Abstract

A system for 'tagging' (possibly ambiguous) words with their correct Part-of-Speech (POS) is constructed. The system has two components: a lexicon containing the set of possible POS tags for a given word, and rules which use a word's context to *eliminate* possible tags for a word. The Inductive Logic Programming (ILP) system Progol is used to induce these rules in the form of definite clauses. The final theory contained 908 clauses. For background knowledge, Progol uses a simple grammar, where the tags are terminals and predicates such as `nounp` (noun phrase) are nonterminals. Progol was altered to allow the caching of information about clauses generated during the induction process which greatly increased efficiency. The system achieved a per-word accuracy of 96.4% on known words drawn from sentences without quotation marks. This is on a par with other tagging systems induced from the same data [5, 2, 4] which all have accuracies in the range 96–97%. The per-sentence accuracy was 49.5%.

1 Introduction

In part-of-speech disambiguation or 'tagging', the aim is to correctly classify word tokens according to their part of speech. For example, in the sentence "Jane sat on the table", "table" is a noun, yet in the sentence "James decided to table the motion", "table" is a verb. Once disambiguation has been carried out, the word can be 'tagged' with its correct part of speech (POS) tag.

As Daelemans et al [5] note

> Automatic tagging is useful for a number of applications: as a preprocessing stage to parsing, in information retrieval, in text to speech systems, in corpus linguistics, etc.

In this report we report on the use of ILP to attack the tagging problem. However, a number of non-ILP approaches have been successfully applied to the problem, and these are now briefly described. Many of the following methods are ably described in [1].

The plan of this report is as follows. In Section 2, we overview related work. In Section 3, the basic approach to tagging is described. In Sections 4 and 5, we describe how ILP is used to induce rules for tagging, with Section 5 concentrating on issues specific to Progol, the ILP system used. Section 6 details our results and Section 7 analyses these results and discusses future work.

2 Related work

2.1 Hidden Markov model approaches

For a proper introduction to Hidden Markov models (HMMs) see [10], here we give only a very brief overview. An HMM consists of

1. A set of states.

2. A set of observation symbols.

3. State transition probabilities a_{ij} which give the probability of moving from state i to state j.

4. Emission probabilities $b_j(k)$, the probability of observing symbol k when in state j.

5. Initial state probabilities π_i, denoting the probability of starting in state i.

An HMM is *Markov* because the probability of moving to state j depends only the current state i, and it is *hidden Markov* because only the observation symbols are observed, not the states themselves.

Given sequences of observations, all of the probabilities that define an HMM (transition, emission and initial) can be estimated using a maximum likelihood procedure: the forward-backward algorithm. Given these probabilities the Viterbi algorithm can be used to find the most likely sequence of (hidden) states given a sequence of observations.

In a bigram HMM tagger, the states will often be POS tags and the observation symbols will be words. This amounts to assuming that the tag for the current word depends only on the preceding tag. From a sequence of observed words, the HMM tagger uses the Viterbi algorithm to find the most likely sequence of unobserved POS tags, and tags the words accordingly. In a trigram HMM tagger, the states are pairs of tags, so the tag for the current word depends only on the preceding two tags.

An HMM tagger can be trained using untagged text—which is an advantage of HMM tagging, although this can lead to unsatisfactory results [1]. Better results can be obtained, as expected, using tagged text.

In [4], the hidden states are again the POS tags, but now the observed symbols are not the words themselves, but *ambiguity classes*—the set of possible tags that a word can take. These ambiguity classes are easy to produce from tagged text. For example in the WSJ corpus the ambiguity class for the word "bear" is $[nn, vb, vbp]$, which means it can be respectively, a singular common noun (the animal), an infinitive (as in "to bear") or a non-3rd person singular present tense verb ("I bear him no grudge"). One can replace words by their ambiguity classes by a simple look up in a lexicon. This means that ambiguity classes can be considered as observables.

Note that this method constrains the problem greatly, since most emission probabilities will now be zero. Given a hidden state, say nn, only ambiguity classes containing nn have a non-zero probability.

Workers using HMM taggers have achieved impressive accuracy figures. For example, [4] report that their HMM tagger correctly tagged 96% of word instances. However, two points should be borne in mind [1]. Firstly, a 4% word error rate translates to a 56% sentence error rate for a 20 word sentence, to achieve 4% error per sentence requires a per-word error rate of 0.2%. Secondly, just tagging each word with its most common POS tag reduces error to 10%, so accuracy increase over the default is not that great.

2.2 Transformation rules

In [2], Brill induces a rule-based tagger based on transformations. Words are initially tagged with their most common tag according to the training corpus. If a word is not in the corpus, a tag is guessed according to a number of features of the word such as capitalisation.

Brill then considers 11 rule templates dictating when a word should have its tag changed. For example, the first rule template is:

> Change tag a to tag b when the preceding (following) word is tagged z.

Lexicalized rules are included such as:

> Change tag a to tag b when one of the two preceding (following) words is w.

Transformation rules are learned (and applied) sequentially. At each stage, one considers *all possible instantiations* of the rule templates, and chooses that instantiation that reduces the tagging error the most. Learning stops when the error can be reduced no further. The following is an example of a transformation rule learned using the Wall Street Journal corpus:

> Change the tag from preposition to adverb if the word two positions to the right is "as".

Using the Wall Street Journal, Brill learned 267 transformation rules (and 148 rules for tagging unknown words), and achieved an accuracy of 96.5%—an impressive score, especially when one considers that the test set included words not seen in the training set.

Rule-based tagging approaches have at least two advantages over statistical approaches such as HMMs. Firstly, a few hundred rules require less storage than an HMM's probability matrix, and secondly the rules can be applied more quickly than HMMs when the tagger is in use.

2.3 Inducing a constraint grammar

EngCG, an English language Constraint Grammar developed at the University of Helsinki, is a "rule-based framework for morphological disambiguation and shallow syntactic parsing, where the rules are hand-coded by a linguistic expert" [11]. Impressive tagging results have been reported for EngCG: an accuracy of 99.7%, albeit with each word receiving on average 1.04–1.09 tags, so some ambiguity remains.

EngCG was "developed and debugged over several years", so there is a motivation for seeing whether a Constraint Grammar (CG) can be induced. In [11], 4 sorts of rules are considered:

local-context Rules such as REMOVE (V) (-1C SET1) which mean "the word can not be a verb if the preceding word has one of the features given in SET1"

barrier Rules such as REMOVE (V) (*-1C (DET) BARRIER NPHEAD) which means "the word can not be a verb if somewhere to its left there is a DET word and there is no NPHEAD in between.

lexical Rules such as REMOVE (V) (0 "<table>")) which means that the word "table" should not be read as a verb.

rare-feature Rules such as REMOVE (SUBJUNCTIVE) which mean that all subjunctive readings should be removed.

We will not give details of the learning approach. Local-context, lexical and rare-feature rules can be induced using straightforward statistical methods such as unigram and bigram statistics. Barrier rule induction is more complex, the details are given in [11].

Training data comprised 55,000 hand-disambiguated words. Two points should be made about this training data. Firstly, it was virtually noise-free: two linguists independently disambiguated the text, a comparison was made, and clerical errors eliminated. Secondly, the EngCG annotation scheme is considerably richer than that used in the Brown or Wall Street corpora. For example, in the EngCG scheme, the sentence, "Thanks for sending me your paper" is annotated as shown in Fig 1.

```
"<*thanks>"
        "thanks" <*> N NOM PL
"<for>"
        "for" PREP
"<sending>"
        "send" <SVOO> <SVOC/A> <SVO> <SV> <P/for> PCP1
"<me>"
        "i" <NonMod> PRON PERS ACC SG1
"<your>"
        "you" PRON PERS GEN SG2/PL2 @GN>
"<paper>"
        "paper" N NOM SG
```

Figure 1: EngCG annotation

Richer annotation schemes are better suited to rule-based approaches. Although a more complex tag has to be given to each word, the richer representation gives greater contextual information which can be used in choosing the correct tag. Unfortunately for us, the Penn Treebank tag set used in this study was designed with stochastic approaches in mind:

> ...the Penn Treebank tagset is based on that of the Brown Corpus. However, the stochastic orientation of the Penn Treebank and the resulting concern with sparse data led us to modify the Brown Corpus tagset by paring it down considerably. [7]

There is hence good grounds for hoping that the current approach would have better results using a richer tagset.

2.4 A memory-based approach to tagging

Daelemans et al [5] take a memory-based approach to the tagging problem. Memory-based learning[1] is a form of supervised feature-based learning, where examples are represented with a vector of features. When a test example is presented for classification, its 'distance' from each of the stored training examples is calculated based on feature vectors. The test example is then assigned the class of the training example which is closest to it.

In [5], memory-based learning is applied as follows. Firstly, a lexicon is constructed which records words which occur in the training data, and assigns them the set of POS tags with which they have been tagged, i.e. their ambiguity class. The set of POS tags is ordered according to the frequency with which a the tags occur. For example, in the training data used in [5], "once" was tagged 330 times as rb (adverb) and 77 times as in (preposition/subordinating conjunction), and hence was tagged as $rb - in$, rather than $in - rb$. Using this lexicon, known words are represented as a feature of vectors representing their left and right context. For example, the word "old" in the sentence "Pierre Vinken, 61 years old, ..." is represented by the vector $(cd, nns, jj - np, ,)$, corresponding to the context (61 years old,). The class value of this example is jj, since "old" is acting as an adjective in this sentence. In general, the tags corresponding to the context may be ambiguous.

The memory-based learning was implemented via *IGTree*, a tree-based approach for efficient indexing and searching of large case base. Together with a method for handling unknown words, Daelemans et al achieved an accuracy of 96.4% overall, and 96.7% on known words.

3 Combining lexical and contextual information for tagging

The tagging strategy adopted comprises both statistical and rule-based methods. We tag one sentence at a time. Each word in the sentence is replaced by its ambiguity class, as given in a lexicon. Each tag in the ambiguity class for a particular word has the frequency with which the word was given that tag. *At present, we have no methods for dealing with sentences containing new, unknown words.* Once this preliminary, usually ambiguous tagging is done, the rule-base is consulted, which contains rules dictating when certain tags should be *eliminated* from an ambiguity class. After as many eliminations as possible have been carried out, we check to see if there remain any ambiguous tags. The tag eliminations for a particular ambiguity class are done sequentially: we attempt to eliminate the first tag, then the second, etc. In rare cases, this may cause problems: suppose our rules dictate that all tags should be removed—in this case it would be best to fall back on the lexical probabilities, since at least one rule must be in error—instead

[1] also called, amongst other terminology, instance-based or nearest-neighbour learning

the last tag in the ambiguity class will be chosen since by the time it is examined the tagging has become unambiguous.

If there remain ambiguities, we eliminate that tag which has the lowest probability of being correct in the whole sentence, according to the tag frequencies in the lexicon's ambiguity classes. We then return to the rules, and attempt to carry out further eliminations. The whole process continues until all ambiguities have been resolved.

The elimination rules all have the following form:

```
rmv(A,B,rbr) :- in(B,C), noun(C,D), nounp1(A,E), nounp1(E,F).
rmv(A,B,rp) :- nounp(B,C), in(C,D), nounp(D,E), cds(E,F).
```

The first clause says that a word can not be a comparative adverb (rbr) if it is succeeded by first a conjunction and then a noun, and is preceded by first a noun phrase, and then another noun phrase. The second clause says that a word can not be a particle (rp) if it is succeeded by the following sequence: a noun phrase, a conjunction, another noun phrase and finally a sequence of numbers.

The tagging system then contains two distinct components: a word-specific statistical part (the lexicon) that does not take the word's context into account, and a rule-based part that ignores the actual word under consideration and uses only its ambiguity class and its context, in terms of the POS tags of the other words in the sentence in which it occurs. At present, these two components are combined in a crude and certainly suboptimal manner. The rule-induction process is biased towards under-generalisation in a attempt to avoid errors of commission. Hence when an elimination rule fires it overrides any word-specific statistical information. Only when no tag can be eliminated using the rules does the lexical statistical information come into play. Future combination methods could be more sophisticated and are discussed in Section 7.2.

Our approach contains many elements of the work discussed in Section 2 but owes most to the approach taken by Tapanainen and Voutilainen [13]. There, as here, both statistical and knowledge-based methods are combined with the knowledge-based disambiguation taking precedence. In [13], statistical tagging is done using the Xerox Tagger based on HMMs and the knowledge-based tagging is done using the EngCG, discussed above. A per-word accuracy of 98.5% on unseen text was achieved on test data including the WSJ corpus. This is an impressive figure, however, one should bear in mind that the EngCG is not an induced system and is the result of many years of development.

Tapanainen and Voutilainen found that in their combined system:

> The knowledge-based disambiguator does not resolve all ambiguities but the choices it makes are nearly always correct. The statistical disambiguator resolves all ambiguities but its decisions are not very reliable.

This phenomenon was also present here, partly because the induced clauses were chosen to err on the side of undergenerality.

Our method of tagging words with their POS tags is fairly simple, but we refer to it as the "tagging system" to emphasise that it has two components: the lexicon and the elimination rules. Here we show how our tagging system tags the first sentence in the set of 5000 test set sentences. This sentence is:

A House-Senate conference last week accepted the provision with no discussion of the potential cost to the government.

The lexicon then replaces each word with its ambiguity class, which gives us:

```
[[[dt,0.947],[np,0.053]], [[jj,0.268],[np,0.732]], nn, jj, nn,
[[vbn,0.523],[vbd,0.477]], dt, nn, in, [[dt,0.873],[rb,0.127]], nn,
in, dt, [[jj,0.784],[nn,0.216]], [[nn,0.790],[vb,0.210]], to, dt,
nn, stp]
```

We then apply the elimination rules to this list to see which tags can be eliminated. This gives us

```
[[[dt,0.947],[np,0.053]], [[jj,0.268],[np,0.732]], nn, jj, nn, vbd,
dt, nn, in, dt, nn, in, dt, [[jj,0.784],[nn,0.216]], nn, to, dt,
nn, stp]
```

Here, the vbn (past participle) tagging for "accepted" has been eliminated due to the rule:

```
rmv(A,B,vbn) :- dt(B,C), nounp1(A,D).
```

which eliminates the possibility that "accepted" is a past participle because it is sandwiched between the noun phrase "week" and the determiner "the". Note that vbn is the more likely tag for "accepted" without contextual information, so here the elimination rules have overridden (correctly) the lexical-statistical information.

The rb (adverb) tagging for "no" is eliminated because "no" is followed by the noun "discussion" due to the rule:

```
rmv(A,B,rb) :- noun(B,C).
```

The vb (infinitive) tag for "cost" has also been eliminated, because it is followed by "to" and preceded by "potential" which may be a noun phrase. If it is, it allows the following rule to fire:

```
rmv(A,B,vb) :- to(B,C), nounp1(A,D).
```

If instead "potential" is an adjective, then the following rule fires:

```
rmv(A,B,vb) :- adjp1(A,C).
```

Either way the vb tagging for "cost" is eliminated. The system copes with ambiguous contexts by simply considering every possible unambiguous context allowed by the ambiguous context. We simply eliminate the tag if it can be eliminated in each unambiguous context.

Since no more tag elimination rules can fire, the lexical statistics take over and eliminate the very unlikely np tagging for "A". Returning to the rules, this allows us to knock out the np tagging for "House-Senate", since the following rule can now fire.

```
rmv(A,B,np) :- dt(A,C), cnoun(B,D), nounp(D,E), sverb(E,F)
```

The final ambiguity caused by the word "potential" (`[[jj,0.784],[nn,0.216]]`) is then resolved using the lexical statistical information, giving us the final sentence of tags:

```
[dt, jj, nn, jj, nn, vbd, dt, nn, in, dt, nn, in, dt, jj, nn, to,
dt, nn, stp]
```

This happens to be a correctly tagged sentence.

4 Using ILP for POS tag disambiguation

4.1 The Wall Street Journal corpus

The data used was the Wall Street Journal corpus, which was divided into 2/3 training and 1/3 testing. The version of the corpus was case-sensitive. The Wall Street Journal corpus contains text taken from the Wall Street Journal where each word has been manually tagged with its correct POS tag. The Wall Street Journal forms part of the Penn Treebank [7], and the tags used are those of the Penn Treebank, see Table 1 for details. The size of the corpus is 26.94 Mbytes (see Table 3).

In order to make the data easier for Progol to use, all tags composed entirely of upper-case letters were translated into lower-case. The other non-alphabetic tags were translated as in Table 2.

4.2 Building the lexicon

We followed [5] in building a lexicon which pairs words occurring in the training data with their ambiguity class. We actually recorded the relative frequency with which the various tags appeared rather than just the order, although, at present only the frequency order is used. An excerpt from the lexicon is given in Fig 2. The lexicon created entries for 66,025 word tokens, 8,156 of which had non-singleton ambiguity classes.

If a tag for a particular word was found to occur less than 5% of the time then that tag was not included in the word's ambiguity class. This was done in order to mitigate the effect of noise in the corpus. However, this crude method of noise-handling inevitably introduced some errors of its own. For example, consider the tag statistics for the word "New" as given in Table 4.

"New" is tagged as jj (an adjective) 61 times, yet this forms only $61/(61+1+2501+2) = 61/2564 = 2.38\% < 5\%$ of all the taggings for "New". Consequently, in the lexicon "New" is unambiguously tagged as np. The problem here was that numerous references to American placenames such as "New York" has swamped out adjectival occurrences of "New" such as: "New measures must be taken".

On the other hand, the single case of "New" being tagged as nn (singular common noun) was an error in the corpus, so there the noise elimination was doing its job. (It occurred in the headline "New Tests May Show If Cancer Drugs Work".) The two cases of "New" tagged as a nps (plural proper noun), both occur in the phrase "New Yorkers"

Tag	Meaning	Frequency
#	£	538
$	“$”	21242
“	“ “ ” or “ ‘ ”	23789
”	“ ” ” or “ ’ ”	23427
(	“(“	4647
)	“)”	4552
,	“,”	163227
.	“.”	127640
:	“;” or “:”	8806
CC	Coordinating Conjunction	67833
CD	Cardinal number	112565
DT	Determiner	263125
EX	Existential ”there”	2826
FW	Foreign word	705
IN	Preposition or Subordinating conjunction	321687
JJ	Adjective	197118
JJR	Adjective, comparative	10574
JJS	Adjective, superlative	6104
LS	List item marker	109
MD	Modal	31516
NN	Noun, singular or mass	424605
NNS	Noun, plural	192883
NP	Proper noun, singular	304396
NPS	Proper noun, plural	8778
PDT	Predeterminer	1075
POS	Possessive ending	28090
PP	Personal pronoun	56094
PP$	Possessive pronoun	27378
RB	Adverb	99788
RBR	Adverb, comparative	5482
RBS	Adverb, superlative	1436
RP	Particle	5391
SYM	Symbol	91
TO	“To”	72202
UH	Interjection	328
VB	Verb, base form	86049
VBD	Verb, past tense	96504
VBG	Verb, gerund or present participle	48325
VBN	Verb, past participle	65663
VBP	Verb, non 3rd-person, singular present	40099
VBZ	Verb, 3rd person singular present	70394
WDT	Wh-determiner	14300
WP	Wh-pronoun	7771
WP$	Possessive wh-pronoun	587
WRB	Wh-adverb	6829

Table 1: Penn Treebank Part of Speech Tags.

Tag	Atom
#	pnd
$	dlr
”	rqt
“	lqt
(	lpn
)	rpn
,	cma
.	stp
:	cln
PP$	ppz
WP$	wpz

Table 2: Translating tags into Prolog atoms.

	Size (Mb)	Words
Training set	17.95	2043935
Test set	8.99	1021961
Corpus	26.94	3065896

Table 3: Size of the Wall Street Journal corpus.

Tag	Frequency
jj	61
nn	1
np	2501
nps	2

Table 4: Frequency of tags for “New” in the training set.

```
beans nns
beanstalk nn
bear [[nn,0.450],[vb,0.450],[vbp,0.100]]
bearable jj
bearded jj
beards nns
bearing [[nn,0.421],[vbg,0.579]]
bearings nns
bearish jj
bears [[nns,0.385],[vbz,0.615]]
beast nn
beasties nns
beasts nns
beat [[vbp,0.080],[nn,0.100],[vb,0.600],[vbd,0.220]]
beat-up jj
beaten [[jj,0.100],[vbn,0.900]]
beatific jj
beating [[nn,0.400],[vbg,0.600]]
beats [[nns,0.111],[vbz,0.889]]
```

Figure 2: Excerpt from lexicon created from the Wall St. Journal.

(i.e. inhabitants of New York). This appears to the author to be the correct tagging, so here the noise elimination has again removed a legitimate member of the ambiguity class for "New".

Clearly, the fact that "New" has been tagged as jj 99 times should be sufficient to include jj in the ambiguity class of "New", despite this being a small proportion of all taggings. Future work will have to take more than the bare proportion of taggings into account. However, even in this unsatisfactory case, where two legitimate tags have been removed leaving only a single tag, one should bear in mind that (if the test set frequencies reflect those of the training set) "New" will be correctly tagged about $2501/2564 = 97.54\%$ of the time, which is a respectable per-word accuracy rate.

4.3 Generating examples for ILP

It was decided to learn rules for eliminations on a tag by tag basis. In other words, Progol learnt rules for eliminating *cc*, *cd*, *dt*, etc from separate data sets, one data set for each tag. Positive and negative examples where generated as follows. The corpus was scanned a sentence at a time. For each word which has an ambiguous tag according to the lexicon, one negative example, and one or more positive examples of tag elimination were created. The example contains the tag to be removed and its entire left and right context. The left context is reversed so that the start of the tag list representing the context contains those tags nearest to the focus word. For example, the first few sentences from the corpus produce the positive and negative examples given in Table 5.

