Sequences of Part of Speech Tags vs. Sequences of Phrase Labels

How Do They Help in Parsing?

Gabriel Infante-Lopez[1] and Maarten de Rijke[2]

[1] FaMAF, Universidad Nacional de Córdoba, Córdoba, Argentina
gabriel@famaf.unc.edu.ar
[2] Informatics Institute, University of Amsterdam, The Netherlands
mdr@science.uva.nl

Abstract. We compare the contributions made by sequences of part of speech tags and sequences of phrase labels for the task of grammatical relation finding. Both are used for grammar induction, and we show that English labels of grammatical relations follow a very strict sequential order, but not as strict as POS tags, resulting in better performance of the latter on the relation finding task.

1 Introduction

Some approaches to parsing can be viewed as a simple context free parser with the special feature that the context free rules of the grammar used by the parser do not exist a priori [1–3]. Instead, there is a device for generating bodies of context free rules on demand. Collins [1] and Eisner [2] use Markov chains as the generative device, while Infante-Lopez and De Rijke [3] use the more general class of probabilistic automata. These devices are induced from sample instances obtained from tree-banks. The learning strategy consists of coping all bodies of rules inside the Penn Tree-bank (PTB) to a bodies of rules sample bag which is then treated as the sample bag of an *unknown* regular language. This unknown regular language is to be induced from the sample bag, which is, later on, used for generating new bodies of rules.

Usually, the induced regular language is described by means of a probabilistic automata. The quality of the resulting automata depends on many things; the alphabet of the target regular language being one. At least two such alphabets have been considered in the literature: Part of Speech (POS) tags and grammatical relations (GRs), where the latter are labels describing the relation between the main verb and its dependents; they can be viewed as a kind of non-terminal labels. Using one or the other alphabets for grammar induction might produce different results on the overall parsing task. Which of the two produces "better" automata, that produce "better rules," which in turn lead to "better" parsing scores? This is our main research question in this paper.

Let us provide some further motivation and explanations. In order to obtain phrase structures like the ones retrieved in [4], the dependents of a POS tag should consist

of pairs of POS tags and non-terminal labels instead of sequences of POS tags alone. Like sequences of POS tags, sequences of pairs of POS tags and non-terminal labels can be viewed as instances of a regular language: one 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 regular languages: one modeling sequences of POS tags, and another modeling sequences of non-terminal labels. Infante-Lopez and De Rijke [3] only use the first regular language for grammar induction, while non-lexicalized approaches [5] use the second regular language, and Markovian rules [4] use a combination of the two. Combining the regular language of POS tags and that of non-terminal labels boosts the overall parsing performance, cf., [4, 5], but it is not clear why this is the case. Infante-Lopez and De Rijke [3] suggest that lexicalization improves the quality of the automata modeling sequences of POS tags, but they do not provide any insight about the differences or the interplay between these two regular languages.

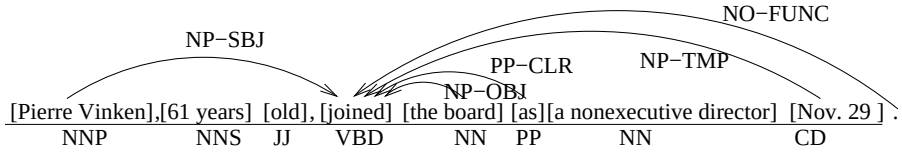

Fig. 1: Information we use from each tree in the PTB.

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. To address our research question we focus on a task that cleary isolates these two regular languages: detecting and labeling dependents of the main verb of a sentence. We present two approaches to dealing with this task. In the first, 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 overall task of detecting and labeling dependents is performed by cascading the two grammars. In the second approach, we build a single grammar that uses sequences of GRs as the main feature for detecting and labeling dependents. The overall task of is done in one go by this grammar. The two approaches differ in that the first uses sequences of GRs and sequences of POS tags, while the second only uses sequences of GRs.

