An efficient and linguistically rich statistical parser

Andreas van Cranenburgh

Huygens ING
Royal Netherlands Academy of Arts and Sciences

Institute for Logic, Language and Computation
University of Amsterdam

April 16, 2015

Linguistics & Statistics

- Linguistically rich parsers HPSG, LFG, &c.
  Non-local relations, function labels,
  morphological information.
  Often handwritten.
- Statistical Parsing
  Automatically induced from treebanks.
  Efficient
  Limited to constituents or projective dependencies.

This talk

1. Mild context-sensitivity
   Parsing with discontinuous constituents.
2. Data-Oriented Parsing
   Parsing with tree fragments.
3. Experiments

Two perspectives

Chomsky (1965):

Competence:
the idealized rules of language

Performance:
actual language use

Formal Grammar theory

Statistical NLP

This talk: Computational Linguistics should focus more on the latter.

Chomsky (1965). Aspects of the Theory of Syntax.

The Chomsky hierarchy

1. Unrestricted undecidable
2. Context-Sensitive PSPACE complete
3. Context-Free $O(n^3)$
4. Regular $O(n)$

Chomsky & Schützenberger (1959). The Algebraic Theory of Context-Free Languages.

Cross-Serial dependencies

Dutch:
dat Karel Marie Peter Hans laat helpen leren zwemmen

English:
that Charles lets Mary help Peter teach Hans to swim

NB: cross-serial easier to process than center embedding!
(Bach et al. 1986)

Bach et al. (1986). Crossed and nested dependencies in German and Dutch: A psycholinguistic study.

Joshi (1985)

Joshi (1985): How much context sensitivity is necessary (. . .)

Goal A grammar formalism that is efficiently parsable yet strong enough to describe natural language

Figure: Aravind K. Joshi

Mild Context-Sensitivity

Definition

Mild Context-Sensitivity

1. limited crossed dependencies
2. constant growth
3. polynomial time parsing

Tree-Adjoining Grammar:

Tree Substitution: combine tree fragments

Tree Adjunction: add adjuncts

Discontinuous Constituents

Example:

- Why did the chicken cross the road?
- The chicken crossed the road to get to the other side.

Non-local information in PTB: traces

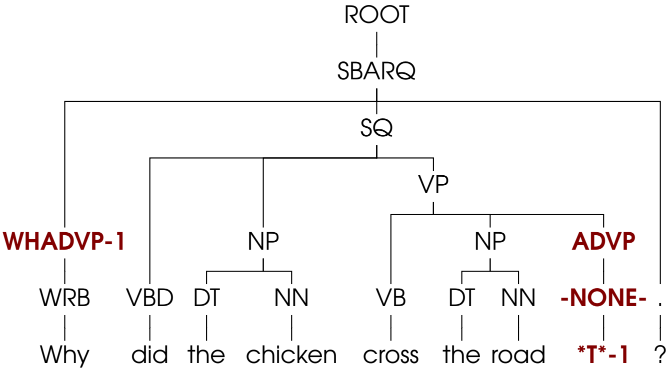


Figure: PTB-style annotation.

Discontinuous trees

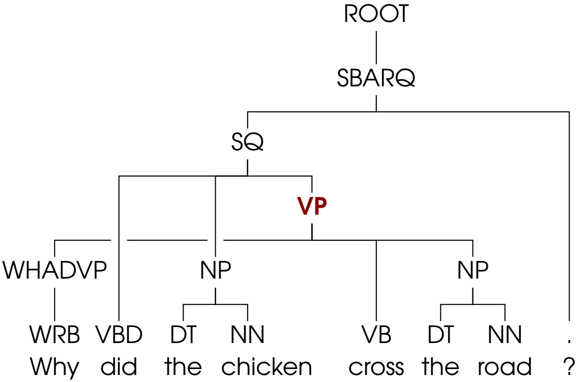


Figure: A tree with a discontinuous constituent.

Discontinuous constituents

Motivation:

- Handle flexible word-order, extraposition, &c.
- Capture argument structure
- Combine information from constituency & dependency structures

(NB: non-projectivity is a subset of discontinuous phenomena)

Discontinuous treebanks

Treebanks with discontinuous constituents:

German/Negra: Skut et al. (1997). An annotation scheme for free word order languages.

Dutch/Alpino: van der Beek (2002). The Alpino dependency treebank.

English/PTB (after conversion): Evang & Kallmeyer (2011). PLCFRS Parsing of English Discontinuous Constituents.