Since it was possible to create vast numbers of examples (at least for the more important tags), we could afford to be quite restrictive about which examples to include in training. Firstly, only sequences which (i) began with a word with an initial capital letter, (ii) ended in a full stop and (iii) *contained no quotes* were used. The no-quote restriction is a considerable one, and further work will have to drop it. It was used in this preliminary work since it made it easier to identify where sentences began and ended and because the lack of embedded speech in sentences simplified their structure.

For many tags, this produced many thousands of examples (positive and negative). In the future, it may be possible to develop Progol to cope with such vast example sets. Although Progol was altered to speed up learning on large data sets, it was still necessary here to sample from the full training set for each tag. So in this work we used only the first 6000 examples found for each tag. Details of the various training sets so produced are given in Table 6. (In this table "Prop" refers to the proportion of the entire training set which the sample amounts to.)

4.4 Background knowledge

4.4.1 Defining a grammar

The entire background knowledge can be found in Figs 3–6. The background predicates fall into three sets: "tag" predicates (Fig 3), forward parsing predicates (Figs 4 and 5) and backward parsing predicates (Fig 6). The background knowledge defines a grammar where the 43 POS tags are terminals and there are 22 nonterminals defining such grammatical constructs as noun phrase (`nounp`) and adjectival phrase (`adjp`). We use the standard logic programming representation for a grammar: `nounp(A,B)` is true if the sequence of terminals `A` begins with a `nounp` and the remainder of this sequence, once the initial `nounp` has been removed is `B`. There is also a number of "utility predicates" such as `adjp_rest` which are used for efficiency.

Because the terminals of the grammar are the tags, which do not represent all the information there is about the underlying words, the grammar so defined is very "sloppy". For example, the definition of simple verb phrase `svp` allows that the tag sequence `[vbz,vbg]` constitutes a simple verb phrase. This means that in any case where a gerund or past participle follows a 3rd person singular present verb, then this is counted as a simple verb phrase. *The singular present verb is not constrained to be an auxiliary verb.* The tag set used here does not distinguish auxiliary verbs. For example, the phrase "is wandering" will be translated, via the lexicon, into `[vbz,vbg]` (neither word is ambiguously tagged in the lexicon, although "wandering" can be a noun). It will then be correctly recognised as a simple verb phrase. However, the phrase "wanders being" which is ungrammatical, will also be marked as a simple verb phrase.

The grammar defined in the background knowledge is clearly over-general, but we are not using it to determine whether sentences are grammatical, or to generate new sentences. We have chosen the definitions of the 22 background predicates in the hope that these will be relevant to the task of tag elimination. The clauses produced show this

Pierre Vinken, 61 years old, will join the board **as** a nonexecutive director Nov. 29.
`:- rmv([nn,dt,vb,md,cma,jj,nns,cd,cma,np,np],[dt,jj,nn, np,cd,stp],in).`
`rmv([nn,dt,vb,md,cma,jj,nns,cd,cma,np,np],[dt,jj,nn,np, cd,stp],rb).`

Mr. Vinken is chairman of Elsevier N.V., the **Dutch** publishing group.
`:- rmv([dt,cma,np,np,in,nn,vbz,np,np],[vbg,nn,stp],np).`
`rmv([dt,cma,np,np,in,nn,vbz,np,np],[vbg,nn,stp],jj).`
`rmv([dt,cma,np,np,in,nn,vbz,np,np],[vbg,nn,stp],nps).`

Mr. Vinken is chairman of Elsevier N.V., the Dutch **publishing** group.
`:- rmv([np,dt,cma,np,np,in,nn,vbz,np,np],[nn,stp],vbg).`
`rmv([np,dt,cma,np,np,in,nn,vbz,np,np],[nn,stp],nn).`

Rudolph Agnew, 55 years old and former chairman of **Consolidated** Gold Fields PLC, was named a nonexecutive director of this British industrial conglomerate.
`:- rmv([in,nn,jj,cc,jj,nns,cd,cma,np,np],[np,np,np,cma, vbd,vbn,dt,jj,nn,in,dt,jj,jj,nn,stp],np).`
`rmv([in,nn,jj,cc,jj,nns,cd,cma,np,np],[np,np,np,cma,vbd, vbn,dt,jj,nn,in,dt,jj,jj,nn,stp],jj).`

Rudolph Agnew, 55 years old and former chairman of Consolidated **Gold** Fields PLC, was named a nonexecutive director of this British industrial conglomerate.
`:- rmv([np,in,nn,jj,cc,jj,nns,cd,cma,np,np],[np,np,cma, vbd,vbn,dt,jj,nn,in,dt,jj,jj,nn,stp],np).`
`rmv([np,in,nn,jj,cc,jj,nns,cd,cma,np,np],[np,np,cma,vbd, vbn,dt,jj,nn,in,dt,jj,jj,nn,stp],nn).`

Rudolph Agnew, 55 years old and former chairman of Consolidated Gold **Fields** PLC, was named a nonexecutive director of this British industrial conglomerate.
`:- rmv([np,np,in,nn,jj,cc,jj,nns,cd,cma,np,np],[np,cma, vbd,vbn,dt,jj,nn,in,dt,jj,jj,nn,stp],np).`
`rmv([np,np,in,nn,jj,cc,jj,nns,cd,cma,np,np],[np,cma,vbd, vbn,dt,jj,nn,in,dt,jj,jj,nn,stp],nps).`

Rudolph Agnew, 55 years old and former chairman of Consolidated Gold Fields PLC, was **named** a nonexecutive director of this British industrial conglomerate.
`:- rmv([vbd,cma,np,np,np,np,in,nn,jj,cc,jj,nns,cd,cma, np,np],[dt,jj,nn,in,dt,jj,jj,nn,stp],vbn).`
`rmv([vbd,cma,np,np,np,np,in,nn,jj,cc,jj,nns,cd,cma,np, np],[dt,jj,nn,in,dt,jj,jj,nn,stp],vbd).`

Table 5: Creating examples from a sentence.

Tag	Pos	Neg	Total	Prop
cc	1192	338	1530	100%
cd	780	2312	3092	100%
cln	8	0	8	100%
cma	0	0	0	100%
dt	3825	2175	6000	32%
dlr	0	0	0	100%
ex	182	660	842	100%
fw	331	162	493	100%
in	2292	3708	6000	14%
jj	3216	2784	6000	8%
jjr	2624	3376	6000	81%
jjs	879	1068	1947	100%
lpn	0	0	0	100%
lqt	0	0	0	100%
ls	5	4	9	100%
md	209	102	311	100%
nn	3025	2975	6000	6%
nns	2266	3734	6000	28%
np	2647	3353	6000	16%
nps	3773	2227	6000	65%
pdt	3231	470	3701	100%
pnd	0	0	0	100%
pos	448	448	896	100%

Tag	Pos	Neg	Total	Prop
pp	511	168	679	100%
ppz	152	511	663	100%
rb	4182	1818	6000	14%
rbr	3850	2150	6000	75%
rbs	1064	720	1784	100%
rp	4539	1461	6000	65%
rpn	0	0	0	100%
rqt	1043	0	1043	100%
stp	11	1	12	100%
sym	11	18	29	100%
to	0	0	0	100%
uh	195	33	228	100%
vb	3566	2434	6000	8%
vbd	3294	2706	6000	12%
vbg	3192	2808	6000	33%
vbn	3087	2913	6000	11%
vbp	4328	1672	6000	15%
vbz	4925	1075	6000	22%
wdt	4433	1567	6000	49%
wp	10	5	15	100%
wpz	0	0	0	100%
wrb	0	0	0	100%

Table 6: Training sets for tagging.

```
cc([cc|S],S).      cd([cd|S],S).      dt([dt|S],S).
ex([ex|S],S).      fw([fw|S],S).      in([in|S],S).
jj([jj|S],S).      jjr([jjr|S],S).    jjs([jjs|S],S).
md([md|S],S).      nn([nn|S],S).      nns([nns|S],S).
np([np|S],S).      nps([nps|S],S).    pos([pos|S],S).
pp([pp|S],S).      rb([rb|S],S).      rbr([rbr|S],S).
rbs([rbs|S],S).    rp([rp|S],S).      sym([sym|S],S).
to([to|S],S).      uh([uh|S],S).      vb([vb|S],S).
vbd([vbd|S],S).    vbg([vbg|S],S).    vbn([vbn|S],S).
vbp([vbp|S],S).    vbz([vbz|S],S).    wdt([wdt|S],S).
wp([wp|S],S).      wrb([wrb|S],S).    pnd([pnd|S],S).
dlr([dlr|S],S).    lqt([lqt|S],S).    rqt([rqt|S],S).
lpn([lpn|S],S).    rpn([rpn|S],S).    cma([cma|S],S).
stp([stp|S],S).    cln([cln|S],S).    ppz([ppz|S],S).
wpz([wpz|S],S).
```

Figure 3: Background predicates for the tags.

to be the case.

As well as the terminals and the "forward parsing" predicates we have defined 4 predicates for parsing backwards. Recall that in producing examples from sentences in the corpus, the context to the left of the focus word was reversed. This was done so that the tags of the words nearest to the focus word were at the beginning of a list. This list of terminals is parsed in the normal way, except now we must take the reversal into account. Consequently, a simple verb phrase in the left context can be a sequence `[vbg,vbz]` ("walking is"). Note that many nonterminals, such as `cds`, have the same definition whether the list of terminals is reversed or not.

4.4.2 Choosing background predicates

Although all background predicates were defined for all Progol runs, not all were allowed to appear in bodies of induced tag elimination clauses. The basic set of background predicates that were allowed to appear in clause bodies is given in Table 7.

The intuition for doing this was that for many of the tags, the missing background predicates were irrelevant. For example, knowing whether a particular tag was an `nn, nns, np` or `nps` was thought not to be that helpful in eliminating the possibility that a certain word could be a `dt`—it is enough to know that it is a noun or indeed a noun phrase.

When learning elimination rules for certain tags it was judged that extra background predicates should be allowed to appear in clause bodies. The details are given in Table 8. Although not all defined background predicates were allowed in the induced rules, the

```
% to mark the beginning of sentences
end([]).

% to mark NOT the beginning of sentences
not_end([_|_]).

% pounds and dollars are both currencies
cncy([pnd|S],S) :- !.  cncy([dlr|S],S).

% sequences of numbers represent numbers
cds([cd|X],S2) :- cd(S1,X), (X=S2 ; cds(X,S2)).

% noun
noun([nn|S],S) :- !.  noun([np|S],S) :- !.
noun([nns|S],S) :- !. noun([nps|S],S).

% proper noun
prnoun([np|S],S) :- !. prnoun([nps|S],S).

% common noun
cnoun([nn|S],S) :- !.  cnoun([nns|S],S).

% singular noun
snoun([nn|S],S) :- !.  snoun([np|S],S).

% plural noun
plnoun([nns|S],S) :- !. plnoun([nps|S],S).

% adjectives
adj([jj|S],S) :- !. adj([jjr|S],S) :- !. adj([jjs|S],S).

% adverbs
adv([rb|S],S) :- !. adv([rbr|S],S) :- !. adv([rbs|S],S).

% adverbial phrase
advp(L1,L2) :- adv(L1,X), (X=L2 ; advp(X,L2)).

% adjectival phrase
adjp(L1,L2) :- adj(L1,X), !,  (X=L2 ; adjp_rest(X,L2)).
adjp(L1,L2) :- advp(L1,L3), adjp(L3,L2).

adjp_rest([cma|L4],L2) :- !, adjp(L4,L2).
adjp_rest([cc|L4],L2) :- !, adjp(L4,L2).
adjp_rest(L3,L2) :- adjp(L3,L2).
```


Figure 4: Forward parsing background predicates.

```
%simple verb
sverb([vbd|S],S) :- !.  sverb([vbp|S],S) :- !.  sverb([vbz|S],S).

% participle
vpart([vbg|S],S) :- !. vpart([vbn|S],S).

% simple verb phrase
svp(S1,S2) :- sverb(S1,X), !, (X=S2 ; svp_rest1(X,S2)).
svp([md|X],S2) :- !, svp_rest2(X,S2).
svp(S1,S2) :- advp(S1,S3), sverb(S3,S2).

svp_rest1(S3,S2) :- vpart(S3,X), !,  (X=S2 ; advp(X,S2)).
svp_rest1(S3,S2) :- advp(S3,X),  (X=S2 ; vpart(X,S2)).

svp_rest2([vb|X],S2) :- !, (X=S2 ; advp(X,S2)).
svp_rest2(S3,S2) :- advp(S3,S4), vb(S4,S2).

% verb phrase
vp(S1,S2) :- svp(S1,X), (X=S2 ; rp(X,S2)).

% simple noun phrase (sequence of nouns)
snp(L1,L2) :- noun(L1,X), (X=L2 ; snp(X,L2)).

% dtz indicates a determiner or possession.
dtz([dt|S],S) :- !.  dtz([ppz|S],S).

% noun phrase
nounp(L1,L2) :- noun(L1,X), !, (X=L2 ; snp(X,L2)). % "Pierre Vinken"
nounp([pp|S],S) :- !. % "us, him"
nounp(L1,L2) :- dtz(L1,L3), !, (snp(L3,L2) ; adjp(L3,L4), snp(L4,L2)).
nounp(L1,L2) :- cncy(L1,L3), !, cds(L3,L2). % "$ 20 billion "
nounp([cd|Y],L2) :- !, (X=Y ; cds(Y,X)),  (X=L2 ; snp(X,L2)). % "20, 20%"
nounp(L1,L2) :- adjp(L1,L3), snp(L3,L2). % "green men"
```

Figure 5: Forward parsing background predicates (continued).

```
% BACKWARDS noun phrase
nounp1([pp|S],S) :- !. % "us, him"
nounp1([cd|S1],S2) :- !, (S3=S1 ; cds(S1,S3)), (S3=S2 ; cncy(S3,S2)).
nounp1(S1,S2) :- snp(S1,S3), nounp1_rest(S3,S2).

nounp1_rest(S,S).
nounp1_rest(S1,S2) :- dtz(S1,S2), !.
nounp1_rest([cd|S1],S2) :- !, (S1=S2 ; cds(S1,S2)).
nounp1_rest(S1,S2) :- adjp1(S1,S3), (S3=S2 ; dtz(S3,S2)).

% BACKWARDS simple verb phrase
svp1(S1,S2) :- sverb(S1,X), !, (X=S2 ; advp(S1,S2)).
svp1(S1,S2) :- vpart(S1,S3), !, (S4=S3 ; advp(S3,S4)), sverb(S4,S2).
svp1([vb|S3],S2) :- !, (S3=S4 ; advp(S3,S4)), md(S4,S2).
svp1(S1,S2) :- advp(S1,X), svp_rest11(X,S2).

svp_rest11(S3,S2) :- sverb(S3,S2), !.
svp_rest11(S3,S2) :- vpart(S3,S4), !, sverb(S4,S2).
svp_rest11([vb|[md|S]],S).

% BACKWARDS verb phrase
vp1([rp|S1],S2) :- !, svp1(S1,S2).
vp1(S1,S2) :- svp1(S1,S2).

% BACKWARDS adjectival phrase
adjp1(L1,L2) :- adj(L1,X), !,  (X=L2 ; adjp_rest1(X,L2)).

adjp_rest1([cma|L4],L2) :- !, adjp1(L4,L2).
adjp_rest1([cc|L4],L2) :- !, adjp1(L4,L2).
adjp_rest1(L3,L2) :- adjp1(L3,L2).
adjp_rest1(L3,L2) :- advp(L3,L2).
```

Figure 6: Backward parsing background predicates.

Tag predicates	cc, dt, ex, fw, in, md, pos, pp, rp, sym, to, uh, wdt, wp, wrb, lqt, rqt, lpn, rpn, cma, stp, ppz, wpz
Forward parsing predicates	cncy, end, cds, advp, adjp, sverb, vpart, noun, vp, svp, nounp
Backward parsing predicates	nounp1, vp1, svp1, adjp1

Table 7: Basic set of background predicates.

basic set still includes 38 background predicates which is quite extensive by ILP standards.

Tag	Extra background predicates
jj	cnoun, prnoun
jjr	cnoun, prnoun
jjs	cnoun, prnoun
nn	nn, nns, np, nps, cnoun, prnoun, plnoun, snoun, not_end
nns	nn, nns, np, nps, cnoun, prnoun, plnoun, snoun, not_end
np	nn, nns, np, nps, cnoun, prnoun, plnoun, snoun
nps	nn, nns, np, nps, cnoun, prnoun, plnoun, snoun
vb	vbd, vbg, vbn, vbp, vbz
vbg	vbd, vbg, vbn, vbp, vbz
vbn	vbd, vbg, vbn, vbp, vbz
vbp	vbd, vbg, vbn, vbp, vbz
vbz	vbd, vbg, vbn, vbp, vbz

Table 8: Extra background predicates

5 Using Progol for POS tag disambiguation

The Progol algorithm is described in detail in [8]. Here we detail those aspects of our use of Progol that are specific to the tagging task. All experiments reported here used the Prolog version of Progol (P-Progol).

5.1 Caching clause statistics

5.1.1 Motivation

The most important challenge for ILP in corpus-based NLP is the sheer size of the corpora. Although only a maximum of 6000 examples were used for each tag, the complexity of and number of predicates in the background knowledge was such that this still formed a formidably large example set.

In Progol, a "bottom" clause is produced from a seed positive example, which is the most specific clause which covers that example, given the declarative bias. Progol then carries out an *exhaustive* top-down search of the subsumption lattice of clauses which are more general than the bottom clause. Progol is *guaranteed* to find a clause which has maximal compression in this lattice. (Recent work [12], has considered using cost/utility functions other than compression in Progol. In this work the standard compression approach was always used.)

Carrying out an *exhaustive* rather than heuristic search is clearly an expensive business. One approach that is often used with Progol is to remove all examples which are covered by an induced clause, i.e. they are replaced by the new clause. In noise-free situations this is always admissible. However, in the current work, where there is no hope of inducing a theory that is either consistent or complete, this greedy cover removal is problematic. To take an extreme example, suppose there exists a clause which covers 300 positive examples and say, 3 negative examples of the original training data, and so has 99% training accuracy. This is clearly a "good" clause and should be included in any final theory. However, now suppose that other previously found clauses cover 250 of these 300 positive examples. If these examples are removed, then the clause will have training accuracy of only 94% accuracy, which would lead to it being rejected in the Progol runs recorded here.

To avoid this problem, in this work, Progol was told *not* to remove covered examples. This means that the entire training set of positive and negative examples persists throughout the entire run. However, covered positive examples were removed from the set of possible seeds for bottom clause construction. (It is possible to run Progol so that even this removal is not done.)

Another factor adding to the computational effort required is that "deep", reasonably long clauses are required. The main motivation for using ILP on the tagging task is the problem of "long-term dependencies". In order to find the correct part of speech for a word it is sometimes necessary to look at words in the surrounding text which are some distance from the focus word. The use of "high-level" grammatical constructs such as noun phrase partially solve this problem, since a noun phrase can often "consume" many terminals (tags), allowing us to reach parts of the sentence some way from the focus word. However, as the results show deep clauses are still required. Consequently, in this work we allowed clauses of up to 4 body literals including variables of up to depth 4.

To sum up, for each bottom clause constructed Progol does an exhaustive search looking for a maximally compressive clause. Clauses can have 4 body literals and variables of depth 4. There are at least 38 background predicates, some quite complex, and the example set can have 6000 example. Finally, all this is done without greedy removal of covered positives.

5.1.2 Implementation

In order to speed up the rate of learning, it was decided to *cache* the positive and negative cover of every clause that was produced during a search of a subsumption lattice. If the clause is produced during a subsequent search, then it is not necessary to do theorem-proving to find which examples it covers, since this information can be quickly recovered from the cache.

In P-Progol, the positive and negative examples are never actually asserted as ground Prolog clauses, but are *recorded* in an indexed internal database, using the `recorda/3` predicate. (P-Progol is implemented in Yap Prolog.) This means the list of positive examples covered by a particular clause can be represented fairly compactly as a list of intervals. For example, the list `[1,3-4,6,8,14-20]` would be used to represent the positive cover of a clause that covered the examples numbered 1, 3, 4, 6, 8, 14, 15, 16, 17, 18, 19 and 20. Similarly, for negative examples. P-Progol has always kept track of the positive and negative cover of clauses in a particular search. This allows Progol to "pass down" cover from a clause to its children. If clause A is a refinement of clause B, then clearly clause A can only possibly cover those examples covered by clause B, there is no point wasting time by considering any other examples. The addition in the current work is simply to store these cover lists away *permanently*, so the cover for a particular clause is only computed once.

In fact, we refined the basic caching approach a little. Many clauses that are produced can never be included in the final theory produced. Perhaps they cover too many negative examples, or insufficiently many positives. For these clauses, there is no point taking up memory by recording which examples they cover, it is merely sufficient to record that they are unacceptable. So for each clause produced during a search, we first check to see if it has been recorded as unacceptable, if not, we then look to see if its cover has been recorded away, only if it has not do we then find its positive and negative cover by theorem-proving.

Yap Prolog uses the functor and first three arguments (if these exist) of a recorded term as keys when searching for terms in its internal database. It is also optimised for storing ground terms. Consequently, in order to make the caching more efficient, ground versions of clauses were stored, and the body predicate symbols were extracted so that they could be subsequently be used as keys.