English GRs are shown to follow a strict sequential order, but not as strict as POS tags of verbal dependents. Counterintuitively, the latter are more effective for detecting and labeling dependents, and, hence, provide a more reliable instrument for detecting GRs. This feature is responsible for boosting parsing performance.

In Section 2 we detail the task on which we focus; Section 3 builds the grammars used in the experiments. Section 4 argues for the appropriateness of the task on which we focus for our main research questions. Section 5 describes our experiments and answers these questions. We present related work in Section 6 and conclude in Section 7.

2 Task Definition

The task we use for our experiments is to find dependents of main verbs and to determine their GR. Given a sentence, the input for the task consists of the following: (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) POS tags for the heads of the chunks. The rest of the information in the sentence is discarded. The information below the line in Figure 1 shows an example of the input data.

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 output has to specify the GR between the POS tags and the main verb. An example of the desired output is shown in Figure 1. Tags labeled `yes` have been replaced by links between the POS tags and the main verb. *Not* all POS tags in our example sentence bare a relation to the main verb. More generally, there may be POS tags that depend on the main verb but whose relation cannot be labeled by any of the labels we define below. 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`; Figure 1 has an example.

In order to define the regular language of GRs, we codify GRs in pre-terminal symbols. As an example, Figure 2 shows the verb dependents from Figure 1, `nnp nn pp`, and `cd`, with labels as pictured, while `nns jj`, and `nn` are not in any relation to the main verb and, consequently, they are not linked or labeled and not shown in Figure 2. One 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.

3 Building Grammars

We build 3 grammars; each is a PCW-grammars (see Section 3.1 for details): G_D, G_L, and G. The grammar G_D aims to *detect* main verb dependents. It uses automata that model sequences of POS tags. The parser using G_D is fed with all POS tags. For each sentence parsed with this grammar, the parser outputs a dependency structure in which the main verb dependents are found. The grammar G_L aims to *label* dependents. It uses

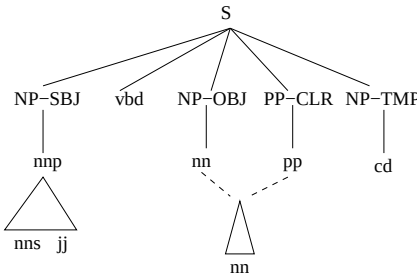

Fig. 2: The desired tree for the input in Figure 1. The subtree pictured as a triangle denotes that it can be adjoined to both points.

automata that model sequences of GRs. The parser using G_L 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. This grammar assumes that (somehow) the right dependents have been identified, and its task is to assign the correct label to the dependents; it assigns a label to all elements in the the input string. The grammar G aims to *detect* and *label* main dependents. It uses automata that model sequences of GRs together with automata that model sequences of POS tags. The input and output of parsing with G are as for the grammar G_D.

Using G_D, G_L, and G we define two ways to address the relation finding task described in Section 2: (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.

The three grammar are PCW-grammars (see Section 3.1). We build them following the same procedure: (1) we build a bodies of rules training set extracted from the PTB (see Section 3.2), (2) we induce an automaton from the training material (see Section 3.4), and, (3) we build a grammar using the automata induced in step 2 (see Section 3.3).

3.1 Grammatical Framework

We need a grammatical framework that models rule bodies as instances of a regular language and that allows us to transform automata to grammars as directly as possible. We use the grammatical framework of CW-grammars [6]. Based on PCFGs, they have a clear and well-understood mathematical background and we do not need to resort to ad-hoc parsing algorithms.

A *probabilistic constrained W-grammar* (PCW-grammar) is a two-level grammar consisting of two sets of PCF-like rules (*pseudo-rules* and *meta-rules*) and three pairwise disjoint sets of symbols (*variables*, *non-terminals* and *terminals*). Pseudo-rules and meta-rules provide mechanisms for building 'real' rewrite rules, which are built by first selecting a pseudo-rule, and then using meta-rules for instantiating the variables in the body of the pseudo-rule.

