Bayesian models of language acquisition
or
Where do the rules come from?

Mark Johnson

joint work with Tom Griffiths and Sharon Goldwater

November, 2007
Outline

Why *computational* linguistics?

Grammars (finite descriptions of languages)

Learning morphology with adaptor grammars

Word segmentation using adaptor grammars

Conclusions

Technical details
Why is there a field of **computational** linguistics?

- Language is a symbolic system (involves manipulation of *meaning-bearing entities*)
  \[\Rightarrow\] linguistic processes are *computational* processes

- Linguistic processes have a computational dimension (alongside formal, psychological, neurological, developmental, etc.)

- Empirical properties of linguistic processes motivating this work:
  - speakers/hearers can produce and comprehend sentences (parsing, generation)
  - children, starting from the same initial state, can learn any human language (acquisition)
  - these processes are faced with an astronomically large number of different possible sentences
### Linguistic processing as inference

<table>
<thead>
<tr>
<th>Comprehension</th>
<th>Acquisition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentence</td>
<td>sentences</td>
</tr>
<tr>
<td>“grammar”</td>
<td>“universal grammar”</td>
</tr>
<tr>
<td>meaning (parse)</td>
<td>grammar</td>
</tr>
</tbody>
</table>

- Research agenda: *What information* is used in these processes?
Bayesian learning

\[ P(\text{Hypothesis} | \text{Data}) \propto P(\text{Data} | \text{Hypothesis}) \cdot P(\text{Hypothesis}) \]

- A Bayesian model \textit{integrates information from multiple sources}
  - \textit{Likelihood} reflects how well grammar fits input data
  - \textit{Prior} encodes a priori preferences for particular grammars
- The \textit{prior is as much a linguistic issue as the grammar}
  - Priors can be sensitive to linguistic structure (e.g., words should contain vowels)
  - Priors can encode \textit{linguistic universals} and \textit{markedness preferences}
- Priors can prefer \textit{smaller grammars} (Occam’s razor, MDL)
- A Bayesian model is \textit{not} an implementation or algorithm
Outline

Why computational linguistics?

Grammars (finite descriptions of languages)

Learning morphology with adaptor grammars

Word segmentation using adaptor grammars

Conclusions

Technical details
Probabilistic context-free grammars

- *Context-Free Grammars* (CFGs) provide rules (building blocks) for constructing phrases and sentences
- In a *Probabilistic CFG* (PCFG), each rule has a probability (cost)
- Probability of a tree is the *product of the probabilities of the rules* used to construct it

<table>
<thead>
<tr>
<th>Rule r</th>
<th>P((\theta_r))</th>
<th>Rule r</th>
<th>P((\theta_r))</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (\rightarrow) NP VP</td>
<td>1.0</td>
<td>S (\rightarrow) NP VP</td>
<td>0.45</td>
</tr>
<tr>
<td>NP (\rightarrow) Hillary</td>
<td>0.75</td>
<td>NP (\rightarrow) Hillary</td>
<td>0.45</td>
</tr>
<tr>
<td>VP (\rightarrow) barks</td>
<td>0.6</td>
<td>VP (\rightarrow) barks</td>
<td>0.4</td>
</tr>
<tr>
<td>NP (\rightarrow) Barack</td>
<td>0.25</td>
<td>VP (\rightarrow) snores</td>
<td>0.4</td>
</tr>
</tbody>
</table>

\[
P \left( \begin{array}{c}
S \\
NP \\
\mid \\
Hillary \\
\mid \\
barks
\end{array} \right) = 0.45
\]

\[
P \left( \begin{array}{c}
S \\
NP \\
\mid \\
Barack \\
\mid \\
snores
\end{array} \right) = 0.1
\]
Estimating PCFG rule probabilities from trees

- Prior over rule probabilities: product of Dirichlet distributions with parameters $\alpha_r$ for each rule $r$
- Conjugacy $\Rightarrow$ posterior is also product of Dirichlets, with parameters $\alpha_r + n_r$, where $n_r$ is number of times $r$ occurs in trees

<table>
<thead>
<tr>
<th>Rule $r$</th>
<th>$\alpha_r$</th>
<th>$n_r$</th>
<th>$\alpha_r + n_r$</th>
<th>Sample $\theta_r$</th>
<th>Sample $\theta_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S $\rightarrow$ NP VP</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NP $\rightarrow$ Hillary</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0.61</td>
<td>0.51</td>
</tr>
<tr>
<td>NP $\rightarrow$ Barack</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.39</td>
<td>0.49</td>
</tr>
<tr>
<td>VP $\rightarrow$ barks</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0.93</td>
<td>0.72</td>
</tr>
<tr>
<td>VP $\rightarrow$ snores</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.07</td>
<td>0.28</td>
</tr>
</tbody>
</table>
The Dirichlet distribution

\[ \text{Dir}(\theta \mid \alpha) = \frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)} \prod_i \theta_i^{\alpha_i - 1} \]

