CYK parsing

The task

Constituency (or *constituent*) *parsing* refers to recovering the derivational structure of a sentence, usually assuming it was generated by a context-free grammar.

Shieber (1986) shows, conclusively, that human language is not even weakly context-free; and one can parse using *mildly context-sensitive grammars* (MCSGs), such as

- lexicalized tree-adjoining grammars (LTAGs) or
- combinatory categorial grammar (CCGs).

CFGs can be parsed in O(n^3) or better; MCSGs can be parsed in O(n^6).

Ambiguity

Non-trivial CFGs tend to have substantial ambiguities, so we use:

- *Probabilistic* productions, making the grammar a PCFG and parsing a special case of the Viterbi algorithm
- Learned (e.g., Klein & Manning 2003) or induced (e.g., Petrov et al. 2006) enrichments to the productions to encode argument structure of verbs, etc.
- Optionally, discriminative reranking (e.g., Collins 1999, Bikel 2004)

None of these has a major impact on the actual parsing algorithm, however.

Structural ambiguity: prepositional phrase attachment

Pope Francis ***on*** *Saturday appointed a victim* ***of*** *sexual abuse and a senior cardinal known* ***for*** *his zero-tolerance approach* ***to*** *a new group charged* ***with*** *advising the Catholic Church* ***on*** *how to respond* ***to*** *the problem* ***of*** *sexual abuse* ***of*** *children.* (Wall St. Journal, 2014-03-22)

- The prepositional phrase *on Saturday* is construed as a modifier of *Pope Francis* rather than of *appointed.*
- The phrase *to a new group charged with advising the Catholic Church on how to respond to the problem of sexual abuse of children* is construed as a modifier of *zero tolerance approach* rather than of *appointed*.

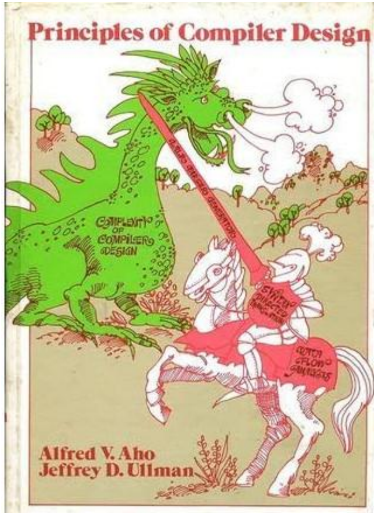
http://languagelog.ldc.upenn.edu/nll/?p=11286

CFGs and compilers

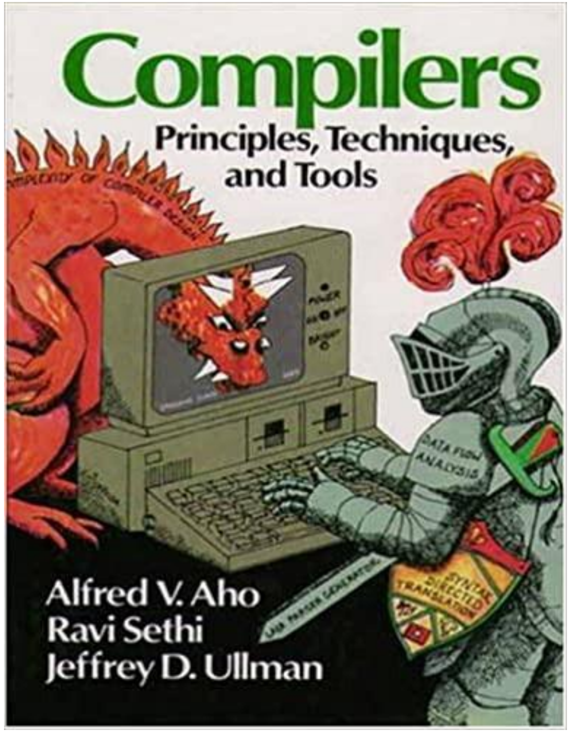
CFG parsing has been studied *extensively* for compiler design long before it was used to parse natural language.

Computer languages are designed to be unambiguous or even deterministic, and compiler designers can take advantages of more-efficient algorithms for these special cases (e.g., the Earley algorithm, the LL and LALR parsers, etc.).