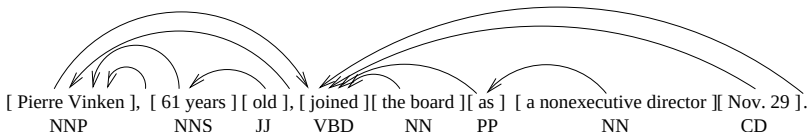

Fig. 3: A dependency tree from which we extracted training material.

Parsing PCW-grammars requires two steps: a generation-rule step followed by a tree-building step. Parsing with PCW-grammars can be viewed as parsing with PCF grammars. The main difference is that in PCW-parsing derivations for variables remain hidden in the final tree [6].

3.2 Training Material

The training material we use for building G_D, G_L and G always comes from sections 11–19 of the PTB. We use `chunklink.pl` [7] for transforming the PTB to labeled dependency structures and for marking all the information we use. Briefly, [7] 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. This chunk correspond to the projection of the pre-terminal level in the original tree. *Labels* are defined as concatenation of the non-terminals labels found in the PTB.

Clearly, `chunklink.pl` does *not* define an invertible procedure, i.e., its output dependency trees can *not* be mapped back to the original phrase structure tree, as labels of some intermediate constituents are deleted during pruning [7, p. 60]; some information regarding the original attachment position of grammatical functions is also lost. Despite this, `chunklink.pl` does not appear to discard too much information; the structures it produces are meaningful. All our experiments use the same type of information and the transformation performed with `chunklink.pl` does not favor one experiment over another.

After the transformation, the resulting trees contain information about chunks and labels (see Figure 1). From such trees, two further trees can be extracted, each containing information relevant to the 3 grammars we want to build. For the tree in Figure 1, the trees in Figures 3 and Figure 4 can be obtained. We use these derived trees for obtaining the training material. The precise tree to be used depends on the grammar we want to induce, as we will now explain.

meta-rules	pseudo-rules
$\overline{Adj} \xrightarrow{m}_{0.5} \overline{Adj}\,Adj$	$S \xrightarrow{s}_{1} \overline{Adj}\,Noun$
$\overline{Adj} \xrightarrow{m}_{0.5} Adj$	$Adj \xrightarrow{s}_{0.1} big$
	$Noun \xrightarrow{s}_{1} ball$
	$\vdots$

Table 1: Example of a PCW-grammar.

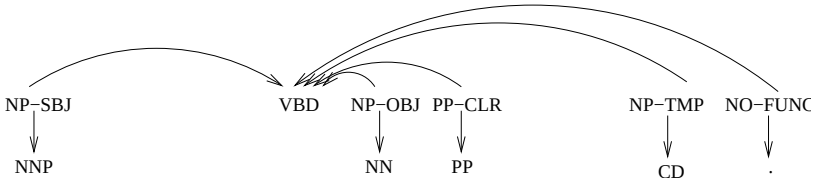

Fig. 4: The tree representation we use, extracted from tree in Figure 1.

For the grammar for detecting dependants G_D, the dependency trees used are like the one shown in Figure 3, and Table 2 shows the sample sets of right and left dependents we extracted from it.

POS	Left	Right
NNP	NNP	NNP COMMA NNS COMMA
COMMA	COMMA	COMMA
NNS	NNS	NNS JJ
JJ	JJ	JJ
COMMA	COMMA	COMMA
VBD	VBD NNP	VBD NN PP CD DOT
NN	NN	NN
PP	PP	PP NN
NN	NN	NN
CD	CD	CD
DOT	DOT	DOT

Table 2: Instances of left and right dependents extracted from the tree in Figure 3. The head always starts the string of dependants. Left dependants should be read backwards.

In contrast, for the grammar G_L we use trees like the one pictured in Figure 4. From such trees, 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 is used to model pseudo-rules that rewrite names of GRs into POS tags, i.e., the third level of the output tree. Table 3 shows all instances to be added to the training material extracted from the tree in Figure 1.