- Increasingly concentrated when \( \alpha_i \gg 1 \) or \( \alpha_i \ll 1 \)
- When \( \alpha_i \ll 1 \), \( P(\theta_i) \) is concentrated around 0

\( \Rightarrow \) prior prefers not to use rule
Estimating rule probabilities from strings alone

- Hillary barks
- Barack barks
- Barack barks

- No closed-form solution, but various *Markov Chain Monte Carlo sampling algorithms* and *Variational Bayes approximations* have been developed

- Guess initial production probabilities

- Repeat:
  - produce sample parses for strings in training corpus
  - count rules in sampled parse trees
  - sample production probabilities from rule counts as before

- Repeat this long enough, converges to samples from posterior

- (It is possible to *integrate out* the rule probabilities)
Estimating rule probabilities for toy grammars

Initial rule probs

<table>
<thead>
<tr>
<th>rule</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>VP $\rightarrow$ V</td>
<td>0.2</td>
</tr>
<tr>
<td>VP $\rightarrow$ V NP</td>
<td>0.2</td>
</tr>
<tr>
<td>VP $\rightarrow$ NP V</td>
<td>0.2</td>
</tr>
<tr>
<td>VP $\rightarrow$ V NP NP</td>
<td>0.2</td>
</tr>
<tr>
<td>VP $\rightarrow$ NP NP V</td>
<td>0.2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Det $\rightarrow$ the</td>
<td>0.1</td>
</tr>
<tr>
<td>N $\rightarrow$ the</td>
<td>0.1</td>
</tr>
<tr>
<td>V $\rightarrow$ the</td>
<td>0.1</td>
</tr>
</tbody>
</table>

“English” input (50 sentences)
the dog bites
the dog bites a man
a man gives the dog a bone
...

“pseudo-Japanese” input (50 sentences)
the dog bites
the dog a man bites
a man the dog a bone gives
...
Probability of “English”

Posterior probability of parses (per sentence)
Rule probabilities from “English”

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V \rightarrow \text{the}$</td>
<td>0.5</td>
</tr>
<tr>
<td>$V \rightarrow \text{Det}$</td>
<td>0.75</td>
</tr>
<tr>
<td>$V \rightarrow \text{NP VP}$</td>
<td>1.0</td>
</tr>
<tr>
<td>$V \rightarrow NP NP VP$</td>
<td>0.75</td>
</tr>
<tr>
<td>$V \rightarrow NP$</td>
<td>0.5</td>
</tr>
<tr>
<td>$V \rightarrow \text{the}$</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Graph showing the probability of different rules over iterations.
Probability of “Japanese”

Posterior probability of parses (per sentence)

Iteration

0 1 2 3 4 5 6 7

0 0.0001 0.001 1e-05

1e-05

76543210
Rule probabilities from “Japanese”

- **Rule probabilities**
  - $V \rightarrow \text{theN}$
  - $\text{theN} \rightarrow \text{theVP}$
  - $\text{VP} \rightarrow \text{NP NP}$
  - $\text{VP} \rightarrow \text{V VP}$
  - $\text{VP} \rightarrow \text{NP NP V}$
  - $\text{Det} \rightarrow \text{the}$
  - $\text{N} \rightarrow \text{the}$
  - $V \rightarrow \text{the}$

- **Iteration and Probability**
  - Iteration 0: 0.25
  - Iteration 1: 0.5
  - Iteration 2: 0.75
  - Iteration 3: 1

- **Graph**: Rule probability vs Iteration
Summary so far

- Simple algorithm for learning rule probabilities: learn from your current “best guesses”
  - requires learner to parse the input sentences
- “Glass box” models: learner’s prior knowledge and learnt generalizations are *explicitly represented*
- We’ve seen how to estimate the rule probabilities

*Where do the rules come from?*
Where do the rules come from?