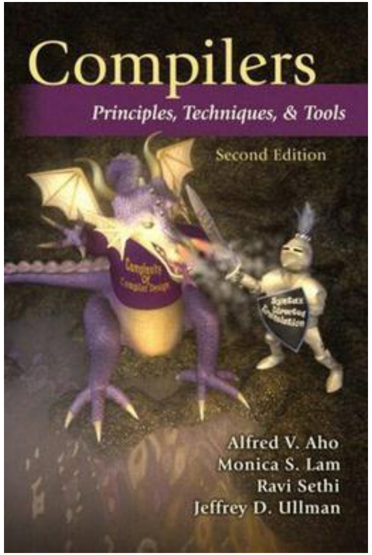
Probably the best known parsing algorithm for natural language is the simultaneously-invented Cocke-Younger-Kasami (CYK or CKY) algorithm, a simple bottom-up parsing algorithm.



Aho & Ullman 1977



Aho, Sethi & Ullman 1986



Aho, Lam, Sethi & Ullman 2006

Example grammar

```
S   →  NP  VP
VP  →  VP  PP        # Permits PPs to adjoin to VP.
VP  →  V   NP        # Permits transitive VPs.
VP  →  V             # Permits intransitive VPs.
PP  →  P NP
NP  →  D N           # Permits article-noun NPs.
NP  →  NP PP         # Permits PPs to adjoin to NP.
NP  →  N             # Permits bare NPs.
NP  →  Pagliaccio
V   →  eats
N   →  spaghetti | fork
P   →  with
D   →  a
```

Eliminate unary productions

Before:

```
VP  →   V
NP  →   N
```

After:

```
VP  →   eats
NP  →   spaghetti, fork
```

Grammar representation structure

The preterminal rules are simple mappings, and only used once.

```
preterminals: Dict[str, List[str]]
```

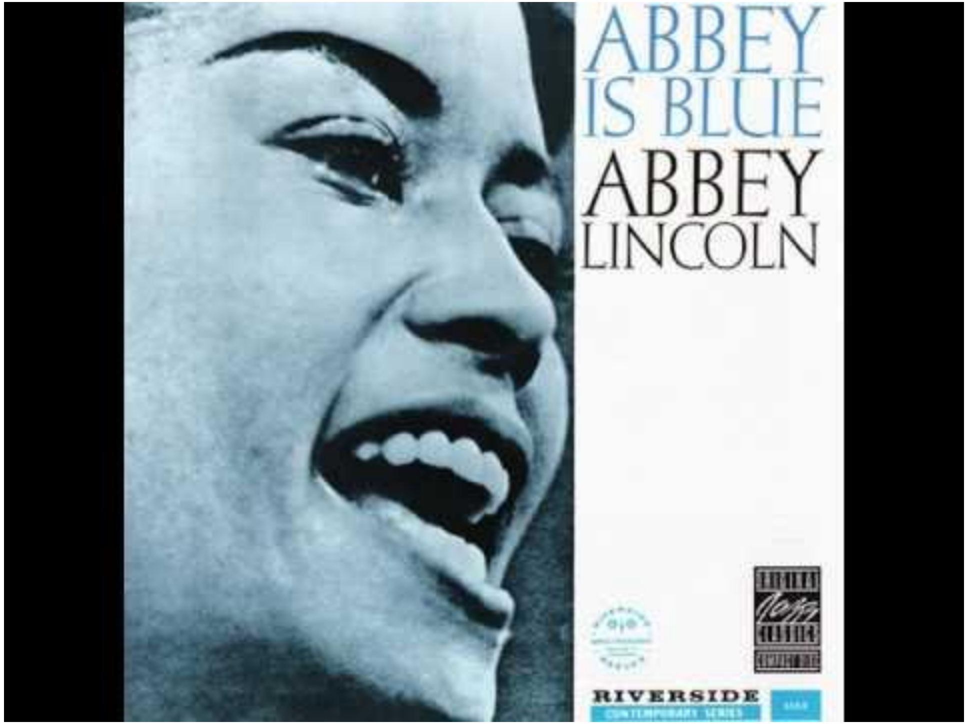
The non-terminal rules map from pairs of strings (the children/daughters/productions) to non-terminals.

```
nonterminals: Dict[Tuple[str, str], List[str]]
```

Looking up productions to find candidate non-terminals is known as *grammar intersection*. Intersection is performed O(n^3) times, so it *has* to be constant-time.

Pagliaccio eats spaghetti with a fork

(Do you see the ambiguity?)


ABBEY
IS BLUE
ABBEY
LINCOLN
RIVERSIDE
CONTEMPORARY SERIES

The ambiguity

1. Pagliaccio $[_{VP}$ $[_{VP}$ eats spaghetti] [with a fork]]

   (The fork is an instrument for eating.)

2. Pagliaccio eats $[_{NP}$ $[_{NP}$ spaghetti] [with a fork]]

   (The fork is a tasty topping for the spaghetti.)

The CYK chart

Given a sentence of length *n*, construct an *n* x *n* table, empty above the diagonal, in which each cell is a container of non-terminal labels.