Probabilities of pseudo-rules in G_D were hand coded, because there is a one to one correspondence with left-hand symbols and the body of rules. For the present grammar, this is no longer the case. Here, left hand symbols of pseudo-rules are GRs, and these names can yield different POS tags. To estimate probabilities, we extracted all pairs of (GR, POS) from the training material and put them aside in only one bag. Table 4 shows

VBD	
Left	Right
NP-SBJ VBD	VBD NP-OBJ PP-CLR NP-TMP NO-FUNC

Table 3: Data extracted from the tree in Fig. 1. Left dependents should be read from right to left.

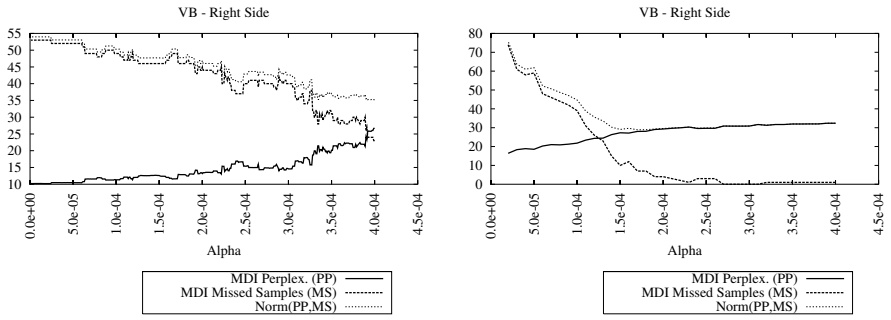

Fig. 5: Left and right plots for automata used in G_L and G_D.

the instances of pairs extracted from the tree in Figure 1. The training material used for building G is the union of the training material for G_L and G_D.

3.3 Defining Grammars

We start by building G_D. Once the training material has been extracted, we build two automata per POS tag, one modeling left dependents, the other right dependents. Let POS be the set of possible POS tags, w an element in POS, and A_L^w and A_R^w 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 [8], 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 $\{S \xrightarrow{s}_1 S_L^v v^* S_R^v : v \in \{\texttt{VB}, \texttt{VBD}, \texttt{VBG}, \texttt{VBN}, \texttt{VBP}, \texttt{VBZ}\}\}$. The grammar is designed in such a way that the start symbol S only yields the head words of the sentences which are marked with the $*$ symbol. That is, all sentences that are parsed using these grammars have one word marked with the $*$ symbol indicating that the marked word is the head of the sentence.

For G_L, automata are used to model sequences of GRs instead of POS tags. GRs are at depth one (see Figure 4) and they are modeled with automata and meta-rules. The yield of the tree is at depth two and it is modeled using pseudo-rules. The latter rewrite GR names into a POS tag and they are read from the tree-bank; their probabilities are computed using maximum likelihood estimation [9]. 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.

GR	NP-SBJ	NP-OBJ	PP-CLR	NP-TMP	NO-FUNC
POS tag	nnp	nn	pp	cd	dot

Table 4: Pairs of GRs and POS tags extracted from tree in Figure 1.

Once the training material for meta-rules has been extracted, we build two automata per GR, one modeling left sequences of GRs, the other right sequences of GRs. Let VS be the set of possible verb tags, 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 $\{S \xrightarrow{s}_1 S_L^v v^* S_R^v : v \in VS\}$, is connects automata modeling left sequences of GRs with automata modeling right sequences of GRs. The second set, given by $\{GR \xrightarrow{s}_p w : w \in POS\}$, where GR is the name of a GR, w is a POS tag, and p the probability associated to the rule, is computed using (GR, POS) pairs extracted from the training material, using maximum likelihood estimation.