Proper extensive comparisons of the time saved and extra memory consumed by the caching approach have not been done yet. One reason is that initial comparisons indicate that to run Progol in non-caching mode on the datasets used here would take an inordinate amount of time! Figure 7 shows the results of running Progol in caching and non-caching mode on a small data set of 1000 positive and 1000 negative examples, with a much simpler background knowledge than above. Also, this run was done before the clause body predicate symbols were used as keys. Even in this experiment, which was set up to be quite unfavourable to the caching approach, the time-saving is impressive. The total time taken without caching was 98494 seconds and with caching it was 28123, a 350% speed up. The theories produced in both cases, were, of course, identical. Note that for the first few searches the caching mode is not quicker: since most clauses are "new" we can not save time by retrieving them.

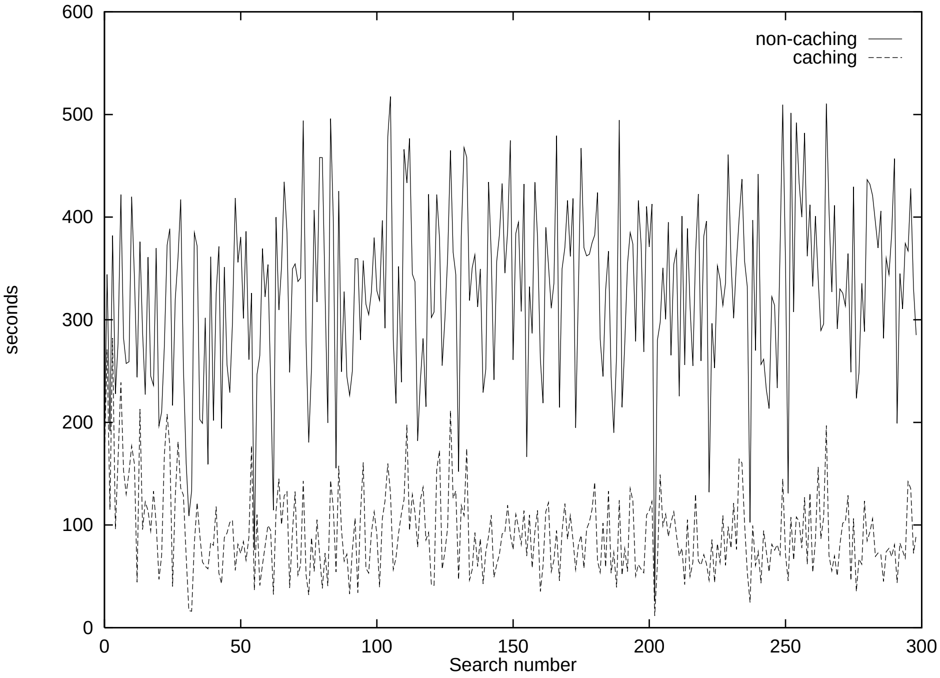

Figure 7: Running Progol with and without caching.

5.1.3 Successes and Problems

In the majority of cases, the caching approach was highly successful. The combination of exhaustive search, deep long clauses, complex background knowledge and large data sets would have become intractable without it. Clearly, the vast speed up in time is at the expense of using large amounts of memory. We have yet to do a proper investigation on the amount of memory used. The strategy here was to simply cache *all* clauses and hope that we did not run out of memory! In a few cases, memory problems were encountered. The large number of extra background predicates used for nn, nns, np and nps (see Table 8) increased the number of clauses produced considerably, and hence the amount of memory used in caching them. On the runs for np this caused one machine to exhaust the amount of memory available for the Progol run. A similar problem occurred, surprisingly, for jj, where the machine ran out of real memory and began "thrashing to disk". The similar run for jjr caused no problems, however. In runs for nn, nns and nps, memory was not exhausted, but learning was clearly proceeding very slowly, due to the large search space. In these cases, the runs were aborted, any induced clauses collected and Progol re-run with a clause length limit of 4, or reduced background knowledge. The theories for jj, nn, nns, np and nps are hence the result of more than one Progol run, and hence useful computation time statistics for them are unavailable (see Table 9.

It should be pointed out that clause caching is currently "bolted on" to P-Progol and that there are a number of obvious avenues for improvements for its implementation. For example, in the original P-Progol, clause cover is recorded for each clause in a particular search. At present, this local information is still recorded, even if caching is turned on and the same information is being recorded permanently. Considerable memory saving would be possible by avoiding this redundancy.

5.2 Declarative bias

Since Progol does an exhaustive search there is considerable motivation for defining as strong a declarative bias as possible. The use of syntactic integrity constraints and the *abandon* predicate are discussed in [9]. As each clause is produced in Progol's search, the system attempts to prove *false* and *abandon*. If *false* is provable, an integrity constraint has been violated and the clause is rejected. If *abandon* is proved, then the clause is rejected *and* the search is pruned at that point. In this work we use no integrity constraints and one *abandon* clause, given in Fig 8.

```
abandon :-
        hypothesis(_,Body,_),
        has_redundant_literals(Body).

has_redundant_literals(Body) :-
        has_literal(Body,Lit1,Rest),
        has_literal(Rest,Lit2,_),
        arg(1,Lit1,X),
        arg(1,Lit2,XX),
        X==XX,
        arg(2,Lit1,Y),
        arg(2,Lit2,YY),
        Y==YY.

has_literal((Lit,Rest),Lit,Rest).
has_literal(Lit,Lit,[]) :- Lit\=[].
has_literal((_,Rest),Lit,Rest1) :- has_literal(Rest,Lit,Rest1).
```

Figure 8: The *abandon* clause.

The abandon clause prevents Progol from considering clauses such as

```
rmv(A,B,jj) :- noun(B,C), nn(B,C).
```

where the literal $noun(B,C)$ is irrelevant (if something is a singular common noun (nn) then it is certainly a noun). This clause is not ideal since it causes the abandonment of clauses such as

```
rmv(A,B,jj) :- prnoun(B,C), snoun(B,C).
```

which is equivalent to

```
rmv(A,B,jj) :- np(B,C).
```

The former clause is worth keeping if *np* is not included in those predicates which may appear in the bodies of induced clauses. On the other hand, this *abandon* clause is very quick to evaluate and cuts out a large swathe of unacceptable clauses in a very simple manner.

5.2.1 Search constraints

The set of search constraints used are given in Figure 9. Taking them in turn, the i parameter sets a limit to the depth of variables, where depth has its usual definition, as described, for example, in [8]. *clauselength* limits the length of clauses, and includes the head literal in the count. *minacc* constrains all acceptable clauses to have at least 95% training set accuracy, and *minpos* sets the minimum amount of positive cover that is acceptable.

Our goal here is to find clauses that will have high test set accuracy. To this end, clauses must have reasonable training set accuracy and coverage. This is a rather crude approach to identifying accurate clauses. A better approach would be to use a Bayesian approach to estimating clause accuracy. This is discussed in [3] and has been implemented in ILP as m-estimation [6]. Note that we are not attempting to induce anything like a complete theory: our final theory will have very many errors of omission. This does not matter greatly—our main concern is that when an induced clause says that a tag should be eliminated then it does so correctly. If no rule eliminates a tag that should be eliminated, the lexical probabilities will come into play, and they effect correct eliminations in 94.1% of the cases.

```
:- set(i,4).
:- set(clauselength,5).
:- set(minacc,0.95).
:- set(minpos,15).
```

Figure 9: Progol search constraints.

The *minacc* and *minpos* constraints lead to very large numbers of clauses to be unacceptable. These are cached away as such and can be rejected very quickly if encountered again. Note that if a clause has insufficient positive cover, then all of its refinements will also, and the search can be pruned at that point.

As is commonly the case, Progol rarely produced clauses towards the end of its run. Positive examples used as ‘seeds’ which are encountered towards the end of a run are very unlikely to be covered by an acceptable clause. This is because of the *minpos* constraint: if an example is covered by a sufficiently accurate clause, *which also covers at least 14 other examples*, then at least one of these examples is likely to be encountered early on in the run. This will usually result in the said clause being induced and the example removed from the set of admissible seeds. To prevent Progol wasting time attempting to produce useful clauses from the last fraction of remaining seeds, the `seeds_prop` parameter was set on some of the runs (Fig 10. This prevents Progol from considering the last 10% 'seed' examples.

```
:- set(seeds_prop,0.9).
```

Figure 10: The seeds proportion parameter.

6 Results

6.1 Theory size and computation time

In this section, we detail the computational effort required to produce the theories. Table 9 gives details, for each tag, of the example sets, the number of subsumption lattice searches, the mean time for a lattice search and the total amount of time spent searching. All times are in seconds. "Search time" is the total time spent performing searches of subsumption lattice, it is not the total time Progol spent producing the final theory. Although searching forms the bulk of work there is also around 1.3 seconds producing the bottom clause. The estimated total time is found be adding 1.3 seconds for each search to the total time spent searching.

The theories produced for jj, nn, nns, np and np were the result of a number of Progol runs, so reliable computation statistics for these runs are unavailable.

To get a good idea of the effectiveness of the caching method it is useful to calculate the time taken for searches as it relates to the number of clauses constructed during a search. We use statistics from the `rp` run since it is the longest. The run-time statistics for all other runs are very similar. As Figs 11 and 12 indicate in the majority of searches Progol takes less than 0.3 seconds per clause. A more naïve approach which did not use indexing or caching would take vastly more time than this, since for each clause we have to determine which of the 6000 examples are covered by that clause.

Figs 13 and 14 are histograms showing the distribution of subsumption lattice search times when learning `rp`. Note that the vast majority of searches take well under a minute. This shows the great effectiveness of the caching technique, since each of these searches involves an exhaustive search for compressive clauses using an example set of 6000 examples.

Tag	Pos	Neg	Searches	Mean	Search time	Total time (est.)	Theory size
cc	1192	338	539	10.61	5717	6417	28
cd	780	2312	780	26.47	20650	21664	0
dt	3825	2175	385	53.83	20724	21225	42
ex	182	660	101	2.85	288	419	2
fw	331	162	294	7.31	2148	2530	2
in	2292	3708	1186	20.75	24605	26146	22
jj	3216	2784	n/a	n/a	n/a	n/a	47
jjr	2624	3376	1101	49.06	54012	55444	39
jjs	879	1068	426	7.98	3401	3955	19
md	209	102	35	0.19	7	52	4
nn	3025	2975	n/a	n/a	n/a	n/a	61
nns	2266	3734	n/a	n/a	n/a	n/a	48
np	2647	3353	n/a	n/a	n/a	n/a	31
nps	3773	2227	n/a	n/a	n/a	n/a	47
pdt	3231	470	21	58.41	1227	1254	14
pos	448	448	21	110.05	2311	2338	6
pp	511	168	32	2.32	74	116	18
ppz	152	511	84	0.41	35	144	2
rb	4182	1818	1305	19.63	25622	27318	87
rbr	3850	2150	1727	21.08	36400	38645	45
rbs	1064	720	250	4.04	1009	1334	25
rp	4539	1461	1883	36.31	68371	71259	48
rqt	1043	0	1	4.65	5	6	1
stp	11	1	10	0.01	0	13	0
sym	11	18	10	0.01	0	13	0
uh	195	33	42	0.21	9	64	3
vb	3566	2434	180	44.09	7935	8169	35
vbd	3294	2706	545	37.35	20354	21062	20
vbg	3192	2808	1589	22.24	35335	37401	44
vbn	3087	2913	881	28.92	25482	26627	57
vbp	4328	1672	222	38.39	8522	8811	25
vbz	4925	1075	462	62.90	29061	29661	83
wdt	4433	1567	5	115.83	579	586	3
wp	10	5	10	0.01	0	13	0
Total	73313	51952	14127	n/a	n/a	n/a	908

Table 9: Theory size and search computation times for each tag.

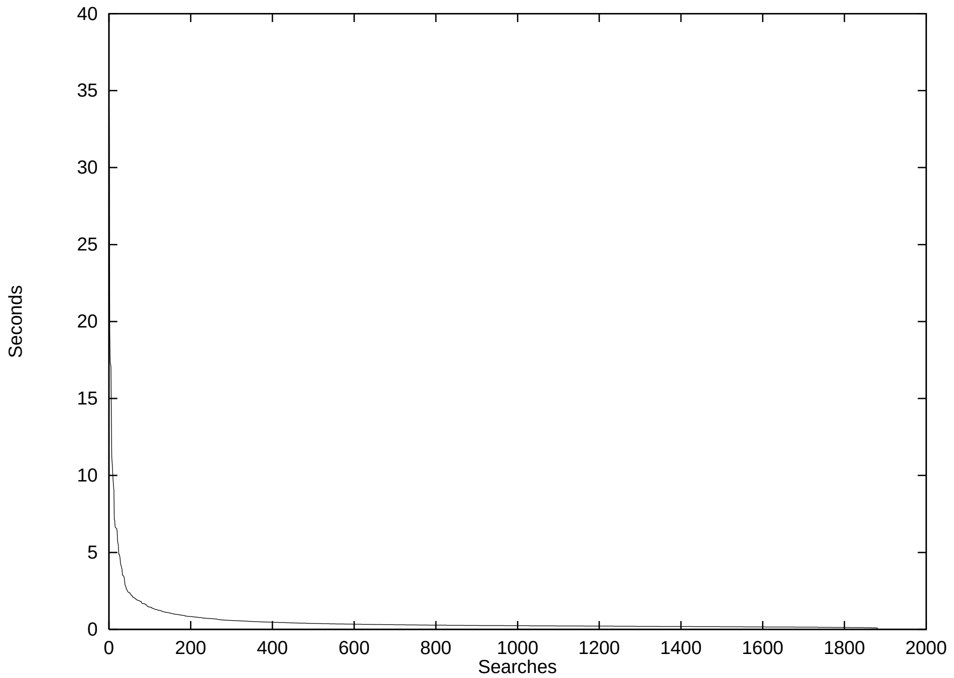

Figure 11: Distribution of search time per clause for **rp**.

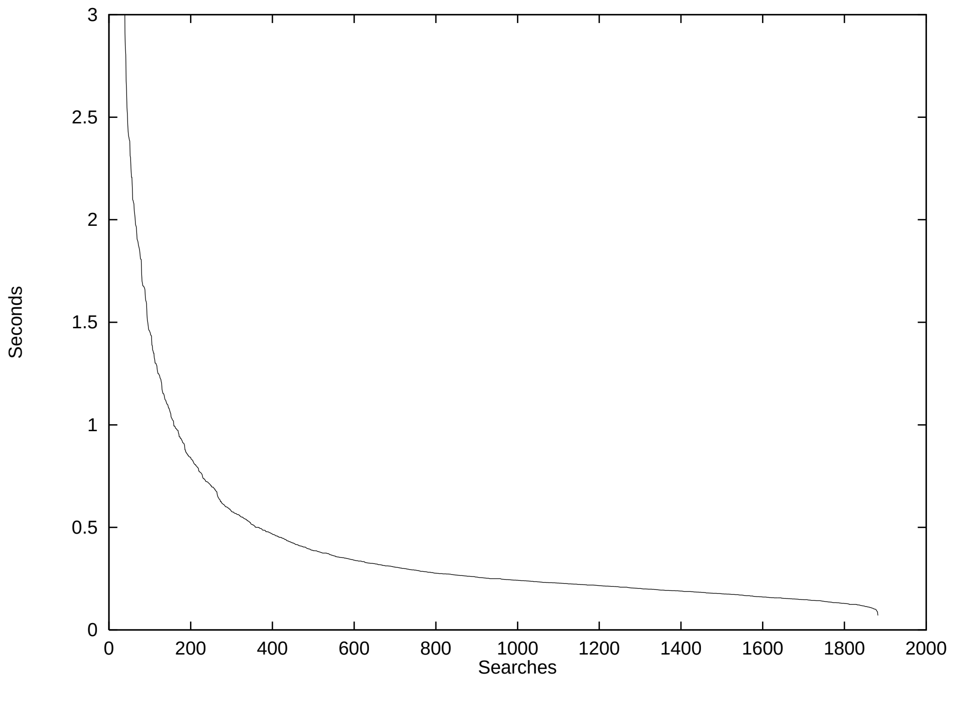

Figure 12: Tail of distribution of search time per clause for **rp**.

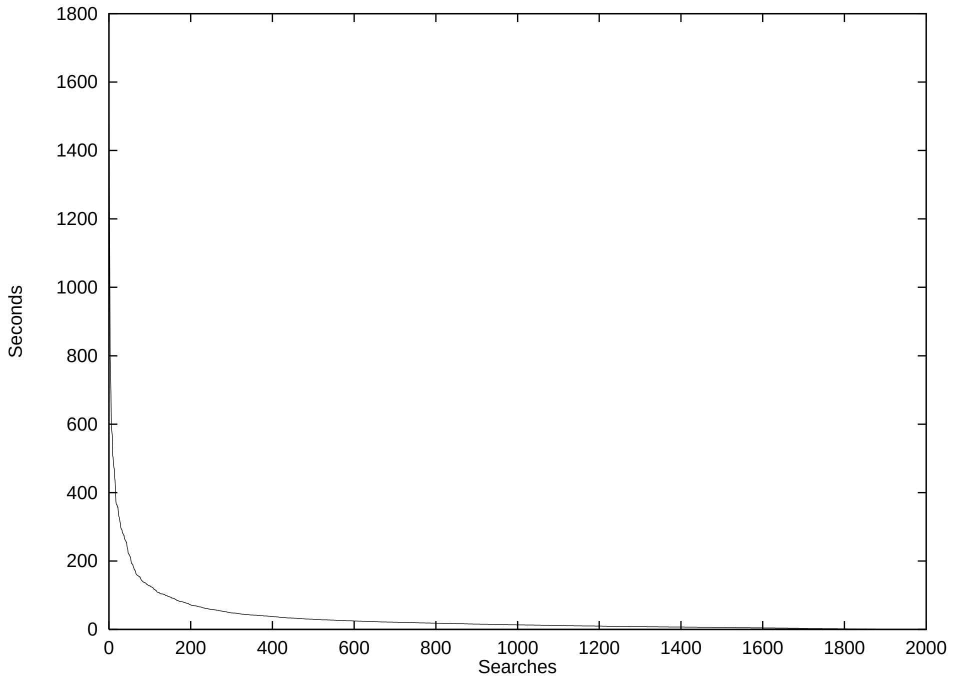

Figure 13: Distribution of search times for `rp`.

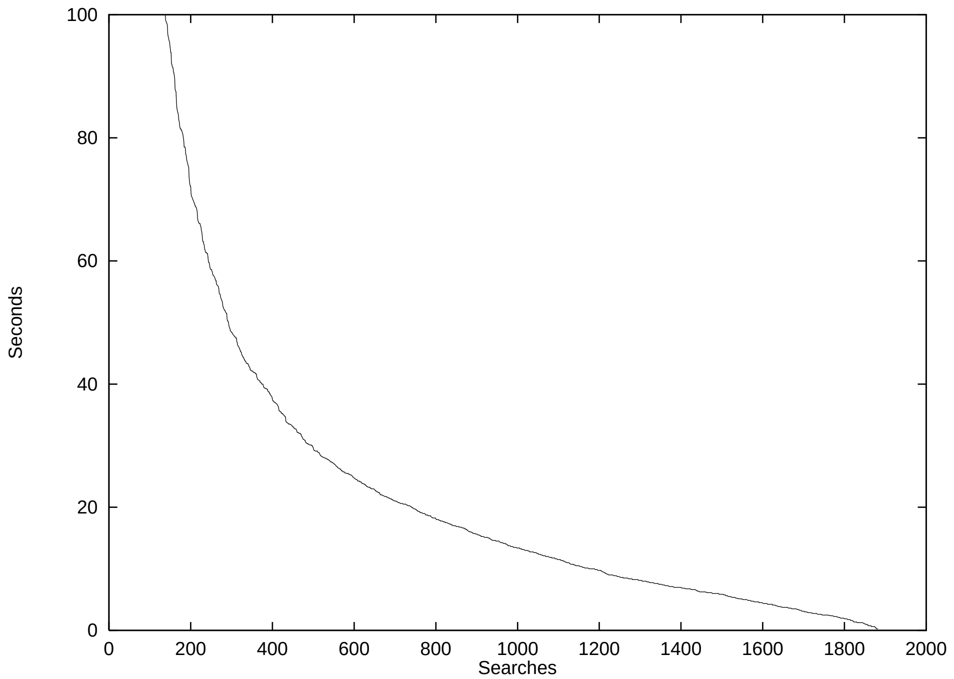

Figure 14: Tail of distribution of search times for `rp`.

6.2 Theory accuracy

Contingency tables showing the performance of the theories for each tag are given in Appendix B. The final theory itself is recorded in Appendix A. These tables give the results of testing each theory against a maximum of 1000 examples sampled from the test set. These were positive and negative examples of tag elimination for the tag in question, and *do not* reflect the overall tag disambiguation performance, since this also involves the lexical-statistical information. Fig 15 gives the contingency table for the entire theory.

	Actual		
Predicted↓	Pos	Neg	Total
Pos	3439	146	3585
Neg	2111	4304	6415
Total	5550	4450	10000

True positives =	62.0%
True negatives =	96.7%
Overall accuracy =	77.4%

Figure 15: Contingency table for complete theory

The real test of the resulting theory is its performance when embedded in the complete tagging system. Recall that, for each sentence of text, the following steps are carried out:

1. Each word in the sentence is replaced by its ambiguity class, as specified in the lexicon.
2. The induced theory is used to eliminate tags from ambiguity classes.
3. If ambiguities remain, the tag frequencies for words are used to eliminate an unlikely tag, and the rules reconsulted.
4. The final tagged sentence is output when all ambiguities have been eliminated.