Swedish, Polish, . . .

Discontinuous trees

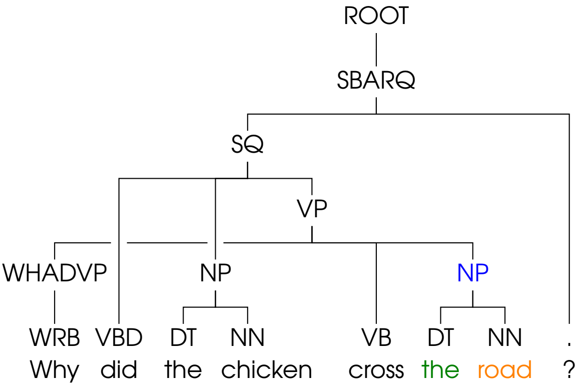


Figure: A tree with a discontinuous constituent.

Context-Free Grammar (CFG)

$$NP(ab) \rightarrow DT(a)\ NN(b)$$

Discontinuous trees

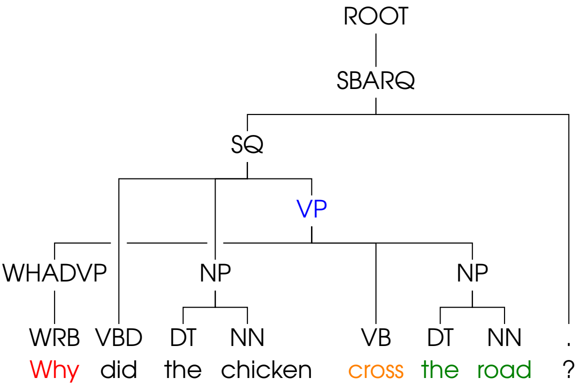


Figure: A tree with a discontinuous constituent.

Linear Context-Free Rewriting System (LCFRS)

$$VP_2(a, bc) \rightarrow WHADVP(a)\ VB(b)\ NP(c)$$

Discontinuous trees

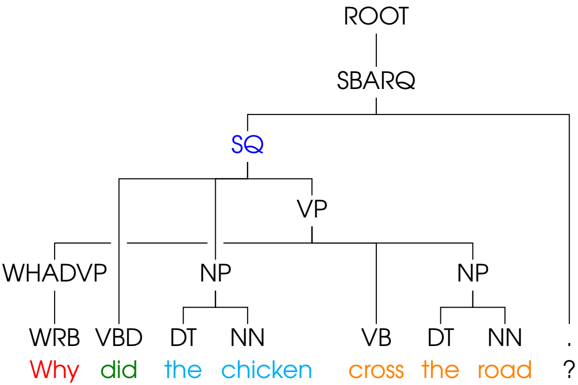


Figure: A tree with a discontinuous constituent.

Linear Context-Free Rewriting System (LCFRS)

$$VP_2(a, bc) \rightarrow WHADVP(a)\ VB(b)\ NP(c)$$
$$SQ(abcd) \rightarrow VBD(b)\ NP(c)\ VP_2(a, d)$$

Linear Context-Free Rewriting Systems

LCFRS are a generalization of CFG:
⇒ rewrite tuples, trees or graphs!

Vijay-Shanker et al. (1987): Structural descriptions (. . .) grammar formalisms.
Kallmeyer & Maier (2010, 2013). Data-driven parsing with probabilistic (LCFRS).

Linear Context-Free Rewriting Systems

LCFRS are a generalization of CFG:
⇒ rewrite tuples, trees or graphs!

linear: each variable on the left occurs once on the right & vice versa

context-free: apply productions based on what they rewrite

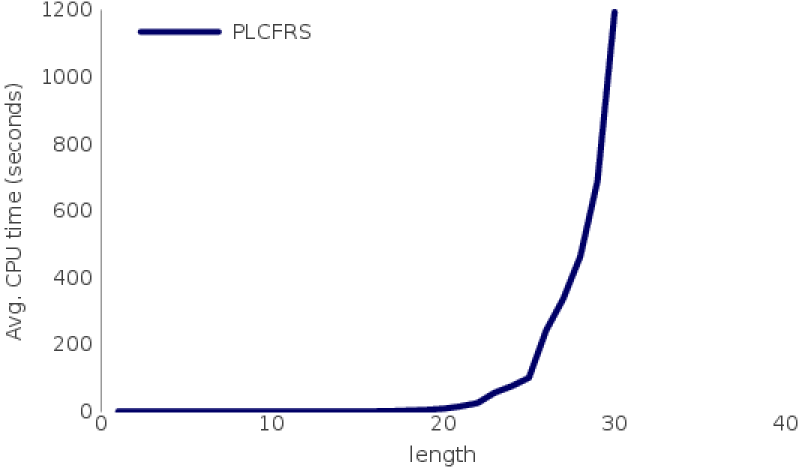
rewriting system: i.e., formal grammar

Parsing a binarized LCFRS has polynomial time complexity:

