

ESSLLI 2004 Introductory Course on *Symbolic Learning of Natural Language*

Dimitar Kazakov and James Cussens

Department of Computer Science

University of York

Heslington, York YO10 5DD, UK

{kazakov|jc}@cs.york.ac.uk

<http://www-users.cs.york.ac.uk/~kazakov>

Machine Learning (ML) of language studies the acquisition of linguistic knowledge from examples. While statistical learning methods are widely spread nowadays, they have their limitations. The need for adequate modelling of semantics, and acquisition of human-comprehensible theories has led to an increased interest in symbolic learning.

The course demonstrates how Symbolic ML (SML) is used across the whole range of natural language processing (NLP) tasks. Word morphology learning is chosen to introduce a number of SML techniques and compare supervised and unsupervised learning. Of these, Transformation-Based Learning is further focussed on and its use for tagging and bracketing discussed. Tagging is also used to illustrate the principles of Inductive Logic Programming (ILP). Detailed discussion of ILP learning of syntax, lexical semantics and ontologies follows. Hybrid SML is the last topic covered.

Prerequisites: good practical knowledge of logic programming and familiarity with the basics of NLP, but no knowledge of ML.

Introduction

Symbolic Learning of Natural Language has three components:

- (Machine) Learning
- Natural Language (Processing): hand-crafted vs corpus-based
- Symbolic (vs statistical) learning of language

2

Symbolic Learning of Natural Language

Dimitar Kazakov
kazakov@cs.york.ac.uk
Department of Computer Science
University of York

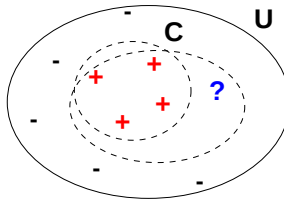
ESSLLI 2004, Nancy, France

Links and Online Resources:

<http://www-users.cs.york.ac.uk/~kazakov/esslli04.html>

Concept and Bias

- Given: a set of observations U called *universe*. All observations are described in an *object language*.
- Define *concept* C as a subset of U . C can also be seen as a binary function (or a *predicate*) from U to $\{0, 1\}$.
- Given: (1) a subset of U split into members and non-members of C , (2) an extra element. Does the latter belong to C ?



- No answer is possible w/o the introduction of a *bias* restricting the *concept language* used to describe C .

4

Machine Learning

the study of computer programs that improve through experience their performance at a certain class of tasks, as measured by a given performance measure.

T.Mitchell, Machine Learning, 1997

3

Inductive Learning

- Given a set of observations relevant to a certain *target concept*, learn to recognise its members.
- Already known concepts or *background knowledge* may be used to derive a theory about the target concept.

6

The Need of Bias (Example)

O_1	black	5-seat	Toyota
O_2	black	5-seat	Renault
O_3	red	2-seat	Ferrari
O_4	red	2-seat	Corvette

Given: $U = \{O_1, O_2, O_3, O_4\}$, $C_1 = \{O_1, O_2\}$, $C_2 = \{O_3, O_4\}$

New observation: a black Ford with 5 seats. Does it belong to C_1 ? What about C_2 ?

5

Taxonomy of Machine Learning

- Lazy/eager learning
- Black/white box learning
- Pos&neg/positive-only learning
- Batch/incremental learning
- Perfect/noisy data

8

Inductive Learning (cont.)

- Concepts of finite size, all instances known: can memorise all members, yet might want to learn a shorter description.
- Otherwise: given a set of observations, a *training set*, recognise the target concept in unseen data (*test set*).
- *Inductive Learning Hypothesis*: theories that work well on (a sufficient amount of) training data, will also do so on unseen data, provided both training and test data come from the same source.

7

Natural Language Processing (2)

- Theoretical framework provided by [Computational] Linguistics
- Two approaches:
 - Hand-crafted tools
 - Corpus-based (i.e. machine learning of natural language)

10

Natural Language Processing

aims at the development of tools allowing to retrieve, extract and classify linguistic information, and, to a certain degree, to understand language and provide translation or relevant feedback.

Statistical Learning of Language

- + Robust, can handle noisy data
- + Often offers faster learning
- ++ Indispensable in processing analogue data (e.g., speech)
- Requires lots of data (ever heard the phrase “statistically significant”?)
- Has its limits: exceptions are by definition insignificant
- The result is a black-box theory/model of data

12

Hand-crafted NLP Tools vs Learning of Natural Language

Hand-crafted tools are costly to develop, and difficult to adapt to specific areas; can achieve very high accuracy after fine-tuning

NLP Learning allows for domain-specific rules, requires (annotated) text corpora

Fact: To collect (and annotate) a corpus requires less experience than to develop the corresponding tool

Fact: A large number of corpora (some of them annotated) are available

11

Summary

- Definition and types of Machine Learning; Concept, Bias, Object Language, Concept Language, Background Knowledge
- Natural Language Processing; Hand-crafted vs Corpora-driven NLP
- Natural Language Learning: Statistical vs Symbolic

14

Symbolic Learning of Language

A richer concept language may result in:

- a brittle theory
- + fewer/no problems with exceptions
- slower learning
- + no need for large training sets: a few well-chosen examples may be enough
- + a theory that can be understood by humans
- + ...therefore allowing for fine-tuning.
- + Better suited to processing structured data

13

Levels of Text Processing

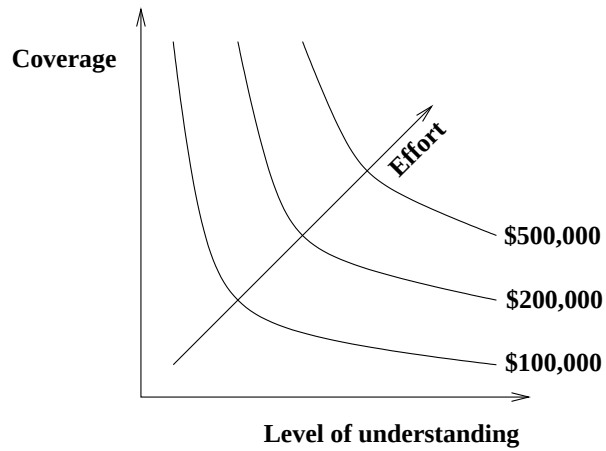
- Several NLP tasks of increasing complexity
- Some can be done in parallel (depending on the formalisms used)
- Can be (partially) ordered: $\{T_1, T_2, \dots\}$, so that there is no T_i that requires T_{i+k} as a pre-processing step.

16

Types of NLP Tasks and Text Corpora

15

Cost vs. Coverage of NLP Tools



18

Levels of Text Processing

Tokenisation	Full parsing
Morpholexical analysis	Lexical semantic disambiguation
Part-of-speech tagging	Semantic analysis
Chunking	Pragmatic analysis
Partial/Shallow parsing	

17

Tokenisation

- Split the text into words/tokens:
 - π • is • equal • to • 3.14 • . • No • , • it • is • n't • . • On • 29/01/2001
 - ... •
- Same kind of task as when processing programming languages
- Standard or hand-written FSA (lex, tr, Perl) are usually used.
- In speech recognition the input does not contain delimiters: *"hii'myourlecturer"*. Learning can be used to segment it.

20

Starting Points for NLP Learning

- The NLP tool exists, but is too bad a value for money.
- No good tool exists for our language, but one can find a corpus with examples of the input and output required.
- There is no (annotated) corpus either.

Another dichotomy: (1) learning may aim to produce the best results or (2) we simply want to see if it can produce anything nearly as good as the hand-crafted tools.

19

Morphology Models

- Analytical vs. generative
- one-, two- or many-level morphology
- Example 1: **Generative morphology** (Chomsky): stick morphemes together, then apply rules in sequence to derive the surface from the lexical level form:

Lexical level:	te+mi	
Rule 1: (vowel raising)	timi	change <i>e</i> into <i>i</i> in front of 0–N consonants followed by <i>i</i>
Rule 2: (palatalisation)	cimi	change <i>t</i> into <i>c</i> in front of an <i>i</i> .
Surface level:	cimi	

Also: uN+Possible → UMpossible → impossible

22

Morpholexical Analysis

- Given a word-form,
find all matching lexical entries and categories:
shows:(PoS:verb,lex.entry:show,pers:3, num:sing,tense:pres)
shows:(PoS:noun,lex.entry:show,num:sing)
- Not difficult, but result often very ambiguous (that's all right!). Can be done by using:

exhaustive lexicons of word-forms	cats	a morphology model, e.g.:	
		<i>stem</i>	<i>ending</i>
work	V_inf	work	–
works	V_3pers		s
worked	V_past		ed
working	V_ing		ing
leaf	N_sing	leaf	
leaves	N_plur	leav	es

21

Spelling vs. Pronunciation

- Two-level models of morphology (lexicons of morphemes + rules) available for many languages (tool PC KIMMO). Hand-written approach still prevailing.
- A similar task is to match spelling and pronunciation. Corpora may contain:

– pairs of unaligned strings (CELEX): `booking` - `bukɪŋ`

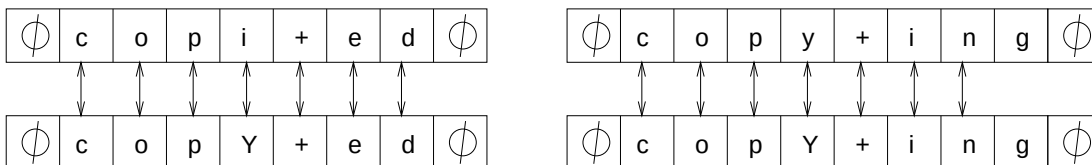
– same as above but characters are aligned:

b	o	o	k	i	n	g
b	u	-	k	i	ŋ	-

24

Morphology Models: Two-Level Morphology

- Example 2: **Two-level morphology** (Koskenniemi): uses a finite state transducer (FST), an automaton with two tapes and two heads moving synchronously. It can be seen as accepting the aligned lexical and surface level strings, or rejecting them according to a set of rules compiled into the FST transition table.



Rules are of the sort:

$[???, i, +, e] \Leftrightarrow [???, Y, +, e]$

$[???, s, s, +, e, s] \Leftrightarrow [???, s, s, +, \emptyset, s]$

23

Word Segmentation

Splitting the word into morphemes (e.g., de-com-press-ed) can be very helpful in:

- Morpholexical analysis: to match each morpheme against a lexicon of morphemes (prefixes/roots/suffixes) to identify the word.
- Text-to-speech: to find the pronunciation of each morpheme: compare *dis-honest* and **di-shone-st*, **di-shon-est*

26

More Corpora for Morphology Learning

- Pairs of word-forms corresponding to two categories, e.g.: sing-plural, present-past, etc. E.g., English Past Tense Data:

```
past([s,l,e,e,p],[s,l,e,p,t]).  
past([w,a,l,k],[w,a,l,k,e,d]).
```

- Lexicons of word-forms and their categories. Can be used to learn rules predicting the category of a given word-form or deriving a new word-form given its lexical entry and desired category:

Word-form₁ Cat_X – standard form of lexical entry
Word-form₂ Cat_Y
Word-form₃ Cat_Z

25

Part-of-Speech Tagging

- Definition: part of speech, form class, word class – one of the traditional categories of words intended to reflect their functions in a grammatical context. (WordNet)

Example: John/NNP loves/VBP books/NNS ./.

- Previous NLP processing required: tokenisation.
- Probably the most common type of annotated corpora: included in Penn Treebank, Susanne,...
- The number of PoS tags can vary greatly: from a couple of dozen (some English corpora) to over 1600 (Czech National Corpus).

28

Word Segmentation Corpora

- There are corpora such as the one derived from CELEX by van den Bosch:

book 10001

books 100011

booking 1000100

- The number of possible segmentations = $2^{\text{word length}-1}$. (Why?)
For a 28-letter word – antidisestablishmentarianism – $2^{27} \approx 10^8$.

27

Partial/Shallow Parsing

- Task: identify syntactic patterns in a sentence. No complete parse of the sentence is needed.
- Syntactic patterns, aka 'elementary semantic links' (Pazienza, Basili): Noun Phrases (NP), Subject-Verb (SV) Phrases, N-V-N, N-Prep-N, etc.:
NASA launched a new satellite this morning. →
N-V-N:(NASA launched satellite), N-V-N:(NASA launched morning)

I saw the boy with the hat. →
SV:(I saw), N-Prep-N:(boy with hat).
- Any existing treebank of complete parses can be used. Lots of existing work in statistical learning.

30

Chunking

- Chunking: dividing sentences into *non-overlapping* phrases.
- NP chunking is a subtask: recognise and mark the chunks that represent noun phrases. Notice: no nested NP chunks possible (by definition): *NP(The boy) with NP(the hat) sang*

One standard dataset is Ramshaw and Marcus's:

<ftp://ftp.cis.upenn.edu/pub/chunker/>

```
Confidence NN I I O B
in IN O O I B
the DT I I O B
pound NN I I O B
is VBZ O O I B
...
```

Format: Attr1:Word Attr2:PoS Attr3:NP_marker (ignore the rest).
'I' marks the beginning of a NP, 'O' the end of it.

29

Full Parsing

- Corpora containing parsed sentences are called *treebanks*.
- Ex.: S(NP(NN:Time) VP(V:flies PP(Prep:like NP(DT:an NN:arrow))))
- Treebanks: Penn Treebank (includes Wall Street Journal (WSJ) Corpus and Brown Corpus), Susanne, . . .
- Building of such treebanks perceived as issue of national importance: Czech National Corpus Institute

32

Bracketing

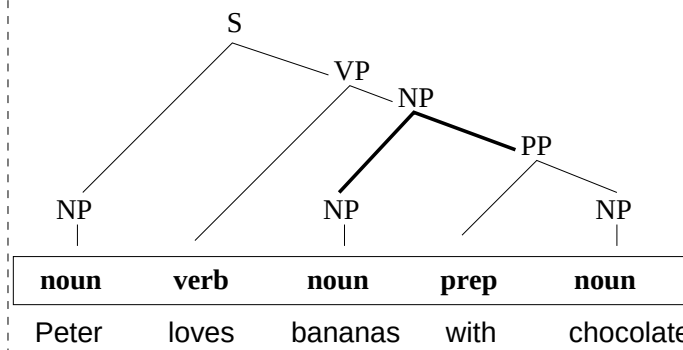
- Bracketing: produces a parse tree without labels:

((((Time) (flies (like (an arrow))))).)
(((The dog) (barked))).)

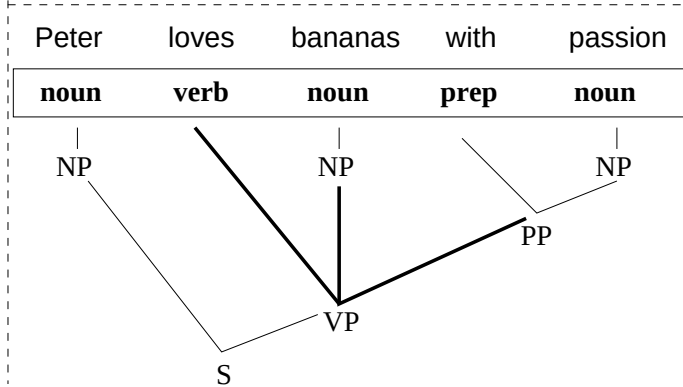
- Result: more informative than chunking – e.g., allows for nested structures –, but simpler than parsing.

31

S $\rightarrow$ NP VP
 VP $\rightarrow$ verb NP
 NP $\rightarrow$ noun
NP $\rightarrow$ **NP PP**
 PP $\rightarrow$ prep NP



S $\rightarrow$ NP VP
VP $\rightarrow$ **verb NP PP**
 NP $\rightarrow$ noun
 PP $\rightarrow$ prep NP



Lexical Semantic Disambiguation

- Semantics: the study of meaning (of isolated words as well as sentences).
- The same word-form may correspond to several lexical entries and meanings:
 On the **bank** of the river Jordan. . . vs.
 Strong restrictions will be imposed on the **bank** of England. . .
- Different meaning $\sim$ different parse: see next slide
- Semcor: a part of the Brown Corpus which contains unambiguous lexical semantic tags. Tag set – taken from WordNet.

QLF

- Quasi-logical forms: neutral with respect to the choice of:
 - referents for pronouns (she, that) and definite descriptions (Mary, the woman)
 - relations implied by ellipsis (*Have you seen my book?* - *No, I haven't.*)
 - alternative scopings of quantifiers: *Every man does not love a woman* contains $\forall, \exists, \neg$. There are 6 possible interpretations according to the order of their application, e.g.:

$$\forall X.man(X)(\exists Y.woman(Y)(\neg loves(X,Y)))$$

$$\exists Y.woman(Y)(\forall X.man(X)(\neg(loves(X,Y))))$$

35

Semantic analysis

- Expresses the meaning of a sentence in terms of entities and their relations: *The little boy loves his mother*:
 $loves(A,B), boy(A), little(A), mother(B,A)$. (A=? B=?)
- First order logic usually used to express the semantic interpretation of a sentence.
- Database terminology (SQL, relations) also suitable:
Which is the capital of Britain?

```
select city
from table
where country = "Britain" and capital = "Yes"
```

34

Logical Forms

- LF: In a given context, *She hired one* could actually mean *Mary Smith hired a car*.

```
quant(exists,C,[car1,C],
      quant(exists,E,[event,E],
            [past,[hire1,E,mary_smith,C]]))).
```

“There is a car C, and an event E such that, in the past, E is a hiring event by Mary Smith of C.” (David Carter, SRI)

- QLFs of several sentences (a paragraph or more) have to be combined to derive the LFs.

37

Real QLF Example

She hired one.

```
[past,
 [hire,
  q_term(<t=quant,n=sing>,
        E,[event,E]),
  a_term(<t=ref,p=pro,l=she,n=sing>,
        Y,[female,Y]),
  q_term(<t=quant,n=sing>,
        C,a_form(<t=pred,l=one>,
                  P,[P,C]))]]).
```

36

Pragmatic Analysis

- Definition: the meaning of sentences in a particular context; opposes *implicatures* to literal meanings and truth conditions of sentences.

Ex.: “Could you pass me the salt?”, “There is someone there.”

39

Corpora with Semantic Annotation of Sentences

- The LLL corpus: has an experimental character; consists of pairs (Sent, QLF); also the grammar used to produce the corpus is available.

- The Geobase corpus – pairs (Sent, Query) in Prolog:
`ftp://ftp.cs.utexas.edu/pub/mooney/nl-ilp-data/geoqueries`

```
parse(  
[what,are,the,major,cities,in,kansas,?],  
answer(B,(major(B),city(B),loc(B,A),const(A,stateid(kansas))))  
).
```

38

Motivation for Unsupervised Learning of Language

- Cost and subjectivity of annotation
- Use Information Theory (Shannon, '48) to study language as just another communication code
- Political correctness vs. technological imperialism

41

Unsupervised Learning

Language bias a new framework for data representation

Preference bias ranks possible representations of the data

- Occam's razor: make your theory short/simple!
- Minimal Description Length (MDL):
 $|Theory| + |Data/Theory| = \min$
- Expectation Maximisation (EM):
 $P(Data/Theory) = \max$

Search strategy defines the order/way in which possible representations will be evaluated.

- E.g.: first try the most general/specific theory

40

Analogy Principle

In a long term, the word-forms in a given language change and tend to form the following patterns (*Course in General Linguistics*, de Saussure):

$$\begin{array}{l} Pref_1 + Suf_1 : Pref_1 + Suf_2 = \\ Pref_2 + Suf_1 : Pref_2 + Suf_2 \end{array} \quad (1)$$

43

Unsupervised Learning of Word Segmentation

- Analogy (F. de Saussure)
- Prefix/suffix trees (Z. Harris)
- Naïve theory of morphology (D. Kazakov)

42

Analogy as Unsupervised Learning

Language bias 4 Words $\rightarrow$ (P1,S1,P2,S2) + analogy rule

Preference bias

- For a single split-point segmentation: the segmentation supported by the highest number of analogy 'squares' (4-tuples).
- For all possible split points: anything in the language (ie, for which an analogy square exists.)

Search strategy Exhaustive search

45

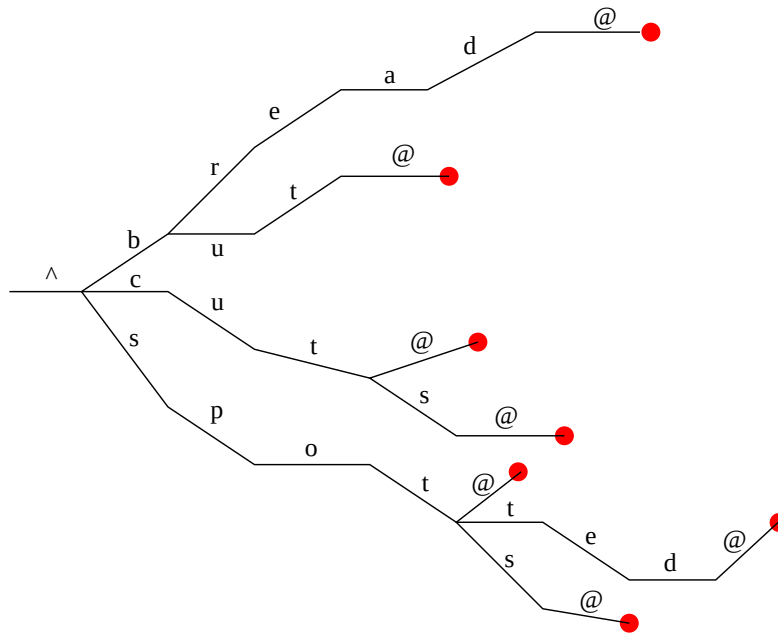
Examples of Analogy

$work + s : work + ing =$ (2)
 $sleep + s : sleep + ing$

$\emptyset + done : \emptyset + limited =$ (3)
 $un + done : un + limited$

44

Example of a Trie



47

Word Segmentation with Tries

- *Trie*: a labelled tree where the labels correspond to a single character.
- Any lexicon can be represented as a trie. For instance, given 'but cut cuts bread spot spots spotted' one could produce the following tree (see next slide).
- Now count the number of branches $br(Prefix)$ for any path *Prefix* from the root downwards.

```

but      : ^-3, b-2, u-1, t-1, @-0
cut      : ^-3, c-1, u-1, t-2, @-0
cut-s    : ^-3, c-1, u-1, t-2, s-1, @-0
bread    : ^-3, b-2, r-1, e-1, a-1, d-1, @-0
spot     : ^-3, s-1, p-1, o-1, t-3, @-0
spot-s   : ^-3, s-1, p-1, o-1, t-3, s-1, @-0
spot-ted : ^-3, s-1, p-1, o-1, t-3, t-1, e-1, d-1, @-0
    
```

46

Harris's Segmentation as Unsupervised Learning

Language bias Word $\rightarrow$ Segments

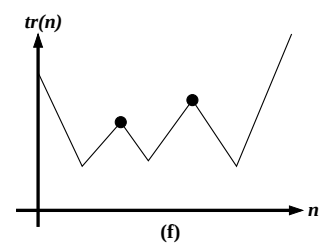
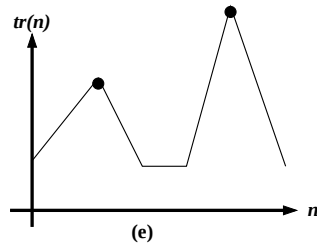
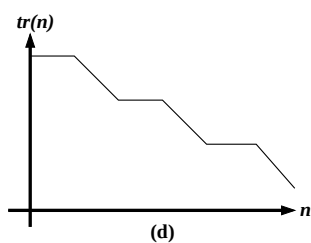
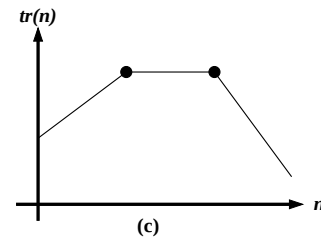
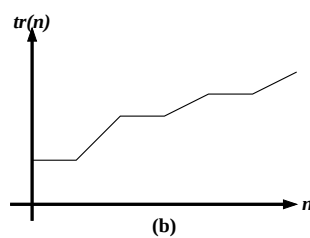
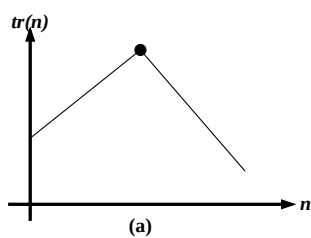
Preference bias Based on the trie + branching factor

Search strategy Exhaustive search

49

Harris's Segmentation Bias

A word w is segmented after the initial substring of length n , if for n : $br(n-1) < br(n) > br(n+1)$. Also, all nodes on a peak plateau are split points.



48

Sample Output of Harris's Segmentation

```
abord-ai      : ^-24, a-12, b-3, o-2, rd-6, a-3, i-3, @-0
abord-aient   : ^-24, a-12, b-3, o-2, rd-6, a-3, i-3, ent@-0
abord-ait     : ^-24, a-12, b-3, o-2, rd-6, a-3, i-3, t@-0
abord-ant     : ^-24, a-12, b-3, o-2, rd-6, a-3, nt@-0
abord-ass-e   : ^-24, a-12, b-3, o-2, rd-6, a-3, ss-2, e@-0
abord-ass-iez : ^-24, a-12, b-3, o-2, rd-6, a-3, ss-2, iez@-0
abord-e       : ^-24, a-12, b-3, o-2, rd-6, e-3, @-0
abord-er      : ^-24, a-12, b-3, o-2, rd-6, e-3, r-3, @-0
abord-erais   : ^-24, a-12, b-3, o-2, rd-6, e-3, r-3, ais@-0
abord-erons   : ^-24, a-12, b-3, o-2, rd-6, e-3, r-3, ons@-0,
abord-ez      : ^-24, a-12, b-3, o-2, rd-6, e-3, z@-0
```

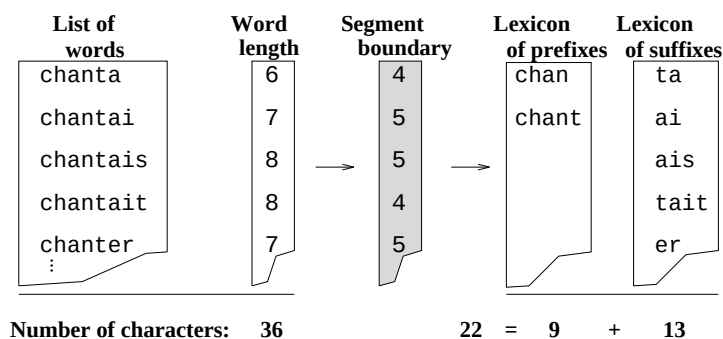
51

Variants of Harris's Segmentation

- Prefix tree (read words from left to right)
- Suffix tree (read words in reverse)
- The union/intersection of the two sets of split points

50

Naïve Theory of Morphology



For a given list of n words, where l_i is the length of word i , every vector $\{b_1, b_2, \dots, b_n\}$, $0 \leq b_i \leq l_i$ will be called a *naïve theory of morphology*.

53

Genetic Algorithms and Unsupervised Word Segmentation with NTM Bias

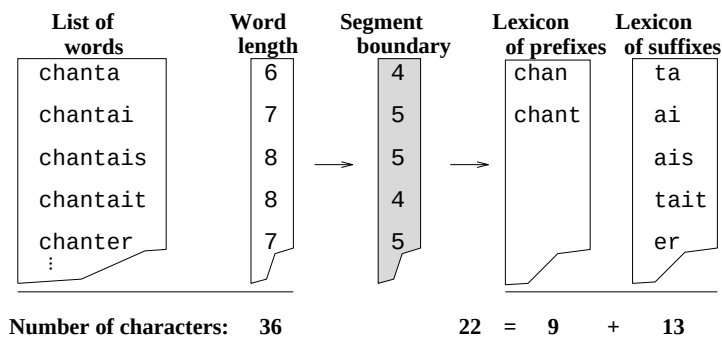
“A struggle for life is constantly going on amongst the words and grammatical forms in each language. The better, the shorter, the easier forms are constantly gaining the upper hand...”

Max Müller: *The Science of Language*, in *Nature*, Jan. 1870, as quoted by Charles Darwin in *The Descent of Man*, 1871.

52

the prefix and suffix lexicons to 2 in each, rather than $P_1 + S_1, P_2 + S_2, P_3 + S_3, P_4 + S_4$.

The NTM Bias



P = total chars. in the prefix lexicon
 S = total chars. in the suffix lexicon
 W = total chars. in the list of words

- **Word segmentation bias:** for a given list of words, choose a NTM that minimises the total lexicon size ($P + S$).
- Compare to analogy: $P_1 + S_1 : P_1 + S_2 = P_2 + S_1 : P_2 + S_2$
 – one could think of it as trying to fix the no of items in

Analogy vs. NTM Comparison

- Analogy: **preference bias** is local, and takes into account a very small part of the data (4 words at a time); produces mini-theories (“analogy squares”), which are local to the word in question and can be tested in isolation (w/o the rest of the data). **Search** can be focussed on the words sharing the same prefix/suffix; one word’s segmentation does not influence the segmentation of another one.
- NTM: Based on a global (holistic) **preference bias** taking into account all data. **Search** is difficult, as it cannot be described as a series of incremental steps. For instance, changing the segmentation of one word may result in changes in the prefix/suffix lexicons, which can potentially influence the segmentation of any other word.

56

Rationale behind NTM Bias

- The NTM bias - based on the hypothesis that substrings composed out of real morphemes occur in the words with a frequency higher than any other left or right substrings. In that way, a theory with a low $P + S$ would produce lexicons where ‘prefixes’ and ‘suffixes’ correspond very often to single morphemes or their combinations.
- Since the word list can be stored as a list of pairs of indices $\langle \text{prefix}, \text{suffix} \rangle$ along with the two lexicons of affixes, the NTM bias can be seen as using *Occam’s razor* to choose the simplest (= shortest) theory corresponding to the given data set.

55

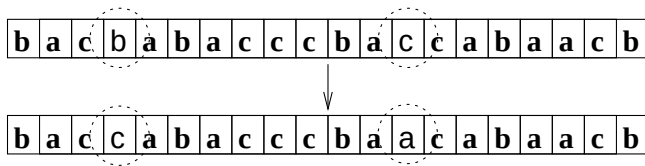
- It is straightforward to represent a NTM as a string of symbols from a given alphabet (integers b/w 0 and, say, 100).
- All that makes the search for NTM a perfect case for GA.

NTM Bias: the problem and the fix

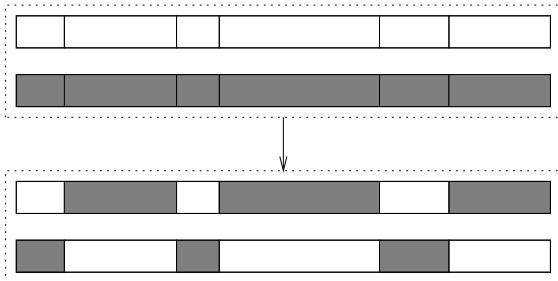
- Search space is huge: $(l_1 + 1).(l_2 + 2). \dots .(l_n + 1)$ where l_i is the length of word i , or, approximately, $(l_{av} + 1)^n$, l_{av} being the average word length and n the number of words in the list.
- No greedy algorithm is known to improve on the solution in an incremental fashion, ultimately leading to the right answer (min. lexicons).
- It is easy to *compare* the quality of any two NTMs, though: just count the letters in each pair of lexicons and compare the two integers!

Genetic Operators

- Mutation



- Crossover

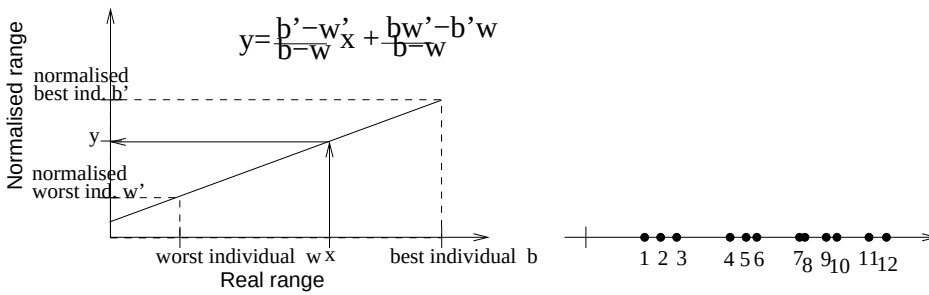


59

Genetic Algorithms

- A GA maintains a set (*population*) of candidate solutions called *chromosomes* or *individuals*
- initial population is generated randomly
- *fitness function* is used to rank individuals according to their goodness
- *selection* is applied to select a new population from the old one, where the chances of one or more clones of the original individual to appear in the new population are proportional to its fitness
- *crossover* and *mutation* are applied to generate new candidate solutions from existing ones.

58



Scaling

- Definition: modify the values of the fitness function in order to make the selection more effective.
- Usually, it involves the change of the individuals' fitness so that the span between the worst and the best corresponds to a given interval.
- Linear scaling
- Ranking

Mapping The NTM Search To GA

Procedure: *Genetic algorithm with fitness scaling*

Initialisation:

- Set *numberOfChromosomes*
- Set *numberOfGenerations*
- Set *mutationMethod*
- Set *crossoverProbability*
- Set *mutationProbability*
- Read *Words*
- Generate randomly *generationZero*,
- compute *fitness[1..numberOfChromosomes]*
- and keep *theBestEver* individual

62

Scaling by Ranking

Order the individuals according to their fitness and then simply assigns them fitness as a function of their rank, $f(x) = rank(x)$, where the order is from the worst to the best one.

61

One Implementation of GA Search for Optimal NTM

- simple GA where each list of segmented words (NTM) represented as an individual – a vector of integers
- fitness function maximises $W - (P + S) \geq 0$
- 1 or 2-point crossover
- mutation defined as one-letter shift to the left or right or as a random change of the segment boundary
- mutation rate is *temporarily* increased by 20% after 3 generations with the same best-ever individual until it changes

63

Main loop:

repeat *numberOfGenerations*-times:

begin

 Compute *scaledFitness*[1..*numberOfChromosomes*]

 Choose intermediate generation *tempChroms* by sampling

 Apply crossover with *crossoverProbability*

 Apply mutation with *mutationProbability*

 Compute *fitness*[1..*numberOfChromosomes*]

 and update *theBestEver* individual

end

return *theBestEver*

Example of GA Output and Evaluation

ébarb+e	ébatt+irent	ébouillant+erez	ébourgeonn+ée
ébarb+ent	ébatt+issent	ébouillant+eriez	ébourgeonn+és
ébarb+erai	ébatt+ra	ébouillant+erions	éb+ouriffais
ébarb+erez	éborgn+es	ébouillant+ez	ébouriff+ait
ébarb+ez	éborgn+ons	ébouillant+ions	ébouriff+asse
ébarb+iez	éborgn+ées	ébouillant+ât	ébouriff+ent
ébarb+ons	ébouillant+assent	éboul+ait	ébouriff+er

	Experiment 1		Experiment 2	
	Items	Characters	Items	Characters
GA input	200	1943	500	4855
Generated prefixes	43	259	75	454
Generated suffixes	64	252	131	638
Compression	0.535	0.263	0.412	0.225

$$\text{Compression} = (P + S)/W$$

64

- To reduce complexity, split input into chunks of $\sim 10^2$ words, then join resulting segmentations of all words (and merge lexicons).

