grammars. In the second approach, we built a single grammar that uses sequences of GRs as the main feature for detecting dependents and for labeling them. The task of detecting and labeling dependents as a whole was done in one go by this grammar. The two approaches differ in that the first one used sequences of GRs and sequences of POS tags, while the second only used sequences of GRs.

We showed that English GRs follow a very strict sequential order, but not as strict as POS tags of verbal dependents. We showed that the latter is more effective for de-
detecting and labeling dependents, and, hence, it provides a more reliable instrument for detecting them. Moreover, we have shown that sequences of POS tags are fundamental for parsing performance: they provide a reliable source for predicting and detecting dependents. Our experiments also show that sequences of GRs are not as reliable as sequences of POS tags.

The usual perspective on parsing is that all features that improve parsing performance are used without clearly stating why these features improve. Our approach aims at changing this perspective; we designed grammars and experiments for isolating, testing and explaining two particular features: sequences of POS tags and sequences of GRs, both for detecting and labeling and labeling dependents. PCW-grammars allow us to do these. Note that trees returned by our parser are flat trees and they cannot be modeled either by PCFGs nor by bilexical grammars. This is the case because some of the dependencies we model using automata in this chapter do not yield terminals but preterminal symbols.