Table 10 records for each tag, the frequency with which the system correctly identified a word as having that tag in a test set of 5000 sentences composed of 110716 “known” word tokens (i.e. those appearing in the lexicon). We record both the overall frequency with which a tag was correctly identified and the frequency with which it was identified given that it formed part of a non-singleton ambiguity class in the lexicon. Table 11 gives the same figures when no elimination rules were used (the ‘empty theory’), and instead we rely entirely on the lexicon for tagging. Table 12 directly compares the overall accuracies for each tag with and without the elimination rules. Note that the results for pos are something of an anomaly: because we have removed all sentences with right quotes, our theory can always correctly tag the token “ ’ ” as pos rather than rqt, (see Fig 39), giving a very high accuracy figure.

Inaccuracies caused by crude noise-handling when building the lexicon, as discussed in Section 4.2 were a significant cause of error. On a number of occasions the correct tag for a word can not be found, since it does not occur in the word’s ambiguity class as recorded in the lexicon. The frequency with which this occurred is recorded in the “Lex-Errs” columns for Tables 10 and 11. For example, “New” has an ambiguity class of np in the lexicon. The sentence “New stabs at automation had snafus.” is consequently tagged

	Ambiguous				Overall			
Tag	Correct	Total	Lex-Errs	Acc	Correct	Total	Lex-Errs	Acc
cc	12	27	1	44.4%	2802	2820	4	99.4%
cd	161	176	1	91.5%	4554	4570	2	99.6%
cln	0	0	0	n/a	261	261	0	100.0%
cma	0	0	0	n/a	5233	5233	0	100.0%
dt	396	533	0	74.3%	9674	9815	4	98.6%
dlr	0	0	0	n/a	852	852	0	100.0%
ex	52	52	0	100.0%	79	79	0	100.0%
fw	9	9	0	100.0%	9	9	0	100.0%
in	1763	1900	3	92.8%	11924	12064	6	98.8%
jj	2053	2598	55	79.0%	6630	7251	131	91.4%
jjr	311	331	2	94.0%	396	417	3	95.0%
jjs	66	81	1	81.5%	219	234	1	93.6%
lpn	0	0	0	n/a	76	76	0	100.0%
ls	0	0	0	n/a	0	1	1	0.0%
md	10	11	0	91.0%	1299	1300	0	99.9%
nn	3295	3740	15	88.1%	15451	16019	138	96.5%
nns	795	891	8	89.2%	7432	7551	31	98.4%
np	1317	1514	11	87.0%	9757	9981	38	97.8%
nps	164	341	13	48.1%	177	366	25	48.4%
pdt	23	37	3	62.2%	23	37	3	62.2%
pos	1035	1039	0	99.6%	1035	1039	0	99.6%
pnd	0	0	0	n/a	23	23	0	100.0%
pp	1	10	4	10.0%	1706	1717	6	99.4%
ppz	12	12	0	100.0%	969	969	0	100.0%
rb	536	991	22	54.1%	3269	3772	70	86.7%
rbr	100	195	0	51.3%	100	196	1	51.0%
rbs	48	51	0	94.1%	48	51	0	94.1%
rp	78	165	2	47.3%	78	182	19	42.9%
rpn	0	0	0	n/a	59	59	0	100.0%
stp	0	0	0	n/a	5016	5016	0	100.0%
sym	0	0	0	n/a	0	0	0	n/a
to	0	0	0	n/a	2778	2778	0	100.0%
uh	2	3	1	66.7%	4	6	2	66.7%
vb	2303	2474	27	93.1%	3259	3504	101	93.1%
vbd	1390	1599	16	87.0%	3224	3449	32	93.5%
vbg	553	661	1	83.7%	1585	1702	10	93.1%
vbn	1849	2077	5	89.0%	2337	2599	39	90.0%
vbp	722	852	52	84.7%	1197	1354	79	88.4%
vbz	263	339	0	77.6%	2283	2385	26	95.7%
wdt	176	243	0	72.4%	467	541	7	86.3%
wp	0	0	0	n/a	188	188	0	100.0%
wpz	0	0	0	n/a	18	18	0	100.0%
wrb	0	0	0	n/a	228	228	0	100.0%

Table 10: System tagging accuracy for each tag.

	Ambiguous				Overall			
Tag	Correct	Total	Lex-Errs	Acc	Correct	Total	Lex-Errs	Acc
cc	10	27	1	37.0%	2800	2820	4	99.3%
cd	161	176	1	91.5%	4554	4570	2	99.6%
cln	0	0	0	n/a	261	261	0	100.0%
cma	0	0	0	n/a	5233	5233	0	100.0%
dlr	0	0	0	n/a	852	852	0	100.0%
dt	392	533	0	73.5%	9670	9815	4	98.5%
ex	52	52	0	100.0%	79	79	0	100.0%
fw	9	9	0	100.0%	9	9	0	100.0%
in	1747	1900	3	91.9%	11908	12064	6	98.7%
jj	2000	2598	55	77.0%	6577	7251	131	90.7%
jjr	304	331	2	91.8%	389	417	3	93.3%
jjs	25	81	1	30.9%	178	234	1	76.1%
lpn	0	0	0	n/a	76	76	0	100.0%
ls	0	0	0	n/a	0	1	0	0.0%
md	10	11	0	90.9%	1299	1300	0	99.9%
nn	3011	3740	15	80.5%	15167	16019	138	94.7%
nns	772	891	8	86.6%	7409	7551	31	98.1%
np	1294	1514	11	85.5%	9734	9981	38	97.5%
nps	151	341	13	44.3%	164	366	25	44.8%
pdt	0	37	3	0.0%	0	37	3	0.0%
pnd	0	0	0	n/a	23	23	0	100.0%
pos	1039	1039	0	100.0%	1039	1039	0	100.0%
pp	0	10	4	0.0%	1705	1717	6	99.3%
ppz	12	12	0	100.0%	969	969	0	100.0%
rb	456	991	22	46.0%	3189	3772	70	84.5%
rbr	42	195	0	21.5%	42	196	1	21.4%
rbs	47	51	0	92.2%	47	51	0	92.2%
rp	93	165	2	56.4%	93	182	19	51.1%
rpn	0	0	0	n/a	59	59	0	100.0%
sym	0	0	0	n/a	0	0	0	n/a
stp	0	0	0	n/a	5016	5016	0	100.0%
to	0	0	0	n/a	2778	2778	0	100.0%
uh	2	3	1	66.7%	4	6	2	66.7%
vb	1644	2474	27	66.5%	2600	3504	101	74.2%
vbd	1040	1599	16	65.0%	2874	3449	32	83.3%
vbg	511	661	1	77.3%	1543	1702	10	90.7%
vbn	1467	2077	5	70.6%	1955	2599	39	75.2%
vbp	514	852	52	60.3%	989	1354	79	73.0%
vbz	132	339	0	38.9%	2152	2385	26	90.2%
wdt	0	243	0	0.0%	291	541	7	53.8%
wp	0	0	0	n/a	188	188	0	100.0%
wpz	0	0	0	n/a	18	18	0	100.0%
wrb	0	0	0	n/a	228	228	0	100.0%

Table 11: System tagging accuracy for each tag with no rules.

Tag	No Rules	Rules
cc	99.3%	99.4%
cd	99.6%	99.6%
cln	100.0%	100.0%
cma	100.0%	100.0%
dt	98.6%	98.6%
dlr	100.0%	100.0%
ex	100.0%	100.0%
fw	100.0%	100.0%
in	98.7%	98.8%
jj	90.7%	91.4%
jjr	93.3%	95.0%
jjs	76.1%	93.6%
lpn	100.0%	100.0%
ls	0.0%	0.0%
md	99.9%	99.9%
nn	94.7%	96.5%
nns	98.1%	98.4%
np	97.5%	97.8%
nps	44.8%	48.4%
pdt	0.0%	62.2%
pos	100.0%	99.6%
pnd	100.0%	100.0%
pp	99.3%	99.4%
ppz	100.0%	100.0%
rb	84.5%	86.7%
rbr	21.4%	51.0%
rbs	92.2%	94.1%
rp	51.1%	42.9%
rpn	100.0%	100.0%
stp	100.0%	100.0%
sym	n/a	n/a
to	100.0%	100.0%
uh	66.7%	66.7%
vb	74.2%	93.1%
vbd	83.3%	93.5%
vbg	90.7%	93.1%
vbn	75.2%	90.0%
vbp	73.0%	88.4%
vbz	90.2%	95.7%
wdt	53.8%	86.3%
wp	100.0%	100.0%
wpz	100.0%	100.0%
wrb	100.0%	100.0%

Table 12: System tagging accuracy with and without rules.

as [np,vbz,in,nn,vbd,nns,stp] by the lexicon, so that the correct jj tag for "New" can not be found. Also, such inaccuracies have a knock-on effect: by ensuring that an incorrect tag is found for one word, the chances of incorrect tag eliminations for other words are increased. To get a measure of the degradation produced by these lexicon-induced errors, Table 13 records per-tag accuracies once these errors have been excluded.

Table 14 summarises the performance of the entire theory, and Table 15 compares the induced theory with the empty theory. Finally, in Table 16, we record the performance of the system on the level of correctly tagging an entire sentence of known words. As before, we give results including lexicon-induced errors and excluding them. The "Lex-Errs" column records the number of sentences which included words whose correct tags were missing in the lexicon. Table 17 compares the performance of the empty theory against the induced theory.

7 Conclusions and Future Work

7.1 Analysis of results

The overall per-word token accuracy of 96.4% (Table 14)is a respectable result, (Daelemans et al [5] report 96.7% on known words) and quite an encouraging one given that this is the first time ILP has been applied to the tagging problem. The per-sentence accuracy although still quite low at 49.5% is as expected given the per-word accuracy.

An advantage of the ILP approach is that the elimination rules are intelligible, and are therefore open to scrutiny. Indeed, there is no reason why a linguist should not input some elimination rules by hand leaving the ILP to find the non-obvious ones. Also, having an elimination theory that falls into a part for each tag allows us to pinpoint the weak parts of the theory.

As expected, almost all the theories are seriously undergeneral. In many applications of ILP, undergeneral theories would introduce serious inaccuracy, since the Closed World Assumption would be invoked to classify all uncovered examples as negatives, many erroneously. In this system when a tag can not be eliminated from an ambiguous tag, we do not then assume that it must be the correct tag: instead we see if other tags can be eliminated, and if not fall back on statistical information in the lexicon. Note that using the lexicon alone allows one to achieve a per-word accuracy of 94.1%

As Table 10 makes clear the is considerable variation in the system's success in correctly identifying different tags. The tag rp is only correctly identified 42.9% of the time. Fortunately, it does not occur particularly often!—which is chiefly why it is so poorly identified. The problem here is that rp is often found in ambiguity classes along with in and rb, such as is the case for the words "along", "around", "down" and "off". Usually in or rb is the correct tag. This probably affected the The theories for elimination of in and rb which are quite undergeneral at a true positives rate of 48.8% and 54.5% respectively. So these tags are not as eliminated as often as they should be. rp though has quite a low true negatives rate at 90.0%. Similar problems also occur for the poor performers nps, pdt, rbr and uh.

Tag	Ambiguous	Overall
cc	46.2%	99.5%
cd	92.0%	99.7%
cln	n/a	100.0%
cma	n/a	100.0%
dt	74.3%	98.6%
dlr	n/a	100.0%
ex	100.0%	100.0%
fw	100.0%	100.0%
in	92.9%	98.9%
jj	80.7%	93.1%
jjr	94.5%	95.7%
jjs	82.5%	94.0%
lpn	n/a	100.0%
ls	n/a	n/a
nn	88.5%	97.3%
nns	90.0%	98.8%
np	87.6%	98.1%
nps	50.0%	51.9%
md	90.9%	99.9%
pdt	67.6%	67.6%
pos	99.6%	99.6%
pnd	n/a	100.0%
pp	16.7%	99.7%
ppz	100.0%	100.0%
rb	55.3%	88.3%
rbr	51.3%	51.3%
rbs	94.1%	94.1%
rp	47.9%	47.9%
rpn	n/a	100.0%
stp	n/a	100.0%
sym	n/a	n/a
to	n/a	100.0%
uh	100.0%	100.0%
vb	94.1%	95.8%
vbd	87.8%	94.4%
vbg	83.8%	93.7%
vbn	89.2%	91.3%
vbp	90.2%	93.9%
vbz	77.6%	96.8%
wdt	72.4%	87.5%
wp	n/a	100.0%
wpz	n/a	100.0%
wrb	n/a	100.0%

Table 13: System tagging accuracy for each tag, excluding lexicon-induced errors.

	Ambiguous				Overall			
Lexical errors?	Correct	Total	Lex-Errs	Acc	Correct	Total	Lex-Errs	Acc
Included	19495	22956	247	84.9%	106719	110716	783	96.4%
Excluded	19495	22709	0	85.8%	106719	109933	0	97.1%

Table 14: System tagging accuracy.

	Ambiguous				Overall			
Theory	Correct	Total	Lex-Errs	Acc	Correct	Total	Lex-Errs	Acc
Full	19495	22956	247	84.9%	106719	110716	783	96.4%
Empty	16937	22956	247	73.8%	104161	110716	783	94.1%

Table 15: System tagging accuracy with and without rules.

Lexical errors?	Correct	Total	Lex-Errs	Acc
Included	2475	5000	698	49.5%
Excluded	2475	4302	0	57.6%

Table 16: System tagging accuracy for sentences.

Theory	Correct	Total	Lex-Errs	Acc
Full	2475	5000	698	49.5%
Empty	1520	5000	698	30.4%

Table 17: System tagging accuracy for sentences with and without rules.

7.2 Future work

Here we give a list of necessary future work on using ILP for tagging. The list is ordered in terms of importance.

Unknown words At present, we have *no* method of dealing with words that have not occurred in the training set. This means that the present system is not yet a practical tagging system. There is nothing to stop us "bolting on" a sensible method of dealing with unknown words which takes an unknown word and returns a tag ambiguity class. Brill [2] and Daelemans et al [5] both have methods for dealing with unknown words; these techniques could be incorporated into future work.

Noise handling Noisy data is particularly problematic when one is aiming for accuracies of over 96%. [11] use two linguists to pore over the training data to weed out noise, and this probably has a lot to do with their impressive accuracy figures. The crude method of noise-handling used in the work reported here is the single biggest cause of inaccuracy. Future work should take into account the absolute frequency with which a tag appears for a particular word, rather than the simply the proportion (relative frequency) with which it appears. We could also use an extra filter when choosing tagged sentences to include in the training data: if a sentence contains a word tagged with a tag that is not included in that word's ambiguity class in the lexicon, then there is either an error in the lexicon or the sentence, and it would be best not to include this sentence. This occurs in about 14% of sentences. (It was intended to use this filter in the current study, but unfortunately it was erroneously omitted. Better results should be obtainable using this filter, since there will be less noise in the training set.)

Combining rules and lexical statistics Our system uses two sources of information to tag a word:

1. The frequencies with which the word has been given particular tags, irrespective of context.
2. The context of the word in terms of the tags of the other words in the sentence.

Essentially, we are trying to find that tag *tag* which is in the word's ambiguity class and has the lowest value for $P(tag|word, context)$ At present, we use the lexicon to estimate $P(tag|word)$ and have a collection of rules, such that if a rule of the form $remove(tag) \leftarrow context$ is present, then we assume, sometimes incorrectly, that $P(tag|context) < 0.05$ and also that if this inequality holds then *tag* has a lower probability than some other tag. If no elimination rule covers the word then we simply use $P(tag|word)$ to estimate $P(tag|word, context)$.

A better approach would be to *label* induced clauses with estimates of their accuracy and to allow (labelled) clauses with much lower estimated accuracy to be included in the final theory. To see how this would help, consider the large number of ambiguity classes with two members of equal frequency, e.g.

```
absurdist [[jj,0.500],[nn,0.500]]
```

If no elimination rule covers an occurrence of "absurdist" in a particular sentence, then the lexicon will choose to eliminate jj, even though this will be the wrong choice a full 50% of the time. Suppose instead a rule of estimated accuracy of only 60% was available which covered the instance of "absurdist". Even though the rule has low accuracy, it would still be better to use it rather than the lexical probabilities. Using labelled rules should increase system accuracy, but will increase theory complexity, since there will be a large number of rules.

Large example sets Although the caching approach made using ILP a practical proposition, learning was still slow and only used a sample of the available data. The broad idea behind caching: that space should be traded for time, seems a good one. In ILP, running out of time is usually more of a danger than running out of memory. The caching approach is very simple; more sophisticated approaches, perhaps involving storing away multiple bottom clauses in some suitable data structure, are likely to bring further improvements in efficiency.

Background knowledge The definitions of noun phrase, etc are probably sub-optimal. These definitions were devised by the author (who is not a linguist) using a mixture of intuition and trail and error. Background knowledge devised by a NLP expert is likely to be far more powerful.

Including quotes The avoidance of sentences with quotes is a non-essential restriction and is primarily a matter of convenience. This should be dropped in future work.

Acknowledgements This work was supported by ESPRIT IV Long Term Resarch Project ILP II (No. 20237). The author would like to thank Walter Daelemans for his help on corpus-based linguistics, Ashwin Srinivasan for his advice on modifying P-Progol and Stephen Muggleton for his comments of previous versions.

References

[1] Steven Abney. Part-of-speech tagging and partial parsing. In Ken Church, Steve Young, and Gerrit Bloothooft, editors, *Corpus-Based Methods in Language and Speech*. Kluwer, Dordrecht, 1996.

[2] Eric Brill. Some advances in transformation-based part of speech tagging. In *AAAI94*, 1994.

[3] J. Cussens. Bayesian Inductive Logic Programming with explicit probabilistic bias. Technical Report PRG-TR-24-96, Oxford University Computing Laboratory, 1996.

[4] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In *Third Conference on Applied Natural LInguistic Processing (ANLP-92)*, pages 133–140, 1992.

[5] W. Daelemans, J. Zavrel, P. Berck, and S. Gillis. MBT: A memory-based part of speech tagger-generator. In *Proceedings of the Fourth Workshop on Very Large Corpora,*, pages 14–27, Copenhagen, 1996.

[6] S. Džeroski and I. Bratko. Using the m-estimate in inductive logic programming. In C. Rouveirol, editor, *Proceedings of the ECAI-92 Workshop on Logical Approaches to Machine Learning*, 1992.

[7] M.P. Marcus, B. Santorini, and Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19, 1993.

[8] S. Muggleton. Inverse entailment and Progol. *New Generation Computing Journal*, 13:245–286, 1995.

[9] S. Muggleton, C.D. Page, and A. Srinivasan. An initial experiment into stereochemistry-based drug design using ILP. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 245–261. Stockholm University, Royal Institute of Technology, 1996.

[10] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.

[11] Christer Samuelsson, Pasi Tapanainen, and Atro Voutilainen. Inducing constraint grammars. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Inference: Learning Syntax from Sentences*, volume 1147 of *Lecture Notes in Artificial Intelligence*, pages 146–155. Springer, 1996.

[12] A. Srinivasan and R. Camacho. Experiments in numerical reasoning with Inductive Logic Programming. Technical Report PRG-TR-22-96, Oxford University Computing Laboratory, 1996. Submitted to the Journal of Logic Programming.

[13] Pasi Tapanainen and Atro Voutilainen. Tagging accurately – Don't guess if you know. In *Proc. ANLP94*, 1994.