Hybrid GA&ILP Approach

Method

- Introduces a language bias for the segmentation of a list of words, and applies a genetic algorithm to the search of the best segmentation (Kazakov '97).
- Uses the segmentations produced with the GA as training examples of an ILP decision list learner. Segmentation rules applicable to isolated (incl. unseen) words obtained as a result (Kazakov&Manandhar '98).

Advantages

- Does not require annotated lists of words
- Can learn from a limited number of words

Limitations

- The number of constituents is limited to 2 per word.

66

Discussion of NTM Segmentation

Advantages

- Does not require annotated lists of words
- Can learn from a limited number of words

Limitations

- The number of constituents is limited to 2 per word
- Rules still not applicable to isolated words
- Rules still not applicable to unseen words

65

Supervised Learning, Transformation-Based Learning and The Brill Tagger

68

Summary

- Naïve Theories of Morphology (NTM)
- NTM Preference Bias
- Genetic algorithms: genetic operators, scaling
- GA search for the best NTM

67

Evaluation of Supervised Learning

In the classical case of learning a single predicate/concept, the following contingency table is used:

	Pos	Neg
Predicted Pos	True Pos	False Pos
Predicted Neg	False Neg	True Neg

$$Accuracy = \frac{True_Pos + True_Neg}{Pos + Neg}$$

Different/disjunctive training and test sets assumed.

70

Supervised Learning

Data in the form (Observation, Concept)

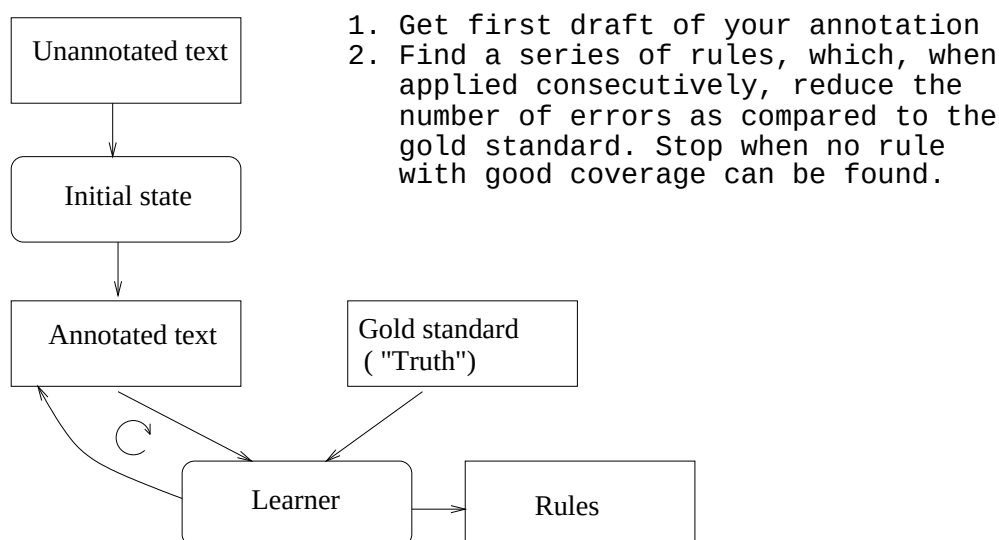
Background knowledge about the domain: explicitly defined or implicitly present in the way search is pruned.

Preference bias Predictive accuracy of the theory learned, i.e. performance on unseen examples.

Search strategy could also impose a preference bias, e.g., *try the most general theories first*.

69

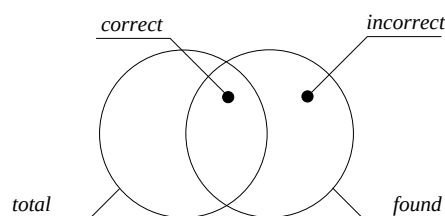
Transformation-Based Learning



72

Evaluation of Supervised Learning in NLP Context

The existence of several correct answers is characteristic:



$$Precision = \frac{correct}{found} \quad (4)$$

$$Recall = \frac{correct}{total} \quad (5)$$

$$F-score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (6)$$

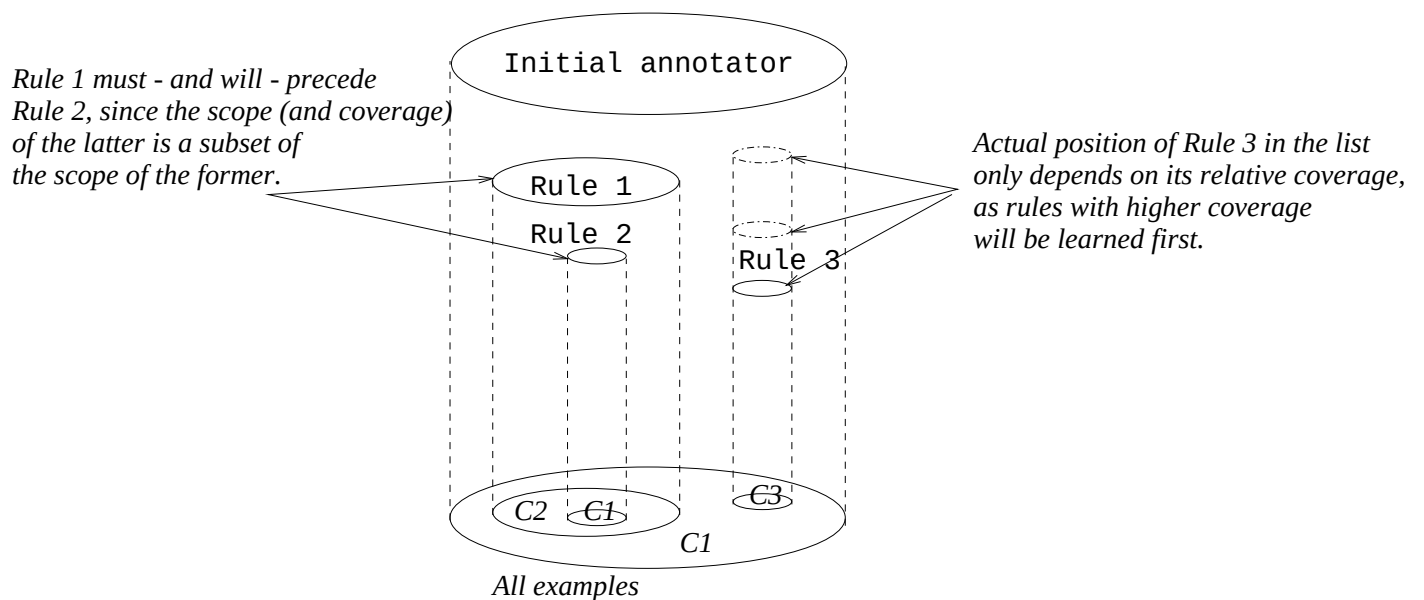
71

Example: Plural of English Nouns

Data:	book	NN	Initial state annotator
	books	NNS	
	shoe	NN	Rule 1
	shoes	NNS	
	dress	NN	Rule 2
	dresses	NNS	
	lemma	NN	Rule 3
	lemmata	NNS	
	cherub	NN	
	cherubim	NNS	

74

Scope of Rules Learned



73

Example of PoS Tagging

Pierre/**NNP** Vinken/**NNP** ,/, 61/**CD** years/**NNS** old/**JJ** ,/,
will/**MD** join/**VB** the/**DT** board/**NN** as/**IN** a/**DT** nonexecu-
tive/**JJ** director/**NN** Nov./**NNP** 29/**CD** ./.

Mr./**NNP** Vinken/**NNP** is/**VBZ** chairman/**NN** of/**IN** Elsevier/**NNP**
N.V./**NNP** ,/, the/**DT** Dutch/**NNP** publishing/**VBG** group/**NN**
./.

See Brill's PhD, pp. 136–137 for the tag set of Pen Treebank
(handout 2).

76

PoS Tagging

A half-way house in between morphosyntactic analysis and pars-
ing:

- In case of M-S ambiguity, it selects just one tag, as parsing would do
- Does not produce a parse tree
- Decisions usually based on local context (a few words to the left and/or right)

75

Rule Templates for PoS Tagging

Change tag **a** to tag **b** when:

1. The preceding/following word is tagged *z*.
2. The word two before/after is tagged *z*.
3. One of the preceding/following 2 words is tagged *z*.
4. One of the preceding/following 3 words is tagged *z*.
5. The preceding word is tagged *z* and the following word is tagged *w*.
6. The preceding/following word is tagged *z* and the word two before/after is tagged *w*.

78

The Brill PoS Tagger

Use Transformation-Based Learning to learn how to produce PoS annotation.

- Initial state annotator: Assign the most probable tag (the one that is assigned the most often to the word in the training data)
- Each rule changes the annotation of words within its scope
- Uses rule templates to describe all possible rules for all words and all parts of speech

77

Lexicalised Rule Templates for PoS Tagging

Add the following templates to the set: Change tag **a** to tag **b** when:

1. The preceding/following word is *w*.
2. The word two before/after is *w*.
3. One of the preceding/following 2 words is *w*.
4. One of the preceding/following 3 words is *w*.
5. The current word is *w* and the preceding/following word is *x*.
6. The current word is *w* and the preceding/following word is tagged *z*.

80

Example of Rules Learned

NN VB PREVTAG TO
VBP VB PREV10R20R3TAG MD
NN VB PREV10R2TAG MD
VB NN PREV10R2TAG DT
VBD VBN PREV10R20R3TAG VBZ
VBN VBD PREVTAG PRP
VBN VBD PREVTAG NNP
VBD VBN PREVTAG VBD
VBP VB PREVTAG TO
POS VBZ PREVTAG PRP
VB VBP PREVTAG NNS

79

Predicting PoS of Unknown Words

- Initial state annotator: very simple, produces what is the majority PoS class in the language
 - default is NNP for capitalised words and NN for the rest
 - can be changed to another tag at the code level
- Rule templates: refer to
 - the first/last 1-2-3-4 letters
 - the presence of letter L in the word
 - word W ever appearing immediately to the left/right
 - adding/deleting affix X (of length up to 4) results in a word.

82

Example of Lexicalised Rules Learned

IN RB WDAND2AFT as as
VBD VBN PREV10R2WD have
IN WDT NEXT10R2TAG VB
VB VBP PREVTAG PRP
VBP VB PREV10R2WD n't
IN WDT NEXTTAG VBZ
IN DT NEXTTAG NN
JJ NNP NEXTTAG NNP
IN WDT NEXTTAG VBD
JJ NN NEXTWD of
VBD VBN PREV10R2WD be

81

Lexical Databases: WordNet

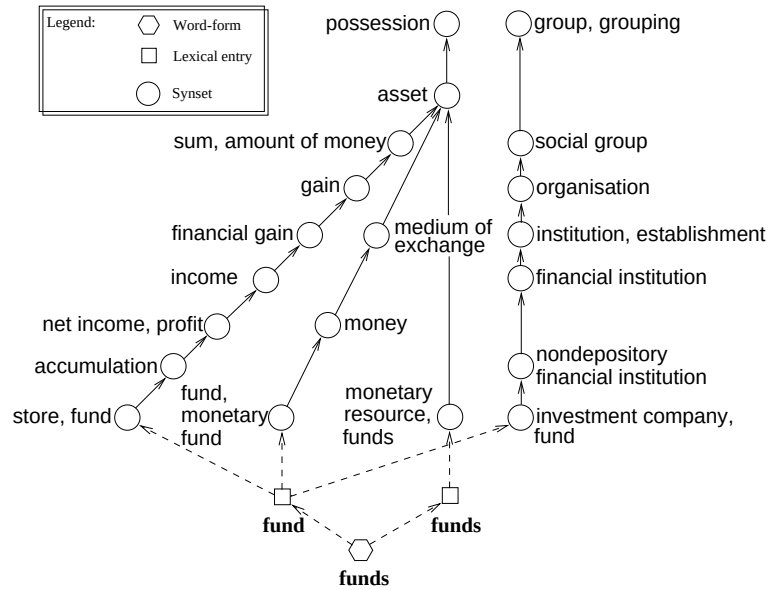
84

Example of Lexical Rules Learned

```
NN s fhassuf 1 NNS x
NN . fchar CD x
NN - fchar JJ x
NN ed fhassuf 2 VBN x
NN ing fhassuf 3 VBG x
ly hassuf 2 RB x
ly addsuf 2 JJ x
NN $ fgoodright CD x
NN al fhassuf 2 JJ x
NN would fgoodright VB x
NN 0 fchar CD x
NN be fgoodright JJ x
NNS us fhassuf 2 JJ x
NNS it fgoodright VBZ x
```

83

WordNet



86

WordNet

- Only covers 4 PoSs: nouns, verbs, adjectives, and adverbs
- Several semantic relations, which are different for each PoS
- IS_A hierarchies available for Verb and Noun
- For adjectives: pairs of direct or indirect antonyms
- Adverbs: antonyms etc.

85

WordNet: Verb Relations

Info available for verb dog

-hyper	Hypernyms
-hypov, -treev	Hyponyms & Hyponym Tree
-synsv	Synonyms (ordered by frequency)
-famlv	Familiarity & Polysemy Count
-framv	Verb Frames
-simsv	Synonyms (grouped by similarity of meaning)
-grepv	List of Compound Words
-over	Overview of Senses

88

WordNet: Noun Relations

%wn dog

Info available for noun dog

-hyphen	Hypernyms
-hypon, -treen	Hyponyms & Hyponym Tree
-synsn	Synonyms (ordered by frequency)
-smemn	Member of Holonyms
-sprtn	Part of Holonyms
-partn	Has Part Meronyms
-meron	All Meronyms
-holon	All Holonyms
-famln	Familiarity & Polysemy Count
-coorn	Coordinate Sisters
-simsn	Synonyms (grouped by similarity of meaning)
-hmern	Hierarchical Meronyms
-hholn	Hierarchical Holonyms
-grepn	List of Compound Words
-over	Overview of Senses

87

Corpus-Based Approach to Syntactic Disambiguation

90

Antonymy b/w Adjectives

Direct vs. indirect antonyms

light (vs. heavy)

heavy (vs. light)

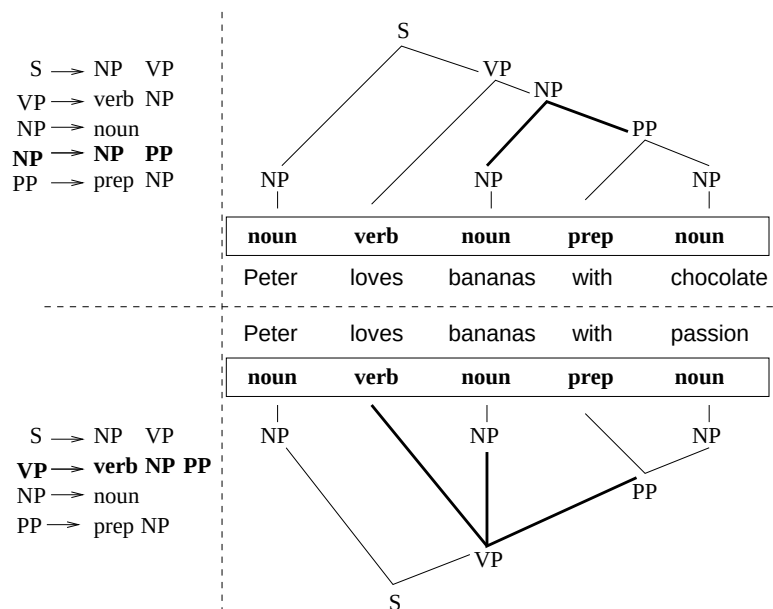
- => dense
- => doughy, soggy
- => heavyweight
- => hefty
- => massive
- => middleweight
- => ponderous
- => welterweight

light (vs. heavy)

- => airy
- => bantamweight
- => buoyant, floaty
- => ...

89

Ambiguity of NL Grammars



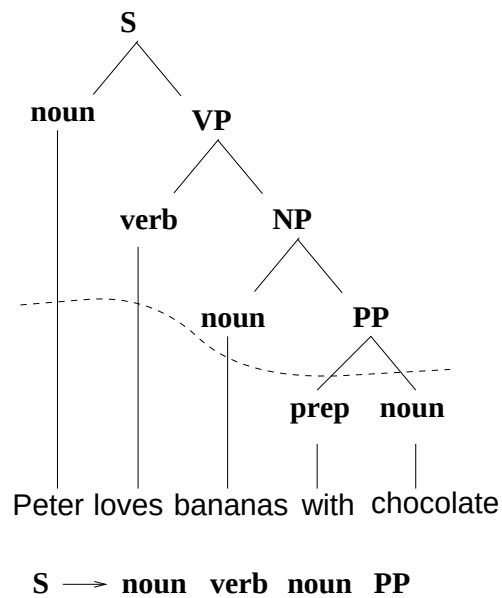
92

Grammars

- Each grammar describes a language, i.e. a set of sequences of symbols from a given alphabet.
- NLP context:
 - ‘symbol’ = word
 - ‘sequence of symbols’ = sentence
 - ‘alphabet’ = parts of speech (Noun, Verb, Prep.)
- Sentence $\xrightarrow{\{\text{Grammar}\}}$ Syntactic structure (parse tree)

91

Resolving Ambiguity (1): Rule Unfolding



94

Parsers

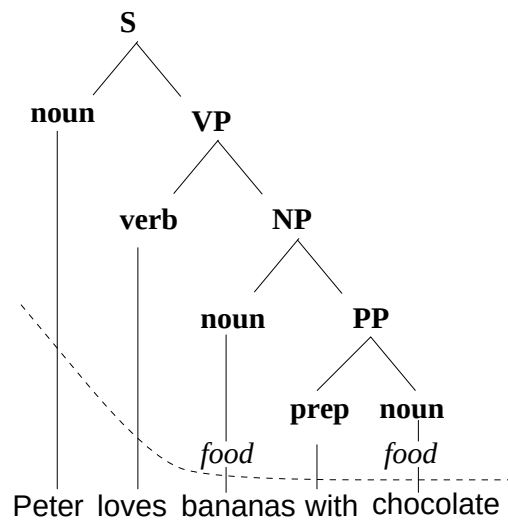
Given a grammar, a parser can

- generate a sentence of the language described by the grammar
- identify whether a sentence belongs to that language
- produce the parse tree

Each parser is an automaton specified by its memory, states and transitions (parser actions).

93

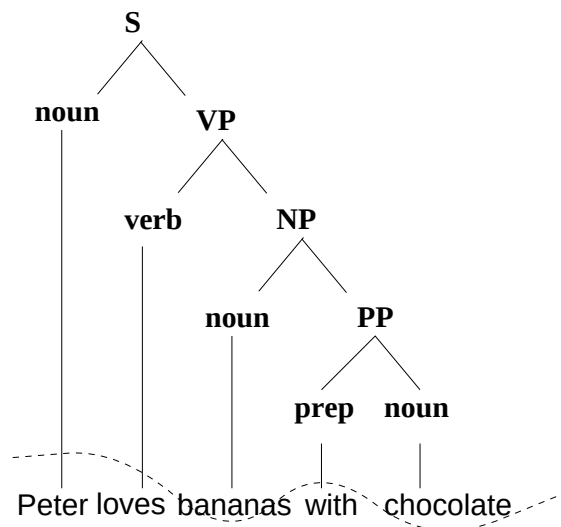
Res. Ambig. (3): Lexical Semantic Constraints



$S \rightarrow \text{noun} \text{ verb } \text{noun:food} \text{ prep } \text{noun:food}$

96

Resolving Ambiguity (2): Rule Lexicalisation



$S \rightarrow \text{noun} \text{ verb } \text{noun:bananas} \text{ prep } \text{noun:chocolate}$

95

EBL v. Chill v. Lapis

System	EBL (Samuelsson)	Chill (Zelle & Mooney)	Lapis (Kazakov)
Training data	Parse trees	Parse trees	Parse trees + lex. sem. tags (WordNet)
Initial concept	CFG rules present in data	Nonterminals (NTs), terminals present in data; rules: NT $\rightarrow$ Any Any NT $\rightarrow$ Any	CFG rules present in data
Target concept	Specialised grammar - unfolding - lexicalisation used by an LR parser	Specialised S-R parser - unfolding - lexicalisation - lex. sem. constraints	Specialised LR parser - unfolding - lexicalisation - lex. sem. constraints
Lex. sem. constraints	none	ad-hoc, sets of words seen in data	standard sem. concepts (WordNet)
Learning algorithm	EBL	ILP - LGG - top-down refinement - predicate invention	ILP - LGG
Issues	- large # of rules - left recursion (NP $\rightarrow$ NP PP)	- very general init. grammar - inefficient S-R parser - ad-hoc sem. constraints	

Applying Inductive Logic Programming to Natural Language Processing

James Cussens

University of York

1

Overview of this lecture

- The basic problem
- Searching
- Scoring
- Using Aleph

The basic ILP problem

- Given B, E - both logic programs, find H , a logic program $\in \mathcal{H}$ such that

$$B, H \vdash E$$

- Need to stop H being 'too strong', e.g. $H = \text{false}$
Use negative examples or syntactic constraints
- Need some way of stopping H being 'too weak', e.g. $H = E$
Bias towards simplicity or syntactic constraints

ESSLLI - 2004

3

The basic ILP problem - Version 2

Normally negative examples are assumed ...

Given: Positive examples P , Negative examples N and Background knowledge B .

Find $H \in \mathcal{H}$ st

- $\forall e \in P : B \wedge H \models e$ (H is complete)
- $\forall e \in N : B \wedge H \not\models e$ (H is consistent)

ESSLLI - 2004

4

Learning daughter/2

INPUT

<i>Training examples</i>		<i>Background knowledge</i>
<i>daughter(mary, ann).</i>	$\oplus$	<i>mother(ann, mary). female(ann).</i>
<i>daughter(eve, tom).</i>	$\oplus$	<i>mother(ann, tom). female(mary).</i>
<i>daughter(tom, ann).</i>	$\ominus$	<i>father(tom, eve). female(eve).</i>
<i>daughter(eve, ann).</i>	$\ominus$	<i>father(tom, ian).</i>
		<i>parent(X, Y) $\leftarrow$ mother(X, Y)</i>
		<i>parent(X, Y) $\leftarrow$ father(X, Y)</i>

OUTPUT

daughter(X, Y) $\leftarrow$ female(X), parent(Y, X)

ESSLLI - 2004

5

Learning daughter/2 – Version 2

INPUT

<i>Training examples</i>		<i>Background knowledge</i>
<i>daughter(mary, ann).</i>	$\oplus$	<i>mother(ann, mary). female(ann).</i>
<i>daughter(eve, tom).</i>	$\oplus$	<i>mother(ann, tom). female(mary).</i>
<i>daughter(tom, ann).</i>	$\ominus$	<i>father(tom, eve). female(eve).</i>
<i>daughter(eve, ann).</i>	$\ominus$	<i>father(tom, ian).</i>

OUTPUT

daughter(X, Y) $\leftarrow$ female(X), mother(Y, X)

daughter(X, Y) $\leftarrow$ female(X), father(Y, X)

ESSLLI - 2004

6

The search

- Even if we know what we're looking for (say, the smallest H satisfying our constraints),
- actually finding it is the difficult bit.
- We have to use a heuristic search in the space $\mathcal{H}$.
- Top-down v bottom-up is a basic dichotomy.

ESSLLI - 2004

7

Clause-at-a-time learning

- Almost always ILP systems induce a multi-clause logic program by finding one clause at a time.
- Not a bad compromise . . .
- . . . especially if H is not recursive (Think about it!)

ESSLLI - 2004

8

Clause-at-a-time learning with Aleph

- Let's see some clauses induced using the Aleph system.
- Example from *Morphosyntactic tagging of Slovene using Prolog* by Cussens, Džeroski and Erjavec, ILP99
- Aleph used to be called P-Progol.

ESSLLI - 2004

9

Understanding the demo-ed application

```
rmv(Left,Right,Focus) :-  
    case(Left,_Case,_), case(Focus,a,_),  
    gender(Left,m,_), disongender(Left,Focus).
```

Left is all tags to the left of Focus.

Right is all tags to the left of Focus.

“Can't have the sequence [Left,Focus] if Left has some case, Focus has accusative case, Left has masculine gender and Focus has some other gender.”

ESSLLI - 2004

10

Top-down search

- Start with most general clause e.g. `rmv(A,B,C)`
- and specialise by adding literals
- (or instantiating variables)
- This is called *refinement*

ESSLLI - 2004

11

Which is the best clause?

- Aleph's default clause evaluation is coverage
- coverage defines clause utility to be $P - N$, where P , N are the number of positive and negative examples covered by the clause.
- Aleph has another 9 evaluation functions to choose from!

ESSLLI - 2004

12

Defining the hypothesis space $\mathcal{H}$

- Probably the most crucial step in using ILP!
- Let's go through the Aleph method for doing this.
- There are many others.

ESSLLI - 2004

13

Choosing background predicates

- The most basic step.
- Which predicates can appear in clauses?

```
:- determination(rmv/3,noun/2).  
:- determination(rmv/3,verb/2).  
:- determination(rmv/3,adjective/2).  
...
```

ESSLLI - 2004

14

User-defined constraints on acceptable clauses

Just write what's allowed in Prolog!

```
prune((_:-Body)) :-  
    has_redundant_literals(Body).  
  
false :-  
    hypothesis(Head,Body,_PosNegCoverage), %Aleph built-in  
    Body=(_,_,_), %only false out at clauselength  
    arg(3,Head,Focus),  
    \+ uses_focus(Body,Focus).
```

`prune/1` disallows a clause *and all its refinements*

ESSLLI - 2004

15

Limiting the length of clauses

- Each clause search can be exponential in the maximum allowed length of a clause.
- So limit it!

```
:- set(clauselength,5).
```

Beware: the Aleph default is `clauselength=4`
This **includes** the head literal.

ESSLLI - 2004

16

Saturation - a bit of bottom up

- *Saturating* an example produces the most specific clause which (together with B) entails the example.
- It's called the *bottom clause*
- Use `sat/1` to do this in Aleph. Like this ...
- Let's see how Aleph did that.

ESSLLI - 2004

17

Mode declarations

- Aleph uses *mode declarations* to build the bottom clause
- Modes declare the types of all arguments of all predicates;
- and whether the argument is
 1. an input variable (+)
 2. an output variable (-)
 3. or a constant term

ESSLLI - 2004

18

Types

- Types do not have to be defined
- Unless you're doing positive-only learning...
- ...in which case use a monadic predicate to define them

ESSLLI - 2004

19

Using mode declarations

```
:- modeh(1,rmv(+msdlist1,+msdlist,+msdlist2)).
```

So each argument of `rmv/3` is an input variable, and they all have different types.

(We'll come back to the number 1.)

ESSLLI - 2004

20

Attaching type labels

```
:- modeh(1,rmv(+msdlist1,+msdlist,+msdlist2)).
```

So when saturating example 21:

```
[rmv([spsg,afpms1..],[spsg,afpsi,...],[mcnsg1])
```

1. `[spsg,afpms1..]` is an input of type `msdlist1`
2. `[spsg,afpsi..]` is an input of type `msdlist`
3. `[mcnsg1]` is an input of type `msdlist2`

ESSLLI - 2004

21

Finding body literals for the bottom clause

```
[spsg,afpms1..] is an input of type msdlist1
```

We have `:- modeb(1,noun(+msdlist1,-msdlist1)).`

So Aleph first calls:

```
noun([spsg,afpms1..],X)
```

which fails

ESSLLI - 2004

22

Finding body literals for the bottom clause

Because of `:- modeb(1,adposition(+msdlist1,-msdlist1)).`, Aleph eventually calls:

```
adposition([spsg,afpmsl..],X)
```

which succeeds with `X = [afpmsl,...]`.

This is how the `adposition(A,D)` got into the bottom clause.

ESSLLI - 2004

23

Variables and constants in the bottom clause

Aleph internally builds up a **ground** clause in this way:

```
rmv([spsg,afpmsl,...],[spsg,...],[mcmsal__y]]) :-  
    adposition([spsg,afpmsl,...],[afpmsl,...]),  
    case([spsg,afpmsl,...],g,[afpmsl,...]),  
    ...
```

But eventually variablises some of these constant terms according to the mode declarations.

ESSLLI - 2004

24

Variable depth

- Variables in the head have depth 0
- (Roughly) an output variable generated by an input variable of depth i has depth $i + 1$.
- `:- set(i,5).` limits variables in the bottom clause to depth 5.
- Default is $i = 2$. Pretty small!

ESSLLI - 2004

25

The recall number - according to Aleph manual

```
mode(RecallNumber,PredicateMode)
```

RecallNumber bounds the non-determinacy of a form of predicate call [when building up the bottom-clause]

RecallNumber can be either (a) a number specifying the number of successful calls to the predicate; or (b) `*` specifying that the predicate has bounded non-determinacy. It is usually easiest to specify RecallNumber as `*`.

ESSLLI - 2004

26

Why build a bottom clause?

- Pure top-down search can waste time constructing clauses *which cover no positive examples!*
- In each Aleph search *only clauses which are generalisations of the bottom clause are constructed.*
- It follows that they all cover the saturated example

ESSLLI - 2004

27

The basic Aleph algorithm

1. Select example (default at random)
2. Saturate it
3. Do top-down search for 'best' clause
4. Remove covered positive examples
5. If there remain positive examples, go to 1

ESSLLI - 2004

28

Advanced use of Aleph

- Each step of Aleph's basic algorithm can be altered.
- Let's just look at:
 1. Noise-handling
 2. Evaluation functions
 3. User-defined refinement

ESSLLI - 2004

29

Noise

- In real applications, we have to accept clauses which cover negative examples.
- Use `set(minacc,+Float)` and/or
- `set(noise,+PositiveInt)`
- Default is noise-free learning

ESSLLI - 2004

30

Evaluating clauses

Here's one way to change the clause evaluation function:

```
:- set(evalfn,mestimate).  
:- set(m,1).
```

This uses a Bayesian smoothed measure of accuracy to evaluate clauses.

User-defined refinement

```
:- set(refine,user).  
  
refine(false,eastbound(_)).  
refine(eastbound(X),(eastbound(X):-has_car(X,_))).  
refine(eastbound(X),(eastbound(X):-has_car(X,Y),short(Y))).  
refine((eastbound(X):-has_car(X,Y),short(Y)),Clause):-  
    Clause = (eastbound(X):-has_car(X,Y),short(Y),closed(Y)).
```

The Aleph clause search is non-greedy

- Aleph keeps searching for a clause, *until it is sure it has found a maximally good one*
- *This* is what can make it slow
- It is not because of some inherent slowness of ILP or Prolog (as is commonly thought).
- `set(openlist,0)` effects greedy search

ESSLLI - 2004

33

On background knowledge

- It is crucial (for reasonable size applications) that your background predicates are written in efficient Prolog.
- *Always* exploit first-argument indexing.

ESSLLI - 2004

34

On the hypothesis space

- Reduce your hypothesis space as much as possible
- You can use Prolog to define it precisely
- This is good (i) computationally and (ii) statistically
- Tip: `set(minpos,+V)` is a good way to avoid overfitting and can give big speed-ups.

ESSLLI - 2004

35

Some recent theses

- *Inductive Logic Programming for Part-of-Speech Tagging* by Martin Eineborg, Stockholm, 2002
- *Relational Rule Induction for Natural Language Disambiguation* by Miloslav Nepil, Brno, 2003
- *Acquisition automatique de lexiques sémantiques pour la recherche d'information* by Vincent Claveau, Rennes, 2003

ESSLLI - 2004

36

Nepil (INDEED)

- Disambiguation grammar for Czech
- Correcting annotation errors (7000 found)
- Heavy bias: tree Horn clauses
- Enables very detailed rules to be induced.

ESSLLI - 2004

37

Example rule

```
discard(A) :- % Rule 5: a prep. affected by the next token
focus_token(A,B), spurious_reading(B,C),
det_has_tag(C,tag(k7,_,_,_,_,_,_,_,Case0)), intended_reading(B,D),
det_has_tag(D,tag(k7,_,_,_,_,_,_,_,Case1)), Case0 \== Case1,
right_context(A,E), next_token(E,F,G),
det_has_tag(F,tag(_,_,_,_,_,_,_,_,Case1)),
not_has_tag(F,tag(_,_,_,_,_,_,_,_,Case0)),
not_has_tag(F,tag(k6,_,_,_,_,_,_,_,_)),
not_has_tag(F,tag(k7,_,_,_,_,_,_,_,_)),
not_has_tag(F,tag(k9,_,_,_,_,_,_,_,_)), (
det_has_tag(F,tag(_,_,_,_,_,_,_,_,c2)), next_token(G,H,_),
(det_has_tag(H,tag(kN,_,_,_,_,_,_,_,_)) ;
det_has_tag(H,tag(k4,xC,_,_,_,_,_,_,c4)) ; det_has_lemma(H,"pa'r") ;
det_has_lemma(H,"vi'ce") ; det_has_lemma(H,"sto") ;
```

38

```
det_has_lemma(H,"tisi'c") ; det_has_lemma(H,"milion") ;  
det_has_lemma(H,"miliarda") ; det_has_lemma(H,"pr^ibliz^ne^") ) ->  
Case0 \== c4 ; true ).
```

ESSLLI - 2004

Claveau (ASARES)

- Semantically annotated data
- Identify noun-verb pairs which are semantically linked (qualia pairs)
- Search bias θ_{NV} -subsumption

ESSLLI - 2004

Example rules

```
is_qualia(A,B) :- precedes(B,A), near_verb(A,B),  
                  infinitive(B), action_verb(B).
```

```
is_qualia(A,B) :- precedes(B,A), singular_common_noun(A),  
                  suc(B,C), colon(C), pred(A,D), punctuation(D).
```

Grammar induction using ILP

James Cussens

University of York, UK

(NB joint work with Stephen Pulman, University of Oxford, UK)

1

Overview

- Chomsky from “Aspects of the Theory of Syntax”
- Inductive chart parsing
- Our grammar induction system

Grammar learning from Chomsky's “Aspects of the Theory of Syntax”

Given:

A hypothesis space of possible grammars $G_1, G_2, \dots$

Sentences paired with structural descriptions $(s_1, SD_1), (s_2, SD_2), \dots$

If your current grammar fails to parse (s_i, SD_i) correctly, move to the the next one that does.