The automata we use for building G are the same as those used in the previous two grammars, but the set of rules differs. Let POS be the set of possible POS tags, let w be an element in POS; let A_L^w and A_R^w be the two automata built for each POS tag for the grammar G_D. Let VS be the set of possible verb tags, v an element in VS; let A_L^v and A_R^v the two automata we built for verb tags for grammar G_L . Let G_L^v, G_R^v, G_L^w, and G_R^w be the PCFGs equivalent to A_L^v, A_R^v, A_L^w and A_R^w, respectively, and let S_L^v, S_R^v, S_L^w and S_R^w be the start symbols of G_L^v and G_R^v, 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^v and G_R^v, for all POS tags and all verbs tags, while its set of pseudo-rules is the union of the following sets: $\{S \xrightarrow{s}_1 S_L^v v^* S_R^v : v \in VS\}$, $\{W \xrightarrow{s}_1 S_L^w w S_R^w : w \in POS\}$, and $\{GR \xrightarrow{s}_p S_L^w w S_R^w : w \in POS\}$, where p is the probability assigned to the rule $\{GR \xrightarrow{s}_p w : w \in POS\}$ using maximun likelihood estimation.

3.4 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. Since we use the same technique for optimizing all automata, we describe the procedure for a general bag. We use *minimum discrimination information* (MDI) [10] algorithm for inducing the automata, and two different measure for evaluating them: perplexity (PP) and missed samples (MS). 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.

The MDI algorithm has one parameter: `alpha`. We search for the value of `alpha` that minimizes $q = \sqrt{PP^2 + MS^2}$ (see [6] for motivation), where both PP and MS depend on α. In Figure 5 we have plotted `alpha` vs. PP, MS and q for the `VB` tag used in the grammars G_L (left) and G_D (right). Even though the PP values for automata modeling sequences of GRs (left) and the PP values for automata modeling POS tags (right) are close to each other, the difference between their MSs is remarkable. Data

sparseness seems to affect the modeling of GRs much more than that of POS tags; it prevents the MDI algorithm from inducing a proper language for GRs.

4 Comparing Probability Distributions

The approach we follow to detect the value of sequences as features is to address the task of detecting and labeling arguments using two different strategies. One is to cascade the grammars G_L and G_D, while the second is to use G in one go. The first approach uses the sequence of POS as a feature while the second one does not. Let us take a closer look. We present the probabilities that each grammar assigns to its tree language. Consider the trees shown in Figure 6. G_D, G_L, and G generate the

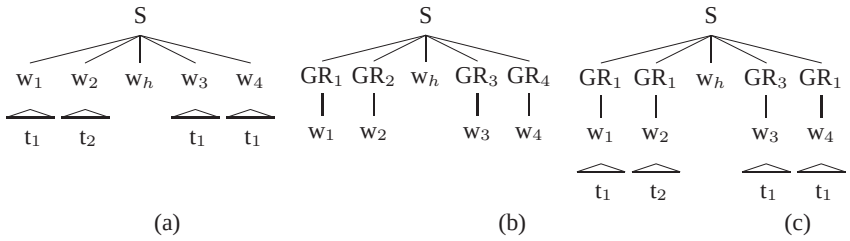

Fig. 6: (a) Example of a structure retrieved by the grammar G_D, (b) An example of a structure retrieved by the grammar G_L, and (c) The result of cascading the grammars for detecting and labeling dependents.

trees in Figure 6 (a), (b) and (c), respectively. The three grammars assigns probabilities $p_{G_D}(t|s)$, $p_L(t|w_1 \dots w_4)$, and $p_G(t|s)$ as defined in Figure 7. There, $p(w_h w_1 w_2)$,

$$
\begin{aligned}
p_{G_D}(t|s) &= p(w_h w_1 w_2) p(w_h w_3 w_4) p(t_1) \dots p(t_4) \\
p_{G_L}(t|w_1 \dots w_4) &= p(w_h GR_1 GR_2) p(w_h GR_3 GR_4) p(GR_1 \rightarrow w_1) \dots p(GR_4 \rightarrow w_4) \\
p_G(t|s)(t) &= p(w_h GR_1 GR_2) p(w_h GR_3 GR_4) p(GR_1 \rightarrow w_1) \dots p(GR_4 \rightarrow w_4) \times \\
&\qquad \times p(t_1) \dots p(t_4) \\
p_G(t|s) &= p_{\textit{one-go}}(t)
\end{aligned}
$$

