

# **Grammar induction using ILP**

**James Cussens**

**University of York, UK**

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

## 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.

## Chomsky on induction

The child who acquires a language in this way of course knows a great deal more than 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*

## 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?

## 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”

**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*

## 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:

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

## 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

## 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.

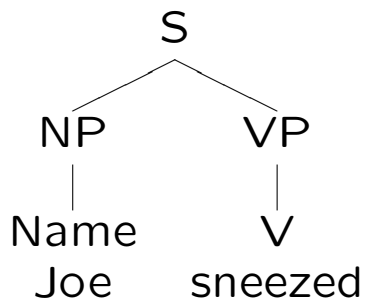
## CHART PARSING

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:

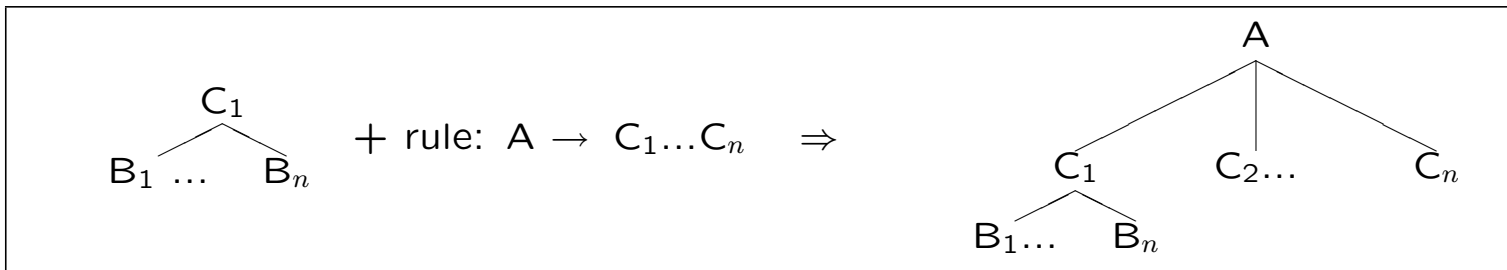


Grammar:  $S \rightarrow NP VP$   
 $VP \rightarrow V$   
 $NP \rightarrow Name$

Lexicon: Joe: Name  
sneezed: V

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

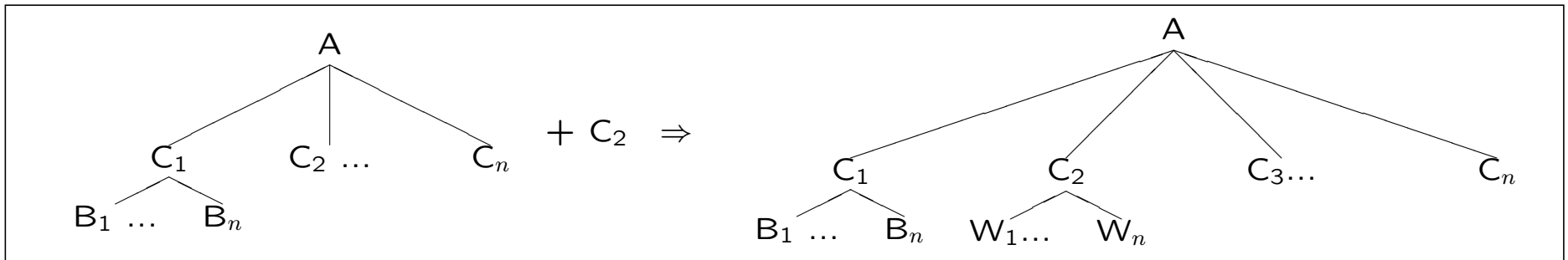
i.e. if there is a constituent  $C_1$  from position  $X$  to position  $Y$ , and there is a grammar rule which builds an  $A$  from a series of constituents beginning with a  $C_1$ , then begin a new constituent, an incomplete  $A$ , consisting just of the recognised  $C_1$ , but expecting to find the 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  $C_2 \dots C_n$ '.



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

Combine:

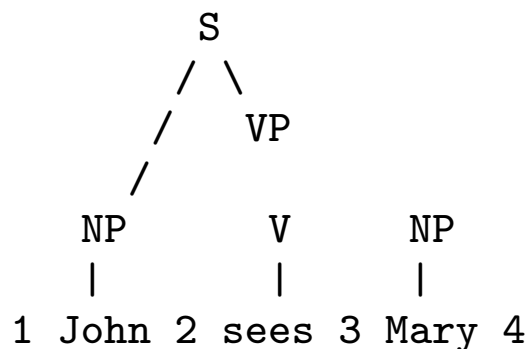
$$\frac{A \rightarrow C_1 \dots \bullet C_i \dots C_m \text{ from } X \text{ to } Y, \text{ and } C_i \text{ from } Y \text{ to } Z}{A \rightarrow C_1 \dots C_i \bullet C_{i+1} \dots C_m \text{ from } X \text{ to } Z}$$



## 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)

## Our grammar system

- Derived from a system built under the FraCaS project, March 1994.
- The software is for research only and is not to be distributed.
- It carries no warranty!

## 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].
```

## 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?

## 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

## 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[ng,ng] → VP[ng,ng] MOD[ng,ng]
2	VP[np,ng] → VP[np,ng] MOD[ng,ng]
3	VP[np,ng] → VP[np,np] MOD[np,ng]
4	VP[np,np] → VP[np,np] MOD[np,np]

## 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),_)]).
```

## 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),_,_)).
```

## Firing it up

- Just compile `sys.pl`
- Enter `p`, if it asks about redefining predicates

## 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

...

GSLT - ILP2

## 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]).`

## 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)))
```

## 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.

## 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]]`

GSLT - ILP2

## Linguistic knowledge: Gap-threading

The gaps also must be threaded properly . . .

## 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.

GSLT - ILP2

## 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.

## 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.

GSLT - ILP2

## 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.

## 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=\_].

## 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*

## 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:

## Over-generalising

Our np ==> [nom] rule allows us to parse:

*Dogs bark*

*School is no fun*

but also *Man bites dog*

Is this a sentence? a WSJ sentence?

## 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).
- `np ==> [nom]` generates **no** extra parses for my current validation corpus.
- `s ==> [np, vp]` rule generated  $111 + 306 = 417$  parses

## 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

## 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.