ESSLLI - 2004

3

Chomsky on induction

The child who acquires a language in this way of course knows a great deal more that he has “learned”. His knowledge of the language, as this is determined by his internalized grammar, goes far beyond the presented linguistic data and is in no sense an “inductive generalization” from these data.

Aspects of the Theory of Syntax

ESSLLI - 2004

4

Hypothesis formation vs hypothesis selection

- In Chomskyan accounts of learning, there is a distinction drawn between hypothesis formation and hypothesis selection.
- Hypothesis Formation: look at data and somehow construct a hypothesis that fits
- Hypothesis Selection: run through the hypotheses in some order, checking each against the data, until one fits
- Is this a valid dichotomy?

ESSLLI - 2004

5

Chomsky's "Aspects..." framework

- This is a Bayesian algorithm; the ordering of the grammars corresponds to a ranking induced by a prior distribution over possible grammars.
- The hypothesis space $G_1, G_2, \dots$ is not generated from the data—but this is the usual case for machine learning and Bayesian statistics . . .
- . . . which leads to the slogan "generalisation as search"

ESSLLI - 2004

6

A slight digression: Chomsky on statistical models

1. Colorless green ideas sleep furiously
2. Furiously sleep ideas green colorless

... It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical model for grammaticalness, these sentences will be ruled out as equally 'remote' from English. Yet (1), though nonsensical, is grammatical, while (2) is not.

Syntactic Structures

ESSLLI - 2004

7

A riposte from Pereira

A simple *aggregated bigram model*

$$p(w_i, w_{i+1}) = p(w_i) \sum_{c \in C} p(w_{i+1}|c)p(c|w_i)$$

$$p(w_1, \dots, w_n) = p(w_1) \prod_{i=2}^n p(w_i|w_{i-1})$$

Setting $|C| = 16$, and estimating parameters from newspaper text Saul & Pereira got:

$$\frac{p(\text{Colorless green ideas sleep furiously})}{p(\text{Furiously sleep ideas green colorless})} \approx 2 \times 10^5$$

ESSLLI - 2004

8

Recap of Inductive Logic Programming

- Given B , E - both logic programs, find H , a logic program $\in \mathcal{H}$ such that

$$B, H \vdash E$$

- Need to stop H being 'too strong', e.g. $H = \text{false}$
Use negative examples or syntactic constraints
- Need some way of stopping H being 'too weak', e.g. $H = E$
Bias towards simplicity or syntactic constraints

9

Grammar learning as ILP

- Given
 - an existing unification grammar and lexicon (B)
 - unparsable (unannotated) sentences (E)
- Find grammar rules and lexical items (H) such that
 - $B, H \vdash E$ (the sentences get parsed)
 - $H \in \mathcal{H}$ (rules, lexicon are linguistically plausible)
- Have to (i) find possible H s and (ii) evaluate them.

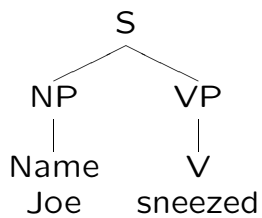
CHART PARSING

0 Joe 1 sneezed 2

A chart of edges:

Id	Constituent	From	To	Containing
c1	Name	0	1	Joe
c2	V	1	2	sneezed
c3	NP	0	1	c1
c4	VP	1	2	c2
c5	S	0	2	c3, c4

The equivalent tree representation would be:



Grammar: $S \rightarrow NP VP$
 $VP \rightarrow V$
 $NP \rightarrow Name$

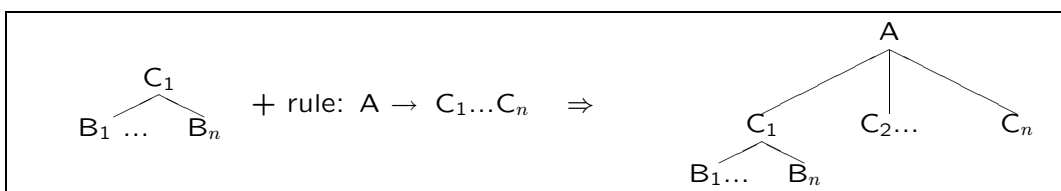
Lexicon: Joe: Name
sneezed: V

11

Propose:

$$\frac{C_1 \text{ from } X \text{ to } Y, \text{ and Rule } A \rightarrow C_1 C_2 \dots C_n}{A \rightarrow C_1 \bullet C_2 \dots C_n \text{ from } X \text{ to } Y}$$

i.e. if there is a constituent C_1 from position X to position Y , and there is a grammar rule which builds an A from a series of constituents beginning with a C_1 , then begin a new constituent, an incomplete A , consisting just of the recognised C_1 , but expecting to find the remainder of the constituents. The notation $A \rightarrow C_1 \bullet C_2 \dots C_n$ is to be read as 'something which will be an A when we have found $C_2 \dots C_n$ '.

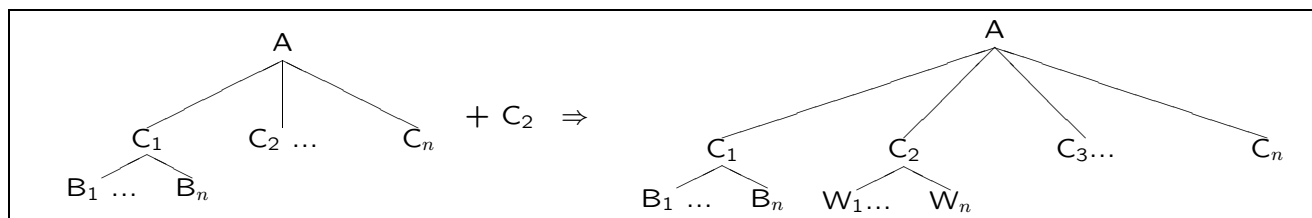


12

The other main rule is 'Combine' which combines an existing incomplete constituent with an existing complete one.

Combine:

$$\frac{A \rightarrow C_1 \dots \bullet C_i \dots C_m \text{ from X to Y, and } C_i \text{ from Y to Z}}{A \rightarrow C_1 \dots C_i \bullet C_{i+1} \dots C_m \text{ from X to Z}}$$

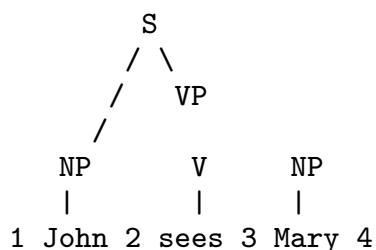


13

INDUCTIVE CHART PARSING

S → NP VP
 VP → V
 V → snores
 V → sees
 NP → John, Mary

This grammar contains a transitive verb but has no rule for the corresponding verb phrase. After trying to parse 'John sees Mary' we will have a chart that contains (among others) the following complete and incomplete constituents, which will give rise to the needs and hypotheses indicated:



Need S from 1 to 4

Need VP from 2 to 4

Got V from 2 to 3

Got NP from 3 to 4

Hypothesise VP → V NP

(Hypothesise S → NP V NP)

14

Our grammar system

- Derived from a system built under the FraCaS project, March 1994.
- Very much work in progress!
- But it illustrates some general ideas nonetheless

ESLLI - 2004

15

Grammar rules and lexical items

```
vp_vp_mod syn vp:[gaps=[A,B],mor=C,aux=n]==>
[vp:[gaps=[A,D],mor=C,aux=n],mod:[gaps=[D,B],of=or(s,vp),type=_]].

have syn v:[mor=or(pl,or(s1,s2)),aux=y,inv=_,
          subc=[vp:[gaps=_,mor=en,aux=_]]].
have syn v:[mor=inf,aux=y,inv=_,subc=[vp:[gaps=_,mor=en,aux=_]]].

'NNP' syn np:[mor=s3].      %for WSJ tagged data
'NNPS' syn np:[mor=pl].
```

ESLLI - 2004

16

The features

mor Value is a boolean combination of morphology properties:
number, person, verb type

type Value is a boolean combination of [n,q,r]: Ordinary noun phrases are n, wh-phrases q, and relative pronouns r

case Values in subj,nonsubj for nounphrases

aux Values in y,n on verbs and verb phrases: is the head verb an auxiliary or not?

ESSLLI - 2004

17

The features (ctd)

inv Values in y,n: is this sentence inverted?

mvt Values in y,n: is there a fronted constituent?

of Values in s,vp,nom: what is this a modifier of?

subc Value is a list of categories, representing the complements to a verb

ESSLLI - 2004

18

The gap feature

The gap feature represents *movement*

1		I did	prepare a talk	quickly.
2	What	did I	prepare	quickly?
		I did	prepare <i>what</i>	quickly?
3	What	did I	prepare a talk	with ?
		I did	prepare a talk with <i>what</i>	?
4	What	did I	prepare a talk	quickly with ?
		I did	prepare a talk	quickly with <i>what</i> ?

1	VP[$\emptyset, \emptyset$]	→	VP[$\emptyset, \emptyset$]	MOD[$\emptyset, \emptyset$]
2	VP[np, $\emptyset$]	→	VP[np, $\emptyset$]	MOD[$\emptyset, \emptyset$]
3	VP[np, $\emptyset$]	→	VP[np, np]	MOD[np, $\emptyset$]
4	VP[np, np]	→	VP[np, np]	MOD[np, np]

ESSLLI - 2004

19

Uncompiled versus compiled grammar rules

```
% vp_vp_mod syn vp: [gaps=[A,B],mor=C,aux=n]==>
%[vp: [gaps=[A,D],mor=C,aux=n],mod: [gaps=[D,B],of=or(s,vp),type=_]].

cmp_synrule(vp_vp_mod, vp([A,B],C,n),
            [vp([A,D],C,n),mod([D,B],f(0,0,_,1),_)]) .
```

ESSLLI - 2004

20

Compiled lexical items

```
cmp_synword(have, v(f(0,0,0,0,_,_,1,1,1),y,_,  
    [vp(_,f(0,1,1,1,1,1,1,1,1),_)])).  
cmp_synword(have, v(f(0,0,1,1,1,1,1,1,1),y,_,  
    [vp(_,f(0,1,1,1,1,1,1,1,1),_)])).  
  
cmp_synword('NNP', np(_,f(0,0,0,0,0,0,0,1,1),_,_)).  
cmp_synword('NNPS', np(_,f(0,0,0,0,1,1,1,1,1),_,_)).
```

ESSLLI - 2004

21

Normal parsing

| ?- test.

...

[a,computer,disappeared].

```
(sigma:dcl):sigma:[]  
s_np_vp:s:[gaps=[ng:[],ng:[]],mor=s3,type=n,inv=n]  
np_det_nom:np:[gaps=[ng:[],ng:[]],mor=s3,type=n,case=subj]  
lex:det:[type=n,mor=s3] a  
lex:nom:[mor=s3] computer  
vp_v:vp:[gaps=[ng:[],ng:[]],mor=s3,aux=n]  
lex:v:[mor=s3,aux=n,inv=n,subc=[]] disappeared  
...
```

ESSLLI - 2004

22

What edges look like

edge(Id,Origin,LeftVertex,RightVertex,Category,Needed,Contents)

Some edges from: *a computer disappeared*

```
edge(1,a,0,1,det(f(0,1,1,1),f(0,0,0,0,0,0,0,1,1)), [], []).
edge(11,np_det_nom,0,1,np([A,A],f(0,0,0,0,0,0,0,1,1),f(0,1,1,1),_),
    [nom(f(0,0,0,0,0,0,0,1,1))], [1]).
edge(2,np_gap,1,1,np([np([ng,ng],A,B,nonsubj),ng],A,B,nonsubj), [], []).
edge(3,mod_gap,1,1,mod([mod([ng,ng],A,_),ng],A,_), [], []).
edge(20,sigma:dcl,0,3,sigma,[], [18]).
```

ESSLLI - 2004

23

Generating naive rules from a failed parse in a unification grammar

₀ All ₁ big ₂ companies ₃ wrote ₄ a ₅ report ₆ quickly ₇

```
%need(Sent,Cat,From,To).
need(1, vp([ng,ng], f(0,0,0,0,1,1,1,1,1),A, 3, 7).

%edge(Sent,Id,Origin,From,To,Cat,...)
edge(1, 39, vp_v_np, 3, 6, vp([X,X],f(0,0,0,0,B,C,D,1,1),n), [],...).
edge(1, 19, quickly, 6, 7, mod([Y,Y],f(0,0,0,1),f(0,1,1,1)), [],...).

cmp_synrule(r0,vp([ng,ng],f(0,0,0,0,1,1,1,1,1),A),
    vp([X,X],f(0,0,0,0,B,C,D,1,1),n),mod([Y,Y],f(0,0,0,1),f(0,1,1,1))).
```

ESSLLI - 2004

24

Being less naive . . .

- The naive approach of matching needed edges to actual edges produces too many rules.
- We therefore effect constraints on the naive rules.
- Only those that meet the constraints get stored.

ESSLLI - 2004

25

Linguistic knowledge: Head-feature constraints

- “prepared” is the head of the verb phrase “prepared a talk quickly”
- “man” is the head of the noun phrase “the man who I saw standing in the bus which was on its way to the street down by the lake which is usually frozen in winter, but was not this year”
- We dictate that a phrase has the same (non-gap) features as its head; so this rule would be rejected:

`vp: [gaps=[ng,ng],mor=pl,aux=n] ==> [vp: [gaps=[ng,ng],mor=inf,aux=n],
mod: [gaps=[ng,ng],of=vp,type=n]]`

ESSLLI - 2004

26

Linguistic knowledge: Gap-threading

The gaps also must be threaded properly ...

ESSLLI - 2004

27

Specialising and filtering with linguistic knowledge

Naive rule:

```
cmp_synrule(r0,  
vp([ng,ng],f(0,0,0,0,1,1,1,1,1),A),  
  vp([X,X],f(0,0,0,0,B,C,D,1,1),n),  
  mod([Y,Y],f(0,0,0,1),f(0,1,1,1))).
```

specialise (with gap threading constraints) to:

```
cmp_synrule(r0,  
vp([ng,ng],f(0,0,0,0,1,1,1,1,1),A),  
  vp([ng,ng],f(0,0,0,0,B,C,D,1,1),n),  
  mod([ng,ng],f(0,0,0,1),f(0,1,1,1))).
```

If we can't specialise to meet constraints then the rule is thrown out.

ESSLLI - 2004

28

Linguistic knowledge: Whatever!

- If you can express your linguistics constraints in Prolog, then it is easy to effect them.
- For example, we have been experimenting with enforcing X-bar schemas on our rules.
- Plenty of WSJ sentences don't satisfy (our) X-bar constraints.

ESSLLI - 2004

29

Least general generalisation

```
%Covers [all,big,companies,wrote,a,report,quickly]
vp:[gaps=[ng,ng],mor=pl,aux=n] ==> [vp:[gaps=[ng,ng],mor=pl,aux=n],
  mod:[gaps=[ng,ng],of=vp,type=n]]

%Covers [what,dont,all,big,companies,read,with,a,machine]
vp:[gaps=[np,ng],mor=inf,aux=n] ==> [vp:[gaps=[np,ng],mor=inf,aux=n],
  mod:[gaps=[ng,ng],of=or(nom,vp),type=n]]
```

lgg is:

```
vp:[gaps=[A,ng:[ ]],mor=or(inf,pl),aux=n] ==>
  [vp:[gaps=[A,ng:[ ]],mor=or(inf,pl),aux=n],
  mod:[gaps=[ng:[ ],ng:[ ]],of=or(nom,vp),type=n]]
```

lgg carried out on **unit** clauses so no blow-up.

ESSLLI - 2004

30

Using the WSJ tagged corpus

17673 sentences like:

```
['Atone'/'VB']. %1  
['Terms'/'NNS', 'were'/'VBD', 'n\'t'/'RB', 'disclosed'/'VBN']. %5  
['Champagne'/'NN', 'and'/'CC', 'dessert'/'NN', 'followed'/'VBD']. %5
```

645 non-sentences lacking a verb previously eliminated.
Punctuation is ignored.

ESLLI - 2004

31

Mapping tags to lexical items

```
% 19939 Verb, gerund or present participle  
'VBG' syn v:[mor=ing,aux=_,inv=n].
```

```
% 27119 Verb, past participle  
'VBN' syn v:[mor=en,aux=y,inv=n].
```

```
% 128 Interjection (no equivalent)  
'UH' syn unk:[].
```

```
% 3156 Wh-pronoun  
'WP' syn np:[gaps=[A,A],mor=or(pl,s3),type=or(r,q),case=_].
```

ESLLI - 2004

32

Before rule induction

- We have an initial grammar that only parses 62 of the 750 tagged WSJ sentences.
- But it also overgenerates, producing parses for *him will every client read* and *kim will every client read*

ESSLLI - 2004

33

Some induced rules

```
r1604 np:[gaps=[A,A],mor=or(pl,s3),type=or(n,q),case=_] ==>
      nom:[mor=or(pl,s3)] Covers: 56 sentences
r5692 vp([A,B],f(0,0,C,D,E,F,G,1,1),H) ==>
      vp([A,B],f(0,0,C,D,E,F,G,1,1),H)
      nom(f(0,0,0,0,I,I,I,1,1)) Covers: 23 sentences:
```

ESSLLI - 2004

34

Over-generalising

Our $\text{np} \Rightarrow [\text{nom}]$ rule allows us to parse:

Dogs bark

School is no fun

but also *Man bites dog*

Is this a sentence? a WSJ sentence?

ESSLLI - 2004

35

Controlling over-generalising

- Grammar learning is apparently hard due to a lack of negative examples; so apparently there's no barrier to over-generation.
- My plan is to evaluate over-generation by the number of extra parses a candidate rule generates (on a some validation corpus).
- $\text{np} \Rightarrow [\text{nom}]$ generates **no** extra parses for my current validation corpus.
- $\text{s} \Rightarrow [\text{np}, \text{vp}]$ rule generated $111 + 306 = 417$ parses

ESSLLI - 2004

36

Current strategy

- Start with the shorter sentences
- Inspect induced rules, and covered/uncovered sentences to track down problems.
- Improve validation corpus. Improve parser
- Think about punctuation
- Code for comparing fracas parse trees with WSJ parse trees

ESSLLI - 2004

37

The grand plan

- The initial motivation for using ILP was to incorporate semantic constraints.
- Current fracas constraints implemented by Prolog hacks (e.g. beta-reduction)
- Plan: replace with cleaner variable-free semantics
- Use type inference to effect semantic constraints.

ESSLLI - 2004

38

A Hybrid Approach to Word Segmentation

Dimitar Kazakov¹ and Suresh Manandhar²

University of York, Heslington, York YO10 5DD, UK,
{kazakov,suresh}@cs.york.ac.uk,
WWW home page: ¹ <http://www.cs.york.ac.uk/mlg/> and
² <http://www.cs.york.ac.uk/~suresh/>

Abstract. This article presents a combination of unsupervised and supervised learning techniques for generation of word segmentation rules from a list of words. First, a bias for word segmentation is introduced and a simple genetic algorithm is used for the search of segmentation that corresponds to the best bias value. In the second phase, the segmentation obtained from the genetic algorithm is used as an input for two inductive logic programming algorithms, namely FOIDL and CLOG. The result is a logic program that can be used for segmentation of unseen words. The learnt program contains affixes which are characteristic for the given language and can be used in other morphology tasks.

1 Introduction

Word segmentation is an important subtask of natural language processing with a range of applications from hyphenation to more detailed morphological analysis and text-to-speech conversion. Several approaches based on learning aiming at word segmentation and morphology have been published recently. Some of them, as in Brill [2] and van den Bosch et al. [15] make use of preprocessed data—a text corpus tagged with the corresponding part of speech in the first case or morphologically analysed words in the second one. At the same time, algorithms for unsupervised learning that use plain text [5] or lists of words are also known. A series of works is based on the notion of *analogy* introducing a bias for word segmentation [13]. This principle has also been used for text-to-speech conversion rules [16].

Inductive logic programming (ILP) [10] has proved to be a feasible way to learn linguistic knowledge in different domains, such as morphological analysis [1], part-of-speech tagging [4] and parsing [8]. Unlike statistical or connectionist approaches, these results are comprehensible and a human expert can easily interpret or modify them. However, ILP remains within the supervised learning paradigm and requires preprocessed (classified) input data. Another possible limitation of ILP approach in the field of linguistics is the very large search space that is to be explored. A typical illustration of the pros and cons of the use of ILP for natural language processing is the learning of the past tense of English verbs [12]. When compared to neural networks and decision trees a more accurate knowledge in the form of decision lists is obtained [12]. Top-down learning algorithms such as FOIDL require theory constants which in our task are

the relevant affixes. These constants are generated by exploring the exhaustive set of possible affixes, which slows down the algorithm and limits the size of the input data the system can handle.

In this work, we describe a hybrid approach combining unsupervised and supervised learning techniques for generation of word segmentation rules from a list of words. A bias for word segmentation introduced in our previous work [9] is reformulated as a fitness function of a simple genetic algorithm, which is used for the search of a word list segmentation that corresponds to the best bias value. In the second phase, the list of segmented words obtained from the genetic algorithm is used as an input for two decision list learning algorithms, namely FOIDL and CLOG [11]. The result is a logic program in a decision list representation that can be used for segmentation of unseen words. This program also contains a number of constants, which are characteristic for the given language and can be used in other learning tasks related to word morphology.

2 Prerequisites

2.1 Genetic Algorithms

Genetic algorithms (GA) [7] are often used as an alternative approach to tasks with a large search space where greedy search methods would often get stuck in local maxima. Genetic algorithms are inspired from Darwinian evolution. A GA maintains a set of candidate solutions (called *chromosomes* or *individuals*) and applies the natural selection operators of *crossover* and *mutation* to successively generate new candidate solutions from existing ones. A *fitness function* is employed to rank the individuals to determine their goodness. The chromosomes are represented as a sequence of characters (or a bit vector). The *crossover* operation constructs two new child chromosomes by splicing two parent chromosomes at n points. The *mutation* operator creates a new chromosome from a single parent by randomly changing a single character (or flipping a bit) of the parent chromosome. Chromosomes are mutated according to some mutation probability known as *mutation rate*. The following algorithm is known as a *simple genetic algorithm*:

1. Initialization
 - (a) Create a random population of candidate solutions (*individuals*) of *popsize* size.
 - (b) Evaluate all individuals using the fitness function.
 - (c) Store the best evaluated individual as *best-ever* individual.
2. Generation and Selection
 - (a) Sample the individuals according to their fitness, so that in the resulting *matching pool* those with higher fitness appear repeatedly with a higher probability.
 - (b) Apply crossover with probability *crossover_rate*.
 - (c) Apply mutation with probability *mutation_rate*.
 - (d) Evaluate all individuals using the fitness function.
 - (e) Update the best-ever individual.
3. If the stopping condition is satisfied, provide the *best-ever* individual as a solution. Otherwise go to step 2.

There are two characteristics that are typical for all GA: 1. they are indeterminate and 2. they provide a solution, which may be not optimal.

2.2 Decision list learning in FOIDL

FOIDL [12] is a system for learning first-order decision lists from examples and is closely related to FOIL[14]. One of the main reasons that a system such as FOIDL is a good candidate for NLP applications is that it can be used to learn from positive examples only. FOIDL uses an assumption known as *output completeness* to generate implicit negative examples from positive data. The notion of output completeness can be best explained by an example. Let the database (in Prolog form) consist of the following positive examples: `past(sleep,slept)`, `past(like,liked)`, `past(walk,walked)`.

Assuming that this database is queried using the mode `past(+,-)` the query corresponding to the example `past(sleep,slept)` will be `past(sleep,X)`. Then the implicitly assumed negative examples will consist of all instantiations of `X` such that `past(sleep,X)` is *false* in the database.

Such a closed-world assumption is quite reasonable in the morphology tasks and provides a method for inducing rules from positive only examples. FOIDL learns a given target predicate from a set of examples by following a greedy top-down hill-climbing strategy by starting from the most general hypothesis and by successively refining the hypothesis by adding body literals which match the user supplied mode declarations.

2.3 Decision list learning in CLOG

CLOG is another system for learning of first-order decision lists. CLOG shares a fair amount of similarity with FOIDL [3]. Like FOIDL, CLOG can learn first-order decision lists from positive examples only using the *output completeness* assumption. The main difference is that CLOG only considers generalisations that are relevant to an example. In the current implementation these generalisations are supplied by a user-defined predicate which takes as input an example and generates a list of all generalisations that cover that example. The reader is referred to [11] for more detailed discussion of CLOG.

For the task of learning segmentation rules described in this paper we coded our own clause generator that took as input an example and generated as output all clauses whose body contained at most two `append/3` clauses and covered the example. The generated clauses included all possible variable binding patterns that were consistent with the mode declarations. CLOG treats the set of generalisations of an example as a *generalisation set*. It then cycles every input example through the generalisation set in a single iteration checking whether a candidate generalisation covers the example positively or negatively. Once this process is complete the “best” candidate generalisation is chosen. The example set is pruned using this candidate and the cycle repeats.

The *gain* function currently used in CLOG is user-defined. For the segmentation problem we chose the following simple gain function: $gain = QP - SN - C$

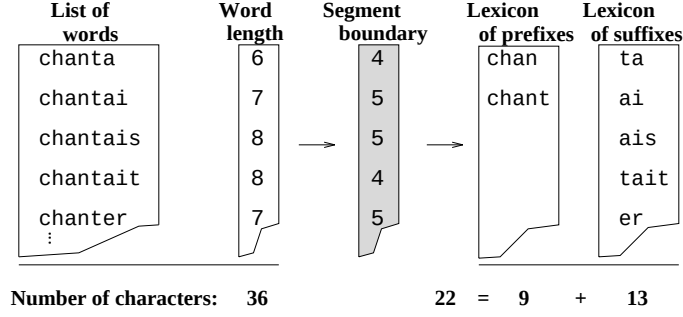


Fig. 1. A naïve theory of word morphology

where QP denotes the number of new examples covered positively, SN denotes the number of previously covered examples that are covered negatively and C is the number of literals in the clause body.

3 Word Segmentation Bias

The segmentation model accepted here limits the number of word segments up to two. These segments will be called further ‘prefixes’, resp. ‘suffixes’ where the terms correspond to their etymological meaning rather than to the one used by linguists. In other words, any left/right substring of the word characters is seen as a prefix/suffix. There are two reasons for this assumption. For practical purposes, such as on-line morpho-lexical analysis, a higher number of word constituents either produces a lot of spurious analyses or requires the use of some additional constraints, which slows down the algorithm and increases the cost of the system development. On the other hand, some of the limitations of the assumption that we make can be overcome by the repeated use of our approach.

For a given set of words the segment boundary positions correspond to a vector of integers, each of them between zero and the word length. The word set along with an arbitrary vector of that kind introduces a *naïve theory of morphology* [9]. Every such theory defines two lexicons where prefixes, resp. suffixes are enumerated without repetition (see fig. 1). The quality of the theory will be estimated by the number of characters $N = P + S$ in the prefix lexicon (P) and the suffix one (S)—the smaller that number, the better the theory. The upper bound N_{max} of that measure is given by the number of characters W in the word list. This case is reached when no prefix nor suffix has been generated twice by the theory. Now, the word segmentation bias can be formulated.

Word Segmentation Bias: *among a set of naïve theories of word morphology, select the one with the lowest number of characters in the corresponding pair of lexicons.*

The Naïve Theory of Morphology (NTM) bias is based on the hypothesis that substrings composed out of real morphemes occur in the words with a frequency higher than any other left or right substrings.¹ In that way, a theory with a low

¹ This presumption is limited to the languages in which the main operator used in derivational and inflectional morphological rules is **concatenation**.

N would produce lexicons where ‘prefixes’ and ‘suffixes’ correspond very often to single morphemes or their combinations. Since the word list can be stored as a list of pairs of indices <prefix, suffix> along with the two lexicons of affixes, the bias described can be seen as using Occam’s razor to choose the simplest theory corresponding to the given dataset.

4 GA Representation of the Learning Task

The approach is straight-forward, making use of a simple genetic algorithm, where the vector of integers corresponding to the theory is directly used as an individual (chromosome) and its fitness function is $N_{max} - N$. The fitness function defined in that way does not take negative values and the GA searches for individuals with maximal fitness. In the context of individuals with non-binary features, mutation is defined either as a shift of the morpheme boundary with one position to the left or right, or, as a random choice of a new boundary position from the allowed interval. The mutation rate is set by the user and is increased by 20% after 10 generations with the same best-ever individual. The higher mutation rate is kept until a better individual is produced and then changed back to its original value. A 1-point crossover is applied. The fittest individual is kept through the generations and is finally given as a result of the learning after a certain number of generations.

5 Results of Unsupervised Learning

A list containing many forms of the same lexical entry or its entire paradigm is suitable for the learning since the difference between N_{max} and N of the best solution is very great. A possible way to obtain such a word list while remaining within the framework of unsupervised learning is to sort the words of a large corpus and remove all duplicates. With the increasing corpus size, the word list converges on an exhaustive lexicon of word forms. A chunk of 700 different word forms from a very large alphabetically ordered French lexicon of word forms was actually used for our experiments.

ébarb+e	ébatt+irent	ébouillant+erez	ébourgeonn+ée
ébarb+ent	ébatt+issent	ébouillant+eriez	ébourgeonn+és
ébarb+erai	ébatt+ra	ébouillant+erions	éb+ouriffais
ébarb+erez	éborgn+es	ébouillant+ez	ébouriff+ait
ébarb+ez	éborgn+ons	ébouillant+ions	ébouriff+asse
ébarb+iez	éborgn+ées	ébouillant+ât	ébouriff+ent
ébarb+ons	ébouillant+assent	éboul+ait	ébouriff+er

Table 1. GA output sample

Two experiments were carried out. In the first, the list of words was randomly sampled and divided into a training set of 200 words and a test set containing

500 words. The genetic algorithm was applied on the training set for 1500 generations.² A sample of the NTM obtained is shown in table 1. For the second experiment, the list of words was divided into a training set of 500 words and a 200-word test set. The training set was randomly split into 5 disjunctive sets of 100 words and the genetic algorithm was separately run on each of them. Then the NTM found were merged and evaluated in the same way as described above. This technique proved to be a feasible trade-off between the input data size and the time needed to find an NTM of high quality. It also makes the GA time complexity linear w.r.t. the input size.

A quantitative description of the first experiment is given in table 2. With regard to the different theoretical approaches [6], it is difficult to give an exact linguistic evaluation of the acquired results. Nevertheless, the quality of the results can be judged by two criteria, namely, whether the segments produced have a linguistic interpretation and whether that interpretation is consistent for the whole set of words. The training data contains a large percentage of inflected

	Experiment 1		Experiment 2	
	Items	Characters	Items	Characters
GA input	200	1943	500	4855
Generated prefixes	43	259	75	454
Generated suffixes	64	252	131	638
$(P + S)/W$	0.535	0.263	0.412	0.225

Table 2. Results of GA segmentation

verb forms, which is typical for the Roman languages. The fragment of the GA output in table 2 displays some of them. The verb stems (*ébarb-*, *ébatt-*, *éborgn-*, *ébouillant-*, *éboul-*, *ébourgeonn-* and *ébourriff-*) have been reliably identified most of the time, therefore many endings corresponding to these verb paradigms have also been enumerated.³ The proposed segments do not have equal value. The results confirmed that, in general, the more frequently an affix appears in the derived theory, the higher is the probability that the affix is really a morpheme or a concatenation of morphemes.

6 Induction of Procedural Rules for Segmentation

Regardless of the quality of the GA results, their reusability is limited by two factors: 1. when several affixes are present in the processed word, there is no way to choose among the possible segmentations, and 2. the NTM derived by GA is most probably not optimal, i.e., some of the segmentations can be seen as “wrong” w.r.t. the given bias. The first problem can be addressed by the

² For *popsize* = 1400 the algorithm ran for about 8 hours on Pentium 200Mhz PC.

³ The affixes generated are saved along with their frequency in separate files.

application of some simple statistical measure, so that, for instance, for a given word and prefix/suffix lexicons containing the relative frequency of the items, the segmentation which maximizes the probability $P_{seg} = Freq_p * Freq_s / W^2$ is selected. However, this criterion needs the prefix *and* the suffix of the best segmentation to be present in the lexicons.

6.1 Setting

The output of GA is represented as clauses of the predicate `seg(W,P,S)`, for instance: `seg([â,n,o,n,n,e,r,a,s],[â,n,o,n,n],[e,r,a,s])`. Then `seg/3` is used as a target predicate with mode `seg(+,?,?)` of two inductive learning algorithms: FOIDL and CLOG. In both cases, the predicate `append(P,S,W)` is used as intentional background knowledge and the range of the theory constants is limited to the set of prefixes and suffixes that appear in `seg/3` predicate. As a matter of fact, either of the last two arguments of `seg/3` is redundant and one of them was omitted in the real input dataset. The preliminary tests showed that `seg(Word,Prefix)` and `seg(Word,Suffix)` settings resulted in almost equal theories and only the latter was used further on.

6.2 Results

The result of ILP learning is an ordered list of rules (logic clauses) preceded by a list of exceptions represented as ground facts, e.g. `seg([â,m,e],[m,e])`. The latter do not have any impact on the segmentation of unseen words. If these exceptions correspond to individual word segmentations from the GA output which are not optimal w.r.t. the NTM bias, the application of the rules on them should result in more uniform (and compact) segmentation than the one produced by the GA. The exceptions that are not covered by any rule can be seen as segmented at their very end or beginning. They can be added to one of the two lexicons and the ratio $(P+S)/W$ [chars] can be computed for the whole dataset in the same way as for the GA evaluation. The results of this experiment⁴ are

	Algorithm	Elapsed time	Rules	Test data		Training data	
				Coverage [items]	$(P+S)/W$ [chars]	Coverage [items]	$(P+S)/W$ [chars]
Exp. 1 200 words	FOIDL	15'	29	85%	0.21	92.0%	0.257
	CLOG	1' 35"	40	89%	0.30	95.5%	0.276
Exp. 2 500 words	FOIDL	4h 28'	50	98.0%	0.390	98.2%	0.227
	CLOG	13' 31"	71	95.5%	0.410	97.2%	0.170

Table 3. Results of ILP learning

⁴ The tests had to be run on different hardware—FOIDL on Sun Ultra Sparc 256MB RAM and CLOG on Sun 4 128MB RAM

shown in the “training data” column of table 3. Indeed, for the 500 words the $83.0\% = (1 - 0.170) * 100\%$ compression reached by the CLOG “rule-only” theory on the training data is better than the $77.5\% = (1 - 0.225) * 100\%$ of GA (see table 2). The difference between the 77.3% compression rate of the FOIDL theory and the 77.5% GA performance approximately corresponds to the worse score of the 9 exceptions not covered by any rule. The remaining 87 exceptions have been segmented in a different way by the rules and the GA, yet the compression rate has not changed. That means that many regular patterns have not been learnt from the GA output but treated as exceptions and some of them have probably been segmented by the rules in a worse manner.