$$\mathcal{O}(n^{3\varphi})$$

Vijay-Shanker et al. (1987): Structural descriptions (. . .) grammar formalisms.
Kallmeyer & Maier (2010, 2013). Data-driven parsing with probabilistic (LCFRS).

But . . .



Negra dev. set, gold tags

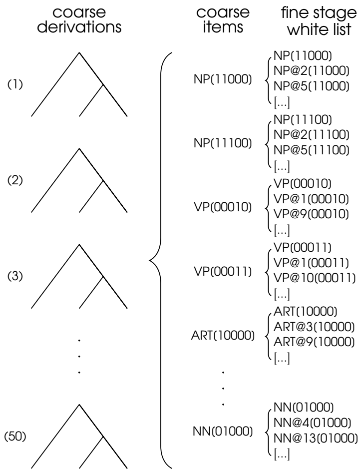
Pruning

Pruning can be based on:

1. Very little: e.g., beam threshold
2. Grammar: e.g., A* or context summary estimates
3. Sentence: e.g., coarse-to-fine parsing

Pauls & Klein (NAACL 2009), Hierarchical search for parsing.

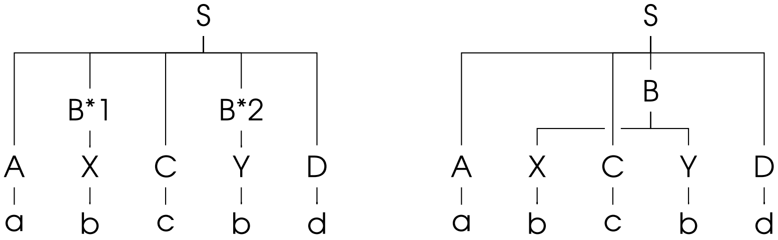
Coarse-to-fine



k-best PLCFRS derivations
help prune DOP derivations.

Charniak et al. (2006), Multi-level coarse-to-fine parsing

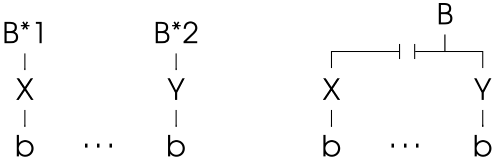
PCFG approximation of PLCFRS



- Transformation is reversible
- Increased independence assumption:
  ⇒ every component is a new node
- Language of PCFG is a superset of original PLCFRS
  ⇒ coarser, overgenerating PCFG ('split-PCFG')

Boyd (2007), Discontinuity revisited.

Coarse-to-fine from PCFG to PLCFRS

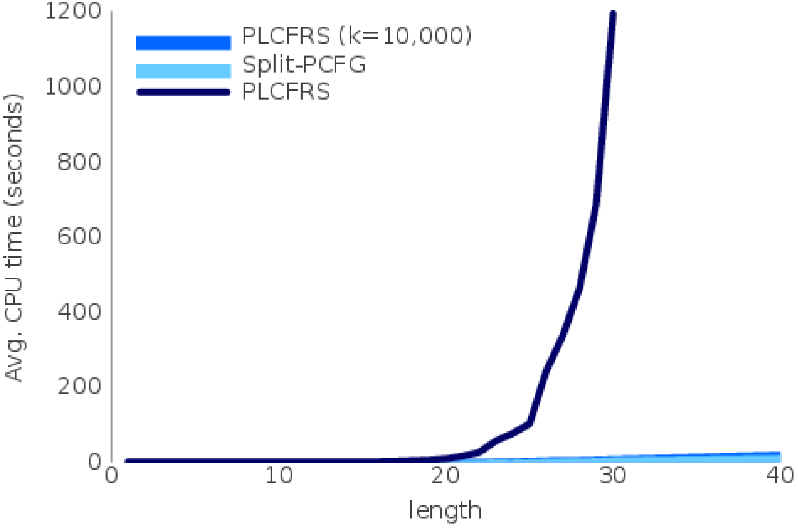


- For a discontinuous item, look up multiple items from PCFG chart (‘splitprune’)
- e.g.:

$$\left\{ \begin{matrix} NP{*}1 : [1, 2], \\ NP{*}2 : [4, 5] \end{matrix} \right\} \Rightarrow NP_2 : [1, 2;\ 4, 5]$$