A Final theory

```
rmv(A,B,cc) :- advp(A,C).
rmv(A,B,cc) :- cma(B,C).
rmv(A,B,cc) :- in(A,C), adjp1(C,D).
rmv(A,B,cc) :- in(A,C), end(C).
rmv(A,B,cc) :- in(A,C), nounp1(C,D), dt(D,E), in(E,F).
rmv(A,B,cc) :- in(A,C), nounp1(C,D), end(D).
rmv(A,B,cc) :- in(A,C), sverb(C,D).
rmv(A,B,cc) :- in(A,C), vp1(C,D), nounp1(D,E).
rmv(A,B,cc) :- noun(B,C), in(A,D).
rmv(A,B,cc) :- noun(B,C), nounp(C,D), nounp(D,E).
rmv(A,B,cc) :- noun(B,C), stp(C,D).
rmv(A,B,cc) :- noun(B,C), vp(C,D), adjp(D,E).
rmv(A,B,cc) :- noun(B,C), vp(C,D), dt(D,E).
rmv(A,B,cc) :- noun(B,C), vp(C,D), vpart(D,E).
rmv(A,B,cc) :- nounp(B,C), in(A,D), noun(D,E), in(E,F).
rmv(A,B,cc) :- nounp(B,C), in(C,D), vp1(A,E).
rmv(A,B,cc) :- nounp(B,C), stp(C,D), in(A,E).
rmv(A,B,cc) :- nounp(B,C), vp(C,D), in(A,E).
rmv(A,B,cc) :- stp(B,C).
rmv(A,B,cc) :- sverb(A,C), wdt(C,D).
rmv(A,B,cc) :- to(B,C).
rmv(A,B,cc) :- vp1(A,C), noun(C,D), dt(D,E), in(E,F).
rmv(A,B,cc) :- vp1(A,C), noun(C,D), nounp1(C,E), end(E).
rmv(A,B,cc) :- vp1(A,C), nounp1(C,D), adjp1(D,E).
rmv(A,B,cc) :- vp1(A,C), nounp1(C,D), cc(D,E).
rmv(A,B,cc) :- vp1(A,C), nounp1(C,D), vp1(D,E).
rmv(A,B,cc) :- vpart(B,C), vp1(A,D), nounp1(D,E).
rmv(A,B,cc) :- vpart(B,C), vpart(C,D).
```

Figure 16: Theory for cc.

```
rmv(A,B,dt) :- adjp1(A,C), nounp1(C,D).
rmv(A,B,dt) :- adjp1(A,C), vp1(C,D).
rmv(A,B,dt) :- advp(B,C), vp(C,D), nounp(D,E).
rmv(A,B,dt) :- advp(B,C), vp1(A,D), nounp1(D,E), end(E).
rmv(A,B,dt) :- dt(B,C).
rmv(A,B,dt) :- ex(B,C).
rmv(A,B,dt) :- in(B,C), adjp(C,D), vp1(A,E).
rmv(A,B,dt) :- in(B,C), noun(C,D), nounp(D,E), vp1(A,F).
rmv(A,B,dt) :- in(B,C), nounp(C,D), vp(D,E), vp1(A,F).
rmv(A,B,dt) :- noun(B,C), noun(C,D), vp1(A,E), advp(E,F).
rmv(A,B,dt) :- nounp(B,C), advp(C,D), vp(D,E), vp1(A,F).
rmv(A,B,dt) :- nounp(B,C), noun(C,D), pos(D,E), nounp(E,F).
rmv(A,B,dt) :- nounp(B,C), nounp(C,D), md(D,E), vp1(A,F).
rmv(A,B,dt) :- nounp(B,C), pos(C,D), noun(D,E), vp1(A,F).
rmv(A,B,dt) :- nounp(B,C), pos(C,D), nounp(D,E), in(E,F).
rmv(A,B,dt) :- nounp(B,C), sverb(C,D), vpart(D,E), noun(E,F).
rmv(A,B,dt) :- nounp(B,C), svp1(A,D), noun(D,E), cc(E,F).
rmv(A,B,dt) :- nounp(B,C), vb(C,D), vp1(A,E).
rmv(A,B,dt) :- nounp(B,C), vp(C,D), ppz(D,E), vp1(A,F).
rmv(A,B,dt) :- nounp(B,C), vp(C,D), vb(A,E), vp1(A,F).
rmv(A,B,dt) :- nounp(B,C), vp(C,D), vb(D,E).
rmv(A,B,dt) :- nounp1(A,C).
rmv(A,B,dt) :- pp(B,C).
rmv(A,B,dt) :- ppz(B,C), adjp(C,D).
rmv(A,B,dt) :- sverb(A,C), cma(C,D), in(B,E).
rmv(A,B,dt) :- sverb(A,C), noun(B,D), noun(D,E), noun(E,F).
rmv(A,B,dt) :- sverb(A,C), noun(B,D), nounp(D,E), vp(E,F).
rmv(A,B,dt) :- sverb(A,C), nounp1(C,D), nounp1(D,E), in(B,F).
rmv(A,B,dt) :- sverb(A,C), nounp1(C,D), pos(D,E).
rmv(A,B,dt) :- sverb(B,C), noun(C,D), noun(D,E).
rmv(A,B,dt) :- vp(B,C), dt(C,D), nounp(D,E), noun(E,F).
rmv(A,B,dt) :- vp(B,C), in(C,D), nounp(D,E), stp(E,F).
rmv(A,B,dt) :- vp(B,C), vpart(C,D), vpart(D,E).
rmv(A,B,dt) :- vp1(A,C), advp(C,D), nounp1(D,E), nounp1(E,F).
rmv(A,B,dt) :- vp1(A,C), noun(C,D), noun(D,E), adjp1(E,F).
rmv(A,B,dt) :- vpart(A,C), nounp(B,D), vp(D,E).
rmv(A,B,dt) :- vpart(A,C), sverb(C,D), nounp1(D,E), end(E).
rmv(A,B,dt) :- vpart(A,C), vp1(A,D), nounp1(D,E), adjp1(E,F).
rmv(A,B,dt) :- vpart(A,C), vp1(A,D), nounp1(D,E), dt(E,F).
rmv(A,B,dt) :- vpart(A,C), vp1(C,D), noun(D,E), dt(E,F).
rmv(A,B,dt) :- vpart(A,C), vp1(C,D), nounp(B,E), nounp(E,F).
rmv(A,B,dt) :- vpart(A,C), vp1(C,D), nounp1(D,E), cma(E,F).
```

Figure 17: Theory for `dt`.

```
rmv(A,B,ex) :- in(B,C).
rmv(A,B,ex) :- stp(B,C).
```

Figure 18: Theory for `ex`.

```
rmv(A,B,fw) :- nounp(B,C), in(C,D), noun(D,E).
rmv(A,B,fw) :- nounp1(A,C), to(C,D).
```

Figure 19: Theory for `fw`.

```
rmv(A,B,in) :- adjp(B,C), dt(A,D).
rmv(A,B,in) :- adjp(B,C), in(A,D).
rmv(A,B,in) :- adjp(B,C), in(C,D), nounp(D,E), cds(E,F).
rmv(A,B,in) :- advp(B,C), cma(A,D), nounp1(D,E), in(E,F).
rmv(A,B,in) :- advp(B,C), in(C,D), cma(A,E).
rmv(A,B,in) :- advp(B,C), in(C,D), nounp(D,E).
rmv(A,B,in) :- advp(B,C), in(C,D), nounp1(A,E).
rmv(A,B,in) :- advp(B,C), noun(A,D), nounp1(D,E).
rmv(A,B,in) :- advp(B,C), vp(C,D).
rmv(A,B,in) :- cds(B,C), cma(A,D), cds(D,E), nounp1(E,F).
rmv(A,B,in) :- cds(B,C), noun(C,D), in(D,E), cma(A,F).
rmv(A,B,in) :- cma(A,C), cds(C,D), nounp1(D,E), cln(E,F).
rmv(A,B,in) :- dt(A,C), in(C,D).
rmv(A,B,in) :- end(A), noun(B,C), vp(C,D).
rmv(A,B,in) :- in(A,C), end(C).
rmv(A,B,in) :- md(B,C).
rmv(A,B,in) :- noun(B,C), cma(C,D), in(A,E).
rmv(A,B,in) :- noun(B,C), dt(A,D).
rmv(A,B,in) :- noun(B,C), nounp(B,D), stp(D,E), in(A,F).
rmv(A,B,in) :- noun(B,C), stp(C,D), in(A,E).
rmv(A,B,in) :- nounp(B,C), nounp(C,D), dt(A,E).
rmv(A,B,in) :- vp(B,C).
```

Figure 20: Theory for `in`.

```
rmv(A,B,jj) :- adjp(B,C), in(C,D).
rmv(A,B,jj) :- advp(A,C), dt(B,D).
rmv(A,B,jj) :- cc(A,C), prnoun(B,D).
rmv(A,B,jj) :- cma(B,C), vp(C,D).
rmv(A,B,jj) :- dt(A,C), in(B,D), dt(D,E).
rmv(A,B,jj) :- dt(A,C), in(B,D), noun(D,E).
rmv(A,B,jj) :- dt(A,C), in(B,D), nounp(D,E), cma(E,F).
rmv(A,B,jj) :- dt(A,C), in(B,D), nounp(D,E), noun(E,F).
rmv(A,B,jj) :- dt(B,C), nounp(C,D), in(D,E).
rmv(A,B,jj) :- dt(B,C), nounp(C,D), nounp(D,E).
rmv(A,B,jj) :- dt(B,C), to(A,D).
rmv(A,B,jj) :- dt(B,C), vp1(A,D).
rmv(A,B,jj) :- end(A), prnoun(B,C).
rmv(A,B,jj) :- in(B,C), adjp(C,D), nounp(D,E), in(E,F).
rmv(A,B,jj) :- in(B,C), adjp1(A,D).
rmv(A,B,jj) :- in(B,C), nounp(C,D), cma(D,E), advp(E,F).
rmv(A,B,jj) :- in(B,C), prnoun(A,D).
rmv(A,B,jj) :- md(A,C).
rmv(A,B,jj) :- noun(A,C), nounp1(C,D), end(D), nounp(B,E).
rmv(A,B,jj) :- noun(A,C), prnoun(B,D), noun(D,E).
rmv(A,B,jj) :- noun(B,C), cc(C,D), prnoun(D,E).
rmv(A,B,jj) :- nounp(B,C), cma(C,D), cc(D,E), prnoun(B,F).
rmv(A,B,jj) :- nounp(B,C), cma(C,D), in(D,E), prnoun(B,F).
rmv(A,B,jj) :- nounp(B,C), nounp(C,D), nounp1(A,E), end(E).
rmv(A,B,jj) :- nounp(B,C), nounp1(A,D), end(D), cnoun(A,E).
rmv(A,B,jj) :- nounp(B,C), pos(C,D), prnoun(B,E).
rmv(A,B,jj) :- nounp(B,C), sverb(C,D), nounp(D,E), prnoun(B,F).
rmv(A,B,jj) :- nounp(B,C), vp(C,D), vpart(D,E), prnoun(B,F).
rmv(A,B,jj) :- nounp1(A,C), cc(C,D), prnoun(A,E).
rmv(A,B,jj) :- nounp1(A,C), dt(C,D), in(D,E), prnoun(B,F).
rmv(A,B,jj) :- nounp1(A,C), nounp1(C,D), cma(D,E), prnoun(C,F).
rmv(A,B,jj) :- nounp1(A,C), prnoun(B,D), cma(D,E).
rmv(A,B,jj) :- pos(B,C).
rmv(A,B,jj) :- pp(A,C).
rmv(A,B,jj) :- ppz(B,C).
rmv(A,B,jj) :- prnoun(A,C), end(C).
rmv(A,B,jj) :- prnoun(B,C), pos(C,D).
rmv(A,B,jj) :- prnoun(B,C), prnoun(A,D).
rmv(A,B,jj) :- prnoun(B,C), prnoun(C,D), cma(D,E).
rmv(A,B,jj) :- prnoun(B,C), vp(C,D).
rmv(A,B,jj) :- stp(B,C), nounp1(A,D), dt(D,E).
rmv(A,B,jj) :- vp(B,C), adjp1(A,D).
rmv(A,B,jj) :- vp(B,C), dt(A,D).
rmv(A,B,jj) :- vp(B,C), in(C,D).
rmv(A,B,jj) :- vp(B,C), nounp(C,D).
rmv(A,B,jj) :- vp(B,C), nounp1(A,D).
rmv(A,B,jj) :- wdt(A,C), noun(C,D).
```

Figure 21: Theory for `jj`.

```
rmv(A,B,jjr) :- adjp(B,C), adjp(C,D), dt(A,E).
rmv(A,B,jjr) :- adjp(B,C), cma(C,D).
rmv(A,B,jjr) :- adjp(B,C), dt(A,D), vb(D,E).
rmv(A,B,jjr) :- adjp(B,C), dt(A,D), vp1(D,E), cnoun(E,F).
rmv(A,B,jjr) :- adjp(B,C), in(C,D).
rmv(A,B,jjr) :- adjp(B,C), noun(A,D).
rmv(A,B,jjr) :- adjp(B,C), nounp(C,D), nounp(D,E), dt(A,F).
rmv(A,B,jjr) :- adjp(B,C), nounp1(A,D), vp1(D,E).
rmv(A,B,jjr) :- adjp(B,C), stp(C,D).
rmv(A,B,jjr) :- adjp(B,C), to(C,D).
rmv(A,B,jjr) :- adjp(B,C), vp1(A,D), nounp1(D,E), cma(E,F).
rmv(A,B,jjr) :- adjp(B,C), vp1(A,D), nounp1(D,E), prnoun(E,F).
rmv(A,B,jjr) :- advp(A,C), dt(C,D), adjp(B,E).
rmv(A,B,jjr) :- advp(A,C), md(C,D).
rmv(A,B,jjr) :- advp(A,C), sverb(C,D), cnoun(D,E), adjp(B,F).
rmv(A,B,jjr) :- advp(A,C), vp(B,D).
rmv(A,B,jjr) :- advp(B,C), cma(C,D), nounp(D,E).
rmv(A,B,jjr) :- advp(B,C), nounp1(A,D).
rmv(A,B,jjr) :- advp(B,C), vpart(A,D).
rmv(A,B,jjr) :- advp(B,C), vpart(C,D).
rmv(A,B,jjr) :- cma(B,C), nounp1(A,D), nounp1(D,E), in(E,F).
rmv(A,B,jjr) :- dt(B,C), nounp(C,D), cma(D,E), noun(E,F).
rmv(A,B,jjr) :- dt(B,C), nounp(C,D), stp(D,E).
rmv(A,B,jjr) :- dt(B,C), nounp(C,D), vp(D,E).
rmv(A,B,jjr) :- dt(B,C), nounp1(A,D).
rmv(A,B,jjr) :- end(A), advp(B,C).
rmv(A,B,jjr) :- end(A), dt(B,C).
rmv(A,B,jjr) :- end(A), nounp(B,C), cma(C,D), nounp(D,E).
rmv(A,B,jjr) :- noun(A,C), cma(B,D), nounp(D,E).
rmv(A,B,jjr) :- noun(A,C), dt(C,D), cma(B,E).
rmv(A,B,jjr) :- nounp(B,C), nounp1(A,D), nounp1(A,E), adjp1(E,D).
rmv(A,B,jjr) :- nounp1(A,C), cds(C,D), nounp1(D,E).
rmv(A,B,jjr) :- nounp1(A,C), in(C,D), noun(D,E), in(E,F).
rmv(A,B,jjr) :- nounp1(A,C), nounp1(C,D), in(D,E), cds(E,F).
rmv(A,B,jjr) :- nounp1(A,C), nounp1(C,D), nounp1(D,E), cncy(E,F).
rmv(A,B,jjr) :- stp(B,C), nounp1(A,D), dt(D,E).
rmv(A,B,jjr) :- sverb(A,C), dt(B,D).
rmv(A,B,jjr) :- vb(B,C).
rmv(A,B,jjr) :- vpart(A,C), dt(B,D).
```

Figure 22: Theory for jjr.

```
rmv(A,B,jjs) :- adjp(B,C), adjp(C,D), dt(A,E).
rmv(A,B,jjs) :- adjp(B,C), cc(A,D), adjp1(D,E).
rmv(A,B,jjs) :- adjp(B,C), cc(C,D).
rmv(A,B,jjs) :- adjp(B,C), cma(C,D).
rmv(A,B,jjs) :- adjp(B,C), cnoun(C,D), vp(D,E), dt(A,F).
rmv(A,B,jjs) :- adjp(B,C), dt(A,D), advp(D,E).
rmv(A,B,jjs) :- adjp(B,C), dt(A,D), cma(D,E).
rmv(A,B,jjs) :- adjp(B,C), dt(A,D), end(D).
rmv(A,B,jjs) :- adjp(B,C), dt(A,D), in(D,E), cds(E,F).
rmv(A,B,jjs) :- adjp(B,C), nounp(C,D), nounp(D,E), dt(A,F).
rmv(A,B,jjs) :- adjp(B,C), nounp(C,D), nounp(D,E), in(E,F).
rmv(A,B,jjs) :- adjp(B,C), nounp(C,D), sverb(D,E), dt(A,F).
rmv(A,B,jjs) :- adjp(B,C), nounp1(A,D), dt(D,E).
rmv(A,B,jjs) :- adjp(B,C), to(C,D).
rmv(A,B,jjs) :- adjp(B,C), vp1(A,D).
rmv(A,B,jjs) :- advp(B,C), cma(C,D).
rmv(A,B,jjs) :- advp(B,C), nounp1(A,D).
rmv(A,B,jjs) :- advp(B,C), vp1(A,D).
rmv(A,B,jjs) :- advp(B,C), vpart(C,D).
```

Figure 23: Theory for `jjs`.

```
rmv(A,B,md) :- cma(B,C).
rmv(A,B,md) :- in(A,C).
rmv(A,B,md) :- noun(B,C).
rmv(A,B,md) :- stp(B,C).
```

Figure 24: Theory for `md`.

```
rmv(A,B,nn) :- adjp(B,C), adjp(C,D).
rmv(A,B,nn) :- adjp(B,C), cnoun(C,D), nounp(D,E).
rmv(A,B,nn) :- adjp(B,C), nns(C,D).
rmv(A,B,nn) :- adjp(B,C), noun(C,D), in(A,E).
rmv(A,B,nn) :- adjp(B,C), plnoun(C,D).
rmv(A,B,nn) :- advp(A,C), md(C,D).
rmv(A,B,nn) :- advp(A,C), nounp(B,D).
rmv(A,B,nn) :- advp(A,C), nounp1(C,D).
rmv(A,B,nn) :- cc(A,C), dt(B,D).
rmv(A,B,nn) :- cc(B,C), nounp(C,D), np(A,E).
rmv(A,B,nn) :- cc(B,C), nounp(C,D), prnoun(A,E).
rmv(A,B,nn) :- cds(B,C), dt(A,D), in(D,E).
rmv(A,B,nn) :- cds(B,C), nns(C,D).
rmv(A,B,nn) :- cds(B,C), plnoun(C,D).
rmv(A,B,nn) :- dt(B,C), adjp(C,D).
rmv(A,B,nn) :- dt(B,C), cma(A,D).
rmv(A,B,nn) :- dt(B,C), in(A,D).
rmv(A,B,nn) :- dt(B,C), nounp(C,D), in(D,E).
rmv(A,B,nn) :- dt(B,C), nounp(C,D), nounp(D,E).
rmv(A,B,nn) :- dt(B,C), plnoun(A,D).
rmv(A,B,nn) :- dt(B,C), plnoun(C,D).
rmv(A,B,nn) :- dt(B,C), vp1(A,D).
rmv(A,B,nn) :- end(A), prnoun(B,C).
rmv(A,B,nn) :- in(A,C), prnoun(C,D).
rmv(A,B,nn) :- md(A,C).
rmv(A,B,nn) :- noun(A,C), nounp(B,D), cma(D,E), noun(E,F).
rmv(A,B,nn) :- noun(A,C), nounp1(C,D), nounp1(D,E), snoun(B,F).
```

Figure 25: Theory for nn.

```
rmv(A,B,nn) :- nounp(B,C), cma(A,D), nounp1(D,E), adjp1(E,F).
rmv(A,B,nn) :- nounp(B,C), cma(C,D), noun(D,E), prnoun(B,F).
rmv(A,B,nn) :- nounp(B,C), cma(C,D), nounp1(A,E), nounp1(E,F).
rmv(A,B,nn) :- nounp(B,C), in(C,D), dt(D,E), plnoun(E,F).
rmv(A,B,nn) :- nounp(B,C), in(C,D), nounp(D,E), vpart(E,F).
rmv(A,B,nn) :- nounp(B,C), in(C,D), plnoun(A,E).
rmv(A,B,nn) :- nounp(B,C), nns(A,D).
rmv(A,B,nn) :- nounp(B,C), noun(C,D), cma(D,E), noun(E,F).
rmv(A,B,nn) :- nounp(B,C), nounp(B,D), nounp(D,C), plnoun(A,E).
rmv(A,B,nn) :- nounp(B,C), nounp(C,D), nounp1(A,E), noun(E,F).
rmv(A,B,nn) :- nounp(B,C), nounp(C,D), prnoun(A,E).
rmv(A,B,nn) :- nounp(B,C), nounp(C,D), snoun(D,E), nounp1(A,F).
rmv(A,B,nn) :- nounp(B,C), nounp1(A,D), nounp1(D,E), prnoun(E,F).
rmv(A,B,nn) :- nounp(B,C), nounp1(A,D), prnoun(D,E), dt(E,F).
rmv(A,B,nn) :- nounp(B,C), plnoun(C,D), dt(A,E).
rmv(A,B,nn) :- nounp(B,C), pos(C,D), prnoun(B,E).
rmv(A,B,nn) :- nounp(B,C), prnoun(A,D), nounp1(D,E).
rmv(A,B,nn) :- nounp(B,C), vp(C,D), prnoun(B,E).
rmv(A,B,nn) :- nounp1(A,C), cc(C,D), nounp1(D,E), np(A,F).
rmv(A,B,nn) :- nounp1(A,C), cma(C,D), cnoun(D,E), prnoun(A,F).
rmv(A,B,nn) :- nounp1(A,C), end(C), snoun(B,D), sverb(D,E).
rmv(A,B,nn) :- nounp1(A,C), nounp1(C,D), cln(D,E).
rmv(A,B,nn) :- pp(A,C).
rmv(A,B,nn) :- ppz(B,C).
rmv(A,B,nn) :- prnoun(A,C), nounp1(C,D), in(D,E), nounp1(E,F).
rmv(A,B,nn) :- prnoun(B,C), cnoun(C,D).
rmv(A,B,nn) :- prnoun(B,C), nounp(C,D), nounp(D,E).
rmv(A,B,nn) :- prnoun(B,C), nounp(C,D), sverb(D,E), nounp(E,F).
rmv(A,B,nn) :- prnoun(B,C), prnoun(A,D).
rmv(A,B,nn) :- rp(B,C).
rmv(A,B,nn) :- to(A,C).
rmv(A,B,nn) :- vp1(A,C), nounp1(C,D), cma(D,E), snoun(E,F).
rmv(A,B,nn) :- vp1(A,C), nounp1(C,D), nounp1(D,E), dt(E,F).
rmv(A,B,nn) :- wdt(A,C).
```