In the second evaluation step, the rules induced were used to generate the segmentation of unseen words (the test dataset) and the results were evaluated as above (see the “test data” column of table 3). The 500-word test dataset for experiment 2 contains 2.5 times more words than the test dataset for experiment 1 when the affixes present in the data remain almost the same. That results in lower compression rates for the second experiment as compared to the first one. The rules derived correspond to four basic patterns:

1. $\text{seg}(A, B) :- \text{append}([\acute{e}, c, a, r, t, e, l], B, A), !.$
2. $\text{seg}(A, [e, r, e, z]) :- \text{append}(_, [e, r, e, z], A), !.$
3. $\text{seg}(A, B) :- \text{append}([\acute{e}, b, l, o, u, i, s, s], B, A),$
4. $\text{seg}(A, B) :- \text{append}([\acute{e}, b, l, o, u, i], B, A),$
 $\text{append}(C, [o, n, s], A), !.$
 $\text{append}([\acute{e}, b, l, o, u, i, s, s], C, A), !.$

Rules of the first type look for a particular prefix. For the given data, most of the prefixes correspond to a word stem. Such rules have a limited scope given by the number of word forms that can be derived from the same stem. The rules of the second type are based on suffixes and are essential for languages such as French and English, in which a closed set of derivational or inflexional suffixes is used. As a result, for any representative dataset the rules based on such suffixes should cover a stable percentage of examples, i.e., the absolute number of words covered by them should increase approximately linearly with the lexicon size.

The last two rule patterns can be a result of the insufficient number of examples or the noise in the data where learning from a larger or less noisy dataset would result in rules of the first two types. However, these rules are of real interest when they identify substrings (morphemes) that have not been or cannot be derived by the GA within the given setting. For instance, the example shown for the third pattern covers two words: *éblouissements* and *éblouissions* where the alternation $-\emptyset/-i-$ is typical for some verb forms and corresponds to the opposition between the indicative present tense and other verb tenses. Similarly, in the fourth example the pattern $\langle \text{stem} \rangle - / \langle \text{stem} \rangle - ss-$ is characteristic for the French verb conjugation.

The theory that CLOG learnt in the second experiment contains 32 rules based on suffixes (as compared to 3 learnt by FOIDL). Thirty-one of these endings can be found in any standard grammar book and the last one ($-r$) still has a clear linguistic interpretation. This theory was used to segment an unseen example of the “-er” verb paradigm covering the large majority of French verbs. Out of 48 word forms, 37 have been consistently split into the verb stem *aim-* (not present in the theory) and the corresponding ending. In five more cases two suffixes corresponding to the imperfect tense of subjunctive *-ass-* or future *-er-*

have been shown as a part of the stem. Only four different word forms have not been covered at all. The example follows where the endings produced are in bold face.

aime	aimais	aimai	aimerai	aime	aimasse	aimerais	aimé
aimes	aimais	aimas	aimeras	aimes	aimasses	aimerais	aimés
aime	aimait	aima	aimera	aime	aimât	aimerait	aimée
aimons	aimions	aimâmes	aimerons	aimions	aimassions	aimerions	aimées
aimiez	aimiez	aimâtes	aimerez	aimiez	aimassiez	aimeriez	aimant
aiment	aimaient	aimèrent	aimeront	aiment	aimassent	aimeraient	aimer

7 Conclusions

The hybrid approach to word segmentation described in this paper is an efficient combination of GA and ILP. The former provides a first approximation of the concept learnt and reduce dramatically the search space. The latter, on its turn, refines the target concept reformulating it as intentional rules, so that no other words are necessary to provide the segmentation context for later use. The results of the tests show that a set of rules for word segmentation can be learnt from a *limited* amount of *unannotated* words, as opposed to other approaches where very large and/or tagged corpora are used.

Comparison to other learning strategies shows that some of them [16] use small subsets of data to derive “microtheories” that are independent from the rest of the data. For instance, in the cited work [16], $P_1 + S_1$ is a legal word segmentation if the words $P_1 + S_2$, $P_2 + S_1$ and $P_2 + S_2$ can be found in the lexicon.⁵ On the other hand, the best segmentation choice can be based on some integral criterion, such as the constituent relative frequency [5]. The NTM bias chooses a single segmentation that is related to the segmentation of most of the other words. In the limit, when only words with the same prefix (or suffix) are provided, the NTM bias, unlike the analogy principle [13], [16] is still able to produce the segmentation. Also, NTM does not need to impose a limitation on the length of the affixes as other approaches [2], [5] do.

Possible limitations of the approach are the computational complexity of the NTM search and the restricted number of constituents derived in each learning step. The time efficiency that CLOG provided is a good premise for experimentation with larger datasets. In our future research, we intend to modify the NTM bias so that it will take into account the amount of information needed to reconstruct the dataset from the NTM. We also intend to explore the possibility to use our approach as a preprocessing phase of other morphology learning tasks in order to provide them with a feasible set of theory constants.

For instance, the Brill part-of-speech tagger [2] derives part-of-speech prediction rules for unknown words from tagged corpora (typically of size 10^5). If only the affixes derived by the GA+ILP approach are used during the search of rules instead of the blind use of all affixes up to a given size present in the words, this should result in a manyfold speed-up.

⁵ There might be several such segmentations.

8 Acknowledgements

The authors wish to thank Stephen Muggleton, James Cussens and the two anonymous reviewers for their thoughtful comments. The first author also wishes to thank Olga Štěpánková for a number of improvements suggested. He has been supported by ESPRIT Long Term Research Action *No 20237* on Inductive Logic Programming II and part of this research was carried out by him in CTU Prague.

References

1. H. Blockeel. Application of inductive logic programming to natural language processing. Master's thesis, Katholieke Universiteit Leuven, 1994.
2. E. Brill. Some advances in transformation-based part of speech tagging. In *Proceedings of AAAI-94*, pages 748–753. AAAI Press/MIT Press, 1994.
3. Mary Elaine Califf and Raymond J. Mooney. Advantages of decision lists and implicit negatives in inductive logic programming. Technical report, University of Texas at Austin, 1996.
4. James Cussens. Part-of-speech tagging using Progol. In *Inductive Logic Programming: Proceedings of the 7th International Workshop (ILP-97)*, pages 93–108, 1997.
5. Sabine Deligne. *Modèles de séquences de longueurs variables: Application au traitement du langage écrit et de la parole*. PhD thesis, ENST Paris, France, 1996.
6. Bernard Fradin. L'approche à deux niveaux en morphologie computationnelle et les développements récents de la morphologie. *Traitement automatique des langues*, 35(2):9–48, 1994.
7. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
8. Dimitar Kazakov. An inductive approach to natural language parser design. In Kemal Oflazer and Harold Somers, editors, *Proceedings of Nemlap-2*, pages 209–217, Ankara, Turkey, September 1996. Bilkent University.
9. Dimitar Kazakov. Unsupervised learning of naïve morphology with genetic algorithms. In W. Daelemans, A. van den Bosch, and A. Weijters, editors, *Workshop Notes of the ECML/MLnet Workshop on Empirical Learning of Natural Language Processing Tasks*, pages 105–112, Prague, April 1997.
10. Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming Techniques and Applications*. Ellis Horwood Ltd., Campus 400, Maylands Avenue, Hemel Hempstead, Hertfordshire, HP2 7EZ, England, 1994.
11. Suresh Manandhar, Sašo Džeroski, and Tomaž Erjavec. Learning Multilingual Morphology with CLOG. In *The Eighth International Conference on Inductive Logic Programming (ILP'98)*, Madison, Wisconsin, USA, 1998.
12. Raymond J. Mooney and Mary Elaine Califf. Induction of first-order decision lists: Results on learning the past tense of English verbs. *JAIR*, June 1995.
13. Vito Pirelli. *Morphology, Analogy and Machine Translation*. PhD thesis, Salford University, UK, 1993.
14. J.R. Quinlan. Learning logical definitions from relations. *ML*, 5:239–266, 1990.
15. Antal van den Bosch, Walter Daelemans, and Ton Weijters. Morphological analysis as classification: an inductive learning approach. In Kemal Oflazer and Harold Somers, editors, *Proceedings of Nemlap-2*, pages 79–89, Ankara, Sep. 1996.
16. François Yvon. *Prononcer par analogie: motivations, formalisations et évaluations*. PhD thesis, ENST Paris, France, 1996.

Achievements and Prospects of Learning Word Morphology with Inductive Logic Programming

Dimitar Kazakov

University of York, Heslington, York YO10 5DD, UK,

kazakov@cs.york.ac.uk,

WWW home page: <http://www.cs.york.ac.uk/~kazakov/>

Abstract. This article presents an overview of existing ILP and non-ILP approaches to word morphology learning, and sets targets for future research. The article claims that new challenges to the ILP community with more appeal to computational linguists should be sought in a whole new range of unexplored learning tasks in which ILP would have to make a more extensive use of relevant linguistic knowledge, and be more closely integrated with other learning techniques for data preprocessing.

1 Introduction

Matthews (1974) defines morphology as “that branch of linguistics which is concerned with the ‘forms of words’ in different uses and constructions”. According to another definition by the same author, morphology is “the study of the grammatical structure of words and the categories realized by them” (Matthews, 1997). In order to compare words and study their differences, words are often *segmented* into a number of constituents. *Word segmentation* — here the expression is used as related to word morphology, and not to *tokenisation*, where lexical constituents are indentified in the text — is an important subtask of Natural Language Processing (NLP) with a range of applications from hyphenation to more detailed morphological analysis and text-to-speech conversion.

Several approaches aiming to learn word morphology have been published recently. In many of them, annotated data is required. For instance, Brill (1994) uses a text corpus tagged with the corresponding part of speech (PoS) to learn morphological rules for the prediction of the PoS of unknown words; van den Bosch, Daelemans, and Weijters (1996) make use of morphologically analysed words to derive a text-to-speech conversion theory in an inductive fashion. There are also algorithms for *unsupervised* learning of word morphology that use plain text (Deligne, 1996) or lists of words (Harris, 1955), (Pirelli, 1993), (Yvon, 1996b) as training data.

Inductive Logic Programming (ILP) aims at the inductive learning of concepts from examples and background knowledge in a first-order logic framework (Lavrač & Džeroski, 1994). ILP has proved to be a feasible way to learn linguistic knowledge in different domains, such as morphological analysis (Blockeel, 1994), (Mooney & Califf, 1995), part-of-speech tagging (Cussens, 1997) and parsing (Zelle & Mooney, 1993), (Kazakov, 1999). Unlike statistical or connectionist

approaches, these results can be easily understood and modified by a human expert.

This article presents an overview of existing applications of ILP to word morphology learning, and sets targets for future research. The article claims that new challenges for the ILP community, and a higher interest from computational linguists should be sought in a whole new range of unexplored learning tasks in which ILP will be more closely integrated with other learning techniques and relevant linguistic knowledge. The exposition starts with an overview of basic concepts and approaches in word morphology and ILP. It then proceeds to describe two of the relevant aspects in machine learning of language:

- training data annotation and the possibilities of its augmentation through learning
- background concepts used by the learner.

The above can be seen as two orthogonal dimensions defining a space of possible learning settings according to the *object language* in which the training data is described, and the *concept language* used to express the theories learned. Each learning task can then be represented as a move along either or both axes, for instance, adding word segmentation to the existing annotation of training data or learning morphology rule templates with the help of some basic linguistic concepts, such as vowels and consonants. This article describes a number of biases for pre-processing of lexical data, and suggests ways of combining the so obtained additional information with linguistic concepts of various complexity for the ILP learning of word morphology tools and theories.

2 Word Morphology

It is usually assumed that a word consists of a number of constituents which cover the entire word, do not overlap and follow each other. The last condition is generally valid for the Indo-European languages. In Arabic, however, a more complex operator than concatenation is used to combine morphemes (Matthews, 1974, p. 131).

2.1 Morphemes

The rôle of the word constituents has varied between the different linguistic schools and with the development of linguistics. The word constituents can be perceived merely as differences allowing us to distinguish between two words with no particular meaning assigned to them (de Saussure, 1916). However, in what is now considered to be the classical approach in morphology, word constituents are paid somewhat more attention. Three basic assumptions are often made, which concern word constituents, the words and their phonological/orthographic representation¹ (Fradin, 1994):

¹ The way in which the words are actually pronounced or written.

1. There are minimal word constituents, *morphemes*, with which morphology operates.
2. The word is made up of morphemes which follow each other.
3. There is a set of rules that produce the actual pronunciation/spelling of each morpheme according to its context.

For instance, it can be said that the word *truthful* is made of two morphemes, *truth* and *ful*. Also, one might want to say that the word *studied* consists of the morphemes *study* and *ed* where a special rule handles the change of the final ‘y’ in *study* into ‘i’ in the context of *-ed*.

So, there are two different meanings of ‘morpheme’:

1. Any sequence of characters that has either grammatical or lexical meaning
2. An invariant lexical or grammatical unit realised by one or more sequences of characters.

In the above definitions, based on Matthews (1997), ‘sequence of characters’ was substituted for ‘configuration of phonological units’, since a lot of NLP research aims at segmentation and morphological analysis of text, and in many languages the pronunciation is not directly reflected in the spelling. That change in terminology may raise the objection that it would make possible to split a single phonological unit represented by two or more characters in the text. However, the requirement that the sequence of characters have grammatical or lexical meaning eliminates those cases.

If ‘morpheme’ is used in its second sense, it represents a certain kind of abstraction over a number of variants, called *morphs*. One can imagine a word represented at two levels. If the *lexical level* contains the morphemes included in the word, the *surface level* shows the actual word as formed by the concatenation of the corresponding morphs. For instance, one can speak of the pairs of morphemes DOG + ‘PLURAL MORPHEME’, BOSS + ‘PLURAL MORPHEME’, LEAF + ‘PLURAL MORPHEME’, ‘NEGATIVE MORPHEME’ + DO which at the surface level produce *dogs*, *bosses*, *leaves*, *undo*, respectively.

A distinction is usually made between *derivational* and *inflectional* morphology. The latter studies how all the forms of each word in the lexicon are produced, e.g. *walk*, *walks*, *walked* given the lexical entry *to walk*, VERB. All these word-forms share the same part of speech and meaning, and it is only the morphosyntactic features, such as Tense or Person, that change. On the other hand, derivational morphology describes transformations changing the word meaning. In this case, the PoS of the initial and resulting words may differ. For instance, adding the suffix *-able* to a verb can produce an adjective confirming the applicability of the action denoted by the verb.

2.2 Morphology Representation

From a practical point of view, it is important to represent a large number of word-forms and the corresponding set of categories in a more concise way than by exhaustive enumeration. Such a compact theory of word-formation has

also a generative power, enabling prediction of new word-forms from the known ones and analysis of unseen words. A certain level of complexity is required if the theory is to reflect the subtleties of the described language; however, a detailed description of language morphology can be costly and, in practice, the complexity of the model used is often chosen to effect a trade-off between its cost and performance.

Since concatenation is the basic operator to arrange morphemes into words in the majority of European languages, simple and efficient representations of word paradigms can be obtained as a combination of an inflectionally invariable stem and a set of endings. The use of several stems corresponding to different parts of the paradigm can simplify the representation, decreasing the number of endings used. The paradigms of several words (lexemes) can be further merged into larger clusters called *inflectional classes*.

The choice of representation also varies according to the purpose. For instance, one may be interested in modelling the idealised native speaker-hearer. Also, one may wish “*to assist the second language learner by constructing an optimal pedagogical grammar, to plumb the depths of what there is to be learned about the history of the language through the method of internal reconstruction, or to predict future directions apt to occur in a language*” (Bender, 1997). For instance, when a second language is studied, practical reasons are put forward, or, in Matthews’s words (Matthews, 1974):

Any language is learned by a mixture of rote-learning, rules and practice and it is the job of the language teacher to work out what combination is the most efficient.

Indeed, it is often simpler for the human learner and the NLP developer alike to cover some of the cases as exceptions by simply memorising them, rather than to use a complex set of rules needed otherwise.

2.3 Generative Phonology and Two-Level Morphology

Several approaches to morphology use the notion of abstract, lexical-level, morphemes. In these formalisms, words are formed by firstly combining morphemes at lexical level, and then applying rules to derive the word surface-level representation. In the framework of generative phonology (Chomsky & Halle, 1968), the initial, lexical-level sequence of morphemes is modified by sequential application of rewriting [generative] rules, and a word can go through several intermediate stages before it takes its final, surface-level form. Some of the criticism of this approach is related to the procedural character of the rules, as they have to be applied in a certain order, and operate only in an underlying-to-surface direction.

The following example, borrowed from Antworth (1991), illustrates the principles of generative phonology. Consider the rules:

Rule 1 (Vowel Raising) *e* becomes (is rewritten as) *i* if it is followed by a zero or more consonants, followed by *i*.

Rule 2 (Palatalisation) *t* becomes *c* preceding *i*.

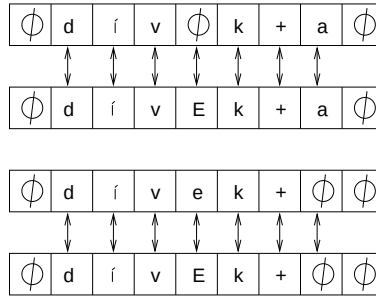


Fig. 1. Lexical vs surface level in two-level morphology

A sample derivation of forms to which these rules apply follows.

Lexical level: temi
 Rule 1: timi
 Rule 2: cimi
 Surface level: cimi

Koskeniemi (1983) introduces the Two-Level Morphology approach. The approach is based on two lexicons containing morphemes (roots and affixes), and a set of morphological rules. The rules establish whether a given sequence of characters at the surface level (as it appears in the text) can correspond to another given sequence of symbols used to represent the morphemes in the lexicon. The morphological rules are implemented as finite state transducers (FST). FST are automata with two bands, i.e. they are based on alphabets composed by pairs of symbols.

For instance, the Nominative Singular of the Czech word *dívka* (girl) can be represented at lexical level as a combination of the morphemes $DÍV\emptyset K+A$. The corresponding string at surface level is $DÍV\emptyset K+A$ (Figure 1). Pairs of corresponding symbols at the two levels have to be aligned, possibly making use of nil ($\emptyset$) symbols. One possible rule may postulate that the symbol E corresponds to the nil symbol $\emptyset$ if the right-hand side context of E is KA. There could be another rule matching E to E if the right-hand side context of E is $K\emptyset$.

From a theoretical point of view, there are several aspects which make the two-level approach interesting. It makes use of morphemes close to the surface level, and it does not need intermediary descriptions which cannot be directly related to empirical observations. Furthermore, this approach is defended by its supporters as declarative, allowing to separate the conditions governing a linguistic phenomenon from their application (Fradin, 1994). Also, two-level rules are bi-directional, i.e. they can operate in an underlying to surface direction (generation mode) or in a surface to underlying direction (recognition mode) (Antworth, 1991). The declarative and bi-directional character of two-level approach makes its logic programming implementation suitable and easy. Another related approach worth mentioning is Kaplan and Kay's multilevel morphology (Kaplan & Kay, 1994).

3 ILP Tools

This section describes the ILP tools which have been applied to morphology learning so far.

3.1 Decision List Learners

A *first-order decision list* is a logic program in which the last literal in all clauses but the last one is a cut (!). For example, in Figure 2, the predicate `number/2` predicts the morpho-syntactic category `NUMBER` of English nouns according to their ending. The order of clauses does matter, and when the predicate is called, only the first of all applicable clauses is fired. The figure shows a decision list

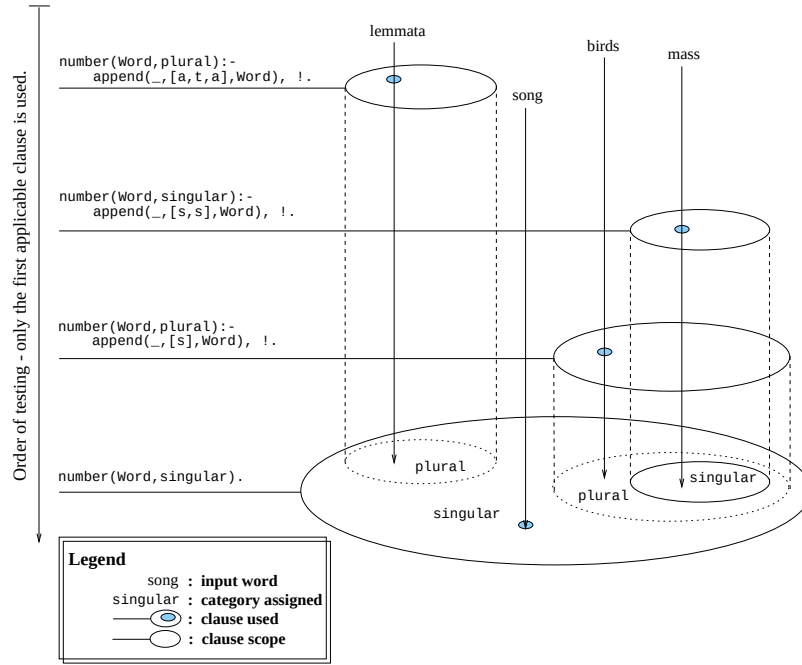


Fig. 2. Scope and priority of decision list clauses

of four clauses and how it is applied on four English nouns. The noun *lemmata* fires the first clause in the list, and is classified as being a plural word-form. The noun *mass* is covered by the second, third, and fourth clauses; however, the clause order guarantees that only the first of the three is used. The scope of each clause is schematically represented with an ellipse; these are projected on a plane to allow for their comparison. The last clause represents the default case, i.e. covers all examples which are not subsumed by any of the preceding clauses.

FOIDL

FOIDL (Mooney & Califf, 1995) is a greedy top-down ILP system for learning first-order decision lists from examples and is closely related to FOIL (Quinlan, 1990).

One of the main reasons that a system such as FOIDL is a good candidate for NLP applications is that it can be used to learn from positive examples only. FOIDL uses an assumption known as *output completeness* to generate implicit negative examples from positive data. The notion of output completeness can be best explained by an example. Let the positive examples be:

`past(sleep,slept), past(like,liked), past(walk,walked).`

Let the target predicate `past/2` be queried using the mode `past(+,-)`. This mode describes that the first argument is an input variable, i.e. it must be instantiated in any query, and the second one is an output variable, i.e. one that is to be instantiated by the clause being learned. The only query about that clause that is allowed by the mode declaration is of the type `past(sleep,X)`. Any instantiation of the output variables that does not exactly match a positive example is perceived as a negative one. Such implicit negative examples for the examples shown are `past(sleep,X), past(sleep,liked), past(sleep,walked).`

CLOG

CLOG (Manandhar, Džeroski, & Erjavec, 1998) is another system for learning first-order decision lists. It shares a fair amount of similarity with FOIDL (Califf & Mooney, 1996). In both FOIDL and CLOG, the decision list is learned from the bottom upwards, i.e. the most general clauses are learned first. Like FOIDL, CLOG can learn first-order decision lists from positive examples only using the output completeness assumption. In the current implementation, the generalisations relevant to an example are supplied by a user-defined predicate which takes as input an example and generates a list of all generalisations that cover that example. CLOG treats the set of generalisations of an example as a *generalisation set*. It then cycles every input example through the generalisation set in a single iteration checking whether a candidate generalisation covers the example positively or negatively. Once this process is complete the best candidate generalisation w.r.t. a user-defined *gain* function is chosen.

A comparison between FOIDL and CLOG is clearly in favour of the latter. Unlike FOIDL, CLOG does not require theory constants to be specified prior to learning. CLOG is much faster than FOIDL (Manandhar et al., 1998), (Kazakov & Manandhar, 1998), and it can process much larger data sets as it does not require all data to be loaded in the memory. A relative shortcoming of CLOG in comparison with FOIDL is that the space of target concepts is limited to a number of hard-coded clauses. A possible way around is to apply FOIDL to a feasible subset of the data, and then pass the type of clauses learned to CLOG and process the whole data set with it.

3.2 Analogical Prediction

Analogical Prediction (AP) is a learning technique that combines some of the main advantages of Instance-Based Learning (IBL) and standard ILP. It takes background knowledge and training examples, then for each test example e it searches for the most compressive single-clause theory that covers e and classifies it similarly to the largest class of training examples covered. AP and IBL share the robustness to the changes in the training data typical for lazy learning approaches. In terms of preference bias, AP replaces the similarity metric needed in IBL with a compression-based metric. Also, AP inherits from ILP its greatest advantage over IBL, namely the explicit hypothesis serving as an explanation of the decisions made. AP has been shown to be particularly well suited for domains in which a large proportion of examples must be treated as exceptions (Muggleton & Bain, 1999). AP has been implemented in Progol4.4 by Muggleton.

4 Aspects of Machine Learning of Natural Language

4.1 Learning from Annotated Data

A look at existing work shows two principal settings in which a computational linguist would learn word morphology. The data in the first case is a corpus (list, lexicon) of annotated words, where each one is tagged with some relevant information. Whether it is a part of speech (Brill, 1994), (Mikheev, 1997), a more complex list of morphosyntactic categories (Kazakov, Manandhar, & Erjavec, 1999) or a string of phonemes describing the word pronunciation (Yvon, 1996b), (van den Bosch, 1997), the information contained in the tag is related to the word as a whole. For instance, if the corpus pairs words with their pronunciation, the two strings of letters and phonemes may be aligned, so that each phoneme corresponds to one or more letters. However, it is the context of the whole word that defines the phonetic value of each letter.

Given a tagged list of words, one can try to identify constituents within the word and find a mapping between these and the information in the tag. One can so attempt to discover whether a certain word ending can be associated with a specific part of speech, or morphological category, such as Number, Gender or Tense. Also, knowing the segments that preserve their pronunciation throughout the corpus can be very valuable, as it can help predict the pronunciation of unknown words composed out of such segments.

One can also imagine the case in which the data set contains words with marked morpheme boundaries. The aim of learning in this case depends on the additional annotation provided. Segmented data can supply the learning algorithm with morphemes or concatenations of morphemes that can be used as theory constants in the rules mapping words to categories. Learning segmentation rules can provide the computational linguist with a tool that can be used as a pre-processor of the morphological analyser or alone. Segmentation producing

the word stem can be seen as assigning a word its meaning as most general-purpose systems for information retrieval assume that the word meaning is fully specified by its stem, ignoring aspects of lexical ambiguity and contextual interpretation.

In the setting described so far, word morphology aims to identify word constituents (morphemes) and assign them certain properties, such as pronunciation, possible parts of speech, etc. The matching between morphemes and word properties can be used both for word generation (Mooney & Califf, 1995), (Mugleton & Bain, 1999) and analysis (Brill, 1994), (Mikheev, 1997).

4.2 Unsupervised Learning of Language

In the second setting in which machine learning is applied to word morphology, the corpus contains no annotation. Here the interest is focussed on the search for the best model of the potentially infinite language or of its finite sample available. The quality of the model can be evaluated by its generative power, i.e. the ability to generate all words of the language, by the likelihood with which the corpus is produced by the model (Deligne, 1996) or by a combination of these two criteria, which are clearly contradictory. The two extremes of the scale could be marked by the most general model, the one that describes words as strings of arbitrary length and composition, and the model represented by the training data itself.

Unsupervised learning combines a new data representation framework (e.g. replace words with concatenations of a prefix and suffix) with a preference bias introducing an order among all possible representations (use the shortest possible lexicons of prefixes and suffixes), and a search technique (e.g. a genetic algorithm) to find the best representation of data w.r.t. the bias. In this context, two principles are often used. Occam's razor recommends the use of the simplest (shortest) model (theory, hypothesis) that can explain how the data was produced. The Minimal Description Length (MDL) principle combines the model length with the length of the data description based on the model; the model which minimises the number of bits encoding both model and data, is given preference (Mitchell, 1997).

Unsupervised learning has been used to annotate texts or lists of words with their segmentations (Deligne, 1996), (Harris, 1955), (Kazakov, 1997), (Brent, Lundberg, & Murthy, 1995), and to assign the most probable pronunciation to each segment (Yvon, 1996b). The biases used in the cited work, and the way they contribute to the data annotation, are shown in the upper part of Table 2 on page 18.

From a practical point of view, unsupervised learning is attractive because of its low cost, as it does not require the presence of a linguist. However, there is also a more fundamental reason to study unsupervised learning. In an area as abundant in models and theories as linguistics is, support from information theory perspective can help consolidate the opinions.

4.3 ILP and Background Knowledge for Language Learning

According to Shieber (1986), there are two classes of linguistic formalisms: linguistic tools and linguistic theories. The former are used to describe natural languages, the latter serve to define the *class of possible natural languages*. One can extend this taxonomy by taking into account the fact that natural languages are yet another class of communication codes, and as such should follow the principles of Information Theory (Shannon & Weaver, 1963), and could be studied and modelled from the its general perspective. Indeed, a considerable part of this article is dedicated to the description of existing research applying general Information Theory principles to natural language.

ILP theory claims that learning is more accurate when the concept language is *extended* to include appropriate background concepts. *Limiting* the concept language to some of these concepts can improve learning efficiency. Even at present, with all ongoing work on predicate invention (Khan, Muggleton, & Parson, 1998) and multi-predicate learning (Muggleton, 1998), the use of well-chosen domain knowledge is crucial. Having existed in its modern form for more than 300 years (Lancelot & Arnauld, 1660), linguistics seems likely to provide such relevant concepts. These could be used in three ways in conjunction with ILP learning.

1. Learning could help specialise an existing theory, e.g. producing rules from templates to obtain a theory or tool for a particular language in a way reminding of Explanation-Based Learning (Mitchell, 1997).
2. On the other hand, it might be possible to use learning for the generalisations of existing theories or tools. For instance, existing morpho-lexical analysers for a number of languages $\mathcal{L}_i$ could be compiled into a single tool for use in a multilingual context. In this case, learning could also result in a metatheory which would subsume each of the languages $\mathcal{L}_i$, and yet be more specific than the general framework initially used.
3. Finally, an existing theory (tool) which is incomplete or inconsistent w.r.t. the corpus could be extended or fine-tuned by learning.

We want to suggest a classification of all possible applications of ILP learning to word morphology based on two factors, the extent to which linguistic concepts are used as background concepts, and the amount of annotation in data. The lower part of Table 2 (see page 18) gives a graphical representation of that classification. The horizontal axis represents a partial ordering between data sets—if two data sets containing the same words are displayed in different columns, the one on the left-hand side contains *all* annotation from the first data set plus some *additional* one. The rows in the table show whether the background concepts used for morphology learning consist only of string relations or make use of increasingly complex linguistic concepts and theories.

Now a learning task can be described as $(O_{in}, C_{in}) \rightarrow (O_{out}, C_{out})$, i.e. as a vector, the initial co-ordinates of which specify the type of data O_{in} and background knowledge C_{in} available. The vector end point is given by the type

of target theory learned C_{out} , and also reflects changes in the type of annotation available in the data.

The application of any preference bias for unsupervised learning from the upper part of Table 2 would result in a right-to-left move in parallel to the horizontal axis. Indeed, each of these biases would add extra annotation to the data yet their contribution to the concept language would usually be limited to the creation of new theory constants, typically segments of words. The vector describing unsupervised learning would usually not have any vertical component, as the (very simple) relations it introduces, such as prefix and suffix (Kazakov, 1997), would already be present in the background knowledge for any learning setting. One could only imagine the hypothetical case in which the use of a preference bias based on a general principle of Information Theory would actually result in a less trivial concept, e.g. lexical-level morpheme (see the last row of the PREFERENCE BIAS part of the table).

Unlike unsupervised learning, one would expect ILP learning to change the conceptual level of representation by introducing new, more complex concepts, and their gradual specialisation, or by generalising existing case-specific theories. In either case, the learning task vector will be vertical. Can ILP be applied to the task usually assigned to unsupervised learners? Strictly speaking, the answer is yes. However, to do so, the ILP learner should incorporate the three already mentioned components of unsupervised learners. As it uses first-order logic as a representation formalism, ILP would hardly have any difficulties incorporating the representation formalism and the preference bias of almost any unsupervised learner. It is the search technique that would make ILP a poor substitute for a given unsupervised learner. To make the search maximally efficient, each of these programs uses a specialised algorithm which would be difficult to replicate in the context of declarative programming or for which a number of standard libraries exist in other, imperative languages. The list of such algorithms includes search for a minimal-cost path in a graph (Deligne, 1996), and genetic algorithms (Kazakov, 1997). The implication is that non-ILP algorithms for unsupervised learning and ILP are not mutually exclusive. Quite the opposite, they could, and should, be combined in fruitful marriages, in which the unsupervised learner is used as a pre-processor for ILP. An example of such hybrid approach using the Naïve Theory of Morphology bias already exists (Kazakov & Manandhar, 1998). This, and several other relevant biases are described in the next section.

5 Preference Biases for Word Segmentation

5.1 Deligne’s Text Segmentation

Deligne (1996) describes a technique for the segmentation of a long single string, possibly containing several sentences with no delimiters between words. The method is looking for a model from which the corpus could be generated with a maximal likelihood. All possible segmentations are represented as a directed graph with one start and one end node. Each edge corresponds to a sequence of letters forming a segment. Each path from the start to the end node represents

a particular segmentation of the whole text. Each edge of the graph is assigned weight representing the probability of this segment. This probability is estimated simply as the relative frequency of the string in the text, and the search for the best segmentation—the one which maximises the product of all segments’ probabilities—is reformulated as a search for a minimal-cost path in the graph. Once an initial segmentation is obtained, it is used to re-estimate the probability of each segment, and to segment the text again. The process stops when the segmentation becomes stable.

This method can be adapted to the task of word segmentation by introducing firm boundaries in the text corresponding to the word borders. The algorithmic complexity of the method requires that the maximal length of segments should be limited (Deligne (1996) sets it to 5 characters). Even in the context of segmenting lists of words, rather than continuous text, removing this restriction may cause problems, as the number of possible constituents becomes exponential w.r.t. the word length. It follows directly from the fact that a segmentation of a string into substrings can be encoded as a string of 1s and 0s where a 1 marks a split point. Then for a string of length n , there are 2^{n-1} possible segmentations. This, additionally multiplied by the complexity $O(n^2)$ of the search for a minimal-cost path (n is the number of nodes in the graph) can easily become an issue, especially for agglutinative languages where words can be of considerable length. For instance, a forty-one-letter word, which is not unheard of in Turkish, has $2^{40} \approx 10^{12}$ possible segmentations.