Barthélemy et al. (2001) Guided parsing of range concatenation languages.
van Cranenburgh (2012), Efficient parsing with LCFRS

With coarse-to-fine



Negra dev. set, gold tags

Coarse-to-fine pipeline

Treebank grammars: G_0 Split-PCFG, G_1 PLCFRS

G_0 Split-PCFG

$\Downarrow$

G_1 PLCFRS

$\Downarrow$

G_2 *a large grammar*

Mildly context-sensitive: G_1 PLCFRS, G_2 *a large grammar*

prune parsing with G_{m+1} by only considering items in k-best G_m derivations.

Data-Oriented Parsing

Treebank grammar

trees ⇒ productions + rel. frequencies

⇒ problematic independence assumptions

Data-Oriented Parsing (DOP)

trees ⇒ fragments + rel. frequencies

fragments are arbitrarily sized chunks from the corpus

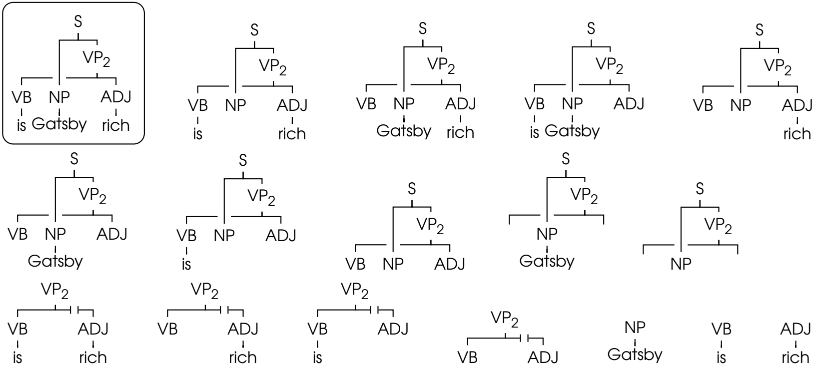
consider all possible fragments from treebank

…and "let the statistics decide"

Scha (1990): Lang. theory and lang. tech.; competence and performance

Bod (1992): A computational model of language performance

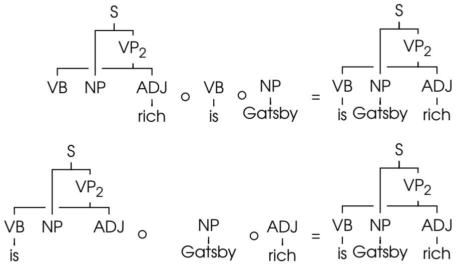
DOP fragments



$$P(f) = \frac{\text{count}(f)}{\sum_{f' \in F} \text{count}(f')} \text{ where } F = \{\, f' \mid root(f') = root(f) \,\}$$

Note: discontinuous frontier non-terminals mark destination of components

DOP derivation



$P(d) = 0.2$

$P(d) = 0.3$

Derivations for this tree $\qquad P(t) = 0.5$

$$P(d) = P(f_1 \circ \cdots \circ f_n) = \prod_{f \in d} p(f)$$

$$P(t) = P(d_1) + \cdots + P(d_n) = \sum_{d \in D(t)} \prod_{f \in d} p(f)$$

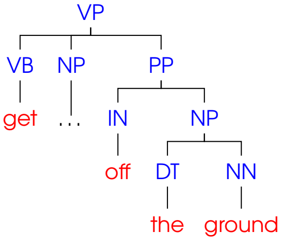
Tree-Substitution Grammar

This DOP model (Bod 1992) is based on Tree-Substitution Grammar (TSG):

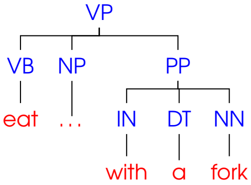
- Weakly equivalent to CFG; typically strongly equivalent as well; advantage is in stochastic power of Probabilistic TSG.
- Same Context-Free property as CFG, but multiple productions applied at once;
  $\Rightarrow$ captures more structural relations than PCFG.
- CFG backbone can be replaced with LCFRS to get Discontinuous Tree-Substitution Grammar ($\text{PTSG}_{\text{LCFRS}}$).