Figure 26: Theory for nn (ctd).

```
rmv(A,B,nns) :- adjp(B,C), nounp(C,D), in(D,E).
rmv(A,B,nns) :- advp(A,C), adjp(B,D).
rmv(A,B,nns) :- advp(A,C), in(B,D).
rmv(A,B,nns) :- advp(A,C), nounp(B,D).
rmv(A,B,nns) :- advp(A,C), nounp1(C,D).
rmv(A,B,nns) :- cc(B,C), prnoun(C,D).
rmv(A,B,nns) :- cma(A,C), prnoun(B,D).
rmv(A,B,nns) :- cma(B,C), prnoun(A,D), nounp1(D,E).
rmv(A,B,nns) :- dt(B,C), adjp(C,D).
rmv(A,B,nns) :- dt(B,C), cma(A,D).
rmv(A,B,nns) :- dt(B,C), nounp(C,D), cma(D,E).
rmv(A,B,nns) :- dt(B,C), nounp(C,D), in(D,E).
rmv(A,B,nns) :- dt(B,C), nounp(C,D), to(D,E).
rmv(A,B,nns) :- dt(B,C), snoun(A,D).
rmv(A,B,nns) :- end(A), prnoun(B,C).
rmv(A,B,nns) :- in(A,C), prnoun(C,D), noun(D,E).
rmv(A,B,nns) :- in(A,C), snoun(C,D), prnoun(B,E).
rmv(A,B,nns) :- noun(A,C), dt(C,D), nounp(B,E).
rmv(A,B,nns) :- noun(A,C), nounp(B,D), to(D,E).
rmv(A,B,nns) :- noun(B,C), noun(C,D), in(A,E).
rmv(A,B,nns) :- noun(B,C), vp(C,D), nounp1(A,E).
rmv(A,B,nns) :- nounp(B,C), cln(C,D), not_end(A).
rmv(A,B,nns) :- nounp(B,C), noun(C,D), cma(D,E).
rmv(A,B,nns) :- nounp(B,C), nounp1(A,D), pos(D,E).
rmv(A,B,nns) :- nounp(B,C), prnoun(A,D).
rmv(A,B,nns) :- nounp(B,C), prnoun(C,D).
rmv(A,B,nns) :- nounp(B,C), snoun(A,D), end(D).
rmv(A,B,nns) :- nounp(B,C), snoun(C,D), cma(A,E).
rmv(A,B,nns) :- nounp(B,C), snoun(C,D), nounp1(A,E).
rmv(A,B,nns) :- nounp1(A,C), pos(C,D), np(A,E).
rmv(A,B,nns) :- np(B,C), vp(C,D).
rmv(A,B,nns) :- nps(A,C).
rmv(A,B,nns) :- pp(A,C).
rmv(A,B,nns) :- ppz(B,C).
rmv(A,B,nns) :- prnoun(A,C), cc(C,D), np(D,E).
rmv(A,B,nns) :- prnoun(A,C), cma(C,D).
rmv(A,B,nns) :- prnoun(A,C), in(C,D), np(D,E).
rmv(A,B,nns) :- prnoun(A,C), noun(C,D), cma(D,E).
rmv(A,B,nns) :- prnoun(A,C), noun(C,D), in(D,E).
rmv(A,B,nns) :- prnoun(A,C), nounp1(C,D), noun(D,E).
rmv(A,B,nns) :- prnoun(B,C), cma(C,D), dt(D,E).
rmv(A,B,nns) :- prnoun(B,C), cma(C,D), nounp(D,E).
rmv(A,B,nns) :- prnoun(B,C), nounp(C,D).
rmv(A,B,nns) :- rp(B,C).
rmv(A,B,nns) :- snoun(B,C), snoun(A,D).
rmv(A,B,nns) :- to(B,C), prnoun(A,D), noun(D,E).
rmv(A,B,nns) :- wdt(A,C).
rmv(A,B,nns) :- wp(A,C).
```

Figure 27: Theory for nns.

```
rmv(A,B,np) :- adjp(B,C), adjp1(A,D).
rmv(A,B,np) :- adjp(B,C), nounp(C,D), vp(D,E), vpart(E,F).
rmv(A,B,np) :- cds(A,C), cds(C,D), cnoun(B,E).
rmv(A,B,np) :- cds(A,C), in(C,D).
rmv(A,B,np) :- cnoun(B,C), in(C,D), dt(D,E), prnoun(E,F).
rmv(A,B,np) :- cnoun(B,C), in(C,D), prnoun(D,E), nounp(E,F).
rmv(A,B,np) :- cnoun(B,C), lpn(C,D).
rmv(A,B,np) :- cnoun(B,C), nounp(C,D), cnoun(D,E).
rmv(A,B,np) :- cnoun(B,C), nounp(C,D), vp(D,E), vpart(E,F).
rmv(A,B,np) :- cnoun(B,C), sverb(C,D), in(D,E), nounp(E,F).
rmv(A,B,np) :- dt(A,C), cnoun(B,D), nounp(D,E), sverb(E,F)
rmv(A,B,np) :- dt(A,C), in(C,D), snoun(D,E), nn(B,F).
rmv(A,B,np) :- end(A), adjp(B,C).
rmv(A,B,np) :- end(A), cnoun(B,C).
rmv(A,B,np) :- end(A), dt(B,C).
rmv(A,B,np) :- end(A), in(B,C), dt(C,D).
rmv(A,B,np) :- end(A), noun(B,C), in(C,D).
rmv(A,B,np) :- end(A), nounp(B,C), advp(C,D).
rmv(A,B,np) :- end(A), sverb(B,C), dt(C,D), cnoun(D,E).
rmv(A,B,np) :- end(A), vp(B,C), adjp(C,D).
rmv(A,B,np) :- end(A), vp(B,C), in(C,D), nounp(D,E).
rmv(A,B,np) :- end(A), vp(B,C), nounp(C,D), to(D,E).
rmv(A,B,np) :- in(A,C), cnoun(B,D), noun(D,E).
rmv(A,B,np) :- in(A,C), cnoun(B,D), stp(D,E).
rmv(A,B,np) :- in(A,C), cnoun(B,D), vp(D,E), nounp(E,F).
rmv(A,B,np) :- in(B,C), adjp(C,D), noun(D,E).
rmv(A,B,np) :- nn(B,C), in(C,D), cnoun(D,E).
rmv(A,B,np) :- nn(B,C), noun(C,D), in(D,E).
rmv(A,B,np) :- nns(B,C), in(C,D), snoun(D,E).
rmv(A,B,np) :- nounp(B,C), vpart(C,D), nn(B,E).
rmv(A,B,np) :- plnoun(B,C), advp(C,D), vp(D,E).
```

Figure 28: Theory for np.

```
rmv(A,B,nps) :- adjp(B,C), adjp1(A,D).
rmv(A,B,nps) :- adjp1(A,C), nounp1(C,D), end(D).
rmv(A,B,nps) :- cln(A,C), noun(C,D).
rmv(A,B,nps) :- cln(B,C), adjp1(A,D).
rmv(A,B,nps) :- cln(B,C), nounp(C,D), nns(D,E).
rmv(A,B,nps) :- cma(A,C), cnoun(B,D).
rmv(A,B,nps) :- cnoun(A,C), end(C).
rmv(A,B,nps) :- cnoun(B,C), cln(C,D).
rmv(A,B,nps) :- cnoun(B,C), in(C,D), dt(D,E), nounp(E,F).
rmv(A,B,nps) :- cnoun(B,C), md(C,D).
rmv(A,B,nps) :- cnoun(B,C), nounp(C,D), cma(D,E).
rmv(A,B,nps) :- cnoun(B,C), sverb(C,D), dt(D,E).
rmv(A,B,nps) :- cnoun(B,C), sverb(C,D), vp(C,E), dt(E,F).
rmv(A,B,nps) :- cnoun(B,C), vp(C,D), advp(D,E).
rmv(A,B,nps) :- dt(A,C), cnoun(B,D).
rmv(A,B,nps) :- end(A), advp(B,C), sverb(C,D).
rmv(A,B,nps) :- end(A), cc(B,C), cnoun(C,D).
rmv(A,B,nps) :- end(A), cln(B,C), nounp(C,D).
rmv(A,B,nps) :- end(A), cma(B,C).
rmv(A,B,nps) :- end(A), cnoun(B,C).
rmv(A,B,nps) :- end(A), in(B,C).
rmv(A,B,nps) :- end(A), sverb(B,C), dt(C,D).
rmv(A,B,nps) :- end(A), sverb(B,C), nounp(C,D).
rmv(A,B,nps) :- end(A), vp(B,C), noun(C,D).
rmv(A,B,nps) :- end(A), vpart(B,C).
rmv(A,B,nps) :- in(A,C), cnoun(B,D), stp(D,E).
rmv(A,B,nps) :- in(A,C), cnoun(C,D), cnoun(B,E).
rmv(A,B,nps) :- in(A,C), nns(B,D), vp(D,E).
rmv(A,B,nps) :- in(A,C), nns(C,D), nounp1(D,E).
rmv(A,B,nps) :- in(A,C), svp1(C,D), cnoun(B,E).
rmv(A,B,nps) :- nns(B,C), nounp(C,D).
rmv(A,B,nps) :- nns(B,C), vp(C,D), adjp(D,E).
rmv(A,B,nps) :- nns(B,C), vp(C,D), in(D,E).
rmv(A,B,nps) :- noun(B,C), vp(C,D), dt(A,E).
rmv(A,B,nps) :- noun(B,C), vp1(A,D), nounp1(D,E).
rmv(A,B,nps) :- nounp(B,C), adjp1(A,D).
rmv(A,B,nps) :- nounp(B,C), cnoun(C,D), in(A,E).
rmv(A,B,nps) :- nounp(B,C), in(A,D), sverb(D,E).
rmv(A,B,nps) :- nounp(B,C), in(A,D), svp1(D,E), nounp1(E,F).
rmv(A,B,nps) :- nounp(B,C), md(C,D), cnoun(B,E).
rmv(A,B,nps) :- nounp(B,C), vb(A,D).
rmv(A,B,nps) :- nounp(B,C), wdt(C,D), vp(D,E).
rmv(A,B,nps) :- nounp1(A,C), end(C), cnoun(A,D).
rmv(A,B,nps) :- plnoun(B,C), vp(C,D), in(D,E).
rmv(A,B,nps) :- plnoun(B,C), vp(C,D), vpart(D,E).
rmv(A,B,nps) :- vp1(A,C), cnoun(B,D).
rmv(A,B,nps) :- vpart(A,C), cnoun(B,D).
```

Figure 29: Theory for nps.

```
rmv(A,B,pdt) :- adjp(B,C).
rmv(A,B,pdt) :- cc(B,C).
rmv(A,B,pdt) :- cds(B,C).
rmv(A,B,pdt) :- cma(A,C).
rmv(A,B,pdt) :- cma(B,C).
rmv(A,B,pdt) :- in(A,C), nounp1(C,D), nounp1(D,E), vb(E,F).
rmv(A,B,pdt) :- in(B,C).
rmv(A,B,pdt) :- noun(B,C).
rmv(A,B,pdt) :- nounp(B,C), adjp(C,D), in(A,E).
rmv(A,B,pdt) :- nounp1(A,C).
rmv(A,B,pdt) :- pp(B,C).
rmv(A,B,pdt) :- stp(B,C).
rmv(A,B,pdt) :- vp(B,C).
rmv(A,B,pdt) :- vpart(B,C).
```

Figure 30: Theory for `pdt`.

```
rmv(A,B,pos) :- adjp(B,C), to(C,D).
rmv(A,B,pos) :- dt(A,C).
rmv(A,B,pos) :- dt(B,C).
rmv(A,B,pos) :- ex(A,C).
rmv(A,B,pos) :- pp(A,C).
rmv(A,B,pos) :- wp(A,C).
```

Figure 31: Theory for `pos`.

```
rmv(A,B,pp) :- adjp(B,C), in(A,D).
rmv(A,B,pp) :- cma(A,C).
rmv(A,B,pp) :- in(A,C), nounp1(C,D), dt(D,E), in(E,F).
rmv(A,B,pp) :- noun(B,C), cc(C,D).
rmv(A,B,pp) :- noun(B,C), in(C,D).
rmv(A,B,pp) :- noun(B,C), nounp(B,D), vp(D,E).
rmv(A,B,pp) :- noun(B,C), stp(C,D).
rmv(A,B,pp) :- noun(B,C), vp(C,D).
rmv(A,B,pp) :- nounp(B,C), cc(A,D).
rmv(A,B,pp) :- nounp(B,C), cma(C,D), nounp(D,E), sverb(E,F).
rmv(A,B,pp) :- nounp(B,C), in(A,D).
rmv(A,B,pp) :- nounp(B,C), in(C,D), dt(D,E), noun(E,F).
rmv(A,B,pp) :- nounp(B,C), pos(C,D).
rmv(A,B,pp) :- nounp(B,C), to(A,D).
rmv(A,B,pp) :- nounp(B,C), to(C,D).
rmv(A,B,pp) :- nounp(B,C), vp(C,D), vp(C,E), vpart(E,D).
rmv(A,B,pp) :- sverb(A,C), nounp(B,D), in(D,E), nounp(E,F).
rmv(A,B,pp) :- vpart(A,C), noun(B,D).
```

Figure 32: Theory for pp.

```
rmv(A,B,ppz) :- in(B,C).
rmv(A,B,ppz) :- to(B,C).
```

Figure 33: Theory for ppz.

```
rmv(A,B,rb) :- adjp(B,C), adjp1(A,D).
rmv(A,B,rb) :- adjp1(A,C), advp(C,D), vb(D,E).
rmv(A,B,rb) :- adjp1(A,C), cma(C,D).
rmv(A,B,rb) :- adjp1(A,C), in(C,D).
rmv(A,B,rb) :- adjp1(A,C), nounp1(C,D), in(D,E).
rmv(A,B,rb) :- advp(A,C), advp(C,D), dt(B,E).
rmv(A,B,rb) :- advp(A,C), advp(C,D), nounp(B,E).
rmv(A,B,rb) :- advp(A,C), cma(C,D), noun(D,E), nounp(B,F).
rmv(A,B,rb) :- advp(A,C), in(C,D).
rmv(A,B,rb) :- advp(A,C), nounp(B,D), vp(D,E).
rmv(A,B,rb) :- advp(A,C), nounp1(C,D), dt(B,E).
rmv(A,B,rb) :- dt(A,C), end(C).
rmv(A,B,rb) :- dt(A,C), in(B,D).
rmv(A,B,rb) :- dt(A,C), in(C,D).
rmv(A,B,rb) :- dt(A,C), sverb(C,D), noun(D,E).
rmv(A,B,rb) :- dt(A,C), to(B,D).
rmv(A,B,rb) :- dt(B,C), adjp(C,D), noun(D,E), cma(E,F).
rmv(A,B,rb) :- dt(B,C), adjp(C,D), noun(D,E), nounp(E,F).
rmv(A,B,rb) :- dt(B,C), adjp(C,D), nounp1(A,E), in(E,F).
rmv(A,B,rb) :- dt(B,C), adjp1(A,D).
rmv(A,B,rb) :- dt(B,C), noun(C,D), nounp(C,E), cc(E,F).
rmv(A,B,rb) :- dt(B,C), noun(C,D), nounp(C,E), stp(E,F).
rmv(A,B,rb) :- dt(B,C), nounp(C,D), nounp(D,E), nounp1(A,F).
rmv(A,B,rb) :- dt(B,C), nounp(C,D), nounp1(A,E), vp1(E,F).
rmv(A,B,rb) :- dt(B,C), nounp(C,D), to(D,E).
rmv(A,B,rb) :- dt(B,C), nounp(C,D), vp(D,E), cma(A,F).
rmv(A,B,rb) :- dt(B,C), nounp(C,D), vp(D,E), dt(E,F).
rmv(A,B,rb) :- dt(B,C), nounp1(A,D), in(D,E), noun(E,F).
rmv(A,B,rb) :- dt(B,C), nounp1(A,D), sverb(D,E).
rmv(A,B,rb) :- end(A), dt(B,C), nounp(C,D), cma(D,E).
rmv(A,B,rb) :- in(B,C), to(A,D).
rmv(A,B,rb) :- noun(A,C), ppz(B,D).
rmv(A,B,rb) :- noun(B,C).
rmv(A,B,rb) :- nounp(B,C), adjp1(A,D).
rmv(A,B,rb) :- nounp(B,C), advp(C,D), vp(D,E).
rmv(A,B,rb) :- nounp(B,C), cc(A,D), end(D).
rmv(A,B,rb) :- nounp(B,C), cc(C,D), nounp(D,E).
rmv(A,B,rb) :- nounp(B,C), cc(C,D), sverb(D,E).
rmv(A,B,rb) :- nounp(B,C), cds(C,D), cma(D,E), nounp(E,F).
rmv(A,B,rb) :- nounp(B,C), cma(C,D), nounp(D,E), cc(E,F).
rmv(A,B,rb) :- nounp(B,C), cma(C,D), nounp(D,E), dt(B,F).
rmv(A,B,rb) :- nounp(B,C), dt(A,D).
rmv(A,B,rb) :- nounp(B,C), dt(B,D), adjp(D,E), nounp1(A,F).
rmv(A,B,rb) :- nounp(B,C), dt(C,D), nounp(D,E), vp(E,F).
```

Figure 34: Theory for `rb`.

```
rmv(A,B,rb) :- nounp(B,C), in(C,D), in(D,E).
rmv(A,B,rb) :- nounp(B,C), in(C,D), noun(D,E), nounp1(A,F).
rmv(A,B,rb) :- nounp(B,C), in(C,D), vp1(A,E), ex(E,F).
rmv(A,B,rb) :- nounp(B,C), noun(C,D), cma(D,E), nounp(E,F).
rmv(A,B,rb) :- nounp(B,C), noun(C,D), nounp(D,E), nounp(E,F).
rmv(A,B,rb) :- nounp(B,C), nounp(C,D), md(D,E).
rmv(A,B,rb) :- nounp(B,C), nounp(C,D), nounp1(A,E).
rmv(A,B,rb) :- nounp(B,C), nounp(C,D), ppz(B,E).
rmv(A,B,rb) :- nounp(B,C), nounp1(A,D), adjp1(D,E).
rmv(A,B,rb) :- nounp(B,C), nounp1(A,D), cc(D,E).
rmv(A,B,rb) :- nounp(B,C), nounp1(A,D), in(D,E).
rmv(A,B,rb) :- nounp(B,C), nounp1(A,D), noun(D,E).
rmv(A,B,rb) :- nounp(B,C), nounp1(A,D), pos(D,E).
rmv(A,B,rb) :- nounp(B,C), nounp1(A,D), sverb(D,E).
rmv(A,B,rb) :- nounp(B,C), nounp1(A,D), vb(D,E), to(E,F).
rmv(A,B,rb) :- nounp(B,C), nounp1(A,D), vpart(D,E).
rmv(A,B,rb) :- nounp(B,C), sverb(C,D), adjp(D,E).
rmv(A,B,rb) :- nounp(B,C), sverb(C,D), nounp(D,E), stp(E,F).
rmv(A,B,rb) :- nounp(B,C), sverb(C,D), to(D,E).
rmv(A,B,rb) :- nounp(B,C), to(C,D), noun(D,E).
rmv(A,B,rb) :- nounp(B,C), to(C,D), vb(D,E), vp1(A,F).
rmv(A,B,rb) :- nounp(B,C), vp(C,D), adjp(D,E), nounp(E,F).
rmv(A,B,rb) :- nounp(B,C), vp(C,D), dt(D,E), noun(E,F).
rmv(A,B,rb) :- nounp(B,C), vp(C,D), nounp(D,E), md(C,F).
rmv(A,B,rb) :- nounp(B,C), vp(C,D), nounp(D,E), to(E,F).
rmv(A,B,rb) :- nounp(B,C), vp(C,D), nounp1(A,E).
rmv(A,B,rb) :- nounp(B,C), vp(C,D), pp(D,E).
rmv(A,B,rb) :- nounp(B,C), vp(C,D), vpart(D,E), nounp(E,F).
rmv(A,B,rb) :- nounp(B,C), wp(C,D).
rmv(A,B,rb) :- nounp1(A,C), adjp1(C,D), nounp1(D,E).
rmv(A,B,rb) :- nounp1(A,C), nounp1(C,D), cds(D,E), cds(E,F).
rmv(A,B,rb) :- nounp1(A,C), pos(C,D), nounp1(D,E), in(E,F).
rmv(A,B,rb) :- pos(A,C).
rmv(A,B,rb) :- pp(B,C).
rmv(A,B,rb) :- ppz(A,C).
rmv(A,B,rb) :- sverb(A,C), ex(C,D).
rmv(A,B,rb) :- sverb(B,C), cma(A,D), nounp1(D,E).
rmv(A,B,rb) :- to(B,C), vb(C,D), dt(D,E), vp1(A,F).
rmv(A,B,rb) :- vp(B,C), cma(A,D), nounp1(D,E), in(E,F).
rmv(A,B,rb) :- vp(B,C), in(A,D).
rmv(A,B,rb) :- vp(B,C), nounp(C,D), in(D,E), cma(A,F).
rmv(A,B,rb) :- vpart(A,C), nounp(B,D), cc(D,E).
rmv(A,B,rb) :- vpart(B,C), nounp(C,D), nounp1(A,E).
```