5.2 Brent et al.

Another word segmentation method based on information theory is employed by Brent et al. (1995). The article describes a binary encoding of a list of words based on lexicons of word constituents, and a table describing how these constituents are combined to form words. Then the Minimal Description Length (MDL) principle is applied, i.e. the table corresponding to the encoding requiring the minimal number of bits is assumed to describe the optimal segmentation of the list of words. As the search space for possible encodings is very large, the approach limits the number of constituents to two per word. Also, suffixes (right-hand-side constituents) which share a suffix between themselves are not allowed.

The suggested search technique is iterative, trying to reduce the binary encoding size by alternately adding new suffixes to their lexicon, or by removing suffixes from that lexicon if the addition of new suffixes does not result in shorter representations. The efficiency of this search technique is unsatisfactory — “*the bad news is search*” (Brent et al., 1995), which leaves the bias with no practical use so far.

5.3 Naïve Theory of Morphology

Kazakov (1997) introduces a bias for the segmentation of all words in a list. It is assumed that each word can be represented as a concatenation of two (possibly

empty) strings *Pref* + *Suf*. For each two segmentations of all words in the list, the bias gives a preference to the one for which the two lexicons of prefixes and suffixes contain less characters in total.

The segmentation of each word can be represented with an integer denoting the prefix length, and the segmentation of a list of words can be encoded as a vector of integers called a Naïve Theory of Morphology (NTM). Then a genetic algorithm can be easily applied to the search for a suitable segmentation of the input where each chromosome is an NTM, and its fitness function is minimising the number of characters in the corresponding prefix and suffix lexicons. Initially, the algorithm generates a number of naïve theories of morphology at random, and then repeatedly applies the genetic operators *crossover*, *mutation* and *selection*.

5.4 Harris's Segmentation

Harris (1955) describes an unsupervised approach to the segmentation of utterances spelt phonetically. The approach counts the number of different phonemes $succ(Prefix)$ which can appear in an utterance of the language after a given initial sequence *Prefix* of phonemes (the notation $succ(n)$, where n is the prefix length, will also be used). For each utterance available, its left substrings $Prefix_i$ are produced and the counts $succ(Prefix_i)$ computed. Then the utterance is segmented where the function $succ(Prefix)$ reaches its local maxima. It is possible to adapt the method for the segmentation of words instead of whole utterances, also replacing phonemes with letters. Figure 3 gives the value of $succ(Prefix)$ for each word in the data set *but*, *cut*, *cuts*, *bread*, *spot*, *spots*, *spotted* — note the use of ^ and @ as unique symbols for a start and end of word. As already

but	: ^-3, b-2, u-1, t-1, @-0
cut	: ^-3, c-1, u-1, t-2, @-0
cut-s	: ^-3, c-1, u-1, t-2, s-1, @-0
bread	: ^-3, b-2, r-1, e-1, a-1, d-1, @-0
spot	: ^-3, s-1, p-1, o-1, t-3, @-0
spot-s	: ^-3, s-1, p-1, o-1, t-3, s-1, @-0
spot-ted	: ^-3, s-1, p-1, o-1, t-3, t-1, e-1, d-1, @-0

Fig. 3. Segmentation points and $succ(Prefix)$ (shown after the last letter of each prefix)

stated, a word w is segmented after the initial substring of length n , if for n $succ(n)$ reaches a local maximum, which is greater than both $succ(n - 1)$ and $succ(n + 1)$. In case a plateau is found, all nodes on the plateau are considered as segmentation points provided a downhill slope follows. Using this criterion one obtains *cut-*, *cut-s*, *spot-*, *spot-s*, *spot-ted* while *but* and *bread* are not segmented. Although not mentioned in the original paper (Harris, 1955), Harris's method can be based on the representation of all utterances (or words) as a prefix, resp. suffix tree.

5.5 de Saussure’s Analogy

In his *Course in General Linguistics*, de Saussure (1916) describes a principle of analogy according to which, in the long term, the word-forms in a given language change and tend to form the following patterns:

$$\begin{array}{l} Pref_1 + Suf_1 : Pref_1 + Suf_2 = \\ Pref_2 + Suf_1 : Pref_2 + Suf_2 \end{array} \quad (1)$$

The principle is often used for word segmentation (Pirelli, 1993), (Yvon, 1996b). A word is split into a prefix and suffix only if one can find in the corpus another three words forming the proportion in Equation 1, for example, *sleep-s*, *sleep-ing*, *read-s*, *read-ing*. A simpler approach using just one row or column of this pattern can be also adopted (Brill, 1994), (Mikheev, 1997). However, using the complete analogy principle helps filter out the numerous spurious segmentations generated when prefixes or suffixes are factored out of pairs of words. For instance, although the prefix *on-* appears in the words *onyx* and *ontology*, one would never segment *on-yx* if the corpus does not contain the words *Pref+ ‘yx’*, *Pref+ ‘tology’* for any value of *Pref*.

There are situations when the principle of analogy will fail even if there is a very obvious segmentation at hand. A look at the following example 2 shows that, for instance, W_1 cannot be segmented, as the word $Pref_2 + Suf_1$ is missing in the data.

$$\begin{array}{ll} W_1 = Pref_1 + Suf_1 & W_4 = Pref_1 + Suf_2 \\ W_2 = Pref_2 + Suf_2 & W_5 = Pref_2 + Suf_3 \\ W_3 = Pref_3 + Suf_3 & W_6 = Pref_3 + Suf_1 \end{array} \quad (2)$$

6 Existing Applications of ILP to Morphology Learning

ILP has been applied to two types of morphology-related learning tasks so far, learning of inflectional morphology and word segmentation. In our new taxonomy, these tasks are encoded as $(O_2, C_1) \rightarrow (O_2, C_4)$ and $(O_1, C_1) \rightarrow (O_2, C_4)$ respectively.

6.1 Learning of Inflectional Morphology

Most ILP applications to word morphology have aimed to learn a theory matching pairs of word-forms of the same lexical entry (lemma), each of which corresponds to a certain set of morphosyntactic categories. The setting can be schematically represented as $W_1/Cat_1 : W_2/Cat_2$, for instance *sleep/VerbPresent*: *slept/VerbPast*.

Typically, one of the forms provided (e.g. W_1) is the standard lexical entry, which has for each language and part of speech a fixed set of morphosyntactic categories, e.g. for the Czech Noun it is Case=Nominative, Gender=Masculine,

Number=Singular. That allows for the data to be represented as a set of facts of a binary predicate which has the form `cat2(LexEntry,W2)`. There are two variations of the task, (1) given the lexical entry find the inflected form, or, (2) given the inflected form find the lexical entry. In ILP terms, it corresponds to learning the target predicate with modes `cat2(+,-)` or `cat2(-,+)`, respectively.

Existing work includes learning of Dutch diminutives (Blockeel, 1994), English Past Tense (Mooney & Califf, 1995), (Muggleton & Bain, 1999), Slovene nominal paradigms (Džeroski & Erjavec, 1997), and nominal and adjectival paradigms in English, Romanian, Czech, Slovene and Estonian (Manandhar et al., 1998). In all but one case, a first-order decision list learner has been used, the exception being the application of Progol and Analogical Prediction to English Past Tense data by Muggleton and Bain (1999). The rules learned are based on the presence of one or two affixes in the word. For instance, one could have rules of the following template:

$$(Stem + Suf_1)/Cat_1 : (Stem + Suf_2)/Cat_2 \quad (3)$$

If the concatenation of a given stem and suffix Suf_1 is a word-form corresponding to category Cat_1 , then by replacing Suf_1 with Suf_2 one can obtain the word-form corresponding to category Cat_2 , e.g. *(appl+y)/Present: (appl+ied)/Past*.

The English Past Tense data set has become a standard test bed for both ILP (Mooney & Califf, 1995), (Muggleton & Bain, 1999) and non-ILP learning approaches related to morphology (Rumelhart & McClelland, 1986), (Ling & Marinov, 1993). The work of Rumelhart and McClelland (1986) is the first in a series of articles to apply learning to the task of producing the past tense of English verbs from their present form. The simple perceptron-based learner used by them was followed by a more sophisticated use of Artificial Neural Networks (ANN) by MacWhinney and Leinbach (1991). Ling and Marinov (1993) have subsequently applied the Symbolic Pattern Associator (SPA), a C4.5-based learner, to the same task. The principle of analogy has also been used by Yvon (1996a).

The data consists of pairs of present and past tense forms of the same verb, for instance, *(sleep, slept)*. It comes in two flavours, alphabetic, using standard spelling, and phonetic, reflecting the pronunciation. Direct comparison of learning methods is made difficult by the fact that different approaches do not use training and test data of the same size and distribution of regular and irregular verb forms. For instance, for a sample of 1200 training and 102 test verb forms spelt phonetically, SPA shows 89% predictive accuracy as compared to the 80% of MacWhinney's ANN (Ling, 1994). The ANN here is an improved version of the one previously published by MacWhinney and Leinbach (1991). The original ANN achieves 55.6% accuracy on a sample of 500 training and 500 test entries in phonetic representation as compared to Ling's 82.8%. The ILP learner FOIL outperforms SPA on the same data with 83.3% (Quinlan, 1994).

The alphabetical data set with the same 500/500 training-to-test sample ratio allows for the best comparison of ILP methods applied to the task. Table 1 shows

the results for FOIDL, IFOIL and FOIL reported by Mooney and Califf (1995) (Quinlan’s result with FOIL for the same data is 81.2%). Results with standard Progol learning pure logic programs and Progol using Analogical Prediction have been reported by Muggleton and Bain (1999). All results should be compared to the majority class baseline of 79.3% (PAST = PRESENT+‘ed’). Comparison shows that ILP overperforms the other approaches. Within ILP, Analogical Prediction achieves the best results (Muggleton & Bain, 1999).

Table 1. Learning of English past tense

Data set	non-ILP learning		ILP learning				
	Neural network	SPA	FOIDL	IFOIL	FOIL	AP	PROGOL
Phonetic	55.6	82.8	N/A	N/A	83.3	N/A	N/A
Alphabetic	N/A	N/A	86	60	82	95.9	87.0

6.2 Hybrid Learning of Word Segmentation Rules

Kazakov and Manandhar (1998) have proposed a hybrid approach to the segmentation of an unannotated list of words in which the words are initially segmented by applying the Naïve Theory of Morphology bias, and then a first-order decision list of segmentation rules is learned with the help of either FOIDL or CLOG. These rules show a good degree of generalisation, and can be used for the segmentation of unknown words. A characteristic feature of the method is that the search technique (GA) used in the unsupervised learner may not be able to find the best segmentations w.r.t. the bias. The words that are not segmented in an optimal way are often covered by an exception in the decision list, that is, a rule applicable to just one word with no potential for generalisation. Removing these exceptions from the decision list does not influence its performance on other, unseen, words, and the application of the remaining rules in the decision list to the training list of words usually results in segmentations that are better both w.r.t. the bias, and from a linguistic point of view.

Experiments with the learning of segmentation rules for Slovene have shown that the segments produced for each word are strongly correlated with its morpho-syntactic categories, thus supporting the hypothesis that these segments can be interpreted as morphemes (Kazakov et al., 1999).

7 Future Work

The taxonomy introduced in this article defines automatically a whole space of learning tasks most of which remain unexplored. The following list contains the defining vectors of some of them and their possible interpretation.

- $(O_1, C_1) \rightarrow (O_2, C_4)$ Replace NTM with other biases, such as Deligne’s, Harris’s or analogy, and learn word segmentation rules.
- $(O_2, C_1) \rightarrow (O_2, C_3)$ Invent new predicates to be used as rule templates in task $(O_2, C_1) \rightarrow (O_2, C_4)$. The template in Equation 3 (page 15) gives a good idea of the type of templates one could expect to learn.
- $(O_1, C_2) \rightarrow (O_1, C_4)$ Use some basic linguistic concepts, such as consonants, vowels, semivowels, diphthongs (unary predicates) or the possibility of mapping several morphs onto a single lexical-level morpheme (a binary predicate with append/3 literals in the body) to learn a model of a given language that is a more-than-one-level morphology.
- $(O_1, C_3) \rightarrow (O_1, C_4)$ Same as above, but use the two-level morphology and, possibly, relevant rule templates as background knowledge.
- $(O_2, C_3) \rightarrow (O_3, C_4)$ Learn the two-level morphology model of language $\mathcal{L}$ from words tagged with their morphosyntactic categories. Use a combination of unsupervised learning to segment the words (Harris, Naïve Theory of Morphology) and ILP.
- $(O_3, C_2) \rightarrow (O_3, C_4)$ Learn the two-level morphology model of language $\mathcal{L}$ from words tagged with their lexical-level morphemes and morphosyntactic categories. This kind of annotation is the typical output of a two-level morpho-lexical analyser. Here is a transliterated example of so analysed Turkish word:
`['CIkarIlmakla', [[cat:noun, stem: [cat:verb, root: 'CIK', voice:caus, voice:pass, sense:pos], suffix:mak, type:infinitive, agr: '3SG', poss: 'NONE', case:ins]]].`
- $(O_3, C_4) \rightarrow (O_3, C_{4+})$ Start with an incomplete and/or inaccurate two-level description of morphology and learn the complete and correct set of rules. This is a particularly interesting application, as it can help to adapt an existing two-level model of the language norm to account for the peculiarities of a vernacular.
- $(O_2, C_4) \rightarrow (O_2, C_3)$ Learn metarules (rule templates) from inflection or segmentation rules. The latter can be provided by linguist, or obtained in previous learning steps.
- $(O_3, C_4) \rightarrow (O_3, C_3)$ Use the existing two-level models of the morphology of several languages, e.g. French, Spanish and Italian, to learn a common set of rules and/or templates potentially relevant to a family of languages (e.g. Romance languages).

8 Conclusions

This paper describes some existing work in ILP learning of word morphology, and identifies the issues addressed so far. In the context of the current achievements in computational morphology, it seems that combining ILP with existing linguistic theories, rather than substituting it for them, would lead to applications which would challenge the ILP community and attract more attention from computational linguists. Research in this direction could be helped by the

Table 2. Below: space of ML applications to word morphology. Above: preference biases used to augment training data annotation.

PREFERENCE BIAS	Deligne's text segmentation		Segmentation ←	No annotation
	Brent et al.		Segmentation ←	No annotation
	NTM		Segmentation ←	No annotation
	Harris		Segmentation ←	No annotation
	de Saussure's analogy		Segmentation ←	No annotation
		Segmentation ←	Pronunciation or morphosynt. cat.	
	?	Segmentation and lexical level morphemes	Morphosyntactic categories	No annotation

Relations between strings - append, prefix, suffix C1		English Past Tense, Slovene nominal paradigms	
		Segmentation (NTM)	Words
Basic linguistic concepts - consonants, vowels, semivowels, diphthongs; - morphemes: lexical vs surface level C2	Words annotated with lex. level morphemes and categories		Words
Abstract machines and rules - 2L morphology: finite state transducers, rule templates - specific rules: vowel raising, palatalisation, nasal assimilation C3	Segmentation (Harris) 2L framework for Romance languages	(Word,Cat) in Language L Rule templates	Words
Implementation of tool T for language L - 2L morphology rules of language L * incomplete/inconsistent * complete/accurate C4	Two-level morphology of French, Spanish, Italian... Two-level morphology	Segmentation rules Conjugation / declension rules	Two-level morphology
CONCEPT LANGUAGE TRAINING DATA LANGUAGE	More annotation additional use of - segmentation or - morphosyntactic categories or - lex. level morphemes O3	Some annotation - segmentation or - morphosyntactic categories, etc. O2	No annotation O1

development of standard ILP libraries of background predicates representing linguistic concepts or theories. Also, the integration of ILP with methods for unsupervised learning in a hybrid framework would increase the ability of ILP to learn from impoverished data containing little or no annotation. The newly introduced taxonomy of learning approaches to morphology describes a variety of tasks that have not been addressed yet, and could serve as an inspiration for future research in the field.

Bibliography

- Antworth, E. (1991). Introduction to two-level phonology. *Notes on Linguistics*, 53, pp. 4–18.
- Bender, B. W. (1997). *Latin Verb Inflection*. University of Hawai'i, http://www2.hawaii.edu/~bender/Latin_Verb.html.
- Blockeel, H. (1994). Application of inductive logic programming to natural language processing. Master's thesis, Katholieke Universiteit Leuven, Belgium.
- Brent, M., Lundberg, A., & Murthy, S. (1995). Discovering morphemic suffixes: A case study in minimum description length induction. In *Proc. of the Fifth International Workshop on Artificial Intelligence and Statistics*.
- Brill, E. (1994). Some advances in transformation-based part of speech tagging. In *Proc. of the Twelfth National Conference on Artificial Intelligence*, pp. 748–753. AAI Press/MIT Press.
- Califf, M. E., & Mooney, R. J. (1996). Advantages of decision lists and implicit negatives in inductive logic programming. Tech. rep., University of Texas at Austin.
- Chomsky, N., & Halle, M. (1968). *The Sound Pattern of English*. Harper and Row, New York.
- Cussens, J. (1997). Part-of-speech tagging using Progol. In *Proc. of the Seventh International Workshop on Inductive Logic Programming*, pp. 93–108.
- de Saussure, F. (1916). *Course in General Linguistics* (1959 edition). Philosophical Library, New York.
- Deligne, S. (1996). *Modèles de séquences de longueurs variables: Application au traitement du langage écrit et de la parole*. Ph.D. thesis, ENST Paris, France.
- Džeroski, S., & Erjavec, T. (1997). Induction of Slovene nominal paradigms. In *Proc. of the Seventh International Workshop on Inductive Logic Programming*, pp. 141–148 Prague, Czech Republic. Springer.
- Fradin, B. (1994). L'approche à deux niveaux en morphologie computationnelle et les développements récents de la morphologie. *Traitement automatique des langues*, 35(2), 9–48.
- Harris, Z. S. (1955). From phoneme to morpheme. *Language*, 31(2).
- Kaplan, R. M., & Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3), 331–379.

- Kazakov, D. (1997). Unsupervised learning of naïve morphology with genetic algorithms. In Daelemans, W., Bosch, A., & Weijters, A. (Eds.), *Workshop Notes of the ECML/MLnet Workshop on Empirical Learning of Natural Language Processing Tasks*, pp. 105–112 Prague, Czech Republic.
- Kazakov, D. (1999). Combining LAPIS and WordNet for the learning of LR parsers with optimal semantic constraints. In Džeroski, S., & Flach, P. (Eds.), *Proc. of The Ninth International Workshop on Inductive Logic Programming*, pp. 140–151 Bled, Slovenia. Springer-Verlag.
- Kazakov, D., & Manandhar, S. (1998). A hybrid approach to word segmentation. In Page, D. (Ed.), *Proc. of the Eighth International Conference on Inductive Logic Programming*, pp. 125–134 Madison, Wisconsin. Springer-Verlag.
- Kazakov, D., Manandhar, S., & Erjavec, T. (1999). Learning word segmentation rules for tag prediction. In Džeroski, S., & Flach, P. (Eds.), *Proc. of The Ninth International Workshop on Inductive Logic Programming*, pp. 152–161 Bled, Slovenia. Springer-Verlag.
- Khan, K., Muggleton, S., & Parson, R. (1998). Repeat learning using predicate invention. In Page, D. (Ed.), *Proc. of the Eighth International Workshop on Inductive Logic Programming*, pp. 165–174 Madison, Wisconsin. Springer-Verlag.
- Koskenniemi, K. (1983). *Two-level morphology: A General Computational Model for Word-Form Recognition and Production*. University of Helsinki, Dept. of General Linguistics, Finland.
- Lancelot, C., & Arnauld, A. (1660). *Grammaire générale et raisonnée (de Port-Royal)* (1967 facsimile edition). The Scolar Press, Menston, England.
- Lavrač, N., & Džeroski, S. (1994). *Inductive Logic Programming Techniques and Applications*. Ellis Horwood, Chichester.
- Ling, C. X. (1994). Learning the past tense of English verbs: The symbolic pattern associator vs. connectionist models. *Journal of Artificial Intelligence Research*, 1, 209–229.
- Ling, C. X., & Marinov, M. (1993). Answering the connectionist challenge: A symbolic model of learning the past tense of English verbs. *Cognition*, 49(3), 235–290.
- MacWhinney, B., & Leinbach, J. (1991). Implementations are not conceptualizations: Revising the verb model. *Cognition*, 40, 291–296.
- Manandhar, S., Džeroski, S., & Erjavec, T. (1998). Learning Multilingual Morphology with CLOG. In Page, D. (Ed.), *Proc. of The Eighth International Conference on Inductive Logic Programming*, pp. 135–144 Madison, Wisconsin.
- Matthews, P. H. (1974). *Morphology: an Introduction to the Theory of Word-Structure* (First edition). Cambridge University Press.
- Matthews, P. H. (1997). *The Concise Oxford Dictionary of Linguistics*. Oxford University Press.
- Mikheev, A. (1997). Automatic rule induction for unknown word guessing. *Computational Linguistics*, 23(3), 405–423.
- Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

- Mooney, R. J., & Califf, M. E. (1995). Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research*, 3, 1–24.
- Muggleton, S. (1998). Advances in ILP theory and implementations. In Page, C. (Ed.), *Proc. of The Eighth International Workshop on Inductive Logic Programming*, p. 9 Madison, Wisconsin. Springer-Verlag. Abstract of keynote presentation.
- Muggleton, S., & Bain, M. (1999). Analogical prediction. In *Proc. of the Ninth International Workshop on Inductive Logic Programming* Bled, Slovenia. Springer-Verlag.
- Pirelli, V. (1993). *Morphology, Analogy and Machine Translation*. Ph.D. thesis, Salford University, UK.
- Quinlan, J. R. (1994). Past tenses of verbs and first-order learning. In Debenham, J., & Lukose, D. (Eds.), *Proceedings of the Seventh Australian Joint Conference on Artificial Intelligence*, pp. 13–20 Singapore. World Scientific.
- Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5, 239–266.
- Rumelhart, D. E., & McClelland, J. (1986). *Parallel Distributed Processing*, Vol. II, chap. On Learning the Past Tense of English Verbs, pp. 216–271. MIT Press, Cambridge, MA.
- Shannon, C. E., & Weaver, W. (1963). *The Mathematical Theory of Communication*. University of Illinois Press, Urbana.
- Shieber, S. (1986). *An introduction to unification-based approaches to grammar*. No. 4 in CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, CA.
- van den Bosch, A. (1997). *Learning to Pronounce Written Words: A Study in Inductive Language Learning*. Ph.D. thesis, University of Maastricht, The Netherlands.
- van den Bosch, A., Daelemans, W., & Weijters, T. (1996). Morphological analysis as classification: an inductive learning approach. In Oflazer, K., & Somers, H. (Eds.), *Proc. of NeMLaP-2*, pp. 79–89 Ankara, Turkey.
- Yvon, F. (1996a). Personal communication.
- Yvon, F. (1996b). *Prononcer par analogie: motivations, formalisations et évaluations*. Ph.D. thesis, ENST Paris, France.
- Zelle, J. M., & Mooney, R. J. (1993). Learning semantic grammars with constructive inductive logic programming. In *Proceedings of AAAI-93*, pp. 817–822. AAI Press/MIT Press.

Combining LAPIS and WordNet for the Learning of LR Parsers with Optimal Semantic Constraints

Dimitar Kazakov

University of York, Heslington, York YO10 5DD, UK

kazakov@cs.york.ac.uk,

WWW home page: <http://www.cs.york.ac.uk/~kazakov/>

Abstract. There is a history of research focussed on learning of shift-reduce parsers from syntactically annotated corpora by the means of machine learning techniques based on logic. The presence of lexical semantic tags in the treebank has proved useful for the learning of semantic constraints limiting the amount of nondeterminism in the parsers. The grain of the semantic tags used is of direct importance to that task. The combination of the system LAPIS with the lexical resource WordNet described here allows to learn parsers while shifting the responsibility for the choice of semantic tags from the corpus annotator to the learning system. The method is tested on an original corpus also described herein.

1 Introduction

In all but the simplest natural language corpora, the grammars used for syntactic annotation are ambiguous, i.e. for some sentences they generate more than one parse. When learning parsers from annotated corpora, two of the central issues in the parser design are the reduction of the number of parses, and the choice of the best parse out of the remaining ones. The former issue can be addressed by adding new constraints to and modification of the grammar used by the parser. If the result is a deterministic grammar, then the latter issue mentioned is automatically settled. Otherwise, probabilistic techniques can be applied to assign a probability to each parse, and choose the most probable one.

There is a history of research focussed on limiting or removing the nondeterminism from shift-reduce parsers using machine learning techniques based on logic [7], [6], [2]. The latter work introduces the system LAPIS in which lexical semantic tags present in the treebank are used to learn semantic constraints limiting the amount of nondeterminism in the LR parsers produced by the system. The grain of the semantic tags used is of direct importance to that task. Too general semantic tags would result in rules with poor discriminating ability, whereas too specific ones would not permit sufficient generalisation. It is unlikely that any set of tags chosen *a priori* would be optimal in that respect.

The present work implements a technique previously outlined by the author [2] allowing the use of a large resource of lexical semantic information,

WordNet, and the shift of responsibility for the choice of semantic tags from the corpus annotator to the learning system. The method is tested on an original corpus with suitable annotation.

2 LR Parsers

LR parsers belong to the family of shift-reduce parsers. The latter are bottom-up directional parsers which use a stack to keep the intermediary results. Two of the basic actions carried out by all shift-reduce parsers are to shift a symbol from the input stream onto the stack, and to reduce a number of symbols on top of the stack, i.e., replace them with another one. Characteristic for the LR parsers is that they cannot handle ambiguous grammars, on the other hand, they can detect a syntactic error as soon as it is possible to do so on a left-to-right scan of the input [1].

The LR parser stack always contains an odd number of elements. Starting from the bottom with the initial state of the parser, all odd slots contain indices of parser states. The even slots of the stack contain grammar symbols, either terminals or nonterminals (see Figures 2–4). The current state of the parser is the one on top of the stack. Each internal state corresponds to a set of partially processed grammar rules which share the RHS symbols seen so far. Each subsequent action of the parser is selected from a *parsing table* according to the current state of the stack and a number of lookahead input symbols (a single one for the parsers studied here). LR parsing tables are always created with the help of a software tool, such as the well-known Yacc [1]. The attempt to construct a parsing table for an ambiguous grammar results in a table containing *conflicts*, i.e. for certain pairs of states and lookahead symbols there are several possible actions in the table. Starting from a start state, four actions are possible for a given **State** and input **Symbol**.

Accept If **State**=**EndState** and **Symbol**=**end_of_string**, then halt. The input has been successfully parsed (see Figure 2).

Shift/NewState Push the current input symbol onto the stack and transit to **NewState**, i.e., put **NewState** on top of the stack (see Figure 3). **NewState** is chosen from a parsing table, where it is selected by **State** and **Symbol**.

Reduce/RuleNo Remove from the top of the stack the number of couples (**state**,**symbol**) corresponding to the number of symbols in the RHS of the grammar rule **Rule**, check out the state **OldState** now on top of the stack, push the LHS of **Rule** onto the stack, and transit to state **NewState**, i.e. push it onto the stack (see Figure 4). **NewState** is selected by **OldState** and LHS in the part **Goto** of the LR parsing table.

Error Fail.

LR parsers are easily represented as logic programs. The representation described here is based on the one used by Zelle and Mooney [7]. If the tagged words are represented as **Word:PoS^SemTag**, then the following are templates or examples of parser actions.

```

%accept(Stack,Input).
accept([1,Axiom|_],[_:e_o_s^_|_]).

%error(Stack,Input).
error([0|_],[_:AnyOf^_|_]):-
\+ member(AnyOf,[v,dt,n,to,ppron]).

%action_s(Stack,Input,NewStack,NewInput).
action_s([State|MoreSt],[LAS|MoreIn],[NewState,LAS,State|MoreSt],MoreIn).

%action_r(Stack,Input,NewStack,NewInput).
action_r([State,SymB,State1,SymA,State2|MoreStack],[LAS|MoreInput],
[NewState,lhs([SymA,SymB]),State2|MoreStack],[LAS|MoreInput]):-
goto(State2,lhs,NewState).

goto(2,lhs,3).
goto(4,lhs2,5).

```

3 Related Work

In his PhD thesis [6], Samuelsson describes a technique for the development of an efficient LR parser based on Explanation-Based Learning (EBL) [4] and entropy-related information measures. The method uses a treebank to modify the provided set of grammar rules and replace some of them with partially unfolded ones. The new rules do not cover part of the syntactic readings, if those are only marginally represented in the treebank. As a result, the grammar is less ambiguous at the price of a certain loss of coverage. Using that grammar also results in considerably faster parsing, due to the pruned search space and the larger number of symbols in the right-hand side of the unfolded grammar rules, which are reduced by the LR parser in a single step.

Zelle and Mooney [7], [8] have developed a method for learning semantic grammars from a treebank containing syntactic trees with semantically tagged nonterminal nodes. The treebank is used to construct an overgeneral shift-reduce parser covering the sentences in the treebank. The parser is then specialised and made deterministic by using ILP. When the semantic tags assigned to nonterminals are not sufficient to remove the ambiguity from the grammar, lexical semantic classes are automatically defined in order to achieve deterministic parsing. Specific for that work is that a structure describing deep cases, such as Subject, Object, Instrument, etc. is directly derived from the sentence without the help of an intermediary syntactic tree.

4 LAPIS

LAPIS is a system which builds on Zelle and Mooney's research on the induction of shift-reduce parsers and extends it to learning of LR parsers, while changing at the same time the focus of the learning task. The existence of sources of lexical semantic knowledge such as WordNet [3] makes the learning of lexical semantic classes done by Zelle and Mooney less attractive. Also, treebanks

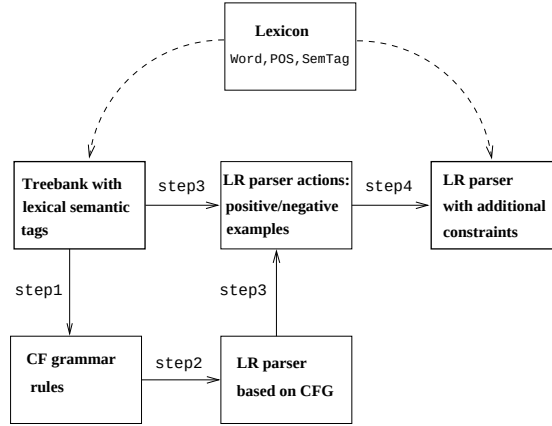


Fig. 1. LAPIS

annotated with phrasal semantic tags are not commonly available. Instead, the system LAPIS constructs LR parsers from treebanks of parse trees annotated with lexical semantic tags. LAPIS aims at the reduction of nondeterminism in the parsers it creates. This is achieved, as it will be shown further, by the means of lexicalisation and partial unfolding of the underlying grammar rules, in combination with the use of lexical semantic constraints.

LAPIS (Learning Algorithm for Parser Induction Systems) consists of the following procedures (see Figure 1).

Step 1 First, the context-free grammar rules implicitly present in the treebank, i.e. all pairs $\langle \text{ParentNode}, \text{Daughters} \rangle$, are found.

Step 2 Using the CFG rules obtained, the parse table of a possibly nondeterministic LR parser is generated with *Yacc*. Each possible parser action (transition of that parser from one state to another) is represented as a Prolog clause. The logic program obtained can use the Prolog built-in backtracking mechanism to deal with any nondeterminism in the parser, and thus produce for a given sentence all possible parses.

Step 3 For each sentence in the treebank, the nondeterministic LR parser from Step 2 is used to generate the sequence of parser actions leading to the correct parse as shown in the treebank. Each of these actions is an instantiation of one of the clauses generated in Step 2.

For each parser state in which nondeterminism occurs, e.g. State 7 in Figure 5, the instantiation of the correct parser action (**r2** in the example) is stored as a positive instance of that action. All other possible instantiations of parser actions for the given state and input symbol (**r5** here) are also generated and stored as negative instances of those actions.

Step 4 Finally, the positive and negative instances of each parser action are used as input of a greedy ILP learning algorithm (shown in Figure 7) based

on syntactic least general generalisation [5] to learn a new definition of that action. The new definition is a set of clauses subsuming all positive examples, but none of the negative (see Figure 5), and is usually more specific than the CFG-based definition of action generated in Step 2. If the former definition is substituted for the latter, the new parser obtained usually contains less nondeterministic states and hence does not produce some of the spurious parses generated by the CFG-based parser from Step 2.

The new parser can be viewed as using a unification grammar based on the backbone CFG of the parser from Step 2, but with rules additionally specialised (their coverage decreased). The difference between the CFG rules and those learned is threefold.

1. Firstly, where the CFG rule is only looking for a terminal with a particular part of speech (e.g. noun), the new rule additionally requires that the terminal should belong to some semantic class.
2. Secondly, a PoS can be narrowed down to a single word. In other words, the grammar rules are partially *lexicalised*.
3. The third important feature of the rules learned is that some of the nonterminals in their RHS are *unfolded*, i.e., replaced by some of their descendants in the parse tree, as shown in Figure 9.

It can be seen from the rules learned that LAPIS combines rule unfolding with the use of lexical semantic tags to disambiguate the grammar. That, along with the partial lexicalisation makes the new grammar rules more specific than the original CFG ones.

5 WordNet as a Source of Semantic Tags

WordNet is an on-line lexical database which contains various syntactic and semantic information for a large number of words and idioms. Originally developed for English [3], WordNet is also being implemented for other languages. The central building element of WordNet is called *synset*, or *lexicographer's entry*. A synset is a set of words or idioms which share a common meaning. For instance: $\{(to) \textit{ shut}, (to) \textit{ close}\}$. To simplify the internal representation, each synset is assigned a large integer used as a unique identifier. For instance, $\{monetary \textit{ resources}, \textit{ funds}\}$ is Synset 109616555 in WordNet1.6.

WordNet uses a set of rules and lists of exceptions to map word-forms to all relevant lexical entries. Figure 8 shows the word-form 'funds' which is recognised by WordNet as corresponding to two lexical entries, 'fund' and 'funds'. The lexical entry 'fund' appears in three synsets: $\{store, fund\}$, $\{fund, monetary \textit{ fund}\}$, and $\{investment \textit{ company}, fund\}$, respectively. The lexical entry 'funds' only appears in the synset $\{monetary \textit{ resource}, funds\}$. WordNet describes several semantic relations between synsets, such as meronymy (PART-OF), hypernymy or hyponymy.