- Maybe they’re all innate?
- Common approach: *generate and prune*
  - generate a large “superset” grammar (from where?)
  - use a “sparse” prior that prefers rules have zero probability
  - estimate rule probabilities
  - discard low probability rules
Estimation from real input

- ATIS treebank consists of 1,300 hand-constructed parse trees
- input consists of POS tags rather than words
- about 1,000 PCFG rules are needed to build these trees
Probability of training strings

$\alpha = 1$

$log P$ vs Iteration
Accuracy of parses of training strings

Accuracy of parses of training strings

Parse Accuracy (labeled f-score)

Iteration

\( \alpha = 1 \)
The PCFG model isn’t a good model of syntax

- Parse accuracy drops as likelihood increases
  - higher likelihood $\nRightarrow$ better parses
  - the statistical model is wrong
- Initialized estimator with correct parse trees
  - started with true rules and their probabilities
    $\Rightarrow$ poor performance not due to search error
- Evaluated on training data
  - poor performance not due to over-learning
Why didn’t it learn the right grammar?

- Higher likelihood $\not\Rightarrow$ better parse accuracy
  $\Rightarrow$ model is wrong

- What could be wrong?
  - Wrong grammar (Klein and Manning, Smith and Eisner)
  - Wrong training data (Yang)
  - Grammar *ignores semantics* (Zettlemoyer and Collins)

$\Rightarrow$ Develop models of syntax/semantics mapping, e.g., from sentences to (visual) contexts

$\Rightarrow$ Study simpler problems
Outline

Why *computational* linguistics?

Grammars (finite descriptions of languages)

**Learning morphology with adaptor grammars**

Word segmentation using adaptor grammars

Conclusions

Technical details
Learning agglutinative morphology

- Words consist of sequence of *morphemes* e.g., *talk* + *ing*, *jump* + *s*, etc.
- Given unanalyzed words as input training data, want to learn a grammar that:
  - generates words as a sequence of morphemes, and
  - correctly generates novel morphological combinations not seen in training data
- Training data: sequences of characters, e.g., `# talking #`
- Where we’re going:
  - CFGs are good ways of generating potentially useful structures
  - but *PCFGs are not good at describing the probability of structures*
A CFG for stem-suffix morphology

\[
\begin{align*}
\text{Word} & \rightarrow \text{Stem \ Suffix} \\
\text{Stem} & \rightarrow \text{Chars} \\
\text{Suffix} & \rightarrow \text{Chars} \\
\text{Chars} & \rightarrow \text{Char} \\
\text{Char} & \rightarrow \text{a \ | \ b \ | \ c \ | \ ...}
\end{align*}
\]

- Grammar generates acceptable structures
- But its units of generalization (rules) are “too small” to learn morphemes
A “CFG” with one rule per possible morpheme

Word $\rightarrow$ Stem Suffix
Stem $\rightarrow$ all possible stems
Suffix $\rightarrow$ all possible suffixes

- A rule for each morpheme
  $\Rightarrow$ “PCFG” can represent probability of each morpheme
- Unbounded number of rules (but only a finite number can be used in any finite training data set)
- *Assumes* $P(\text{Word}) = P(\text{Stem})P(\text{Suffix})$, which is false ...
Relative frequencies of inflected verb forms

Graph showing relative frequencies of verb forms.

- Expect
- Include
- Add
- Continue
- Report
Adaptor grammars: informal description

- An adaptor grammar has a set of PCFG rules
- These determine the possible structures as in a CFG
- A subset of the nonterminals are adapted
- Unadapted nonterminals expand by picking a rule and recursively expanding its children, as in a PCFG
- Adapted nonterminals can expand in two ways:
  - by picking a rule and recursively expanding its children, or
  - by generating a previously generated tree (with probability proportional to the number of times previously generated)
- Each adapted subtree behaves like a new rule added to the grammar
- The PCFG rules of the adapted nonterminals determine the prior over these trees
Adaptor grammars as generative processes

- The sequence of trees generated by an adaptor grammar are not independent
  - it learns from the trees it generates
  - if an adapted subtree has been used frequently in the past, it’s more likely to be used again
- (but the sequence of trees is exchangable)
- An unadapted nonterminal $A$ expands using $A \rightarrow \beta$ with probability $\theta_A \rightarrow \beta$
- An adapted nonterminal $A$ expands:
  - to a tree $\tau$ rooted in $A$ with probability proportional to the number of times $\tau$ was previously generated
  - using $A \rightarrow \beta$ with probability proportional to $\alpha_A \theta_A \rightarrow \beta$
Adaptor grammar morphology example

- Stem and Suffix rules generate all possible stems and suffixes
- Adapt Word, Stem and Suffix nonterminals
- Sampler uses “Chinese restaurant” processes
Morphology adaptor grammar (0)

**Word** restaurant
Word → Stem Suffix

**Stem** restaurant
Stem → #
Stem → # Chars

**Suffix** restaurant
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (1a)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → # Chars

**Suffix restaurant**
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (1b)

Word **restaurant**
Word → Stem Suffix

Stem **restaurant**
Stem → #
Stem → # Chars

Suffix **restaurant**
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (1c)