Figure 35: Theory for rb (ctd.).

```
rmv(A,B,rbr) :- advp(A,C), in(B,D), cds(D,E).
rmv(A,B,rbr) :- advp(A,C), in(C,D), vp1(D,E).
rmv(A,B,rbr) :- cc(A,C), in(B,D), dt(D,E), nounp(E,F).
rmv(A,B,rbr) :- cc(A,C), in(B,D), noun(D,E).
rmv(A,B,rbr) :- cc(A,C), in(B,D), nounp(D,E), in(E,F).
rmv(A,B,rbr) :- cc(A,C), in(B,D), nounp(D,E), nounp(E,F).
rmv(A,B,rbr) :- cc(A,C), nounp1(C,D), cncy(D,E).
rmv(A,B,rbr) :- cc(A,C), nounp1(C,D), in(B,E).
rmv(A,B,rbr) :- cc(A,C), nounp1(C,D), in(D,E), noun(E,F).
rmv(A,B,rbr) :- cc(A,C), nounp1(C,D), nounp1(D,E).
rmv(A,B,rbr) :- cc(A,C), nounp1(C,D), vpart(D,E).
rmv(A,B,rbr) :- cc(A,C), stp(B,D).
rmv(A,B,rbr) :- cds(A,C), nounp1(A,D), in(D,E).
rmv(A,B,rbr) :- cds(A,C), vp1(C,D), nounp1(D,E).
rmv(A,B,rbr) :- cma(A,C), noun(C,D), in(D,E).
rmv(A,B,rbr) :- cma(B,C), nounp(C,D), vp(D,E), vp1(A,F).
rmv(A,B,rbr) :- cma(B,C), vp1(A,D), wp(D,E).
rmv(A,B,rbr) :- in(A,C), advp(C,D).
rmv(A,B,rbr) :- in(A,C), noun(C,D), dt(D,E), vb(E,F).
rmv(A,B,rbr) :- in(B,C), cncy(C,D), vb(A,E), to(E,F).
rmv(A,B,rbr) :- in(B,C), noun(C,D), nounp1(A,E), nounp1(E,F).
rmv(A,B,rbr) :- in(B,C), nounp1(A,D), cma(D,E).
rmv(A,B,rbr) :- noun(B,C).
rmv(A,B,rbr) :- nounp(B,C), cc(A,D), nounp1(D,E).
rmv(A,B,rbr) :- nounp(B,C), cc(C,D), in(A,E).
rmv(A,B,rbr) :- nounp(B,C), cma(A,D), nounp1(D,E), in(E,F).
rmv(A,B,rbr) :- nounp(B,C), cma(C,D), nounp(D,E), in(A,F).
rmv(A,B,rbr) :- nounp(B,C), in(C,D), dt(D,E), vb(A,F).
rmv(A,B,rbr) :- nounp(B,C), in(C,D), nounp(D,E), ppz(D,F).
rmv(A,B,rbr) :- nounp(B,C), in(C,D), to(A,E).
rmv(A,B,rbr) :- nounp(B,C), md(C,D).
rmv(A,B,rbr) :- nounp(B,C), noun(C,D), sverb(D,E).
rmv(A,B,rbr) :- nounp(B,C), nounp(C,D), cc(D,E).
rmv(A,B,rbr) :- nounp(B,C), nounp(C,D), to(A,E).
rmv(A,B,rbr) :- nounp(B,C), nounp(C,D), vp1(A,E), nounp1(E,F).
rmv(A,B,rbr) :- nounp(B,C), stp(C,D), in(A,E), noun(E,F).
rmv(A,B,rbr) :- nounp(B,C), stp(C,D), to(A,E).
rmv(A,B,rbr) :- nounp(B,C), to(A,D), svp1(D,E).
rmv(A,B,rbr) :- nounp(B,C), to(C,D), vb(A,E).
rmv(A,B,rbr) :- nounp(B,C), vp(C,D), in(A,E).
rmv(A,B,rbr) :- nounp(B,C), vp(C,D), vpart(D,E), in(E,F).
rmv(A,B,rbr) :- nounp(B,C), vp1(A,D), nounp1(D,E), cc(E,F).
rmv(A,B,rbr) :- nounp(B,C), vp1(A,D), nounp1(D,E), sverb(E,F).
rmv(A,B,rbr) :- sverb(A,C), in(B,D), cncy(D,E).
rmv(A,B,rbr) :- vpart(A,C), cma(C,D), nounp(B,E).
```

Figure 36: Theory for `rbr`.

```
rmv(A,B,rbs) :- cc(A,C), end(C).
rmv(A,B,rbs) :- dt(A,C), in(B,D).
rmv(A,B,rbs) :- end(A), nounp(B,C), vp(C,D), nounp(D,E).
rmv(A,B,rbs) :- in(A,C), advp(C,D).
rmv(A,B,rbs) :- in(A,C), nounp1(C,D), in(D,E), nounp1(E,F).
rmv(A,B,rbs) :- in(A,C), nounp1(C,D), nounp1(D,E).
rmv(A,B,rbs) :- in(A,C), nounp1(C,D), vp1(D,E).
rmv(A,B,rbs) :- in(B,C), cma(A,D), nounp1(D,E), nounp1(E,F).
rmv(A,B,rbs) :- in(B,C), dt(C,D), in(A,E), vp1(E,F).
rmv(A,B,rbs) :- in(B,C), nounp(C,D), nounp(D,E).
rmv(A,B,rbs) :- in(B,C), nounp(C,D), pos(D,E).
rmv(A,B,rbs) :- in(B,C), nounp(C,D), ppz(C,E).
rmv(A,B,rbs) :- in(B,C), vb(A,D).
rmv(A,B,rbs) :- noun(B,C).
rmv(A,B,rbs) :- nounp(B,C), in(A,D), end(D).
rmv(A,B,rbs) :- nounp(B,C), in(A,D), nounp1(D,E), adjp1(E,F).
rmv(A,B,rbs) :- nounp(B,C), in(A,D), nounp1(D,E), dt(E,F).
rmv(A,B,rbs) :- nounp(B,C), in(A,D), vp1(D,E).
rmv(A,B,rbs) :- nounp(B,C), in(C,D), in(A,E).
rmv(A,B,rbs) :- nounp(B,C), noun(C,D), vp(D,E), in(E,F).
rmv(A,B,rbs) :- nounp(B,C), vp(C,D), cma(A,E), nounp1(E,F).
rmv(A,B,rbs) :- nounp(B,C), vp(C,D), in(A,E).
rmv(A,B,rbs) :- nounp(B,C), vp(C,D), nounp(D,E), cma(A,F).
rmv(A,B,rbs) :- nounp(B,C), vp(C,D), vpart(D,E), in(E,F).
rmv(A,B,rbs) :- vpart(A,C), in(B,D).
```

Figure 37: Theory for rbs.

```
rmv(A,B,rp) :- adjp(B,C), nounp1(A,D).
rmv(A,B,rp) :- advp(A,C).
rmv(A,B,rp) :- advp(B,C), nounp(C,D), sverb(A,E), nounp1(E,F).
rmv(A,B,rp) :- cc(A,C).
rmv(A,B,rp) :- cds(A,C).
rmv(A,B,rp) :- cds(B,C).
rmv(A,B,rp) :- cma(A,C).
rmv(A,B,rp) :- cncy(B,C), cds(C,D), cma(D,E).
rmv(A,B,rp) :- cncy(B,C), cds(C,D), stp(D,E).
rmv(A,B,rp) :- cncy(B,C), nounp(C,D), dt(D,E), nounp(E,F).
rmv(A,B,rp) :- dt(A,C), in(C,D).
rmv(A,B,rp) :- dt(B,C), adjp(C,D), cds(D,E).
rmv(A,B,rp) :- dt(B,C), nounp1(A,D).
rmv(A,B,rp) :- in(A,C).
rmv(A,B,rp) :- in(B,C), adjp(C,D), noun(D,E), nounp1(A,F).
rmv(A,B,rp) :- in(B,C), cds(C,D), nounp(D,E).
rmv(A,B,rp) :- in(B,C), dt(C,D), nounp1(A,E), vb(E,F).
rmv(A,B,rp) :- in(B,C), nounp(C,D), cma(D,E), vp(E,F).
rmv(A,B,rp) :- in(B,C), nounp(C,D), nounp(D,E), dt(E,F).
rmv(A,B,rp) :- in(B,C), nounp(C,D), to(D,E), vp1(A,F).
rmv(A,B,rp) :- in(B,C), nounp(C,D), vp(D,E), nounp1(A,F).
rmv(A,B,rp) :- in(B,C), nounp1(A,D), vb(D,E), to(E,F).
rmv(A,B,rp) :- noun(A,C), in(B,D), nounp(D,E), nounp(E,F).
rmv(A,B,rp) :- nounp(B,C), cds(C,D), nounp(C,E), stp(E,F).
rmv(A,B,rp) :- nounp(B,C), cds(C,D), stp(D,E).
rmv(A,B,rp) :- nounp(B,C), cln(C,D).
rmv(A,B,rp) :- nounp(B,C), cma(C,D), in(D,E), nounp(E,F).
rmv(A,B,rp) :- nounp(B,C), dt(A,D).
rmv(A,B,rp) :- nounp(B,C), in(C,D), cncy(D,E).
rmv(A,B,rp) :- nounp(B,C), in(C,D), nounp(D,E), cds(E,F).
rmv(A,B,rp) :- nounp(B,C), nounp(C,D), in(D,E), cncy(E,F).
rmv(A,B,rp) :- nounp(B,C), nounp1(A,D).
rmv(A,B,rp) :- nounp(B,C), to(C,D), cncy(D,E).
rmv(A,B,rp) :- nounp1(A,C), adjp1(C,D).
rmv(A,B,rp) :- nounp1(A,C), cc(C,D).
rmv(A,B,rp) :- nounp1(A,C), in(C,D).
rmv(A,B,rp) :- nounp1(A,C), nounp1(C,D).
rmv(A,B,rp) :- nounp1(A,C), pos(C,D).
rmv(A,B,rp) :- nounp1(A,C), vb(C,D), to(D,E), vpart(E,F).
rmv(A,B,rp) :- nounp1(A,C), vp1(C,D), wdt(D,E).
rmv(A,B,rp) :- to(A,C).
rmv(A,B,rp) :- to(B,C), cds(C,D).
rmv(A,B,rp) :- to(B,C), cncy(C,D).
rmv(A,B,rp) :- to(B,C), nounp(C,D), cds(D,E).
rmv(A,B,rp) :- to(B,C), nounp(C,D), nounp(D,E), nounp(E,F).
rmv(A,B,rp) :- to(B,C), nounp(C,D), nounp(D,E), vb(A,F).
rmv(A,B,rp) :- vp1(A,C), cma(C,D), nounp1(D,E), cds(E,F).
rmv(A,B,rp) :- vpart(A,C), in(B,D), ppz(D,E).
```

Figure 38: Theory for `rp`.

```
rmv(A,B,rqt).
```

Figure 39: Theory for `rqt`.

```
rmv(A,B,uh) :- adjp(B,C).
rmv(A,B,uh) :- noun(A,C).
rmv(A,B,uh) :- nounp(B,C).
```

Figure 40: Theory for `uh`.

```
rmv(A,B,vb) :- adjp(B,C), noun(A,D).
rmv(A,B,vb) :- adjp1(A,C).
rmv(A,B,vb) :- advp(A,C), nounp1(C,D), nounp1(D,E).
rmv(A,B,vb) :- advp(A,C), pp(C,D).
rmv(A,B,vb) :- cc(B,C), nounp(C,D).
rmv(A,B,vb) :- cc(B,C), vp(C,D).
rmv(A,B,vb) :- cma(A,C), nounp1(C,D), nounp1(D,E).
rmv(A,B,vb) :- cma(B,C), noun(C,D).
rmv(A,B,vb) :- cma(B,C), nounp1(A,D).
rmv(A,B,vb) :- dt(A,C).
rmv(A,B,vb) :- in(A,C).
rmv(A,B,vb) :- in(B,C), noun(C,D), nounp(D,E), vp(E,F).
rmv(A,B,vb) :- in(B,C), nounp1(A,D).
rmv(A,B,vb) :- in(B,C), vbd(A,D).
rmv(A,B,vb) :- noun(B,C), cma(A,D), nounp1(D,E).
rmv(A,B,vb) :- noun(B,C), nounp(B,D), vbd(D,E).
rmv(A,B,vb) :- noun(B,C), vp(C,D).
rmv(A,B,vb) :- nounp(B,C), cma(C,D), nounp1(A,E).
rmv(A,B,vb) :- nounp1(A,C), adjp1(C,D).
rmv(A,B,vb) :- nounp1(A,C), cc(C,D).
rmv(A,B,vb) :- nounp1(A,C), cma(C,D).
rmv(A,B,vb) :- nounp1(A,C), end(C).
rmv(A,B,vb) :- nounp1(A,C), in(C,D).
rmv(A,B,vb) :- nounp1(A,C), nounp1(C,D).
rmv(A,B,vb) :- nounp1(A,C), pos(C,D).
rmv(A,B,vb) :- nounp1(A,C), wrb(C,D).
rmv(A,B,vb) :- pos(A,C).
rmv(A,B,vb) :- ppz(A,C).
rmv(A,B,vb) :- stp(B,C), end(C), nounp1(A,D).
rmv(A,B,vb) :- to(B,C), nounp1(A,D).
rmv(A,B,vb) :- vp(B,C).
rmv(A,B,vb) :- vpart(A,C).
rmv(A,B,vb) :- vpart(B,C), nounp1(A,D).
rmv(A,B,vb) :- wdt(A,C).
rmv(A,B,vb) :- wp(A,C).
```

Figure 41: Theory for vb.

```
rmv(A,B,vbd) :- adjp1(A,C), dt(C,D).
rmv(A,B,vbd) :- advp(A,C), dt(C,D).
rmv(A,B,vbd) :- advp(A,C), in(C,D).
rmv(A,B,vbd) :- advp(A,C), vb(C,D).
rmv(A,B,vbd) :- advp(A,C), vp1(C,D), stp(B,E).
rmv(A,B,vbd) :- cma(A,C), nounp1(C,D), end(D).
rmv(A,B,vbd) :- dt(A,C), in(C,D).
rmv(A,B,vbd) :- dt(A,C), vp1(C,D).
rmv(A,B,vbd) :- in(A,C).
rmv(A,B,vbd) :- in(B,C), dt(C,D), noun(D,E), cc(E,F).
rmv(A,B,vbd) :- in(B,C), noun(C,D), nounp(D,E), cc(E,F).
rmv(A,B,vbd) :- in(B,C), nounp(C,D), cc(D,E), noun(E,F).
rmv(A,B,vbd) :- in(B,C), nounp(C,D), cma(D,E), cc(E,F).
rmv(A,B,vbd) :- md(A,C).
rmv(A,B,vbd) :- noun(A,C), nounp1(C,D), svp1(D,E), in(B,F).
rmv(A,B,vbd) :- nounp(B,C), dt(A,D).
rmv(A,B,vbd) :- to(A,C).
rmv(A,B,vbd) :- vb(A,C).
rmv(A,B,vbd) :- vp1(A,C).
rmv(A,B,vbd) :- vpart(A,C).
```

Figure 42: Theory for `vbd`.

```
rmv(A,B,vbg) :- adjp1(A,C), adjp1(C,D), dt(D,E), in(E,F).
rmv(A,B,vbg) :- adjp1(A,C), adjp1(C,D), in(D,E).
rmv(A,B,vbg) :- adjp1(A,C), cma(C,D), nounp1(D,E).
rmv(A,B,vbg) :- adjp1(A,C), dt(C,D), in(D,E), vp1(E,F).
rmv(A,B,vbg) :- adjp1(A,C), dt(C,D), vp1(D,E), noun(E,F).
rmv(A,B,vbg) :- adjp1(A,C), in(C,D), end(D).
rmv(A,B,vbg) :- adjp1(A,C), noun(C,D).
rmv(A,B,vbg) :- adjp1(A,C), nounp1(C,D), in(D,E).
rmv(A,B,vbg) :- adjp1(A,C), nounp1(C,D), nounp1(D,E).
rmv(A,B,vbg) :- adjp1(A,C), vb(C,D).
rmv(A,B,vbg) :- cc(B,C), adjp1(A,D).
rmv(A,B,vbg) :- cc(B,C), dt(A,D).
rmv(A,B,vbg) :- cc(B,C), nounp(C,D), cma(A,E).
rmv(A,B,vbg) :- cc(B,C), nounp(C,D), nounp(D,E).
rmv(A,B,vbg) :- cc(B,C), nounp(C,D), stp(D,E).
rmv(A,B,vbg) :- cc(B,C), nounp1(A,D), in(D,E).
rmv(A,B,vbg) :- cma(B,C), adjp1(A,D).
rmv(A,B,vbg) :- cma(B,C), noun(C,D), nounp(C,E), vbd(E,F).
rmv(A,B,vbg) :- cma(B,C), noun(C,D), vp(D,E).
rmv(A,B,vbg) :- cma(B,C), nounp(C,D), cc(D,E).
rmv(A,B,vbg) :- cma(B,C), nounp(C,D), cma(D,E), nounp(E,F).
rmv(A,B,vbg) :- cma(B,C), nounp1(A,D), in(D,E).
rmv(A,B,vbg) :- cma(B,C), nounp1(A,D), nounp1(D,E).
rmv(A,B,vbg) :- cma(B,C), sverb(C,D).
rmv(A,B,vbg) :- cma(B,C), wdt(C,D).
rmv(A,B,vbg) :- dt(A,C), cma(B,D).
rmv(A,B,vbg) :- dt(A,C), in(B,D).
rmv(A,B,vbg) :- dt(A,C), stp(B,D).
rmv(A,B,vbg) :- in(B,C), adjp1(A,D).
rmv(A,B,vbg) :- in(B,C), nounp(C,D), cds(D,E), nounp(E,F).
rmv(A,B,vbg) :- in(B,C), pos(A,D).
rmv(A,B,vbg) :- noun(A,C), end(C).
rmv(A,B,vbg) :- nounp(B,C), cma(C,D), noun(D,E), vbd(E,F).
rmv(A,B,vbg) :- stp(B,C), adjp1(A,D).
rmv(A,B,vbg) :- stp(B,C), nounp1(A,D), in(D,E).
rmv(A,B,vbg) :- sverb(B,C), advp(C,D).
rmv(A,B,vbg) :- sverb(B,C), dt(C,D), nounp(D,E).
rmv(A,B,vbg) :- sverb(B,C), in(A,D).
rmv(A,B,vbg) :- vbz(B,C).
rmv(A,B,vbg) :- vp(B,C), adjp1(A,D).
rmv(A,B,vbg) :- vp(B,C), nounp(C,D).
rmv(A,B,vbg) :- vp(B,C), nounp1(A,D).
rmv(A,B,vbg) :- vp(B,C), vpart(C,D).
rmv(A,B,vbg) :- wdt(B,C), vp(C,D).
```

Figure 43: Theory for vbg.

```
rmv(A,B,vbn) :- adjp(B,C), nounp1(A,D).
rmv(A,B,vbn) :- advp(A,C), noun(C,D), nounp(B,E).
rmv(A,B,vbn) :- advp(A,C), nounp1(C,D), dt(B,E).
rmv(A,B,vbn) :- advp(A,C), nounp1(C,D), end(D).
rmv(A,B,vbn) :- advp(A,C), nounp1(C,D), nounp1(D,E), nounp(B,F).
rmv(A,B,vbn) :- advp(A,C), pp(C,D).
rmv(A,B,vbn) :- advp(B,C), nounp1(A,D), cma(D,E).
rmv(A,B,vbn) :- cc(A,C), pp(B,D).
rmv(A,B,vbn) :- cds(B,C), cma(A,D).
rmv(A,B,vbn) :- cds(B,C), nounp1(A,D).
rmv(A,B,vbn) :- cncy(B,C), nounp1(A,D).
rmv(A,B,vbn) :- dt(B,C), cma(A,D), nounp1(D,E), nounp1(E,F).
rmv(A,B,vbn) :- dt(B,C), nounp(C,D), cma(A,E).
rmv(A,B,vbn) :- dt(B,C), nounp1(A,D).
rmv(A,B,vbn) :- dt(B,C), to(A,D).
rmv(A,B,vbn) :- in(B,C), in(C,D), nounp1(A,E).
rmv(A,B,vbn) :- md(A,C).
rmv(A,B,vbn) :- noun(A,C), cma(C,D).
rmv(A,B,vbn) :- noun(A,C), in(C,D), nounp(B,E).
rmv(A,B,vbn) :- noun(A,C), noun(C,D), nounp1(C,E), end(E).
rmv(A,B,vbn) :- noun(A,C), noun(C,D), stp(B,E).
```