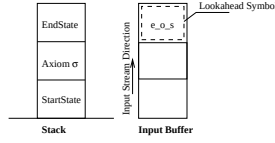


Fig. 2. LR parser accept action

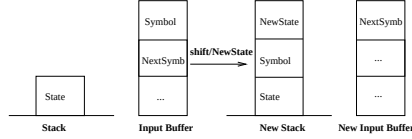


Fig. 3. LR parser shift action

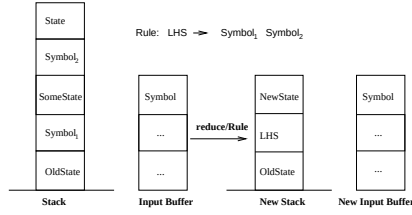


Fig. 4. LR parser reduce action

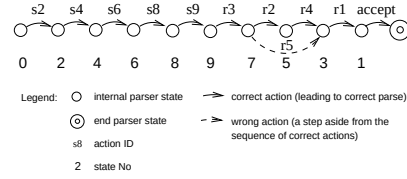


Fig. 5. Generation of positive and negative examples of actions

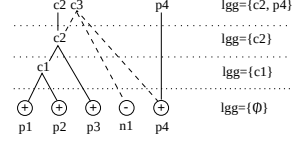


Fig. 6. Least general generalisations of parser actions

Procedure Inductive learning of parser actions
 Initialise $e^+ :=$ List of Positive Examples
 Initialise $e^- :=$ List of Negative Examples
 Initialise $\text{SetOfGenCl} := \{\emptyset\}$
repeat
 Get a positive example e_{next}^+
repeat
 Get next clause NextGenAction from SetOfGenCl
 Find $\text{LGG} := \text{lgg}(\text{NextGenAction}, e_{next}^+)$
until ((LGG does not subsume any example from e^- , $\text{NewLGGFound} := \text{true}$)
 or (there are no more clauses in SetOfGenCl , $\text{NewLGGFound} := \text{false}$))
if $\text{NewLGGFound} := \text{true}$ **then**
 Retract NextGenAction from SetOfGenCl
 Retract from SetOfGenCl all clauses subsumed by LGG
 Put LGG in SetOfGenCl
else Add e_{next}^+ to SetOfGenCl
 Retract e_{next}^+ from e^+
until e^+ is empty
return SetOfGenCl

Fig. 7. Inductive learning of parser actions

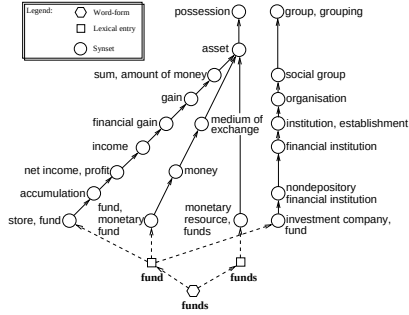


Fig. 8. Mapping from word-forms and lexical entries to synsets and their hypernyms in WordNet

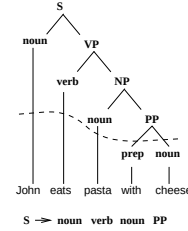


Fig. 9. Partial unfolding of grammar rule

The latter two are identical up to the order of their arguments, and represent the two directions of the IS-A relationship, i.e. they bind a synset to a more general one: ‘medium of exchange’ is the hypernym of ‘money’, resp. ‘money’ is a hyponym of ‘medium of exchange’. Once the mappings *Word-form* $\rightarrow$ *Lexical entry* and *Lexical entry* $\rightarrow$ *Synset* are carried out, WordNet semantic relations can be queried. Figure 8 displays all hypernyms possibly related to the word-form ‘funds’. WordNet also contains a similar hierarchy of hypernyms for the verbs. In terms of graph theory, each of these two hierarchies can be described as a directed acyclic graph.

In the context of corpus-based parser learning, where the grain of semantic classes is difficult to be foreseen, such hierarchy of semantic tags can provide the necessary material for a flexible choice of the most appropriate semantic tags to be embedded in the parser. The idea is to allow the learning algorithm to choose automatically the appropriate level of generality, keeping only the generic semantic classes unless they cover negative examples.

That goal can be achieved by using representing the lexical semantic tags in the treebank as lists. For each tagged word, the last element in the list is the WordNet synset representing the meaning of that word in the given context. In the list used as a semantic tag, that synset is preceded by the *list* of its hypernyms ordered according to their generality, from the most general to the most specific. Typical examples of words so tagged follow; to improve readability, synset indices have been replaced with one of the synonyms to which they correspond.

```
%Word:PoS^SemanticTag  pos/neg example
chauffeur:n^[entity,causal_agent,operator,driver,chauffeur]  (+)
driver:n^[entity,causal_agent,operator,driver]                (+)
friend:n^[entity,causal_agent,person,friend]                  (-)
```

Imagine the words so tagged appearing in otherwise identical instantiations of the same parser action, with the first two examples being positive and the last one negative. The generalisation step of the ILP algorithm will result in the following constraint for that terminal: `Word:n^[entity,causal_agent,operator,driver|_]`. Note that the internal representation of lists as binary trees makes possible the least general generalisation of lists of different length.

6 Dataset

The lexicon Lex5000, used in the experiments with structured semantic tags, consists of the following files and predicates:¹

1. The files `noun.lex` and `verb.lex` contain the predicates `nlx/2`, and `vlx/2` respectively, which map noun and verb word-forms into lexical entries. For instance: `vlx(beginning,begin)`.
2. The files `noun_path_i.pl` and `verb_path_i.pl` contain the predicates `npath/2` and `vpath/2` respectively, which map a lexical entry into one of the possible

¹ The structure of the lexicon benefitted from the joint work with James Cussens on a related task.

semantic tags. There are at least as many clauses per lexical entry, as there are synsets which include that entry. Moreover, there may be more than one “path” of hypernyms per synset, since some of the synsets in WordNet have more than one hypernym. For the sake of brevity, the original synset indices used in WordNet have been replaced by shorter, yet unique identifiers, e.g. `vpath(begin, [i948, i949, i950])`.

3. The file `i_map.pl` contains a single predicate, `imap/2` which maps the identifiers used in the predicates `npath/2` and `vpath/2` to the corresponding synset indices in WordNet1.6, e.g. `imap(i10, '100011937')`.
4. The file `ambig_lex.pl` contains a mapping from word-forms to lexical entries for the words other than nouns and verbs, along with a single top-level predicate `dict(WordForm, PoS, TopDownPath)` serving as a uniform interface to the whole lexicon.

The ratio between word-forms, lexical entries and semantic tags (paths of hypernyms) is 73 : 74 : 265 for the nouns, and 26 : 27 : 183 for the verbs. There are 84 different word-forms in the whole lexicon, and 468 combinations of a word-form, PoS and semantic tag, which corresponds to an average of 5.57 tags per word-form.

The treebank T5000 is an artificial resource, generated from 12 sentence templates. The lexique used is the one present in Lex5000. The treebank contains 5046 sentences and their parse trees, in which nouns and verbs are semantically tagged. Each semantic tag is of the type used in Lex5000. Although the words in Lex5000 can have more than one semantic tag assigned, the treebank is generated with the help of an additional lexicon (`disamb_lex.i.pl`), from which only the correct semantic tag is taken for each lexical slot in the templates. Some of the 12 templates are shown below, with the variable slots in bold face:

1. **I want** to book **the first** flight to **London** on **Monday**
2. **I want** to book **the first** flight to the capital of **Bulgaria** on **Monday**
6. **I want** to go to the **museum** at the **end** of **the street**
7. **I want** to go to the **museum** at the **end** of **the week**
8. **I want** to go to the **museum** by the **river**

The treebank consists of clauses of the predicate `parse(Sentence, Tree)`. Each of the templates has a unique parse tree assigned to the sentences it generates, except for Template 8, which is treated as ambiguous. Indeed, it can be ‘the museum by the river’ that one wants to visit, or, alternatively, one can go to the museum by walking along the river. For each of the sentences corresponding to this template, there is a *pair* of `parse/2` clauses sharing the same first argument, i.e. referring to the same sentence, but assigning a different parse tree to it.

7 Results and Evaluation

The treebank T5000 has been split into a training set containing 4036 pairs `<sentence, parse tree>`, and a test set containing 1010 such pairs. Samples of different size have been subsequently taken from the training set, and LAPIS applied on them. For each training set sample, 2 different parsers were produced: (1) the CFG-based parser produced in Step 2 of the LAPIS algorithm; (2) the

specialised parser learned in Step 4. Two modifications of the latter were considered: (2a) the parser as is, and (2b) the parser in which the lexical semantic tags were not used as semantic constraints, i.e. it only differed from the CFG-based parser in its use of partially unfolded and lexicalised rules.

The performance of these three parsers was compared on the test data set for two outputs: the first parse, resp. all the parses produced for each sentence (see Figure 10). The two main criteria used for evaluation were accuracy and precision. If c is the number of correct parses, w the number of the wrong ones, and u the number of examples for which no parse is generated, then accuracy is defined as $acc = \frac{c}{c+w+u}$, and precision as $prec = \frac{c}{c+w}$. Additionally, we measured the percentage of sentences for which a parse was generated, $cov = \frac{c+w}{c+w+u}$. In the experiments where all possible parses were produced, the number of parses produced for each sentence $\#p$, and the number of correct parses per sentence $corr$ were also considered. All results are displayed in Figure 10.

Comparison between the performance of the CFG-based parser and the one learned with LAPIS is strongly in favour of the latter. Even from a training set so small as 80 examples, which approximately corresponds to only 6 examples per parse tree template, the specialised parser not using semantic constraints achieves 80.10% accuracy and 92.99% precision on the ‘first-parse-only’ task, and 78.91% accuracy and 97.08% precision when the semantic constraints are

Tr. trees		CFG-based		Specialised parser											
		1 st	aver.	Without semantic constraints								With sem. c.			
				First parse only				Average of all parses				First parse only			
no	[%]	acc	acc	acc	prec	cov	acc	prec	cov	#p	corr	acc	prec	cov	
40	1	5.05	11.30	81.68	90.86	89.90	90.86	90.86	100	1.09	0.997	80.69	90.86	88.81	
80	2	5.05	11.30	80.10	92.99	86.14	92.99	92.99	100	1.00	0.930	78.91	97.08	81.29	
121	3	5.05	11.30	91.98	95.28	96.53	95.91	95.91	100	1.04	0.986	86.14	98.98	87.03	
202	5	5.05	11.30	91.29	93.79	97.33	92.07	92.07	100	1.28	1.145	91.49	95.55	95.74	
404	10	5.05	11.30	92.38	94.24	98.02	92.54	92.54	100	1.25	1.096	91.49	95.06	96.23	
807	20	5.05	11.30	94.46	94.46	100.00	92.81	92.81	100	1.60	1.424	92.77	94.55	98.12	
1211	30	5.05	11.30	93.56	93.66	99.90	92.58	92.58	100	1.34	1.159	92.18	94.14	97.92	
1615	40	5.05	11.30	94.46	94.74	99.70	92.80	92.80	100	1.36	1.168	93.96	95.28	98.61	
2018	50	5.05	11.30	94.16	94.16	100.00	92.64	92.64	100	1.41	1.219	93.47	94.68	98.71	
2421	60	5.05	11.30	94.16	94.16	100.00	92.62	92.62	100	1.24	1.059	93.76	94.79	98.91	
2825	70	5.05	11.30	94.36	94.36	100.00	92.90	92.90	100	1.26	1.072	93.56	94.59	98.91	
3228	80	5.05	11.30	94.36	94.36	100.00	93.15	93.15	100	1.34	1.157	93.66	94.69	98.91	
3632	90	5.05	11.30	94.36	94.36	100.00	93.02	93.01	100	1.49	1.300	93.66	94.69	98.91	
3834	95	5.05	11.30	94.46	94.46	100.00	92.99	92.99	100	1.36	1.173	93.96	94.99	98.91	
4036	100	5.05	11.30	94.36	94.36	100.00	92.99	92.99	100	1.36	1.172	93.66	94.69	98.91	

Legend: $acc = \frac{\#correct\ parses}{\#test\ sentences} * 100 [\%]$

$prec = \frac{\#correct\ parses}{\#parsed\ sentences} * 100 [\%]$

$cov = \frac{\#parsed\ sentences}{\#test\ sentences} * 100 [\%]$

$\#p = \frac{\#parses\ produced}{\#test\ sentences}$

$corr = \frac{\#correct\ parses}{\#test\ sentences}$

Fig. 10. Results

used, as compared to a baseline of just 5.05% accuracy and precision for the CFG parser.

Using the lexical semantic constraints in the specialised parser in general decreases accuracy ($\Delta acc \in < -5.84\%, +0.20\% >$), since it decreases the number of covered sentences. However, the precision is almost always increased: ($\Delta prec \in < 0.00\%, +4.09\% >$) thus confirming the positive effect of the semantic constraints learned. Even when the semantic constraints are not applied, the average number of parses produced for each sentence $1.00 < \#p < 1.49$ shows reduction of the number of parses by order of magnitude 1. For the extreme case $\#p = 1.49$ an average number of 1.3 parses are correct, due to the presence in the corpus of some semantically ambiguous sentences, and also probably because some of the specialised clauses representing parser actions partially overlap.

8 Conclusions

The combination of LAPIS and WordNet proves its efficiency for the learning of specialised parsers, containing a very limited amount of nondeterminism. The dataset used, a treebank of semantically annotated parse trees, along with the semantic lexicon WordNet leads to the construction of parsers based on grammars with partially lexicalised and/or unfolded rules, with the additional help of lexical semantic constraints. A promising future line of research is to use treebanks with all nodes annotated with their semantics in logic form, with the possible result being a parser which only produces parse trees corresponding to valid semantic interpretations of the whole sentence.

References

1. Alfred Aho, Ravi Sethi, and Jeffrey Ullman. *Compilateurs - Principes, techniques et outils*. InterEditions, Paris, 1989.
2. Dimitar Kazakov. An inductive approach to natural language parser design. In Kemal Oflazer and Harold Somers, editors, *Proceedings of the Second International Conference on New Methods in Natural Language Processing*, pages 209–217, Ankara, Turkey, September 1996. Bilkent University. ISBN 975-7679-16-X.
3. George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Introduction to WordNet: An on-line lexical database. Technical report, University of Princeton, 1993.
4. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
5. G. Plotkin. *A note of inductive generalization: in Machine Intelligence 5*. Edinburgh University Press, 1969.
6. Ch. Samuelsson. *Fast Natural-Language Parsing Using Explanation-Based Learning*. PhD thesis, The Royal Institute of Technology and Stockholm University, 1994.
7. John M. Zelle and Raymond J. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 817–822. AAI Press/MIT Press, 1993.
8. John M. Zelle and Raymond J. Mooney. Inducing deterministic Prolog parsers from treebanks: A machine learning approach. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 748–753. AAI Press/MIT Press, 1994.

An Introduction to Inductive Logic Programming and Learning Language in Logic^{*}

Sašo Džeroski¹, James Cussens², and Suresh Manandhar²

¹ Jožef Stefan Institute

Jamova 39, 1000 Ljubljana, Slovenia

² Department of Computer Science, University of York
Heslington, York, YO10 5DD, UK

1 Introduction

Learning Language in Logic (LLL) is an emerging research area lying at the intersection of computational logic, machine learning and natural language processing (Fig 1). To see what is to be gained by focussing effort at this intersection begin by considering the role of logic in natural language processing (NLP). The flexibility and expressivity of logic-based representations have led to much effort being invested in developing logic-based resources for NLP. However, manually developed logic language descriptions have proved to be brittle, expensive to build and maintain, and still do not achieve high coverage on unrestricted text. As a result, rule-based natural language processing is currently viewed as too costly and fragile for commercial applications. Natural language learning based on statistical approaches, such as the use of n -gram models for part-of-speech tagging or lexicalised stochastic grammars for robust parsing, partly alleviate these problems. However, they result in linguistically impoverished descriptions that are difficult to edit, extend and interpret.

Work in LLL attempts to overcome both the limitations of existing statistical approaches and the brittleness of hand-crafted rule-based methods. LLL is, roughly, the application of Inductive Logic Programming (See Section 3) to NLP. ILP works within a logical framework but uses data and logically encoded background knowledge to learn or revise logical representations. From the NLP point of view the promise of ILP is that it will be able to steer a mid-course between the two alternatives of large scale, but shallow levels of analysis, and small scale, but deep and precise analyses, and produce a better balance between breadth of coverage and depth of analysis. From the ILP point of view, NLP is an ideal application area. The existence within NLP problems of hierarchically defined, structured data with large amounts of relevant, often logically defined, background knowledge provides a perfect testbed for stretching ILP technology in a way that would also be beneficial in other application areas.

The aim of this introductory chapter is to provide the background required to make ILP and LLL accessible to the widest possible audience. The sections

^{*} This is a non-final version of a chapter of *Learning Language in Logic*, Copyright, Springer, 2000. Distributed at ESSLLI-2000 with kind permission of Springer.

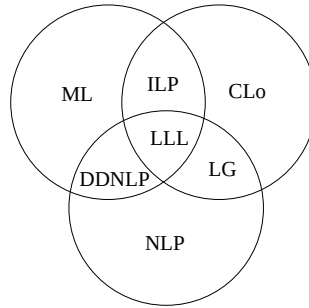


Fig. 1. Situating LLL (CLo = computational logic, ML = machine learning, DDNLP = data-driven NLP, LLL = learning language in logic, LG = logic grammars, NLP = natural language processing, ILP = inductive logic programming.)

on logic programming and ILP (Sections 2–8) give a concise account of the main ideas in the hope that, for example, computational linguists unfamiliar with either or both of these topics can appreciate what LLL has to offer. Section 9 examines which NLP problems are most suitable as LLL application areas. In both cases, we start our explanations from square one so that quite elementary topics are covered. The compressed nature of the expositions mean that readers will have much to gain by following up this chapter with one of the many papers cited in the text.

Section 11 tries to give a coherent picture of the contributions contained in this volume by picking out themes and pointing to the distinctive characteristics that of this research from an ILP perspective. We conclude by pointing to possible future work in Section ??.

2 Introduction to logic programming

In this section, the basic concepts of logic programming are introduced. These include the language (syntax) of logic programs, as well as basic notions from model and proof theory. The syntax defines what are legal sentences/statements in the language of logic programs. Model theory is concerned with assigning meaning (truth values) to such statements. Proof theory focuses on (deductive) reasoning with such statements. For a thorough treatment of logic programming we refer to the standard textbook of Lloyd (1987). The overview below is mostly based on the comprehensive and easily readable text by Hogger (1990).

2.1 The language

A first-order alphabet consists of variables, predicate symbols and function symbols (which include constants). A variable is a term, and a function symbol immediately followed by a bracketed n -tuple of terms is a term. Thus $f(g(X), h)$

is a term when f , g and h are function symbols and X is a variable—strings starting with lower-case letters denote predicate and function symbols, while strings starting with upper-case letters denote variables. A constant is a function symbol of arity 0 (i.e. followed by a bracketed 0-tuple of terms, which is usually left implicit). A predicate symbol immediately followed by a bracketed n -tuple of terms is called an atomic formula or atom. For example, $mother(maja, filip)$ and $father(X, Y)$ are atoms.

A well-formed formula is either an atomic formula or takes one of the following forms: (F) , $\overline{F}$, $F \vee G$, $F \wedge G$, $F \leftarrow G$, $F \leftrightarrow G$, $\forall X : F$ and $\exists X : F$, where F and G are well-formed formulae and X is a variable. $\overline{F}$ denotes the negation of F , $\vee$ denotes logical disjunction (or), and $\wedge$ logical conjunction (and). $F \leftarrow G$ stands for implication (F if G , $F \vee \overline{G}$) and $F \leftrightarrow G$ stands for equivalence (F if and only if G). $\forall$ and $\exists$ are the universal (for all X F holds) and existential quantifier (there exists an X such that F holds). In the formulae $\forall X : F$ and $\exists X : F$, all occurrences of X are said to be bound. A sentence or a closed formula is a well-formed formula in which every occurrence of every variable symbol is bound. For example, $\forall Y \exists X father(X, Y)$ is a sentence, while $father(X, andy)$ is not.

Clausal form is a normal form for first-order sentences. A clause is a disjunction of literals—a positive literal is an atom, a negative literal the negation of an atom—preceded by a prefix of universal quantifiers, one for each variable appearing in the disjunction. In other words, a clause is a formula of the form $\forall X_1 \forall X_2 \dots \forall X_s (L_1 \vee L_2 \vee \dots \vee L_m)$, where each L_i is a literal and $X_1, X_2, \dots, X_s$ are all the variables occurring in $L_1 \vee L_2 \vee \dots \vee L_m$.

A clause can also be represented as a finite set (possibly empty) of literals. The set $\{A_1, A_2, \dots, A_h, \overline{B_1}, \overline{B_2}, \dots, \overline{B_b}\}$, where A_i and B_i are atoms, stands for the clause $(A_1 \vee \dots \vee A_h \vee \overline{B_1} \vee \dots \vee \overline{B_b})$, which is equivalently represented as $A_1 \vee \dots \vee A_h \leftarrow B_1 \wedge \dots \wedge B_b$. Most commonly, this same clause is written as $A_1, \dots, A_h \leftarrow B_1, \dots, B_b$, where $A_1, \dots, A_h$ is called the head and $B_1, \dots, B_b$ the body of the clause. A finite set of clauses is called a clausal theory and represents the conjunction of its clauses.

A clause is a Horn clause if it contains at most one positive literal; it is a definite clause if it contains exactly one positive literal. A set of definite clauses is called a definite logic program. A fact is a definite clause with an empty body, e.g., $parent(mother(X), X) \leftarrow$, also written simply as $parent(mother(X), X)$. A goal (also called a query) is a Horn clause with no positive literals.

A program clause is a clause of the form $A \leftarrow L_1, \dots, L_m$ where A is an atom, and each of $L_1, \dots, L_m$ is a positive or negative literal. A negative literal in the body of a program clause is written in the form *not* B , where B is an atom. A normal program (or logic program) is a set of program clauses. A predicate definition is a set of program clauses with the same predicate symbol (and arity) in their heads.

Let us now illustrate the above definitions with some examples. The clause

$$daughter(X, Y) \leftarrow female(X), mother(Y, X).$$

is a definite program clause, while the clause

$$daughter(X, Y) \leftarrow not\ male(X), father(Y, X).$$

is a normal program clause. Together, the two clauses constitute a predicate definition of the predicate *daughter*/2. This predicate definition is also a normal logic program. The first clause is an abbreviated representation of the formula

$$\forall X \forall Y : daughter(X, Y) \vee \overline{female(X)} \vee \overline{mother(Y, X)}$$

and can also be written in set notation as

$$\{daughter(X, Y), \overline{female(X)}, \overline{mother(Y, X)}\}$$

The set of variables in a term, atom or clause F , is denoted by $vars(F)$. A substitution $\theta = \{V_1/t_1, \dots, V_n/t_n\}$ is an assignment of terms t_i to variables V_i . Applying a substitution θ to a term, atom, or clause F yields the instantiated term, atom, or clause $F\theta$ where all occurrences of the variables V_i are simultaneously replaced by the term t_i . A term, atom or clause F is called ground when there is no variable occurring in F , i.e., $vars(F) = \emptyset$. The fact $daughter(maty, ann)$ is thus ground.

A clause or clausal theory is called function free if it contains only variables as terms, i.e., contains no function symbols (this also means no constants). The clause $daughter(X, Y) \leftarrow female(X), mother(Y, X)$ is function free and the clause $even(s(s(X))) \leftarrow even(X)$ is not. A Datalog clause (program) is a definite clause (program) that contains no function symbols of non-zero arity. This means that only variables and constants can be used as predicate arguments. The size of a term, atom, clause, or a clausal theory T is the number of symbols that appear in T , i.e., the number of all occurrences in T of predicate symbols, function symbols and variables.

2.2 Model theory

Model theory is concerned with attributing meaning (truth value) to sentences in a first-order language. Informally, the sentence is mapped to some statement about a chosen domain through a process known as interpretation. An interpretation is determined by the set of ground facts (ground atomic formulae) to which it assigns the value true. Sentences involving variables and quantifiers are interpreted by using the truth values of the ground atomic formulae and a fixed set of rules for interpreting logical operations and quantifiers, such as “ $\overline{F}$ is true if and only if F is false”.

An interpretation which gives the value true to a sentence is said to satisfy the sentence; such an interpretation is called a model for the sentence. An interpretation which does not satisfy a sentence is called a counter-model for that sentence. By extension, we also have the notion of a model (counter-model) for a set of sentences (e.g., for a clausal theory): an interpretation is a model for the

set if and only if it is a model for each of the set's members. A sentence (set of sentences) is satisfiable if it has at least one model; otherwise it is unsatisfiable.

A sentence F logically implies a sentence G if and only if every model for F is also a model for G . We denote this by $F \models G$. Alternatively, we say that G is a logical (or semantic) consequence of F . By extension, we have the notion of logical implication between sets of sentences.

A Herbrand interpretation over a first-order alphabet is a set of ground facts constructed with the predicate symbols in the alphabet and the ground terms from the corresponding Herbrand domain of function symbols; this is the set of ground atoms deemed to be true by the interpretation. A Herbrand interpretation I is a model for a clause c if and only if for all substitutions θ such that $c\theta$ is ground $body(c)\theta \subset I$ implies $head(c)\theta \cap I \neq \emptyset$. In that case, we say c is true in I . A Herbrand interpretation I is a model for a clausal theory T if and only if it is a model for all clauses in T . We say that I is a Herbrand model of c , respectively T .

Roughly speaking, the truth of a clause c in a (finite) interpretation I can be determined by running the goal (query) $body(c), not\ head(c)$ on a database containing I , using a theorem prover such as PROLOG. If the query succeeds, the clause is false in I ; if it fails, the clause is true. Analogously, one can determine the truth of a clause c in the minimal (least) Herbrand model of a theory T by running the goal $body(c), not\ head(c)$ on a database containing T .

To illustrate the above notions, consider the Herbrand interpretation

$$i = \{parent(saso, filip), parent(maja, filip), son(filip, saso), son(filip, maja)\}$$

The clause $c = parent(X, Y) \leftarrow son(Y, X)$ is true in i , i.e., i is a model of c . On the other hand, i is not a model of the clause $parent(X, X) \leftarrow$ (which means that everybody is their own parent).

2.3 Proof theory

Proof theory focuses on (deductive) reasoning with logic programs. Whereas model theory considers the assignment of meaning to sentences, proof theory considers the generation of sentences (conclusions) from other sentences (premises). More specifically, proof theory considers the derivability of sentences in the context of some set of inference rules, i.e., rules for sentence derivation. An inference rule has the following schematic form: "from a set of sentences of that kind, derive a sentence of this kind". Formally, an inference system consists of an initial set S of sentences (axioms) and a set R of inference rules.

Using the inference rules, we can derive new sentences from S and/or other derived sentences. The fact that sentence s can be derived from S is denoted $S \vdash s$. A proof is a sequence $s_1, s_2, \dots, s_n$, such that each s_i is either in S or derivable using R from S and $s_1, \dots, s_{i-1}$. Such a proof is also called a derivation or deduction. Note that the above notions are of entirely syntactic nature. They are directly relevant to the computational aspects of automated deductive inference.

The set of inference rules R defines the derivability relation $\vdash$. A set of inference rules is sound if the corresponding derivability relation is a subset of the logical implication relation, i.e., for all S and s , if $S \vdash s$ then $S \models s$. It is complete if the other direction of the implication holds, i.e., for all S and s , if $S \models s$ then $S \vdash s$. The properties of soundness and completeness establish a relation between the notions of syntactic ($\vdash$) and semantic ($\models$) entailment in logic programming and first-order logic. When the set of inference rules is both sound and complete, the two notions coincide.

Resolution comprises a single inference rule applicable to clausal-form logic. From any two clauses having an appropriate form, resolution derives a new clause as their consequence. For example, the clauses

$$\begin{aligned} &daughter(X, Y) \leftarrow female(X), parent(Y, X) \\ &female(sonja) \leftarrow \end{aligned}$$

resolve into the clause

$$daughter(sonja, Y) \leftarrow parent(Y, sonja)$$

Resolution is sound: every resolvent is implied by its parents. It is also refutation complete: the empty clause is derivable by resolution from any set S of Horn clauses if S is unsatisfiable. Resolution is discussed further in Section 3.

2.4 Logic programming for NLP

Using clauses Even when restricted to clausal form, the language of first-order logic is sufficiently flexible to cleanly represent a wide range of linguistic information. For example, grammar rules are easy to represent as clauses. Consider the grammar rule

$$VP \rightarrow VP \text{ MOD}$$

where VP stands for verb phrase and MOD stands for modifier. If a grammar can establish that the word *ran* constitutes a verb phrase and that the word *quickly* constitutes a modifier, then this rule allows us to infer that *ran quickly* is a verb phrase.

We can represent this rule by the definite clause

$$vp(Start, End) \leftarrow vp(Start, Middle), mod(Middle, End)$$

where the variables stand for vertices in between words. For example if, from the sentence $_0 \text{ She } _1 \text{ ran } _2 \text{ quickly } _3$, we derive $vp(1, 2), mod(2, 3)$, then the given clause allows us to infer $vp(1, 3)$. A clause such as this, where a predicate appears in both the head and the body is called a recursive clause.

Grammars represented as a set of definite clauses are, unsurprisingly, known as definite clause grammars (DCGs). The logic programming language PROLOG has a special DCG syntax so that the PROLOG clause

$$vp(Start, End) :- vp(Start, Middle), mod(Middle, End).$$

(where the $:-$ represents a $\leftarrow$) may be more compactly written as

$$vp \text{ --> } vp, \text{ mod.}$$

Using terms First-order terms are important in representing structured information. The most common sort of term used in logic programming is the list. Lists are recursive terms: a list is either the empty list or of the form $.(Head, Tail)$, where $Tail$ is a list. In PROLOG, $.(Head, Tail)$ can be more intuitively represented as $[Head|Tail]$. Sentences are often represented as lists, so *She ran quickly* would be represented in PROLOG as $['She'|[ran|[quickly|[]]]]$. In fact, as a convenience, PROLOG allows one to write this as $['She',ran,quickly]$. Note that the constant *She* has been quoted since otherwise its initial capital letter would mean that it would be interpreted as a variable.

Structured terms allow great representational flexibility. For example, in the grammar framework used by Cussens and Pulman (this volume), it is more convenient to represent the grammar rule

$$VP \rightarrow VP \text{ MOD}$$

as the following non-ground fact, using terms in place of literals

cmp_synrule(vp(Start, End), [vp(Start, Middle), mod(Middle, End)]).

First-order terms are sufficiently flexible to represent semantic, as well as syntactic, information. Boström (this volume) uses ILP to translate between quasi-logical forms (QLFs)—terms intended to represent the meaning of sentences. QLFs are often very complex; here is one representing the meaning of the sentence *List the prices*

```
[imp, form(_, verb(no, no, no, imp, y), A,
    B^ [B, [list_Enumerate, A,
        term(_, ref(pro, you, _, l([])), _,
            C^ [personal, C], _, _),
        term(_, q(_, bare, plur), _,
            D^ [fare_Price, D], _, _)]], _)]
```

The underscores are a PROLOG convention used to represent ‘anonymous variables’, each underscore is implicitly replaced by a distinct variable.

3 Introduction to ILP

While logic programming (and in particular its proof theory) is concerned with deductive inference, inductive logic programming is concerned with inductive inference. It generalizes from individual examples/observations in the presence of background knowledge, finding regularities/hypotheses. The most commonly addressed task in ILP is the task of learning logical definitions of relations (Quinlan, 1990), where tuples that belong or do not belong to the target relation are given as examples. ILP then induces a logic program (predicate definition) defining the target relation in terms of other relations that are given as background knowledge.

We assume a set of examples is given, i.e., tuples that belong to the target relation p (positive examples) and tuples that do not belong to p (negative examples). Also given are background relations (or background predicates) q_i that constitute the background knowledge and can be used in the learned definition of p . Finally, a hypothesis language, specifying syntactic restrictions on the definition of p is also given (either explicitly or implicitly). The task is to find a definition of the target relation p that is consistent and complete. Informally, it has to explain all the positive and none of the negative examples.

More formally, we have a set of examples $E = P \cup N$, where P contains positive and N negative examples, and background knowledge B . The task is to find a hypothesis H such that $\forall e \in P : B \wedge H \models e$ (H is complete) and $\forall e \in N : B \wedge H \not\models e$ (H is consistent). This setting was introduced by Muggleton (1991). In an alternative setting proposed by De Raedt and Džeroski (1994), the requirement that $B \wedge H \models e$ is replaced by the requirement that H be true in the minimal Herbrand model of $B \wedge e$: this setting is called learning from interpretations.

In the most general formulation, each e , as well as B and H can be a clausal theory. In practice, each e is most often a ground example and H and B are definite logic programs. Recall that $\models$ denotes logical implication (semantic entailment). Semantic entailment ($\models$) is in practice replaced with syntactic entailment ($\vdash$) / provability, where the resolution inference rule (as implemented in PROLOG) is most often used to prove examples from a hypothesis and the background knowledge.

As an illustration, consider the task of defining the relation $daughter(X, Y)$, which states that person X is a daughter of person Y , in terms of the background knowledge relations $female$ and $parent$. These relations, as well as two more background relations $mother$ and $father$ are given in Table 1. There are two positive and two negative examples of the *target* relation $daughter$.

Table 1. A simple ILP problem: learning the *daughter* relation.

<i>Training examples</i>	<i>Background knowledge</i>
$daughter(mary, ann). \oplus$	$mother(ann, mary). \quad female(ann).$
$daughter(eve, tom). \oplus$	$mother(ann, tom). \quad female(mary).$
$daughter(tom, ann). \ominus$	$father(tom, eve). \quad female(eve).$
$daughter(eve, ann). \ominus$	$father(tom, ian).$
	$parent(X, Y) \leftarrow mother(X, Y)$
	$parent(X, Y) \leftarrow father(X, Y)$

In the hypothesis language of definite program clauses it is possible to formulate the following definition of the target relation,

$$daughter(X, Y) \leftarrow female(X), parent(Y, X).$$

which is consistent and complete with respect to the background knowledge and the training examples.

In general, depending on the background knowledge, the hypothesis language and the complexity of the target concept, the target predicate definition may consist of a set of clauses, such as

$$\begin{aligned} daughter(X, Y) &\leftarrow female(X), mother(Y, X). \\ daughter(X, Y) &\leftarrow female(X), father(Y, X). \end{aligned}$$

The hypothesis language is typically a subset of the language of program clauses. Since the complexity of learning grows with the expressiveness of the hypothesis language, restrictions have to be imposed on hypothesised clauses. Typical restrictions include a bound on the number of literals in a clause and restrictions on variables that appear in the body of the clause but not in its head (so-called new variables).