**Word** restaurant
Word → Stem Suffix

**Stem** restaurant
Stem → #
Stem → # Chars

**Suffix** restaurant
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a…z
Morphology adaptor grammar (1d)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → #Chars

**Suffix restaurant**
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (2a)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → # Char

**Suffix restaurant**
Suffix → #
Suffix → Char #

**Chars factory**
Chars → Char
Chars → Char Char
Char → a...z
Morphology adaptor grammar (2b)

- **Word** *restaurant*
  - Word → Stem Suffix
  - Stem
  - Stem → #
  - Stem → # Chars
  - Suffix
  - Suffix → #
  - Suffix → Char #
  - Suffix → Chars #

- **Chars factory**
  - Chars → Char
  - Chars → Char Chars
  - Char → a...z
Morphology adaptor grammar (2c)

**Word** restaurant  
Word → Stem Suffix

**Stem** restaurant  
Stem → #  
Stem → # Chars

**Suffix** restaurant  
Suffix → #  
Suffix → Chars #

**Chars factory**  
Chars → Char  
Chars → Char Chars  
Char → a ... z
Morphology adaptor grammar (2d)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → # Chars

**Suffix restaurant**
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (3)

**Word** restaurant
Word → Stem Suffix

**Stem** restaurant
Stem → #
Stem → # Chars

**Suffix** restaurant
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (4a)

**Word** restaurant
Word → Stem Suffix

**Stem** restaurant
Stem → #
Stem → # Char

**Suffix** restaurant
Suffix → #
Suffix → Char

**Chars factory**
Chars → Char
Chars → Char Char
Char → a...z
Morphology adaptor grammar (4b)

**Word** restaurant
Word → Stem Suffix

**Stem** restaurant
Stem → #

**Suffix** restaurant
Suffix → #

**Chars factory**
Chars → Char
Chars → Char Char
Char → a...z
Morphology adaptor grammar (4c)

**Word** restaurant
Word → Stem Suffix

**Stem** restaurant
Stem → #
Stem → # Chars

**Suffix** restaurant
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (4d)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → # Chars

**Suffix restaurant**
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Properties of adaptor grammars

- Possible trees generated by CFG rules but the probability of each adapted tree is estimated separately.
- Probability of a tree is:
  - proportional to the number of times seen before
  - “rich get richer” dynamics (Zipf distributions)
  - plus a constant times the probability of generating it via PCFG expansion.

⇒ Useful compound structures can be more probable than their parts.

- PCFG rule probabilities estimated from table labels
  - learns from types, not tokens
  - dampens frequency variation.
Learning Sesotho verbal morphology using an adaptor grammar

\[ \text{Word} \rightarrow (\text{Prefix1}) (\text{Prefix2}) (\text{Prefix3}) \text{ Stem} (\text{Suffix}) \]

- Sesotho is a Bantu language with complex morphology, not much phonology
- Demuth's Sesotho corpus contains morphological parses for 2,283 distinct verb types
- An adaptor grammar finds morphological analyses for these verbs
  - 62% f-score (morpheme accuracy)
  - 41% words completely correct
Outline

Why *computational* linguistics?

Grammars (finite descriptions of languages)

Learning morphology with adaptor grammars

**Word segmentation using adaptor grammars**

Conclusions

Technical details
Unigram model of word segmentation

- Unigram model: each word is generated independently
- Input is *unsegmented broad phonemic transcription* (Brent)
  Example: \( y u w a n t t u s i D 6 b u k \)
- Adaptor for *Word* non-terminal caches previously seen words

```
Words → Word^+
Word → Char^+
```

- Unigram word segmentation on Brent corpus: 54% token f-score, 59% type f-score
Unigram model often finds collocations

- Unigram word segmentation model assumes each word is generated independently
- But there are strong inter-word dependencies (collocations)
- Unigram model can only capture such dependencies by analyzing collocations as words
Combining morphology and word segmentation

Words $\rightarrow$ Word$^+$
Word $\rightarrow$ Stem Suffix
Word $\rightarrow$ Stem
Stem $\rightarrow$ Char$^+$
Suffix $\rightarrow$ Char$^+$

- Adaptors for Word, Stem and Suffix terminals
- Doesn’t do a good job of learning morphology, but does find interesting collocations
Modeling collocations improves segmentation

Sentence $\rightarrow$ Colloc$^+$
Colloc $\rightarrow$ Word$^+$
Word $\rightarrow$ Char$^*$

- A collocation consists of one or more words
- Both words and collocations are adapted
- Significantly improves word segmentation accuracy over unigram model (64% token f-score)
Simultaneously learning word segmentation and syllable structure

Sentence → Word+
Word → Syllable+
Syllable → (Onset) Rhyme
Onset → Consonant+
Rhyme → Nucleus (Coda)
Nucleus → Vowel+
Coda → Consonant+

• Word, Syllable, Onset, Nucleus and Coda are all adapted
• Seems to do a fairly good job of identifying syllable boundaries
• Doesn’t do as well at segmentation as unigram model (46% token f-score)
• but I haven’t tried tweaking the prior, or sampling longer . . .
Simultaneous word segmentation and syllable structure
Outline

Why computational linguistics?