Tree Fragments

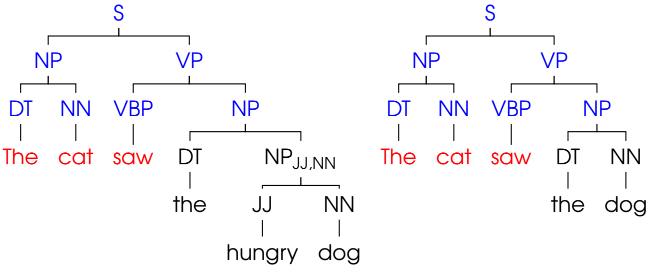
Multiword expressions (MWE):



Statistical regularities:



Double-DOP



Problem: Exponential number of fragments due to all-fragments assumption

- Extract fragments that occur at least twice in treebank
- For every pair of trees, extract maximal overlapping fragments
- Number of fragments is small enough to parse with directly

Sangati & Zuidema (2011). Accurate parsing w/compact TSGs: Double-DOP

Extract recurring fragments in linear average time

Tree kernel: find similarities in trees of treebank

- Worst case: need to compare every node to all other nodes in treebank
- Speed up comparisons by sorting nodes of trees:
  ⇒ Aligns potentially equal nodes, allowing us to skip the rest! (Moschitti 2006)
- Figure out fragments from list of matching nodes

Moschitti (2006): Making tree kernels practical for natural language learning
van Cranenburgh (2014), Extraction of (...) fragments w/linear average time

Extract recurring fragments in linear average time

Method, Corpus	Number of Trees	Number of Fragments	Time (hr:min) Wall	Time (hr:min) CPU
Sangati et al. (2010):				
QTK, WSJ 2–21	39,832	990,156	8:23	124:04
van Cranenburgh (2014):				
FTK*, WSJ 2–21	39,832	990,890	0:05	1:16
FTK, Gigaword, subset	502,424	9.7 million	9:54	$\sim$ 160

Wall clock time is when using 16 cores.

* Includes roaring bitmap datastructure (Chambi et al. 2014).

Sangati et al. (2010): Efficiently extract recurring tree fragments
van Cranenburgh (2014), Extraction of (. . .) fragments with a linear average

Experimental setup

English: Penn treebank, WSJ section
German: Tiger
Dutch: Lassy

Function labels

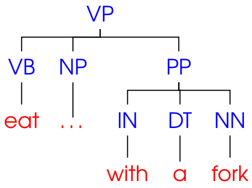
Syntactic categories (form): NP, VP, S, . . .

Function labels (function): SBJ, OBJ, TMP, LOC, . . .

- Classifier:
  - Blaheta & Charniak (2000), Assigning Function Tags to Parsed Text
- Integrate in grammar:
  - Gabbard et al. (2006), Fully parsing the Penn treebank
  - Fraser et al. (2013), Knowledge sources for constituent parsing of German

Evaluation: function tag accuracy over correctly parsed labeled bracketings.

State splits



- Tree fragments and state splits are (relatively) complementary:
  tree fragments include more context, but substitution is only restricted by the fine-grainedness of labels.
- Combine tree-substitution with manual state splits from:

  English: Klein & Manning (2003)
  German: Fraser et al. (2013)
  Dutch: new work

Preprocessing

- Binarize w/markovization (h=1, v=1)
- Simple unknown word model
  - Rare words replaced by features (model 4 from Stanford parser): `forty-two' ⇒ `_UNK-L-H-o`

Not reproduced: morphological tags, secondary parents

Can DOP handle discontinuity without LCFRS?

Negra dev set, gold tags:

Split-PCFG	Split-PCFG
$\Downarrow$	
PLCFRS	$\Downarrow$
$\Downarrow$	
PLCFRS Double-DOP	Split-Double-DOP
77.7 % F1	
41.5 % EX	

Can DOP handle discontinuity without LCFRS?

Negra dev set, gold tags:

Split-PCFG	Split-PCFG
$\Downarrow$	
PLCFRS	$\Downarrow$
$\Downarrow$	
PLCFRS Double-DOP	Split-Double-DOP
77.7 % F1	78.1 % F1
41.5 % EX	42.0 % EX

Answer: Yes!