ILP systems typically adopt the covering approach of rule induction systems. In a main loop, they construct a clause explaining some of the positive examples, add this clause to the hypothesis, remove the positive examples explained and repeat this until all positive examples are explained (the hypothesis is complete). In an inner loop, individual clauses are constructed by (heuristically) searching the space of possible clauses, structured by a specialization or generalization operator. Typically, search starts with a very general rule (clause with no conditions in the body), then proceeds to add literals (conditions) to this clause until it only covers (explains) positive examples (the clause is consistent). This search can be bound from below by using so-called bottom clauses, constructed by least general generalization or inverse resolution/entailment. We discuss these issues in detail in Sections 6–8.

When dealing with incomplete or noisy data, which is most often the case, the criteria of consistency and completeness are relaxed. Statistical criteria are typically used instead. These are based on the number of positive and negative examples explained by the definition and the individual constituent clauses.

4 ILP for LLL

In many cases, ‘standard’ single predicate learning from positive and negative examples as outlined in Section 3 can be used fairly straightforwardly for natural language learning (NLL) tasks. For example, many of the papers reviewed by Eineborg and Lindberg (this volume) show how the standard ILP approach can be applied to part-of-speech tagging.

In this section, we outline NLL problems that require a more complex ILP approach. We will use grammar learning as a running example. Consider the very simple language learning task described in Table 2. The background knowledge is a set of facts about the sentence *She ran quickly*, which, in practice, would have been derived using a background grammar and lexicon, which we do not include here.

An ILP algorithm would easily construct the following single clause theory H :

$$vp(S, F) \leftarrow vp(S, M), mod(M, F)$$

Table 2. A simple LLL problem: learning VPs (verb phrases).

<i>Training examples</i>		<i>Background knowledge</i>	
$vp(1, 3)$	$\oplus$	$np(0, 1)$	$word('She', 0, 1)$
		$vp(1, 2)$	$word(ran, 1, 2)$
		$mod(2, 3)$	$word(quickly, 2, 3)$

which represents the grammar rule $VP \rightarrow VP \text{ MOD}$ and is consistent and complete with respect to the background knowledge and the training example. However many other clauses are also consistent and complete, such as:

$$\begin{aligned}
vp(1, 3) &\leftarrow \\
vp(1, 3) &\leftarrow word(ran, 1, 2) \\
vp(1, 3) &\leftarrow vp(1, 2), mod(2, 3) \\
vp(1, 3) &\leftarrow word(ran, 1, 2), word(quickly, 2, 3) \\
vp(S, F) &\leftarrow \\
vp(S, F) &\leftarrow word(ran, S, M) \\
vp(S, F) &\leftarrow vp(S, M) \\
vp(S, F) &\leftarrow word(ran, S, M), word(quickly, M, F)
\end{aligned}$$

So an ILP algorithm needs something more than consistency and completeness if it is to select the correct hypothesis. This problem stems from a common feature of NLP learning tasks: the absence of explicit negative examples. This absence allowed the wildly overgeneral rule $vp(S, F) \leftarrow$ (VPs are everywhere!) to be consistent. A suitably defined hypothesis language which simply disallows many overgeneral clauses can only partially overcome this problem. One option is to invoke a Closed World Assumption (CWA) to produce implicit negative examples. In our example LLL problem this would amount to the (correct) assumption that VPs only occur at the places stated in the examples and background knowledge. We make these negative examples explicit in Table 3. Although correct in this case, such an assumption will not always be so. If the fact $vp(1, 2)$ were missing from background knowledge the CWA would incorrectly assert that the clause $vp(1, 2)$ were false.

Since a CWA is generally too strong, an alternative is to assume that *most* other potential examples are negative. Muggleton (2000) gives a probabilistic method for learning from positive examples where the probability of a hypothesis decreases with its generality and increases with its compactness. Boström (1998) also uses a probabilistic approach to positive-only learning based on learning Hidden Markov Models (HMMs).

A different approach rests on the recognition that grammar learning, and many other NLL tasks, are not primarily classification tasks. In this view a grammar is not primarily concerned with differentiating sentences (positive examples) from non-sentences (negative examples). Instead, given a string of words

Table 3. A simple LLL problem: learning VPs (verb phrases), with explicit negatives.

<i>Training examples</i>		<i>Background knowledge</i>
$vp(1, 3)$	$\oplus$	$np(0, 1) \quad word('She', 0, 1)$
$vp(0, 1)$	$\ominus$	$vp(1, 2) \quad word(ran, 1, 2)$
$vp(0, 2)$	$\ominus$	$mod(2, 3) \quad word(quickly, 2, 3)$
$vp(0, 3)$	$\ominus$	
$vp(1, 3)$	$\ominus$	
$vp(2, 3)$	$\ominus$	

assumed to be a sentence, the goal is to find the correct *parse* of that sentence. From this view, the problem with $vp(S, F) \leftarrow$ is not so much that it may lead to non-sentences being incorrectly classified as sentences; but rather that it will lead to a given sentence having many different parses—making it harder to find the correct parse. A shift from classification to disambiguation raises many important issues for the application of ILP to NLP which we can not discuss here. This issue is addressed in the papers by Mooney (this volume) and Thompson and Califf (this volume) which present the CHILL algorithm which ‘learns to parse’ rather than classify. Riezler’s paper (this volume) also focusses on disambiguation, employing a statistical approach.

So far we have been looking at cases where we learn from examples of a single target predicate. In grammar learning we may have to learn clauses for several predicates (multiple predicate learning) and also the examples may not be examples of any of the predicates we need to learn. Consider the learning problem described in Table 4. The target theory is the two clause theory:

$$\begin{aligned} v(X, Y) &\leftarrow word(ran, X, Y) \\ vp(S, F) &\leftarrow vp(S, M), mod(M, F) \end{aligned}$$

The problem in Table 4 is to induce a theory H with certain properties. For example, it must be in the hypothesis language, not be over-general, not increase the ambiguity of the grammar too much, etc. However here we will focus on the constraint that $s(0, 3)$ must follow from $B \wedge H$. Neither of our two clauses *directly* define the predicate $s/2$, the predicate symbol of our example. But we can use *abduction* to generate positive examples with the right predicate symbol.

By resolution, we can derive $s(0, 3) \leftarrow vp(1, 3)$ from B . It follows that if $B \wedge H \models vp(1, 3)$, then $B \wedge H \models s(0, 3)$. We can then treat $vp(1, 3)$ as it were a normal positive example and attempt to learn $vp(S, F) \leftarrow vp(S, M), mod(M, F)$.¹ Note that in using abduction we are just guessing that $vp(1, 3)$ is true. Abduction like all inductive operations is unsound: for example from $vp(1, 3)$ and $vp(X, Y) \leftarrow v(X, Y)$, which are both true, we can abductively derive $v(1, 3)$ which is not. Abductive approaches applied to LLL problems can be found in (Muggleton & Bryant, 2000; Cussens & Pulman, 2000) and Cussens and Pulman and Tjong and Nerbonne (this volume).

¹ Of course, an ILP algorithm would not know that this is the correct theory.

Table 4. Grammar completion

<i>Training examples</i>		<i>Background knowledge</i>
$s(0, 3)$	$\oplus$	$np(X, Y) \leftarrow word('She', X, Y)$ $word('She', 0, 1)$ $mod(X, Y) \leftarrow word(quickly, X, Y)$ $word(quickly, 2, 3)$ $s(X, Y) \leftarrow np(X, Z), vp(Z, Y)$ $word(ran, 1, 2)$ $vp(X, Y) \leftarrow v(X, Y)$

5 Decision List Learning for NLP

Propositional decision lists

Quinlan's Work FOIL

First-order decision lists are essentially Prolog clauses in which every clause ends in a cut (!). More generally decision lists can be thought of as an ordered set of clauses with specific clauses at the top and general clauses at the bottom. For example, Figure 2 shows a decision list learnt by CLOG (?) from the list of examples given in Figure 3.

```
plural([s,p,y],[s,p,i,e,s]):- !.
plural(X,Y) :- mate(X,Y,[],[],[a,n],[e,n]),!.
plural(X,Y) :- mate(X,Y,[],[],[a,s,s],[a,s,s,e,s]),!.
plural(X,Y) :- mate(X,Y,[],[],[],[s]),!.
```

Fig. 2. Rules induced by CLOG from the data in Figure 3.

The first clause in Figure 2 is an exception. Although the general rule in English is to change the ending *-y* to *-ies*, CLOG was not able to learn this rule because there is only one example in the training set (in Figure 3) showing this pattern. The second rule states that if a word ends in an *-an* then the plural form replaces it with *-en* (see Figure ?? for the definition of the `mate/6` predicate). The last rule is the default rule that will apply if the previous rules fail. This default rule will add an *-s* ending to any word.

There are several reasons that make decision lists attractive for language learning applications. We summarise some of these below:

1. Unlike the case for learning pure logic programs which usually require negative examples, decision lists can be learned from positive only data. This is especially relevant for natural language learning since negative examples are usually not available in most language corpora.
2. Again unlike the case for pure logic programs which can return multiple answers nondeterministically, decision lists automatically return the most likely answer when multiple solutions are possible. For example, in a tagging application, given the left and the right context of word, one is interested in the most probable tag and not the set of plausible tags.

```

plural([l,i,p], [l,i,p,s]).
plural([m,e,m,b,e,r], [m,e,m,b,e,r,s]).
plural([d,a,y], [d,a,y,s]).
plural([s,e,c,o,n,d], [s,e,c,o,n,d,s]).
plural([o,t,h,e,r], [o,t,h,e,r,s]).
plural([l,i,e], [l,i,e,s]).
plural([m,a,s,s], [m,a,s,s,e,s]).
plural([c,l,a,s,s], [c,l,a,s,s,e,s]).
plural([s,p,y], [s,p,i,e,s]).
plural([m,a,n], [m,e,n]).
plural([w,o,m,a,n], [w,o,m,e,n]).
plural([f,a,c,e], [f,a,c,e,s]).

```

Fig. 3. Sample training data set for CLOG.

3. In common to all ILP approaches, the first-order representation means that a richer set of rules can be learnt as opposed to rules that only employ simpler propositional representations. This means that first-order decision lists can serve as a representation language for a wide range of linguistic applications.
4. In common to all ILP approaches, decision list rules can easily be understood and manipulated by linguists.
5. As existing work (?, ?, ?) demonstrate, there exists relatively efficient decision list learning algorithms.

Output Completeness is a notion introduced by Mooney (?) which allows a decision-list learning system to learn from positive only data - an important consideration for language learning applications. We illustrate the notion of output completeness using a simple example.

```

plural([f,i,s,h], [f,i,s,h]).
plural([f,i,s,h], [f,i,s,h,e,s]).
plural([l,i,p], [l,i,p,s]).

```

In the above example, there are *two* possible plural forms for *fish* whereas there is only one plural form for *lip*. Given these examples, under the output completeness assumption, every example of the form `plural([l,i,p], Y)` where $Y \neq [l,i,p,s]$ is a negative example. Similarly, every example of the form `plural([f,i,s,h], Y)` where $Y \neq [f,i,s,h]$ and $Y \neq [f,i,s,h,e,s]$ is a negative example. Thus, output completeness assumption allows us to assume that the training data set contains, for every example, the complete set of output values, for every input value present in the data set.

In the following we provide an overview of the decision list learning system, CLOG, that has successfully been applied for morphology learning and for

transformation rule learning for base NP bracketing (also known as NP chunking) (?).

CLOG, described in Figure 4, is a system for learning first-order decision lists. CLOG shares a fair amount of similarity with FOIDL (?). In particular, both employ the notion of output completeness to compensate for the lack of negative examples. The central difference between FOIDL and CLOG is that FOIDL is pure top-down learning algorithm whereas CLOG is a example-driven decision list learning system which learns rules by generalising from examples.

The input to CLOG is a list of examples. For instance, the examples in Figure 3 describe the plural of some common English nouns. From the training examples we are interested in learning an ordered set of rules. For example, given appropriate background knowledge, from the examples in Figure 3, CLOG will produce the set of rules and exceptions given in Figure 2.

The CLOG algorithm is a clause-at-a-time sequential cover algorithm (?), which learns a decision-list starting with the last clause. CLOG employs bottom-up guidance only to consider generalisations that are relevant to an example. This is similar to the strategy used in Progol (Muggleton, 1995a).

Currently, these generalisations are generated by the user-defined predicate `generate_clauses(+Example,-Clauses)`. As an example, consider the following scheme used by (?). Note that the definition of `generate_clauses` here is different from the one used in the experiments described in the next section. The following example is provided as an example for its ease of understanding.

```
generate_clauses(plural(U,V),Clauses):-
    bagof( ( plural(X,Y) :- (mate(X,Y,P1,P2,S1,S2), !)),
           mate(U,V,P1,P2,S1,S2),
           Clauses).
```

Here `mate(U,V,P1,P2,S1,S2)` is true if there exists `Stm` such that `U = P1+Stm+S1` and `V = P2+Stm+S2` ('+' here denotes concatenation). Intuitively, the plural of `U` is `V` if `P1` is a prefix of `U`; `P2` is a prefix of `V`; `S1` is a suffix of `U`; `S2` is a suffix of `V` and both `U` and `V` share a common stem `Stm`.

```
mate(W1,W2,[],[],S1,[]):- split(W1,W2,S1).
mate(W1,W2,[],[],[],S2):- split(W2,W1,S2).
mate(W1,W2,[],[],S1,S2):- split(W1,X,S1), split(W2,X,S2).
mate(W1,W2,P1,P2,S1,S2):- split(W1,P1,W11), split(W2,P2,W22),
                           split(W11,X,S1), split(W22,X,S2).

split([X,Y|Z],[X],[Y|Z]).
split([X|Y],[X|Z],W) :- split(Y,Z,W).
```

Given the query `mate([l,i,p],[l,i,p,s],P1,S1,P2,S2)`, the following solutions are obtained:

```
P1 = [], S1 = [], P2 = [], S2 = [s]
P1 = [], S1 = [], P2 = [i,p], S2 = [i,p,s]
P1 = [], S1 = [], P2 = [p], S2 = [p,s]
P1 = [l], S1 = [p], P2 = [p], S2 = [p,s]
```

Thus `generate_clauses(plural([l,i,p],[l,i,p,s]),Clauses)` will give:

```

Clauses = [ ( plural(X,Y) :- (mate(X,Y, [], [], [], [s]), !) ),
              ( plural(X,Y) :- (mate(X,Y, [], [], [i,p], [i,p,s]), !) ),
              ( plural(X,Y) :- (mate(X,Y, [], [], [p], [p,s]), !) ),
              ( plural(X,Y) :- (mate(X,Y, [1], [1], [p], [p,s]), !) )
            ]

```

Each of the above clauses are generalisations of the example `plural([1,i,p], [1,i,p,s])`.

The efficiency of CLOG comes from the fact that it does not use the generate-clause-then-evaluate strategy, on which learners such as FOIDL are based. Instead, CLOG employs a generate-all-relevant-clauses-then-evaluate-all strategy which computes in a single iteration the heuristic goodness of all clauses. This results in a very significant gain in efficiency. For the previous example, the training set will be used for the simultaneous evaluation of all four clauses to determine the best one with respect to the *gain* function (see below).

CLOG has lower memory requirements and can train from large data sets since it keeps only a single training example in memory at any one time. All remaining training examples are stored in files. This contrasts with most other ILP learners which keep all training examples in memory.

The set of generalisations of an example produced by the predicate `generate_clauses/2` is called a *generalisation set*. This is denoted by GC in Figure 4. CLOG cycles every input example through the generalisation set in a single iteration checking whether a candidate generalisation covers the example positively or negatively. Once this process is completed, the best candidate generalisation is chosen. The example set is pruned using this candidate and the cycle repeats. The following notation is used in Figure 4:

- QP denotes the number of examples covered correctly;
- QN denotes the number of examples incorrectly covered;
- SP denotes the number of previously covered examples covered correctly;
- SN denotes the number of previously covered examples incorrectly covered.

Some further explanation is in order to explain the CLOG algorithm given in Figure 4. `covers(G,e)` is a function that returns (1,0) if the clause G covers the example *e* correctly. It returns (0,1) if G applies to *e* but gives an incorrect answer. It returns (0,0) if G does not apply to *e*. These binary answers can be generalised to n-ary ones if application demands it.

In contrast to FOIDL, CLOG learns rules sequentially from the training examples, therefore, changing the ordering in the examples may result in a different ordering of the learnt rules.

The *gain* function used in CLOG is user-defined. For the morphology learning experiments in (?) we used the following gain function which favours the most compressive hypothesis:

$$gain = QP - SN$$

The experiments in (?) show that CLOG is significantly more efficient than FOIDL in the task of learning morphology rules. On the task of analysis of the plural of English nouns, the running times for FOIDL and CLOG are given in Table 5 (all times on a SUN SPARC 10).

```

Let Pos be the (positive) examples to be covered
Let Covered be the set of covered (positive) examples initially empty
Let DL be the decision list being learnt initially empty
While Pos not empty
do
  Let x be an (arbitrary) example in Pos
  Let GC = { (G,0,0,0,0) | G is a clause that covers x }
  For each example e in Pos
    For each (G,SP,SN,QP,QN) in GC
      covers(G, e) = (P,N);
      (G,SP,SN,QP,QN) := (G,SP,SN,QP+P,QN+N)
  For each example e in Covered
    For each (G,SP,SN,QP,QN) in GC
      covers(G, e) = (P,N);
      (G,SP,SN,QP,QN) := (G,SP+P,SN+N,QP,QN)
  Let Best ∈ GC be such that gain(Best) is maximum
  For each example e in Pos
    if Best(e) is correct
      then
        Covered := Covered ∪ {e} ;
        Pos := Pos − {e} ;
      endif
  For each example e in Covered
    if Best(e) is incorrect
      then
        Covered := Covered − {e} ;
        Pos := Pos ∪ {e} ;
      endif
  Add Best to top of DL
enddo

```

Fig. 4. CLOG algorithm

Training set size	100	200	300	400	500	600	700	800	900	1063
CLOG [secs]	21	61	112	204	277	477	646	941	1198	1771
FOIDL [secs]	254	1290	2191	5190	9521	18353	33915	—	—	—

Table 5. Running times for CLOG and FOIDL on the task of learning rules for the analysis of English nouns reported in (?).

5.1 Decision lists = Transformation lists

Transformation lists have been popularised by Brill's work (?) on learning of tagging rules. As far as the syntax for representing both transformation lists and decision lists is concerned there is no difference between the two. The only difference being that in a decision list once a rule has been applied to a test example no further changes to the example take place. On the other hand, in a transformation list, once a rule has been applied to a test example, the example is still subject to further rule application and it is not removed from the test set. In fact, the rules in a transformation list apply to all the examples in the test set repeatedly until no further application is possible. Thus a transformation list successively transforms an input set of examples until no further application is possible. The resulting set is the solution produced by the transformation list.

Given this background, it is straightforward to generalise learning of first order decision lists to learning of first order transformation lists thereby lifting Brill's transformation based learning into an ILP setting. The modification required to the CLOG algorithm is given in Figure 5.

As an example, consider the task of learning base NP bracketing or NP chunking rules introduced in (?). NP chunking is the task of bracketing simple non-overlapping noun phrases in a text. This bracketing can be represented by annotating every word with one of the following three labels: I if the word is inside a base NP; O if the word is outside a base NP; B if the word is the start of a base NP and the previous word is also part of a base NP (?).

For example, the following bracketing and labelling are equivalent:

In [the 1988 third quarter], [the Chicago-based oil company] earned [\$ 552 million], or [\$ 1.07] [a share].

In/O the/I 1988/I third/I quarter/I ,/O the/I Chicago-based/I oil/I company/I earned/O \$/I 552/I million/I ,/O or/O \$/I 1.07/I a/B share/I ./O

Initially a tagger such as the Brill tagger is employed to tag a sentence to be chunked. Then every noun (annotated as NN, NNS, NNP or NNPS), every pronoun (annotated as PRP, WP or EX), and every determiner (annotated as DT, CD or WDT) is itself a whole noun phrase. Given this initial annotation, we are interested in learning rules such as the following, based on the left and right context taking into account the PoS tag and the current IOB annotation:

- Move an opening bracket one position to the left


```
substantial/JJ/O improvement/NN/I from /IN/O July/NNP/I
substantial/JJ/I improvement/NN/I from /IN/O July/NNP/I
```

```
np_bracket(SIn, SOut, P, N):-
  move(left,LeftContext,item(Word,Tag,IOB),RightContext,SIn,SOut,P,N).
```
- Move an closing bracket one position to the right


```
the/DT/I man/NN/I laughing/VBG/O the/DT/I man/NN/I laughing/VBG/I
```

```
np_bracket(SIn, SOut, P, N):-
  move(right,LeftContext,item(Word,Tag,IOB),RightContext,SIn,SOut,P,N).
```

```

– Merge two noun phrases
  the/DT/I pound/NN/B is/vbz/O the/DT/I pound/NN/I is/vbz/O
np_bracket(SIn, SOut, P, N):-
  move(merge,LeftContext,item(Word,Tag,IOB),RightContext,SIn,SOut,P,N).

For each example e in Pos
  Pos := Pos - {e} ;
  if Best(e) is correct
  then
    Covered := Covered ∪ {Best(e)} ;
  else
    Pos := Pos ∪ {Best(e)};
  endif
For each example e in Covered
  if Best(e) is incorrect
  then
    Covered := Covered - {e} ;
    Pos := Pos ∪ {Best(e)} ;
  endif
endif

```

Fig. 5. Modification to CLOG to learn transformation lists

6 Structuring the space of clauses

In order to search the space of clauses (program clauses) systematically, it is useful to impose some structure upon it, e.g., an ordering. One such ordering is based on θ -subsumption, defined below.

Recall first that a substitution $\theta = \{V_1/t_1, \dots, V_n/t_n\}$ is an assignment of terms t_i to variables V_i . Applying a substitution θ to a term, atom, or clause F yields the instantiated term, atom, or clause $F\theta$ where all occurrences of the variables V_i are simultaneously replaced by the term t_i .

Let c and c' be two program clauses. Clause c θ -subsumes c' if there exists a substitution θ , such that $c\theta \subseteq c'$ (Plotkin, 1969).

To illustrate the above notions, consider the clause c

$$c = \text{daughter}(X, Y) \leftarrow \text{parent}(Y, X).$$

Applying the substitution $\theta = \{X/\text{mary}, Y/\text{ann}\}$ to clause c yields

$$c\theta = \text{daughter}(\text{mary}, \text{ann}) \leftarrow \text{parent}(\text{ann}, \text{mary}).$$

Recall that the clausal notation $\text{daughter}(X, Y) \leftarrow \text{parent}(Y, X)$ stands for

$$\{\text{daughter}(X, Y), \overline{\text{parent}(Y, X)}\}$$

where all variables are assumed to be universally quantified and the commas denote disjunction. According to the definition, clause c θ -subsumes c' if there is a substitution θ that can be applied to c such that every literal in the resulting clause occurs in c' . Clause c θ -subsumes the clause

$$c' = \text{daughter}(X, Y) \leftarrow \text{female}(X), \text{parent}(Y, X)$$

under the empty substitution $\theta = \emptyset$, since $\{\text{daughter}(X, Y), \overline{\text{parent}(Y, X)}\}$ is a proper subset of $\{\text{daughter}(X, Y), \text{female}(X), \text{parent}(Y, X)\}$. Furthermore, clause c θ -subsumes the clause

$$c' = \text{daughter}(\text{mary}, \text{ann}) \leftarrow \text{female}(\text{mary}), \text{parent}(\text{ann}, \text{mary}), \text{parent}(\text{ann}, \text{tom})$$

under the substitution $\theta = \{X/\text{mary}, Y/\text{ann}\}$.

θ -subsumption introduces a syntactic notion of generality. Clause c is *at least as general as* clause c' ($c \leq c'$) if c θ -subsumes c' . Clause c is *more general than* c' ($c < c'$) if $c \leq c'$ holds and $c' \leq c$ does not. In this case, we say that c' is a *specialization of* c and c is a *generalization of* c' . If the clause c' is a specialization of c then c' is also called a *refinement* of c . The only clause refinements usually considered by ILP systems are the minimal (most general) specializations of the clause.

There are two important properties of θ -subsumption:

- If c θ -subsumes c' then c logically entails c' , $c \models c'$. The reverse is not always true. As an example, Flach (1992) gives the following two clauses $c = \text{list}([V|W]) \leftarrow \text{list}(W)$ and $c' = \text{list}([X, Y|Z]) \leftarrow \text{list}(Z)$. Given the empty list, c constructs lists of any given length, while c' constructs lists of even length only, and thus $c \models c'$. However, no substitution exists that can be applied to c to yield c' , since it should map W both to $[Y|Z]$ and to Z which is impossible. Therefore, c does not θ -subsume c' .
- The relation $\leq$ introduces a *lattice* on the set of all clauses (Plotkin, 1969). This means that any two clauses have a least upper bound (*lub*) and a greatest lower bound (*glb*). Both the *lub* and the *glb* are unique up to equivalence (renaming of variables) under θ -subsumption. For example, the clauses $\text{daughter}(X, Y) \leftarrow \text{parent}(Y, X), \text{parent}(W, V)$ and $\text{daughter}(X, Y) \leftarrow \text{parent}(Y, X)$ θ -subsume one another.

The second property of θ -subsumption leads to the following definition: The *least general generalization* (*lgg*) of two clauses c and c' , denoted by $\text{lgg}(c, c')$, is the least upper bound of c and c' in the θ -subsumption lattice (Plotkin, 1969). The rules for computing the *lgg* of two clauses are outlined later in this chapter.

Note that θ -subsumption and least general generalization are purely syntactic notions since they do not take into account any background knowledge. Their computation is therefore simple and easy to be implemented in an ILP system. The same holds for the notion of generality based on θ -subsumption. On the other hand, taking background knowledge into account would lead to the notion of *semantic generality* (Niblett, 1988; Buntine, 1988), defined as follows: Clause c is *at least as general as* clause c' with respect to background theory B if $B \cup \{c\} \models c'$.

The syntactic, θ -subsumption based, generality is computationally more feasible. Namely, semantic generality is in general undecidable and does not introduce a lattice on a set of clauses. Because of these problems, syntactic generality is more frequently used in ILP systems.

θ -subsumption is important for inductive logic programming for the following reasons:

- As shown above, it provides a generality ordering for hypotheses, thus structuring the hypothesis space. It can be used to prune large parts of the search space.
 - θ -subsumption provides the basis for two important ILP techniques:
 - top-down *searching of refinement graphs*, and
 - *building of least general generalizations* from training examples, relative to background knowledge, which can be used to bound the search of refinement graphs from below or as part of a bottom-up search.
- These two techniques will be elaborated upon in the following sections.

7 Searching the space of clauses

Most ILP approaches search the hypothesis space (of program clauses) in a top-down manner, from general to specific hypotheses, using a θ -subsumption-based *specialization operator*. A specialization operator is usually called a *refinement operator* (Shapiro, 1983). Given a hypothesis language $\mathcal{L}$, a refinement operator ρ maps a clause c to a set of clauses $\rho(c)$ which are specializations (refinements) of c : $\rho(c) = \{c' \mid c' \in \mathcal{L}, c < c'\}$.

A refinement operator typically computes only the set of minimal (most general) specializations of a clause under θ -subsumption. It employs two basic syntactic operations on a clause:

- apply a substitution to the clause, and
- add a literal to the body of the clause.

The hypothesis space of program clauses is a lattice, structured by the θ -subsumption generality ordering. In this lattice, a *refinement graph* can be defined as a directed, acyclic graph in which *nodes* are program clauses and *arcs* correspond to the basic refinement operations: substituting a variable with a term, and adding a literal to the body of a clause.

Figure 6 depicts a part of the refinement graph for the family relations problem defined in Table 1, where the task is to learn a definition of the *daughter* relation in terms of the relations *female*, and *parent*.

At the top of the refinement graph (lattice) is the clause

$$c = \text{daughter}(X, Y) \leftarrow$$

where an empty body is written instead of the body *true*. The refinement operator ρ generates the refinements of c , which are of the form

$$\rho(c) = \{\text{daughter}(X, Y) \leftarrow L\}$$

where L is one of following literals:

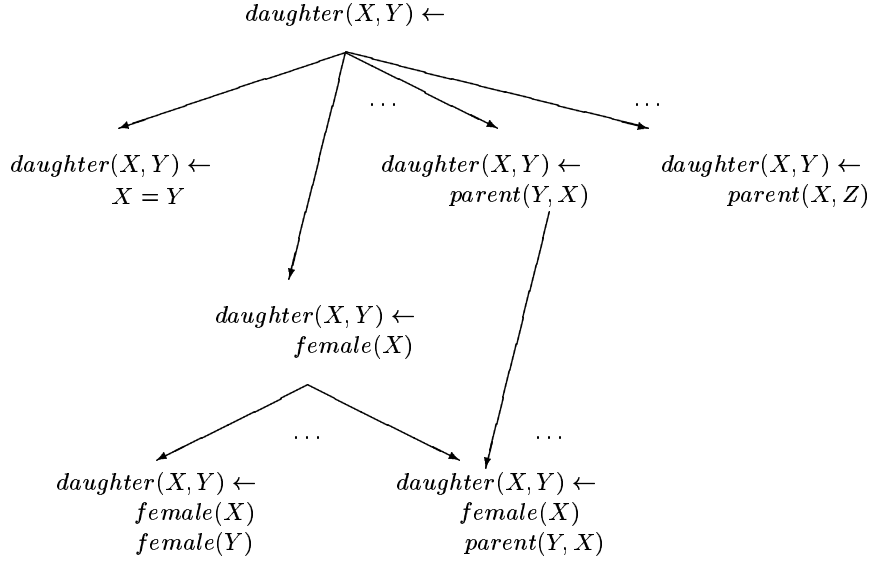


Fig. 6. Part of the refinement graph for the family relations problem.

- literals having as arguments the variables from the head of the clause: $X = Y$ (this corresponds to applying a substitution X/Y), $female(X)$, $female(Y)$, $parent(X, X)$, $parent(X, Y)$, $parent(Y, X)$, and $parent(Y, Y)$, and
- literals that introduce a new distinct variable Z ($Z \neq X$ and $Z \neq Y$) in the clause body: $parent(X, Z)$, $parent(Z, X)$, $parent(Y, Z)$, and $parent(Z, Y)$.

The search for a clause starts at the top of the lattice, with the clause that covers all example (positive and negative). Its refinements are then considered, then their refinements in turn, and this is repeated until a clause is found which covers only positive examples. In the example above, the clause $daughter(X, Y) \leftarrow female(X), parent(Y, X)$ is such a clause. Note that this clause can be reached in several ways from the top of the lattice, e.g., by first adding $female(X)$, then $parent(Y, X)$ or vice versa.

The refinement graph is typically searched heuristically level-wise, using heuristics based on the number of positive and negative examples covered by a clause. As the branching factor is very large, greedy search methods are typically applied which only consider a limited number of alternatives at each level. Hill-climbing considers only one alternative at each level, while beam search considers n alternatives, where n is the beam width. Occasionally, complete search is used, e.g., A^* best-first search or breadth-first search. Often the search can be pruned. For example, since we are only interested in clauses that (together with background knowledge) entail at least one example, we can prune the search if

we ever construct a clause which entails no positive examples. This is because if a clause covers no positive examples then neither will any of its refinements.

8 Bounding the search for clauses

The branching factor of a refinement graph, i.e., the number of refinements a clause has, is very large. This is especially true for clauses deeper in the lattice that contain many variables. It is thus necessary to find ways to reduce the space of clauses actually searched.

One approach is to make the refinement graph smaller by making the refinement operator take into account the types of predicate arguments, as well as input/output mode declarations. For example, we might restrict the *parents*(Y, X) predicate to only give us the parents of a given *child* and not give us persons that are the offspring of a given person. Also we could restrict *parents*(Y, X), so that X can only be instantiated with a term of type *child*. This can be done with a mode declaration *parents*($-person, +child$).

Type and mode declarations can be combined with the construction of a bottom clause that bounds the search of the refinement lattice from below. This is the most specific clause covering a given example (or examples). Only clauses on the path between the top and the bottom clause are considered, significantly improving efficiency. This approach is implemented in the Prolog algorithm (Muggleton, 1995b).

The bottom clause can be constructed as the relative least general generalization of two (or more) examples (Muggleton & Feng, 1990) or the most specific inverse resolvent of an example (Muggleton, 1991), both with respect to a given background knowledge B . In the case where the target clause is recursive and is resolved with itself several times to derive the example at hand inverse resolution is not sufficient and inverse entailment (Muggleton, 1995b) has to be applied. Below we discuss relative least general generalization, inverse resolution and inverse entailment.

8.1 Relative least general generalization

Plotkin's notion of least general generalization (*lgg*) (Plotkin, 1969) forms the basis of cautious generalization: the latter assumes that if two clauses c_1 and c_2 are true, it is very likely that *lgg*(c_1, c_2) will also be true.

The least general generalization of two clauses c and c' , denoted *lgg*(c, c'), is the least upper bound of c and c' in the θ -subsumption lattice. It is the most specific clause that θ -subsumes c and c' . If a clause d θ -subsumes c and c' , it has to subsume *lgg*(c, c') as well. To actually compute the *lgg* of two clauses, *lgg* of terms and literals need to be defined first (Plotkin, 1969).

The *lgg* of two terms *lgg*(t_1, t_2) is computed as

1. *lgg*(t, t) = t ,
2. *lgg*($f(s_1, \dots, s_n), f(t_1, \dots, t_n)$) = $f(\textit{lgg}(s_1, t_1), \dots, \textit{lgg}(s_n, t_n))$.

3. $lgg(f(s_1, \dots, s_m), g(t_1, \dots, t_n)) = V$, where $f \neq g$, and V is a variable which represents $lgg(f(s_1, \dots, s_m), g(t_1, \dots, t_n))$,
4. $lgg(s, t) = V$, where $s \neq t$ and at least one of s and t is a variable; in this case, V is a variable which represents $lgg(s, t)$.

For example, $lgg([a, b, c], [a, c, d]) = [a, X, Y]$, and $lgg(f(a, a), f(b, b)) = f(lgg(a, b), lgg(a, b)) = f(V, V)$, where V stands for $lgg(a, b)$. When computing $lggs$ one must be careful to use the same variable for multiple occurrences of the $lggs$ of subterms, i.e., $lgg(a, b)$ in this example. This holds for $lggs$ of terms, atoms and clauses alike.

The lgg of two atoms $lgg(A_1, A_2)$ is computed as follows:

1. $lgg(p(s_1, \dots, s_n), p(t_1, \dots, t_n)) = p(lgg(s_1, t_1), \dots, lgg(s_n, t_n))$, if atoms have the same predicate symbol p ,
2. $lgg(p(s_1, \dots, s_m), q(t_1, \dots, t_n))$ is undefined if $p \neq q$.