Grammars (finite descriptions of languages)

Learning morphology with adaptor grammars

Word segmentation using adaptor grammars

Conclusions

Technical details
Summary and future work

- Adaptor grammars “adapt” their distribution to the strings they have generated
- They learn the subtrees of an adapted nonterminal they generate
- This makes adaptor grammars non-parametric; the number of subtrees they track depends on the data
- A variety of different linguistic phenomena can be described with adaptor grammars
- Because they are grammars, they are easy to design and compose
- But they still have a “context-freeness” that makes it impossible to express e.g., Goldwater’s bigram word segmentation model. Can we add context-sensitivity in a manageable way?
- The MCMC sampling algorithm used does not seem to scale well to large data or complicated grammars. Are there better estimators?
Outline

Why *computational* linguistics?

Grammars (finite descriptions of languages)

Learning morphology with adaptor grammars

Word segmentation using adaptor grammars

Conclusions

Technical details
From Chinese restaurants to Dirichlet processes

- Labeled Chinese restaurant processes take a base distribution $P_G$ and return a stream of samples from a different distribution with the same support
- The Chinese restaurant process is a sequential process, generating the next item conditioned on the previous ones
- We can get a different distribution each time we run a CRP (placing customers on tables and labeling tables are random)
- Abstracting away from sequential generation, a CRP maps $P_G$ to a distribution over distributions $\text{DP}(\alpha, P_G)$
- $\text{DP}(\alpha, P_G)$ is called a Dirichlet process with concentration parameter $\alpha$ and base distribution $P_G$
- Distributions in $\text{DP}(\alpha, P_G)$ are discrete (w.p. 1) even if the base distribution $P_G$ is continuous
PCFGs as recursive mixtures

The distributions over strings induced by a PCFG in *Chomsky-normal form* (i.e., all productions are of the form $A \rightarrow B \ C$ or $A \rightarrow w$, where $A, B, C \in N$ and $w \in T$) is $G_S$ where:

$$G_A = \sum_{A \rightarrow B \ C \in R_A} \theta_{A \rightarrow B \ C} G_B \cdot G_C + \sum_{A \rightarrow w \in R_A} \theta_{A \rightarrow w} \delta_w$$

$$(P \cdot Q)(z) = \sum_{xy=z} P(x)Q(y)$$

$$\delta_w(x) = 1 \text{ if } w = x \text{ and } 0 \text{ otherwise}$$

In fact, $G_A(x) = P(A \Rightarrow^* x | \theta)$, the sum of the probability of all trees with root node $A$ and yield $x$
Adaptor grammars

An adaptor grammar \((G, \theta, \alpha)\) is a PCFG \((G, \theta)\) together with a parameter vector \(\alpha\) where for each \(A \in N\), \(\alpha_A\) is the parameter of the Dirichlet process associated with \(A\).

\[
G_A \sim \text{DP}(\alpha_A, H_A) \text{ if } \alpha_A > 0 \\
\quad = H_A \text{ if } \alpha_A = 0
\]

\[
H_A = \sum_{A \rightarrow BC \in R_A} \theta_{A \rightarrow BC} G_B \cdot G_C + \sum_{A \rightarrow w \in R_A} \theta_{A \rightarrow w} \delta_w
\]

The probabilistic language defined by the grammar is \(G_S\).

There is one Dirichlet Process for each non-terminal \(A\) where \(\alpha_A > 0\).

Its base distribution \(H_A\) is a mixture of the language generated by the Dirichlet processes associated with other non-terminals.
Estimating adaptor grammars

• Need to estimate:
  ▶ table labels and customer count for each table
  ▶ (optional) probabilities of productions labeling tables

• Component-wise Metropolis-Hastings sampler
  ▶ $i$th component is the parse tree for input string $i$
  ▶ sample parse for input $i$ using grammar estimated from parses for other inputs

• Sampling directly from conditional distribution of parses seems intractable
  ▶ construct PCFG approximation on the fly
  ▶ each table label corresponds to a production in PCFG approximation