Figure 44: Theory for `vbn`.

```
rmv(A,B,vbn) :- nounp(B,C), cc(C,D), nounp1(A,E).
rmv(A,B,vbn) :- nounp(B,C), in(C,D), cma(A,E), nounp1(E,F).
rmv(A,B,vbn) :- nounp(B,C), in(C,D), nounp(D,E), cma(A,F).
rmv(A,B,vbn) :- nounp(B,C), in(C,D), nounp1(A,E).
rmv(A,B,vbn) :- nounp(B,C), noun(C,D), nounp1(A,E).
rmv(A,B,vbn) :- nounp(B,C), nounp(B,D), nounp(D,C), nounp1(A,E).
rmv(A,B,vbn) :- nounp(B,C), nounp(C,D), in(D,E), nounp1(A,F).
rmv(A,B,vbn) :- nounp(B,C), nounp(C,D), to(D,E), nounp1(A,F).
rmv(A,B,vbn) :- nounp(B,C), nounp1(A,D), cma(D,E).
rmv(A,B,vbn) :- nounp(B,C), nounp1(A,D), end(D).
rmv(A,B,vbn) :- nounp(B,C), nounp1(A,D), noun(D,E).
rmv(A,B,vbn) :- nounp(B,C), nounp1(A,D), nounp1(D,E), nounp1(E,F).
rmv(A,B,vbn) :- nounp(B,C), nounp1(A,D), sverb(D,E).
rmv(A,B,vbn) :- nounp(B,C), pos(C,D), nounp1(A,E).
rmv(A,B,vbn) :- nounp(B,C), to(C,D), cds(D,E).
rmv(A,B,vbn) :- nounp(B,C), to(C,D), cma(A,E).
rmv(A,B,vbn) :- nounp(B,C), to(C,D), nounp1(A,E).
rmv(A,B,vbn) :- nounp1(A,C), adjp1(C,D), cma(D,E).
rmv(A,B,vbn) :- nounp1(A,C), cc(C,D), nounp1(D,E), end(E).
rmv(A,B,vbn) :- nounp1(A,C), cma(C,D), noun(D,E), adjp1(E,F).
rmv(A,B,vbn) :- nounp1(A,C), cma(C,D), nounp1(D,E), dt(E,F).
rmv(A,B,vbn) :- nounp1(A,C), cma(C,D), nounp1(D,E), to(E,F).
rmv(A,B,vbn) :- nounp1(A,C), nounp1(A,D), noun(D,C), cc(C,E).
rmv(A,B,vbn) :- nounp1(A,C), nounp1(A,D), noun(D,C), pos(C,E).
rmv(A,B,vbn) :- nounp1(A,C), nounp1(C,D), cma(D,E).
rmv(A,B,vbn) :- nounp1(A,C), nounp1(C,D), noun(D,E), nounp1(E,F).
rmv(A,B,vbn) :- nounp1(A,C), nounp1(C,D), nounp1(D,E), end(E).
rmv(A,B,vbn) :- nounp1(A,C), pos(C,D), nounp1(D,E), dt(E,F).
rmv(A,B,vbn) :- nounp1(A,C), vbd(C,D), nounp1(D,E), nounp1(E,F).
rmv(A,B,vbn) :- nounp1(A,C), wrb(C,D).
rmv(A,B,vbn) :- pp(A,C).
rmv(A,B,vbn) :- pp(B,C), nounp1(A,D).
rmv(A,B,vbn) :- ppz(B,C), nounp1(A,D).
rmv(A,B,vbn) :- to(B,C), vb(C,D), nounp1(A,E), end(E).
rmv(A,B,vbn) :- wdt(A,C).
rmv(A,B,vbn) :- wp(A,C).
```

Figure 45: Theory for `vbn` (ctd) .

```
rmv(A,B,vbp) :- adjp1(A,C), dt(C,D).
rmv(A,B,vbp) :- advp(A,C), md(C,D).
rmv(A,B,vbp) :- cc(A,C), nounp1(C,D), vb(D,E).
rmv(A,B,vbp) :- cc(B,C), nounp(C,D).
rmv(A,B,vbp) :- cc(B,C), vb(C,D).
rmv(A,B,vbp) :- cma(B,C), adjp1(A,D).
rmv(A,B,vbp) :- dt(A,C), in(B,D).
rmv(A,B,vbp) :- dt(A,C), in(C,D), nounp1(D,E).
rmv(A,B,vbp) :- dt(A,C), stp(B,D).
rmv(A,B,vbp) :- dt(A,C), vb(C,D).
rmv(A,B,vbp) :- dt(A,C), vp1(C,D).
rmv(A,B,vbp) :- in(A,C), nounp1(C,D), dt(D,E).
rmv(A,B,vbp) :- in(B,C), adjp1(A,D).
rmv(A,B,vbp) :- in(B,C), in(A,D).
rmv(A,B,vbp) :- md(A,C).
rmv(A,B,vbp) :- nounp(B,C), in(C,D), nounp(B,E), cds(E,C).
rmv(A,B,vbp) :- nounp1(A,C), cncy(C,D).
rmv(A,B,vbp) :- pos(A,C).
rmv(A,B,vbp) :- ppz(A,C).
rmv(A,B,vbp) :- ppz(B,C), nounp(C,D), nounp(D,E).
rmv(A,B,vbp) :- stp(B,C), adjp1(A,D).
rmv(A,B,vbp) :- to(A,C).
rmv(A,B,vbp) :- vb(A,C).
rmv(A,B,vbp) :- vp(B,C), dt(A,D).
rmv(A,B,vbp) :- vp1(A,C).
```

Figure 46: Theory for `vbp`.

```
rmv(A,B,vbz) :- adjp(B,C), noun(A,D), cma(D,E).
rmv(A,B,vbz) :- adjp(B,C), noun(A,D), in(D,E).
rmv(A,B,vbz) :- adjp(B,C), noun(A,D), nounp1(A,E), vbd(E,F).
rmv(A,B,vbz) :- adjp(B,C), nounp(C,D), sverb(D,E), noun(A,F).
rmv(A,B,vbz) :- adjp1(A,C).
rmv(A,B,vbz) :- cc(B,C), nounp(C,D).
rmv(A,B,vbz) :- cds(A,C), cma(B,D).
rmv(A,B,vbz) :- cds(A,C), in(C,D).
rmv(A,B,vbz) :- cds(A,C), vp1(C,D).
rmv(A,B,vbz) :- cma(B,C), vpart(C,D).
rmv(A,B,vbz) :- dt(A,C), in(C,D), in(B,E).
rmv(A,B,vbz) :- dt(A,C), stp(B,D).
rmv(A,B,vbz) :- in(A,C), nounp1(C,D), in(D,E).
rmv(A,B,vbz) :- in(B,C), in(A,D).
rmv(A,B,vbz) :- in(B,C), vp1(A,D).
rmv(A,B,vbz) :- noun(A,C), in(C,D), adjp1(D,E).
rmv(A,B,vbz) :- noun(A,C), in(C,D), nounp(B,E), nounp(E,F).
rmv(A,B,vbz) :- noun(A,C), in(C,D), nounp(B,E), stp(E,F).
rmv(A,B,vbz) :- noun(A,C), in(C,D), nounp1(D,E), cds(E,F).
rmv(A,B,vbz) :- noun(A,C), in(C,D), nounp1(D,E), vb(E,F).
rmv(A,B,vbz) :- noun(A,C), in(C,D), nounp1(D,E), vp1(E,F).
rmv(A,B,vbz) :- noun(A,C), in(C,D), nounp1(D,E), vpart(E,F).
rmv(A,B,vbz) :- noun(A,C), in(C,D), vbn(D,E).
rmv(A,B,vbz) :- noun(A,C), nounp(B,D), cma(D,E), vbg(E,F).
rmv(A,B,vbz) :- noun(A,C), nounp(B,D), nounp(D,E), vp(E,F).
rmv(A,B,vbz) :- noun(A,C), nounp(B,D), nounp1(A,E), vbz(E,F).
rmv(A,B,vbz) :- noun(A,C), nounp(B,D), vbd(D,E).
rmv(A,B,vbz) :- noun(A,C), nounp(B,D), vp(D,E), to(E,F).
rmv(A,B,vbz) :- noun(A,C), nounp1(A,D), vp1(D,E), vbn(D,F).
rmv(A,B,vbz) :- noun(A,C), to(C,D).
rmv(A,B,vbz) :- noun(A,C), vbp(C,D).
rmv(A,B,vbz) :- noun(A,C), vbz(C,D).
rmv(A,B,vbz) :- noun(A,C), vp1(C,D), nounp(B,E).
rmv(A,B,vbz) :- noun(B,C), cc(C,D), nounp(D,E), noun(E,F).
rmv(A,B,vbz) :- noun(B,C), cma(C,D), nounp(D,E), noun(A,F).
rmv(A,B,vbz) :- noun(B,C), cma(C,D), nounp(D,E), noun(E,F).
rmv(A,B,vbz) :- noun(B,C), cma(C,D), nounp1(A,E), noun(E,F).
rmv(A,B,vbz) :- noun(B,C), in(C,D), nounp(D,E), vbd(E,F).
```

Figure 47: Theory for `vbz`.

```
rmv(A,B,vbz) :- noun(B,C), noun(A,D), cc(D,E).
rmv(A,B,vbz) :- noun(B,C), noun(A,D), nounp(B,E), vp(E,F).
rmv(A,B,vbz) :- noun(B,C), noun(A,D), nounp1(A,E), vp1(E,F).
rmv(A,B,vbz) :- noun(B,C), nounp(C,D), noun(A,E), cma(E,F).
rmv(A,B,vbz) :- noun(B,C), nounp(C,D), noun(A,E), end(E).
rmv(A,B,vbz) :- noun(B,C), nounp(C,D), stp(D,E), noun(A,F).
rmv(A,B,vbz) :- noun(B,C), nounp1(A,D), cc(D,E), nounp1(E,F).
rmv(A,B,vbz) :- noun(B,C), nounp1(A,D), dt(D,E), vpart(E,F).
rmv(A,B,vbz) :- noun(B,C), nounp1(A,D), in(D,E).
rmv(A,B,vbz) :- noun(B,C), nounp1(A,D), noun(D,E), cc(E,F).
rmv(A,B,vbz) :- noun(B,C), nounp1(A,D), noun(D,E), cma(E,F).
rmv(A,B,vbz) :- noun(B,C), nounp1(A,D), nounp1(D,E), in(E,F).
rmv(A,B,vbz) :- noun(B,C), nounp1(A,D), vbn(D,E).
rmv(A,B,vbz) :- noun(B,C), stp(C,D), noun(A,E).
rmv(A,B,vbz) :- noun(B,C), to(C,D), noun(A,E), dt(E,F).
rmv(A,B,vbz) :- noun(B,C), to(C,D), vb(D,E), noun(A,F).
rmv(A,B,vbz) :- nounp(B,C), cc(C,D), nounp1(A,E), nounp1(E,F).
rmv(A,B,vbz) :- nounp(B,C), cma(C,D), nounp1(A,E), cc(E,F).
rmv(A,B,vbz) :- nounp(B,C), cma(C,D), vbd(D,E), nounp1(A,F).
rmv(A,B,vbz) :- nounp(B,C), nounp1(A,D), to(D,E).
rmv(A,B,vbz) :- nounp(B,C), nounp1(A,D), vpart(D,E), vp1(E,F).
rmv(A,B,vbz) :- nounp(B,C), vp(C,D), in(D,E), vbd(C,F).
rmv(A,B,vbz) :- nounp(B,C), vp(C,D), nounp1(A,E), in(E,F).
rmv(A,B,vbz) :- nounp1(A,C), advp(C,D), vp1(D,E).
rmv(A,B,vbz) :- nounp1(A,C), dt(C,D), in(D,E), adjp1(E,F).
rmv(A,B,vbz) :- nounp1(A,C), in(C,D), adjp1(D,E), vpart(E,F).
rmv(A,B,vbz) :- nounp1(A,C), to(C,D), nounp1(D,E), adjp1(E,F).
rmv(A,B,vbz) :- nounp1(A,C), to(C,D), vpart(D,E).
rmv(A,B,vbz) :- nounp1(A,C), vb(C,D).
rmv(A,B,vbz) :- nounp1(A,C), vp1(C,D), nounp1(A,E), advp(E,F).
rmv(A,B,vbz) :- nounp1(A,C), vpart(C,D), in(D,E).
rmv(A,B,vbz) :- pos(A,C).
rmv(A,B,vbz) :- ppz(A,C).
rmv(A,B,vbz) :- sverb(B,C), adjp(C,D).
rmv(A,B,vbz) :- vb(A,C).
rmv(A,B,vbz) :- vbp(B,C).
rmv(A,B,vbz) :- vp(B,C), dt(A,D).
rmv(A,B,vbz) :- vp(B,C), dt(C,D), nounp(D,E).
rmv(A,B,vbz) :- vp(B,C), dt(C,D), nounp1(A,E).
rmv(A,B,vbz) :- vp(B,C), noun(C,D).
rmv(A,B,vbz) :- vp(B,C), nounp(C,D), in(D,E).
rmv(A,B,vbz) :- vp(B,C), nounp(C,D), noun(D,E).
rmv(A,B,vbz) :- vp(B,C), to(C,D).
rmv(A,B,vbz) :- vp(B,C), vpart(C,D).
rmv(A,B,vbz) :- vpart(A,C).
```

Figure 48: Theory for `vbz` (ctd) .

```
rmv(A,B,wdt) :- adjp1(A,C), vp1(C,D).
rmv(A,B,wdt) :- nounp(B,C).
rmv(A,B,wdt) :- vp1(A,C).
```

Figure 49: Theory for `wdt`.

B Contingency tables for each tag

Predicted↓	Actual Pos	Neg	Total
Pos	203	22	225
Neg	188	101	289
Total	391	123	514

True positives = 51.9%
True negatives = 82.1%
Overall accuracy = 59.1%

Figure 50: Contingency table for cc

Predicted↓	Actual Pos	Neg	Total
Pos	0	0	0
Neg	248	749	997
Total	248	749	997

True positives = 0.0%
True negatives = 100.0%
Overall accuracy = 75.1%

Figure 51: Contingency table for cd

Predicted↓	Actual Pos	Neg	Total
Pos	495	42	537
Neg	129	334	463
Total	624	376	1000

True positives = 79.3%
True negatives = 88.8%
Overall accuracy = 82.9%

Figure 52: Contingency table for dt

Predicted↓	Actual Pos	Neg	Total
Pos	34	0	34
Neg	38	246	284
Total	72	246	318

True positives = 47.2%
True negatives = 100.0%
Overall accuracy = 88.1%

Figure 53: Contingency table for ex

Predicted↓	Actual Pos	Neg	Total
Pos	6	0	6
Neg	68	21	89
Total	74	21	95

True positives = 8.1%
True negatives = 100.0%
Overall accuracy = 28.4%

Figure 54: Contingency table for fw

Predicted↓	Actual Pos	Neg	Total
Pos	161	11	172
Neg	169	659	828
Total	330	670	1000

True positives = 48.8%
True negatives = 98.4%
Overall accuracy = 82.0%

Figure 55: Contingency table for in

Predicted↓	Actual Pos	Neg	Total
Pos	165	10	175
Neg	363	462	825
Total	528	472	1000

True positives = 31.2%
True negatives = 97.9%
Overall accuracy = 62.7%

Figure 56: Contingency table for jj

Predicted↓	Actual Pos	Neg	Total
Pos	217	9	226
Neg	225	549	774
Total	442	558	1000

True positives = 49.1%
True negatives = 98.4%
Overall accuracy = 76.6%

Figure 57: Contingency table for jjr

Predicted↓	Actual Pos	Neg	Total
Pos	99	3	102
Neg	176	348	524
Total	275	351	626

True positives = 36.0%
True negatives = 99.1%
Overall accuracy = 71.4%

Figure 58: Contingency table for jjs

Predicted↓	Actual Pos	Neg	Total
Pos	71	0	71
Neg	7	40	47
Total	78	40	118

True positives = 91.0%
True negatives = 100.0%
Overall accuracy = 94.1%

Figure 59: Contingency table for md

Predicted↓	Actual Pos	Neg	Total
Pos	261	21	282
Neg	241	477	718
Total	502	498	1000

True positives = 52.0%
True negatives = 95.8%
Overall accuracy = 73.8%

Figure 60: Contingency table for nn

Predicted↓	Actual Pos	Neg	Total
Pos	219	18	237
Neg	188	575	763
Total	407	593	1000

True positives = 53.8%
True negatives = 97.0%
Overall accuracy = 79.4%

Figure 61: Contingency table for nns

Predicted↓	Actual Pos	Neg	Total
Pos	158	22	180
Neg	350	470	820
Total	508	492	1000

True positives = 31.1%
True negatives = 95.5%
Overall accuracy = 62.8%

Figure 62: Contingency table for np

Predicted↓	Actual Pos	Neg	Total
Pos	182	15	197
Neg	335	468	803
Total	517	483	1000

True positives = 35.2%
True negatives = 96.9%
Overall accuracy = 65.0%

Figure 63: Contingency table for nps

Predicted↓	Actual Pos	Neg	Total
Pos	773	11	784
Neg	81	135	216
Total	854	146	1000

True positives = 90.5%
True negatives = 92.5%
Overall accuracy = 90.8%

Figure 64: Contingency table for pdt

Predicted↓	Actual Pos	Neg	Total
Pos	65	7	72
Neg	5	923	928
Total	70	930	1000

True positives =	92.9%
True negatives =	99.2%
Overall accuracy =	98.8%

Figure 65: Contingency table for pos

Predicted↓	Actual Pos	Neg	Total
Pos	79	11	90
Neg	9	24	33
Total	88	35	123

True positives =	89.8%
True negatives =	68.6%
Overall accuracy =	83.7%

Figure 66: Contingency table for pp

Predicted↓	Actual Pos	Neg	Total
Pos	5	0	5
Neg	20	87	107
Total	25	87	112

True positives =	20.0%
True negatives =	100.0%
Overall accuracy =	82.1%

Figure 67: Contingency table for ppz

Predicted↓	Actual Pos	Neg	Total
Pos	398	21	419
Neg	332	249	581
Total	730	270	1000

True positives = 54.5%
True negatives = 92.2%
Overall accuracy = 64.7%

Figure 68: Contingency table for rb

Predicted↓	Actual Pos	Neg	Total
Pos	274	24	298
Neg	408	294	702
Total	682	318	1000

True positives = 40.2%
True negatives = 92.5%
Overall accuracy = 56.8%

Figure 69: Contingency table for rbr

Predicted↓	Actual Pos	Neg	Total
Pos	248	20	268
Neg	105	204	309
Total	353	224	577

True positives = 70.3%
True negatives = 91.1%
Overall accuracy = 78.3%

Figure 70: Contingency table for rbs

Predicted↓	Actual Pos	Neg	Total
Pos	401	24	425
Neg	359	216	575
Total	760	240	1000

True positives = 52.8%
True negatives = 90.0%
Overall accuracy = 61.7%

Figure 71: Contingency table for rp

Predicted↓	Actual Pos	Neg	Total
Pos	0	0	0
Neg	0	0	0
Total	0	0	0

True positives is undefined
True negatives is undefined
Overall accuracy is undefined

Figure 72: Contingency table for rpn

Predicted↓	Actual Pos	Neg	Total
Pos	355	0	355
Neg	0	0	0
Total	355	0	355

True positives = 100.0%
True negatives is undefined
Overall accuracy = 100.0%

Figure 73: Contingency table for rqt

Predicted↓	Actual Pos	Neg	Total
Pos	0	0	0
Neg	1	0	1
Total	1	0	1

True positives = 0.0%
True negatives is undefined
Overall accuracy = 0.0%

Figure 74: Contingency table for stp

Predicted↓	Actual Pos	Neg	Total
Pos	0	0	0
Neg	0	1	1
Total	0	1	1

True positives is undefined
True negatives = 100.0%
Overall accuracy = 100.0%

Figure 75: Contingency table for sym

Predicted↓	Actual Pos	Neg	Total
Pos	0	0	0
Neg	0	0	0
Total	0	0	0

True positives is undefined
True negatives is undefined
Overall accuracy is undefined

Figure 76: Contingency table for to

Predicted↓	Actual Pos	Neg	Total
Pos	59	0	59
Neg	15	11	26
Total	74	11	85

True positives = 79.7%
True negatives = 100.0%
Overall accuracy = 82.4%

Figure 77: Contingency table for uh

Predicted↓	Actual Pos	Neg	Total
Pos	494	16	510
Neg	60	430	490
Total	554	446	1000

True positives = 89.2%
True negatives = 96.4%
Overall accuracy = 92.4%

Figure 78: Contingency table for vb

Predicted↓	Actual Pos	Neg	Total
Pos	443	12	455
Neg	139	406	545
Total	582	418	1000

True positives = 76.1%
True negatives = 97.1%
Overall accuracy = 84.9%

Figure 79: Contingency table for vbd

Predicted↓	Actual Pos	Neg	Total
Pos	198	13	211
Neg	308	481	789
Total	506	494	1000

True positives = 39.1%
True negatives = 97.4%
Overall accuracy = 67.9%

Figure 80: Contingency table for vbg

Predicted↓	Actual Pos	Neg	Total
Pos	299	15	314
Neg	203	483	686
Total	502	498	1000

True positives = 59.6%
True negatives = 97.0%
Overall accuracy = 78.2%

Figure 81: Contingency table for vbn

Predicted↓	Actual Pos	Neg	Total
Pos	640	3	643
Neg	105	252	357
Total	745	255	1000

True positives = 85.9%
True negatives = 98.8%
Overall accuracy = 89.2%

Figure 82: Contingency table for vbp

Predicted↓	Actual Pos	Neg	Total
Pos	642	12	654
Neg	201	145	346
Total	843	157	1000

True positives = 76.2%
True negatives = 92.4%
Overall accuracy = 78.7%

Figure 83: Contingency table for vbz

Predicted↓	Actual Pos	Neg	Total
Pos	675	7	682
Neg	68	250	318
Total	743	257	1000

True positives = 90.8%
True negatives = 97.3%
Overall accuracy = 92.5%

Figure 84: Contingency table for wdt

Predicted↓	Actual Pos	Neg	Total
Pos	0	0	0
Neg	4	1	5
Total	4	1	5

True positives = 0.0%
True negatives = 100.0%
Overall accuracy = 20.0%

Figure 85: Contingency table for wp