The lgg of two literals $lgg(L_1, L_2)$ is defined as follows:

1. if L_1 and L_2 are atoms, then $lgg(L_1, L_2)$ is computed as defined above,
2. if both L_1 and L_2 are negative literals, $L_1 = \overline{A_1}$ and $L_2 = \overline{A_2}$, then $lgg(L_1, L_2) = lgg(\overline{A_1}, \overline{A_2}) = \overline{lgg(A_1, A_2)}$,
3. if L_1 is a positive and L_2 is a negative literal, or vice versa, $lgg(L_1, L_2)$ is undefined.

For example,

$$\begin{aligned} lgg(\text{parent}(\text{ann}, \text{mary}), \text{parent}(\text{ann}, \text{tom})) &= \text{parent}(\text{ann}, X) \\ lgg(\text{parent}(\text{ann}, \text{mary}), \overline{\text{parent}(\text{ann}, \text{tom})}) &\text{ is undefined} \\ lgg(\text{parent}(\text{ann}, X), \text{daughter}(\text{mary}, \text{ann})) &\text{ is undefined} \end{aligned}$$

Taking into account that clauses are sets of literals, the lgg of two clauses is defined as follows. Let $c_1 = \{L_1, \dots, L_n\}$ and $c_2 = \{K_1, \dots, K_m\}$. Then $lgg(c_1, c_2) = \{M_{ij} = lgg(L_i, K_j) \mid L_i \in c_1, K_j \in c_2, lgg(L_i, K_j) \text{ is defined}\}$. If

$$c_1 = \text{daughter}(\text{mary}, \text{ann}) \leftarrow \text{female}(\text{mary}), \text{parent}(\text{ann}, \text{mary})$$

and

$$c_2 = \text{daughter}(\text{eve}, \text{tom}) \leftarrow \text{female}(\text{eve}), \text{parent}(\text{tom}, \text{eve})$$

then

$$lgg(c_1, c_2) = \text{daughter}(X, Y) \leftarrow \text{female}(X), \text{parent}(Y, X)$$

where X stands for $lgg(\text{mary}, \text{eve})$ and Y stands for $lgg(\text{ann}, \text{tom})$.

In (Cussens & Pulman, 2000), lgg is used as the basis for a bottom-up search for clauses representing grammar rules. The search is implemented in Prolog using the Sicstus Prolog library built-in `term_subsumer/3` to construct $lggs$. `term_subsumer/3` finds $lggs$ of terms, not clauses, but since the clauses in (Cussens & Pulman, 2000) are always unit (single-literal) clauses we can find $lggs$ by presenting these unit clauses to `term_subsumer/3` as if they were terms. Figure 7 shows two grammar rules and their lgg. Figure 8 gives human-readable

decompiled versions of the three rules in Figure 7. Rule **r67** involves, amongst other things, a generalisation of the ‘gap-threading’ patterns seen in Rules **r3** and **r24**. Gap-threading is a technique for dealing with movement phenomena in syntax (Pereira, 1981). Adding rule **r3** to the initial incomplete grammar in (Cussens & Pulman, 2000) allows the sentence *All big companies wrote a report quickly.* to be parsed, adding rule **r24** allows *What don’t all big companies read with a machine?*; adding the lgg rule **r67** allows both to be parsed.

```

cmp_synrule(
  vp([ng,ng],f(0,0,0,0,1,1,1,1,1),n),          %Rule r3
    [vp([ng,ng],f(0,0,0,0,1,1,1,1,1),n),
      mod([ng,ng],f(0,0,0,1),f(0,1,1,1))])
).

cmp_synrule(
  vp([np([ng,ng],f(0,0,0,0,_257,_257,_257,1,1), %Rule r24
      f(0,1,1,1),nonsubj),ng],f(0,0,1,1,1,1,1,1,1),n),
    [vp([np([ng,ng],f(0,0,0,0,_257,_257,_257,1,1),
      f(0,1,1,1),nonsubj),ng],f(0,0,1,1,1,1,1,1,1),n),
      mod([ng,ng],f(0,_187,_187,1),f(0,1,1,1))])
).

%Rule r67 = lgg(r3,r24)

cmp_synrule(
  vp([_231,ng],f(0,0,_223,_223,1,1,1,1,1),n),    %Rule r67
    [vp([_231,ng],f(0,0,_223,_223,1,1,1,1,1),n),
      mod([ng,ng],f(0,_187,_187,1),f(0,1,1,1))])
).

```

Fig. 7. Two grammar rules and their lgg

The definition of *relative least general generalization* (*rlgg*) is based on the semantic notion of generality. The *rlgg* of two clauses c_1 and c_2 is the least general clause which is more general than both c_1 and c_2 with respect (*relative*) to background knowledge B . The notion of *rlgg* was used in the ILP system GOLEM (Muggleton & Feng, 1990). To avoid the problems with the semantic notion of generality, the background knowledge B in GOLEM is restricted to ground facts. If K denotes the conjunction of all these facts, the *rlgg* of two ground atoms A_1 and A_2 (positive examples), relative to B can be computed as: $rlgg(A_1, A_2) = lgg((A_1 \leftarrow K), (A_2 \leftarrow K))$.

Given the positive examples $e_1 = daughter(mary, ann)$ and $e_2 = daughter(eve, tom)$ and the background knowledge B for the family example, the least general generalization of e_1 and e_2 relative to B is computed as:

```

r3 vp ==> [vp,mod]
vp: [gaps=[ng: [],ng: []],mor=pl,aux=n]==>
    [vp: [gaps=[ng: [],ng: []],mor=pl,aux=n],
      mod: [gaps=[ng: [],ng: []],of=vp,type=n]]

r24 vp ==> [vp,mod]
vp: [gaps=[np: [gaps=[ng: [],ng: []],mor=or(pl,s3),type=n,case=nonsubj],
            ng: []],mor=inf,aux=n]==>
    [vp: [gaps=[np: [gaps=[ng: [],ng: []],mor=or(pl,s3),type=n,case=nonsubj],
            ng: []],mor=inf,aux=n],
      mod: [gaps=[ng: [],ng: []],of=or(nom,vp),type=n]]

r67 vp ==> [vp,mod]
vp: [gaps=[_283,ng: []],mor=or(inf,pl),aux=n]==>
    [vp: [gaps=[_283,ng: []],mor=or(inf,pl),aux=n],
      mod: [gaps=[ng: [],ng: []],of=or(nom,vp),type=n]]

```

Fig. 8. Two grammar rules and their lgg (uncompiled version)

$rlgg(e_1, e_2) = lgg((e_1 \leftarrow K), (e_2 \leftarrow K))$ where K denotes the conjunction of the literals $parent(ann, mary)$, $parent(ann, tom)$, $parent(tom, eve)$, $parent(tom, ian)$, $female(ann)$, $female(mary)$, and $female(eve)$.

For notational convenience, the following abbreviations are used: *d-daughter*, *p-parent*, *f-female*, *a-ann*, *e-eve*, *m-mary*, *t-tom*, *i-ian*. The conjunction of facts from the background knowledge (comma stands for conjunction) is

$$K = p(a, m), p(a, t), p(t, e), p(t, i), f(a), f(m), f(e).$$

The computation of $rlgg(e_1, e_2) = lgg((e_1 \leftarrow K), (e_2 \leftarrow K))$, produces the following clause

$$\begin{aligned}
& d(\mathbf{V}_{\mathbf{m},\mathbf{e}}, \mathbf{V}_{\mathbf{a},\mathbf{t}}) \leftarrow p(a, m), p(a, t), p(t, e), p(t, i), f(a), f(m), f(e), \\
& p(a, V_{m,t}), \mathbf{p}(\mathbf{V}_{\mathbf{a},\mathbf{t}}, \mathbf{V}_{\mathbf{m},\mathbf{e}}), p(V_{a,t}, V_{m,i}), p(V_{a,t}, V_{t,e}), \\
& p(V_{a,t}, V_{t,i}), p(t, V_{e,i}), f(V_{a,m}), f(V_{a,e}), \mathbf{f}(\mathbf{V}_{\mathbf{m},\mathbf{e}}).
\end{aligned}$$

In the above clause, $V_{x,y}$ stands for $lgg(x, y)$, for each x and y .

In general, a *rlgg* of training examples can contain infinitely many literals or at least grow exponentially with the number of examples. Since such a clause can be intractably large, constraints are used on introducing new variables into the body of the *rlgg*. For example, literals in the body that are not connected to the head by a chain of variables are removed. In the above example, this yields the clause $d(\mathbf{V}_{\mathbf{m},\mathbf{e}}, \mathbf{V}_{\mathbf{a},\mathbf{t}}) \leftarrow \mathbf{p}(\mathbf{V}_{\mathbf{a},\mathbf{t}}, \mathbf{V}_{\mathbf{m},\mathbf{e}}), p(V_{a,t}, V_{m,i}), p(V_{a,t}, V_{t,e}), p(V_{a,t}, V_{t,i}), \mathbf{f}(\mathbf{V}_{\mathbf{m},\mathbf{e}})$. Also, nondeterminate literals (that can give more than one value of output arguments for a single value of the input arguments) may be eliminated. The literals $p(V_{a,t}, V_{m,i}), p(V_{a,t}, V_{t,e}), p(V_{a,t}, V_{t,i})$ are nondeterminate since $V_{a,t}$ is the input argument and a parent can have more than one child. Eliminating these yields the bottom clause $d(V_{m,e}, V_{a,t}) \leftarrow p(V_{a,t}, V_{m,e}), f(V_{m,e})$, i.e.,

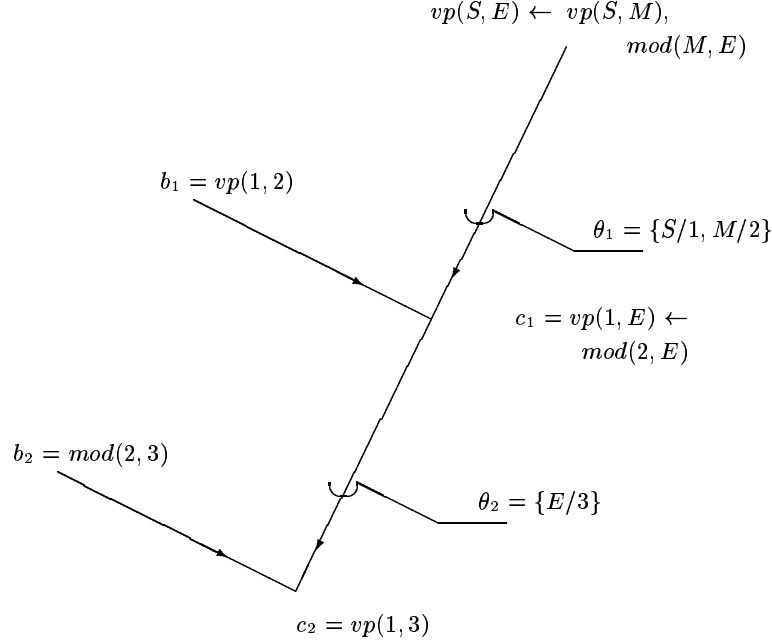


Fig. 9. A linear derivation tree.

$daughter(X, Y) \leftarrow female(X), parent(Y, X)$. In this simple example, the bottom clause is our target clause. In practice, the bottom clause is typically very large, containing hundreds of literals.

8.2 Inverse resolution

The basic idea of *inverse resolution* introduced by Muggleton and Buntine (1988), is to invert the *resolution* rule of deductive inference (Robinson, 1965), i.e., to invert the SLD-resolution proof procedure for definite programs (Lloyd, 1987). The basic resolution step in propositional logic derives the proposition $p \vee r$ given the premises $p \vee \bar{q}$ and $q \vee r$. In a first-order case, resolution is more complicated, involving substitutions. Let $res(c, d)$ denote the resolvent of clauses c and d .

To illustrate resolution in first-order logic, we use the grammar example from earlier. Suppose that background knowledge B consists of the clauses $b_1 = vp(1, 2)$ and $b_2 = mod(2, 3)$ and $H = \{c\} = \{vp(S, E) \leftarrow vp(S, M), mod(M, E)\}$. Let $T = H \cup B$. Suppose we want to derive the fact $vp(1, 3)$ from T . To this end, we proceed as follows:

- First, the resolvent $c_1 = res(c, b_1)$ is computed under the substitution $\theta_1 = \{S/1, M/2\}$. This means that the substitution θ_1 is first applied to clause c

- to obtain $vp(1, E) \leftarrow vp(1, 2), mod(2, E)$, which is then resolved with b_1 as in the propositional case. The resolvent of $vp(S, E) \leftarrow vp(S, M), mod(M, E)$ and $vp(1, 2)$ is thus $c_1 = res(c, b_1) = vp(1, E) \leftarrow mod(2, E)$.
- The next resolvent $c_2 = res(c_1, b_2)$ is computed under the substitution $\theta_2 = \{E/3\}$. The clauses $vp(1, E) \leftarrow mod(2, E)$ and $mod(2, 3)$ resolve in $c_2 = res(c_1, b_2) = vp(1, 3)$.

The linear derivation tree for this resolution process is given in Figure 9.

Inverse resolution, used in the ILP system CIGOL (Muggleton & Buntine, 1988), inverts the resolution process using generalization operators based on inverting substitution (Buntine, 1988). Given a well-formed formula W , an *inverse substitution* θ^{-1} of a substitution θ is a function that maps terms in $W\theta$ to variables, such that $W\theta\theta^{-1} = W$.

Let $c = vp(S, E) \leftarrow vp(S, M), mod(M, E)$ and $\theta = \{S/1, M/2, E/3\}$: then $c' = c\theta = vp(1, 3) \leftarrow vp(1, 2), mod(2, 3)$. By applying the inverse substitution $\theta^{-1} = \{1/S, 2/M, 3/E\}$ to c_1 , the original clause c is restored: $c = c'\theta^{-1} = vp(S, E) \leftarrow vp(S, M), mod(M, E)$. In the general case, inverse substitution is substantially more complex. It involves the *places* of terms in order to ensure that the variables in the initial W are appropriately restored in $W\theta\theta^{-1}$. In fact, each occurrence of a term can be replaced by a different variable in an inverse substitution.

We will not treat inverse resolution in detail, but will rather illustrate it by an example. Let $ires(c, d)$ denote the inverse resolvent of clauses c and d . As in the example above, let background knowledge B consist of the two clauses $b_1 = vp(1, 2)$ and $b_2 = mod(2, 3)$. The inverse resolution process might then proceed as follows:

- In the first step, inverse resolution attempts to find a clause c_1 which will, together with b_2 , entail e_1 . Using the inverse substitution $\theta_2^{-1} = \{3/E\}$, an inverse resolution step generates the clause $c_1 = ires(b_2, e_1) = vp(1, E) \leftarrow mod(2, E)$.
- Inverse resolution then takes $b_1 = vp(1, 2)$ and c_1 . It computes $c' = ires(b_1, c_1)$, using the inverse substitution $\theta_1^{-1} = \{1/S, 2/M\}$, yielding $c' = vp(S, E) \leftarrow vp(S, M), mod(M, E)$.

The corresponding inverse linear derivation tree is illustrated in Figure 10.

Most specific inverse resolution uses only empty inverse substitutions. In the above example, this would yield the clause $vp(1, 3) \leftarrow vp(1, 2), mod(2, 3)$ as the final inverse resolvent. In practice, inverse substitutions replacing the occurrence of each constant with the same variable are considered to yield the most specific inverse resolvent, which can then be used as a bottom clause. In our case, this is again the target clause, while in practice the bottom clause will be much more specific/larger, containing a large number of literals.

8.3 Inverse entailment

As mentioned previously, inverse resolution can not construct a target clause from an example if the clause is recursive and is used more than once to de-

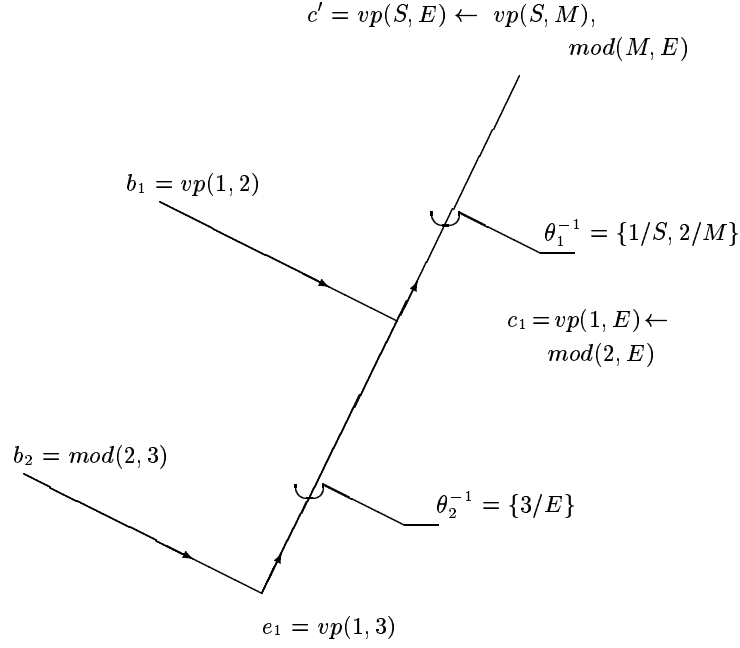


Fig. 10. An inverse linear derivation tree.

duce the example. This is particularly problematic in LLL applications such as grammar learning. Suppose we had an incomplete grammar lacking the target rule $vp(S, E) \leftarrow vp(S, M), mod(M, E)$ and so were unable to parse the sentence *She ran quickly from the telephone*. Suppose now that we had used abduction to (correctly) guess that there should be a VP from vertex 1 to 6, i.e. spanning the words *ran quickly from the telephone*. Suppose also that the rest of the grammar is sufficiently complete to infer that *ran* is a VP, *quickly* is a modifier and *from the telephone* is also a modifier. This gives us the following background B , positive example E and target H :

$$\begin{aligned}
 E &= vp(1, 6). \\
 B &= vp(1, 2), mod(2, 3), mod(3, 6) \\
 H &= vp(S, E) \leftarrow vp(S, M), mod(M, E)
 \end{aligned} \tag{1}$$

Consider now the derivation of E from B and H . This is illustrated in Fig 11 where the last three resolution steps have been squashed into one to conserve space. If we attempted to go up the tree by inverse resolution, then we would fail since we need H to derive H in the final step.

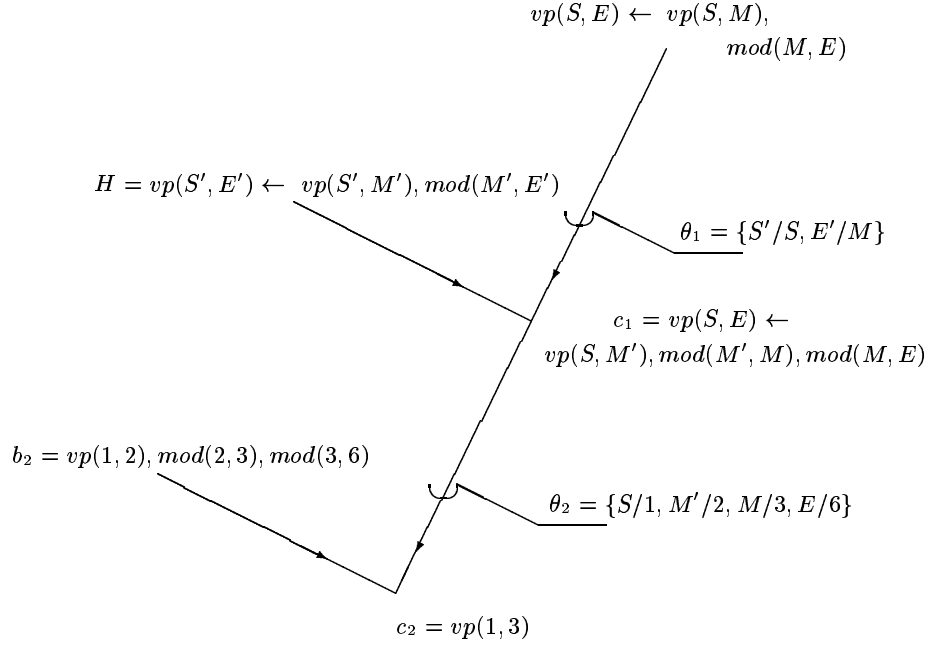


Fig. 11. An abbreviated linear derivation tree.

Inverse entailment was introduced in (Muggleton, 1995b) and we follow Muggleton's description. First note that if

$$B \wedge H \models E$$

then

$$B \wedge \overline{E} \models \overline{H}$$

Next let $\perp$ be the potentially infinite collection of ground literals which are true in every model of $B, \overline{E}$. Note that $\perp$ is *not* being used to denote falsity as in standard logical notation. It follows that

$$B \wedge \overline{E} \models \perp \models \overline{H}$$

and

$$H \models \perp$$

It follows that a subset of solutions for H can be found by considering clauses that θ -subsume the bottom clause $\perp$. If a clause H does subsume $\perp$ we say that H follows from B and E by inverse entailment.

Table 6 gives examples of two bottom clauses. Note that in the first example, the bottom clause is not a definite clause; this example shows how the construction of the bottom clause can be used to essentially do abduction.

Table 6. Most specific ‘bottom’ clauses $\perp$ for various B and E

B	E	$\perp$
$vp(X, Y) \leftarrow v(X, Y)$ $v(X, Y) \leftarrow word(ran, X, Y)$	$vp(1, 3)$	$vp(1, 3) \vee v(1, 3) \vee word(ran, 1, 3)$
$vp(X, Y) \leftarrow v(X, Y)$ $v(X, Y) \leftarrow word(ran, X, Y)$	$is_english(X, Y) \leftarrow$ $word(ran, X, Y)$	$is_english(X, Y) \leftarrow ran(X, Y),$ $vp(X, Y), v(X, Y)$

Returning to the situation given by Equations (1), we find that

$$\begin{aligned}\perp &= vp(1, 6) \leftarrow vp(1, 2), mod(2, 3), mod(3, 6) \\ &= \{vp(1, 6) \overline{vp(1, 2)}, \overline{mod(2, 3)}, \overline{mod(3, 6)}\}\end{aligned}$$

Although $H = vp(S, E) \leftarrow vp(S, M), mod(M, E) \models vp(1, 6) \leftarrow vp(1, 2), mod(2, 3), mod(3, 6) = \perp$, H does not *subsume* $\perp$. This is because there is no substitution θ such that $\{vp(S, E), \overline{vp(S, M)}, \overline{mod(M, E)}\}\theta \subseteq \{vp(1, 6) \overline{vp(1, 2)}, \overline{mod(2, 3)}, \overline{mod(3, 6)}\}$.

In (Muggleton, 1998), inverse entailment based on an enlarged bottom clause is defined. The bottom clause is enlarged by adding all possible positive literals except those which follow from $B \wedge E$. In our example, this enlarged bottom clause, denoted $BOT(B, E)$ is

$$\begin{aligned}\{ &vp(1, 1), \overline{vp(1, 2)}, vp(1, 3), vp(1, 6), \\ &vp(2, 1), \overline{vp(2, 2)}, vp(2, 3), vp(2, 6), \\ &vp(3, 1), \overline{vp(3, 2)}, vp(3, 3), vp(3, 6), \\ &vp(6, 1), \overline{vp(6, 2)}, vp(6, 3), vp(6, 6), \\ &mod(1, 1), mod(1, 2), \overline{mod(1, 3)}, mod(1, 6), \\ &mod(2, 1), mod(2, 2), \overline{mod(2, 3)}, mod(2, 6), \\ &mod(3, 1), mod(3, 2), mod(3, 3), \overline{mod(3, 6)}, \\ &mod(6, 1), mod(6, 2), mod(6, 3), mod(6, 6)\}\end{aligned}$$

The clause $H = vp(S, E) \leftarrow vp(S, M), mod(M, E)$ does subsume this enlarged bottom clause, so H follows from B and E by inverse entailment with an enlarged bottom clause.

9 Some challenges in language learning for ILP

10 Some challenges in language learning for ILP

The main advantages of ILP from a NLP perspective are that 1) the rules learnt by an ILP system are easily understandable by a linguist, 2) ILP allows easy integration of linguistic background knowledge such as language universal/specific syntactic/morphological/phonological principles, 3) ILP offers a first-order representation both for the hypothesis language and the background language. Each

of these advantages are either unavailable or hard to integrate within a purely statistical learning system.

On the other hand, ILP systems suffer from certain disadvantages that make them less competitive compared to well engineered statistical systems. Although first-order representations are learnt, a disadvantage of ILP systems is efficiency and its effect on rule accuracy. ILP algorithms are essentially search algorithms that search the hypothesis space specified by the background knowledge and declarative bias. The search spaces involved are usually large which makes it necessary to put informed restrictions to limit the search. For instance, the hypothesis space is restricted by putting suitable restrictions on clause length and restrictions on literals that can appear in the body of a clause. In addition, the search is restricted by using hill-climbing in systems such as Progol (?) and CLOG (?). These steps make learning using ILP tractable at the cost of reducing rule accuracy. One reason it is difficult to solve the efficiency issue is that most ILP implementations are general purpose learning systems that will accept any background knowledge and any training data set. In contrast, most statistical systems are engineered for a specific task.

ILP systems are ideally suited in language learning tasks which involve learning of complex representations. By “complex representation” we mean representations which are not purely attribute-value. Some of these tasks are listed below:

- Unsupervised learning of complex grammars/lexicons

A grand challenge for ILP would be the learning, from unannotated corpora or a small corpora, of grammars for formalisms such as HPSG (?, ?), LFG (?) and Categorical grammars (?). It is important to note that unsupervised learning is useful here since annotated corpora is unavailable. Initial work on unsupervised learning of categorical grammars is reported in (?). Statistical learning of categorical grammars from tagged input is reported in (?).

- Learning of information extraction rules

Information extraction is the task of identifying and extracting structured information from text. For instance, one of the tasks in the third *Message Understanding Conference* (MUC-3) (?) was to build an information extraction system that automatically extracts from free text a database of Latin American terrorist activities. An example record from MUC-3 is given in Figure ??.

Information extraction often involves tasks such as 1) NP chunking and robust shallow parsing; 2) anaphora resolution which including determining the reference of pronouns; 3) *named entity recognition* which includes the task of identifying and separating names related to people, organisations and places and 4) identifying relevant events. In each of these tasks, rule learning systems have been employed successfully. For instance, (?) employs rule learning system for named entity recognition. Mikheev’s system uses PoS tagged input for training. Shallow parsing rules were learnt by Cardie’s system (?). Information extraction is a potential domain in which it is likely that diverse background knowledge such a domain specific terminology can be beneficially employed for correct extraction of information. Since ILP

allows easy integration of background knowledge it would be valuable technique for learning complex information extraction rules.

- Learning of semantic hierarchies such as WordNet
As an information extraction system is deployed it will need to be updated as new domain specific terminology comes into use. A conceptual hierarchy (similar to the hierarchy used in WordNet) is often employed (?). Automatic methods for acquisition of domain specific terminology and its automatic integration into a semantic hierarchy is necessary since it will not be possible to collect training data and retrain an IE system once it is deployed.
- learning of contextual disambiguation rules/integration with parsing strategies
- learning of generation rules/text planning rules

Message: Id	TST1-MUC3-0002
Message: Template	1
Incident: Date	17 FEB 90 - 18 FEB 90
Incident: Location	El Salvador: San Salvador (City)
Incident: Type	ATTACK
Incident: Stage	ACCOMPLISHED
Incident: Instrument id	-
Incident: Instrument type	-
Perp: Incident category	TERRORIST ACT
Perp: Individual Id	"Members of the Farabundo Marti National Liberation Front [FMLN]" / "Members of the Farabundo Marti National Liberation Front"
Perp: Organization Id	"Farabundo Marti National Liberation Front" / "FMLN"
Perp: Org. confidence	Reported as fact: "Farabundo Marti National Liberation Front" / "FMLN"
Phys tgt: Id	"electric power substation"
Phys tgt: Type	"Energy: electric power substation"
Phys tgt: Number	1: "electric power substation"
Phys tgt: Foreign nation	-
Phys tgt: Effect	-
Phys tgt: Total number	-
Hum tgt: Name	-
Hum tgt: Description	"soldier"
Hum tgt: Type	Active military: "soldier"
Hum tgt: Number	1: "soldier"
Hum tgt: Foreign nation	-
Hum tgt: Effect	INJURY: "soldier"
Hum tgt: Total number	-

Fig. 12. Sample database record from MUC-3

11 Volume overview

11.1 Summaries of papers

Grammars Cussens & Pulman and also Osborne assume the existence of a given initial grammar that must be completed to parse sentences in a training set of unparsable sentences. In both cases, candidate rules are generated from an unparsable sentence via the chart created by the failed parse. From an ILP point of view the chart is that part of the background knowledge *relevant* to the example sentence that produced it.

In Osborne’s paper an SCFG is defined on a feature grammar via a context-free (backbone) grammar formed by mapping each category, distinct in terms of features, to an atomic symbol. Riezler takes a more direct approach, defining a log-linear model on the parses of constraint-based grammars—the goal here is disambiguation rather than finding ‘missing’ grammar rules. Riezler describes how to learn both features (structure) and parameters of such models from unannotated data. Note that the features of the model are not (necessarily) grammar rules.

Semantics Going against the grain of work in statistical computational linguistics, a number of contributions focus on semantics. Indeed, Mooney’s paper argues that logical approaches to language learning have “most to offer at the level of producing semantic interpretations of complete sentences” since logical representations of meaning are “important and useful”. Mooney backs this up with an overview of work on the CHILL system for learning semantic parsers. CHILL is also examined in the paper from Thompson & Califf, who demonstrate (empirically) the effectiveness of *active learning* where a learning algorithm selects the most informative inputs for a human to annotate. This theme of “how best to marry machine learning with human labor” is central to Brill’s paper arguing that we should be “capitalizing on the relative strengths of people and machines”.

Boström reports on learning *transfer rules* which translate between semantic representations (quasi-logical forms) for different languages; in this case French and English. Nédellec continues the semantic theme with the ILP system ASIUM, which learns ontologies and verb subcategorization frames from a parsed corpora.

Information extraction As well as CHILL, Thompson & Califf also describe and give experimental results for active learning with the RAPIER system. RAPIER is a system that learns information extraction (IE) rules using limited syntactic and semantic information. Junker et al also learn IE rules, examining how standard ILP approaches can be used. They also learn rules for text categorisation.

PoS tagging, morphological analysis and phonotactics Part-of-speech tagging is a popular test-bed for the application of ILP to NLP. An overview of this area is provided in the paper from Eineborg and Lindberg. ILP approaches

to tagging often use the tags of neighbouring words to disambiguate the focus word—however, typically, these neighbouring tags are (initially) unknown. Jorge & de Andrade Lopes tackle this problem by learning a succession of theories, starting with a base theory which only has lexical rules and where a theory makes use of the disambiguation achieved by previous theories.

Kazakov gives an overview of ILP approaches to morphology and argues that future work should make more extensive use of linguistic knowledge and be more closely integrated with other learning approaches. Džeroski & Erjavec use CLOG to learn rules which assign the correct lemma (canonical form) to a word given the word and its tag. A statistical tagger is used to find the tags.

Tjong & Nerbonne examine learning phonotactics to distinguish (Dutch) words from non-words, finding that the inclusion of background knowledge improves performance.

11.2 Themes

Classification versus disambiguation Much of ILP is concerned with concept learning—identifying a set from positive and (usually) negative examples. In LLL the focus is on finding the right analysis: usually this is disambiguation: finding the right analysis from a set of possible analyses. Syntactic or semantics analysis of a sentence, tag or lemma for a word. Bostrom: Many rules required to 'cover' one example.

Interactive learning As Brill notes it is wasteful not to use humans. ILP with its emphasis on background knowledge makes it easier to integrate human-derived linguistic knowledge. Leave the drudgery to the machine. Requires linguists and MLers to work together, one of the primary aims of this volume. Greater transparency of logical representations. This can be exploited by

Hybrid approaches The statistical NLP revolution has occurred because statistical approaches such as HMMs have empirically demonstrated their effectiveness. This motivates hybrid statistical-logical approaches.

Bibliography

- Boström, H. (1998). Predicate invention and learning from positive examples only. In *Proc. of the Tenth European Conference on Machine Learning*, pp. 226–237. Springer Verlag.
- Buntine, W. (1988). Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36(2), 149–176.
- Cussens, J., & Pulman, S. (2000). Incorporating linguistics constraints into inductive logic programming. In *Proc. LLL2000* Lisbon. Submitted.
- De Raedt, L., & Džeroski, S. (1994). First order *jk*-clausal theories are PAC-learnable. *Artificial Intelligence*, 70, 375–392.

- Flach, P. (1992). Logical approaches to machine learning - an overview. *THINK*, 1(2), 25–36.
- Hogger, C. (1990). *Essentials of Logic Programming*. Clarendon Press, Oxford.
- Lloyd, J. (1987). *Foundations of Logic Programming* (2nd edition). Springer, Berlin.
- Muggleton, S. (1995a). Inverse entailment and Progol. *New Generation Computing*, 13, 245–286.
- Muggleton, S. (1995b). Inverse entailment and Progol. *New Generation Computing Journal*, 13, 245–286.
- Muggleton, S. (1998). Completing inverse entailment. In Page, C. (Ed.), *Proceedings of the Eighth International Workshop on Inductive Logic Programming (ILP-98)*, LNAI 1446, pp. 245–249. Springer-Verlag, Berlin.
- Muggleton, S. (2000). Learning from positive data. *Machine Learning*. Accepted subject to revision.
- Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4), 295–318.
- Muggleton, S., & Bryant, C. (2000). Theory completion using inverse entailment. In *Proc. of the 10th International Workshop on Inductive Logic Programming (ILP-00)* Berlin. Springer-Verlag. In press.
- Muggleton, S., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proc. Fifth International Conference on Machine Learning*, pp. 339–352 San Mateo, CA. Morgan Kaufmann.
- Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. In *Proc. First Conference on Algorithmic Learning Theory*, pp. 368–381 Tokyo. Ohmsha.
- Niblett, T. (1988). A study of generalisation in logic programs. In *Proc. Third European Working Session on Learning*, pp. 131–138 London. Pitman.
- Pereira, F. (1981). Extraposition grammars. *Computational Linguistics*, 7, 243–256.
- Plotkin, G. (1969). A note on inductive generalization. In Meltzer, B., & Michie, D. (Eds.), *Machine Intelligence 5*, pp. 153–163 Edinburgh. Edinburgh University Press.
- Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3), 239–266.
- Robinson, J. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1), 23–41.
- Shapiro, E. (1983). *Algorithmic Program Debugging*. MIT Press, Cambridge, MA.