Fig. 7: Probabilities $p_{G_D}(t|s)$, $p_{G_L}(t|w_1 \dots w_4)$, and $p_G(t|s)$ assigned by G_D, G_L and G, respectively.

$p(w_h w_3 w_4)$, $p(w_h GR_1 GR_2)$ and $p(w_h GR_3 GR_4)$ are the probabilities assigned by the automata to the strings $w_h w_1 w_2$, $w_h w_3 w_4$, $w_h GR_1 GR_2$, and $w_h GR_3 GR_4$, respectively, and similarly for $w_h GR_1 GR_2$) and $w_h GR_3 GR_4$). Further, $p(GR_i \xrightarrow{s} w_i)$ refers to the probability assigned to the rule $GR_i \xrightarrow{s} w_i$ and s is the concatenation of $yield(t_1) yield(t_2) w_h yield(t_3) yield(t_4)$.

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 pic-

tured in Figure 6.(c) is as follows

$$
\begin{aligned}
& p_{cascading}(t|s) = p_D(t) \times p_L(t) = \\
& = p(GR_1 \dots GR_4) \times \\
& \quad p(GR_1 \xrightarrow{s} w_1) \dots p(GR_4 \xrightarrow{s} w_4) \times \\
& \quad p(w_h w_1 w_2) p(w_h w_3 w_4) p(t_1) \dots p(t_4)
\end{aligned} \tag{1}
$$

We can now establish the relation between the two probabilities behind the two strategies we defined for solving the task. Let $p_{cascading}$ be the probability distribution generated over trees by cascading the two first grammars, and $p_{one\text{-}go}$ the probability distribution generated by G. Both $p_{one\text{-}go}$ and $p_{cascading}$ assign probabilities to the same set of trees, and the two are related as follows:

$$
p_{cascading}(t) = p_{one\text{-}go}(t) \times p(w_h w_1 w_2) p(w_h w_3 w_4). \tag{2}
$$

The difference between the two distributions is the probability of the sequence of POS tags $w_1 \dots w_4$.

Summing up, we have two probability distributions for the very same task, one uses an additional feature, namely, the sequence $w_1 \dots w_4$. An empirical comparison of these two distributions will provide us with information about the value of the additional feature; this is what we turn to in next.

5 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 samples were extracted from Section 00. All sentences fed to the parser have the main head marked; all sentences whose main head was not tagged as a verb are filtered out. First, we perform the whole task (detecting dependents and labeling their relation with the main verb) according to the two strategies; results are shown in Table 5; we observe a 10% difference in $f_{\beta=1}$ between the cascaded strategy and the "direct" strategy. This helps us answer our main research question (What is the importance of the sequences of POS tags for parsing?). Recall from Equation 2 that the only difference between $p_{one\text{-}go}$ and $p_{cascading}$ is that $p_{cascading}$ associates to sequences of POS tags. In other words, the 10% difference in performance between the two strategies is due to the use of this information.

The grammar G_L for labeling dependents allows us to quantify the effectiveness of sequences of GRs together with pseudo-rules $GR \xrightarrow{s} w$ for labeling GRs. To this end, we used grammar the G_L for labeling dependents that are known to be the right

Approach	Precision	Recall	$f_{\beta=1}$
Cascading	0.73	0.73	0.73
One Go	0.65	0.67	0.66

Table 5: The results on detecting and labeling main verbs dependents.