For efficient memory use you **might** want to construct a `Chart` class that has-a *n* x *n* / 2 + 1 array and overloads `__getitem__` and `__setitem__` so you can **pretend** you really do have a *n x n* table without wasting the extra space.

(If you find an error in the demo, let me know.)

Initial chart (plus an extra row at the bottom, for display)

Pagliaccio	eats	spaghetti	with	a	fork

Insert preterminals

NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 2 spans

S					
NP	**VP**	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 2 spans

S	**VP**				
NP	**VP**	**NP**	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 2 spans

S	VP				
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 2 spans

S	VP				
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 2 spans

S	VP			**NP**	
NP	VP	NP	P	**D**	**NP**
Pagliaccio	eats	spaghetti	with	a	fork

Size 3 spans

S					
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 3 spans

S					
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 3 spans

S					
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 3 spans

S					
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 3 spans

S					
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 3 spans

S					
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 3 spans

S			**PP**		
S	VP			**NP**	
NP	VP	NP	**P**	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 3 spans

S			**PP**		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 4 spans

S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 4 spans

S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 4 spans

S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 4 spans

S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 4 spans

S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 4 spans

S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 4 spans

		NP			
S			**PP**		
S	VP			NP	
NP	VP	**NP**	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

(This is the NP-adjunction analysis.)

Size 4 spans

		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 4 spans

		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 5 spans

		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 5 spans

		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 5 spans

		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 5 spans

		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 5 spans

	VP				
		NP			
S			PP		
S	VP			NP	
NP	**VP**	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

(This is the NP-adjunction analysis.)

Size 5 spans

	VP, **VP**				
		NP			
S			**PP**		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

(This is the VP-adjunction analysis.)

Size 5 spans

	VP, VP				
		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 5 spans

	VP, VP				
		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 6 spans

S, S					
	VP, VP				
		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 6 spans

S, S					
	VP, VP				
		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 6 spans

S, S					
	VP, VP				
		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 6 spans

S, S					
	VP, VP				
		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Size 6 spans

S, S					
	VP, VP				
		NP			
S			PP		
S	VP			NP	
NP	VP	NP	P	D	NP
Pagliaccio	eats	spaghetti	with	a	fork

Recognition vs. parsing

When there an S in the top-left cell of the chart, the sentence is *recognized* by the grammar; i.e., there is at least one complete parse of the sentence.

To actually *parse* (and to extract the parses), we have to keep backtraces similar to Viterbi decoding.

Generalizing to PCFGs

Whereas above each cell was a container of non-terminals, in PCFG parsing each cell is a container of triples of;

1. non-terminal
2. log-probability
3. backtraces

Much like with the Viterbi algorithm, we obtain the **highest-probability parse** by tracing back from the **highest-probability *S*-triple in the top-left cell**.

Data

Unlike dependency parsing (the subject of next week's lecture), constituency parsing treebanks are available for maybe a dozen languages, and many are proprietary. Two widely studied ones are:

- English: The Wall St. Journal portion of the Penn Treebank, 3rd edition (Marcus et al. 1999; first release was 1993, I think)
- German: The Tiger2 corpus (Brants et al. 2002)

For German, which is "morphologically rich and less-configurational", it is generally agreed that parsing is poor without the assistance of a high-quality morphological analyzer (e.g., Dubey 2005, Fraser et al. 2013, Dehouck and Denis 2018).

Evaluation

Parsing performance is usually evaluated using the *F*-score of the constituents:

- Precision: the number of the constituents in the candidate/hypothesis parse that are present in the gold/reference parse
- Recall: the number of constituents in the gold/reference parse that are present in the candidate/hypothesis parse
- F-score: the harmonic mean of precision and recall

PARSEVAL (Black et al. 1991), a widely used evaluation script, also includes a measure that counts *crossing brackets*.

Software

- Charniak (2000) is the best-known of the PCFG-plus-discriminative-reranking parsers and is still used for feature extraction (etc.)
- The BUBS "grammar-agnostic" parser (Bodenstab et al. 2011, Dunlop et al. 2011) is a good tradeoff between simplicity and generality, speed and accuracy; unfortunately it's in Java

This week

Please read:

- Eisenstein ch. 10, and
- (optionally) Jurafsky & Martin ch. 12-14.

Project ideas

- Implement either of the "Berkeley" parsers (Klein & Manning or Petrov et al.) and evaluate using the standard split of the Wall St. Journal portion of PTB-3.
- Perform a hybrid (automatic-manual) error analysis of prepositional phrase attachment ambiguity.
- Study the effects of morphological features in constituency parsing along the lines of Fraser et al. 2013, etc.