Fragments can capture discontinuous contexts

Parsing results

Parser	F1	EX	func
GERMAN: Tiger			
Dep: HaNi2008	75.3	32.6	
2DOP: Cr et al	**78.2**	**40.0**	93.5
Dep: FeMa2015	82.6	45.9	
ENGLISH: WSJ			
PLCFRS: EvKa2011	79.0		
2DOP: Cr et al, WSJ	**87.0**	34.4	86.3
2DOP: SaZu2011, no disc.	87.9	33.7	
DUTCH: Lassy			
2DOP: Cr et al	76.6	34.0	92.8

HaNi: Hall & Nivre (2008); FeMa: Fernández-González & Martins (2015); SaZu: Sangati & Zuidema (2011); EvKa: Evang & Kallmeyer (2011); Cr et al: van Cranenburgh, Scha, Bod (submitted).

Recap

Linguistically rich: non-local relations, function tags
Efficiency: CFG base grammer, tree fragment extraction

Competence: idealized rules
Performance: actual language use

Tree fragments increase the abilities of a performance model w.r.t. discontinuous constituents, without increasing formal complexity.

THE END

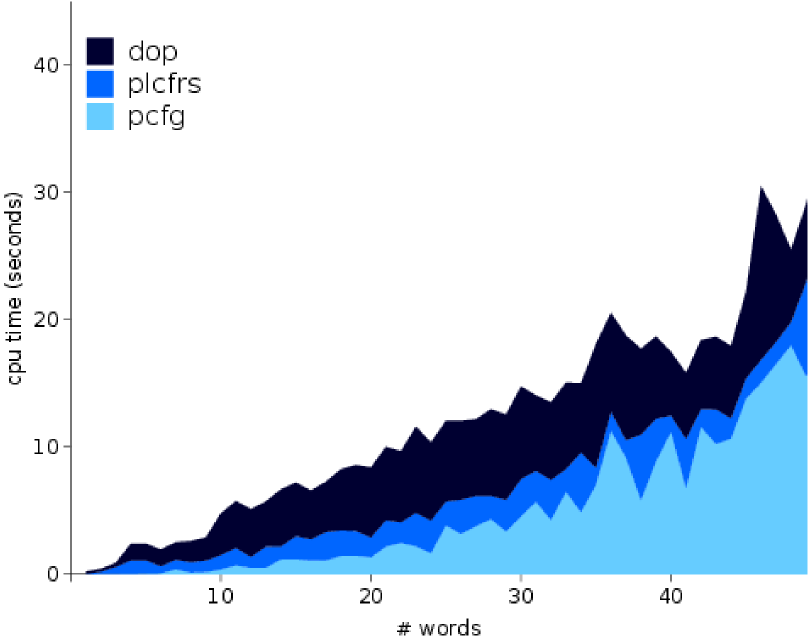
Codes: `http://github.com/andreasvc/disco-dop`

Papers: `http://andreasvc.github.io`

Wait ... there's more

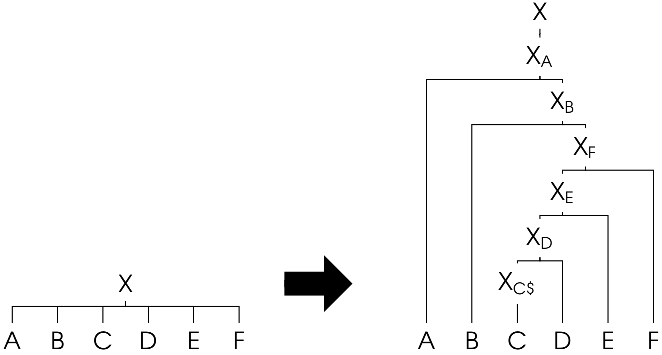
BACKUP SLIDES

Efficiency (Negra dev set)



Binarization

- mark heads of constituents
- head-outward binarization (parse head first)
- no parent annotation: $v = 1$
- horizontal Markovization: $h = 1$



Klein & Manning (2003): Accurate unlexicalized parsing.

Implementation details

- Cython: combines best of both worlds
  C speed, Python convenience.
- Where it matters, manual memory management & layout;
- e.g., grammar rules & edges compactly packed in arrays of `structs`.
- FWIW, lines of code:

Collins parser	C	3k	(!?)
bitpar	C++	6k	
disco-dop parser	Cython	21k	
Berkeley parser	Java	58k	
Charniak & Johnson parser	C++	62k	
Stanford parser	Java	151k	