dependents. We extracted the correct sequences of dependents from the gold standard and used the grammar G_L for labeling them. Table 6 shows the results of this experiment; the results show that labeling is not a trivial task. The scores obtained are low, especially if we take into account that the sentences fed to the parser consisted only of correct dependents. The poor performance of this grammar is due to the data sparseness problem mentioned above: there is a large number of MS in the automata that model GRs. Moreover, the two grammars in the cascaded approach allow us to quantify how errors percolate from detecting dependents to labeling them. Now, the only aspect of the task that is left is to study is the detection of dependents. In Table 6 we see how sensitive the task of labeling dependents is to errors in its input: the labeling precision drops from 0.76 to 0.73 when only the 85% of the arguments fed to the labeling grammar are correct.

6 Related Work

The task of finding GRs has mostly been considered as a classification task [7]. A classifier is trained to find relations and to decide the label of the relations found. The training material consists of sequences of 3-tuples (main verb, label, and context). In contrast to approaches based on classifiers, we view the task of finding GRs as a parsing task. We build grammars that specifically try to find GRs. 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 [7]. She reports 0.86 and 0.80 for precision and recall respectively. Thesescores are better than ours, and the differences are probably due to the restricted amount of information we used for performing the task. In contrast, Buchholz [7] uses all kinds of features for detecting and labeling dependents.

7 Conclusions

The standard practice in parsing is to use all features that improve parsing performance without clearly stating why they improve. In contrast, we designed grammars and experiments for isolating and explaining two particular types of features: sequences of POS tags and sequences of GRs, both for detecting and labeling and labeling dependents.

We designed and implemented experiments for exploring the differences in contribution to the overall task of parsing between the regular language of POS tags and the regular language of GRs. To assess the contribution of these two features, we carried out an evaluation in terms of a task that clearly isolates the two regular languages.

Approach	Precision	Recall	$f_{\beta=1}$
Labeling Gold Standard	0.76	0.76	0.76
Detecting Dependents	0.85	0.88	0.86

Table 6: Results of the experiment on labeling gold standard dependents and detecting dependents.

We used the task of detecting and labeling dependents of the main verb of a sentence. We presented two approaches for addressing this task. For the first, 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 overall task of detecting and labeling dependents 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; here, the overall task was done in one go by this grammar. The first approach 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. The latter are more effective for detecting and labeling dependents, and, hence, provide a more reliable instrument for detecting them. We also showed that sequences of POS tags are fundamental for parsing performance: they provide a reliable source for predicting and detecting dependents.

Acknowledgments. Maarten de Rijke was supported by the Netherlands Organization for Scientific Research (NWO) under project numbers 017.001.190, 220-80-001, 264-70-050, 354-20-005, 612-13-001, 612.000.106, 612.000.207, 612.066.302, 612.069.-006, and 640.001.501.

References

1. Collins, M.: Three generative, lexicalized models for statistical parsing. In: Proc. 35th ACL. (1997)
2. Eisner, J.: Three new probabilistic models for dependency parsing: An exploration. In: Proc. COLING 1996. (1996)
3. Infante-Lopez, G., de Rijke, M.: Alternative approaches for generating bodies of grammar rules. In: Proc. 42nd ACL. (2004)
4. Collins, M.: Head-Driven Statistical Models for Natural Language Parsing. PhD thesis, University of Pennsylvania, PA. (1999)
5. Charniak, E.: Tree-bank Grammars. In: Proceedings *AAAI'96*, Portland, Oregon (1996)
6. Infante-Lopez, G.: Two-Level Probabilistic Grammars for Natural Language Parsing. PhD thesis, Universiteit van Amsterdam (2005)
7. Buchholz, S.: Memory-Based Grammatical Relation Finding. PhD thesis, Universiteit van Tilburg (2002)
8. Abney, S., McAllester, D., Pereira, F.: Relating probabilistic grammars and automata. In: Proc. 37th ACL. (1999) 542–549
9. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, MA (1999)
10. Thollard, F., Dupont, P., de la Higuera, C.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In: Proc. ICML, Stanford (2000)