Syntax-based decoding

JHU Machine Translation class
April 1, 2014
Administrative

- Homework 4 out, due April 14
- Final project proposals due today
Where do grammars come from?

• We left off on Thursday with
  – a formalism for describing the relationship between two languages,
  – an loosely-sketched algorithm for producing translations

• Questions for today:
  – Where do synchronous grammars come from?
  – How do we decode with an ngram language model?
Data-driven grammar extraction

- Grammar rules are not written by hand, they are extracted from bilingual parallel corpora

<table>
<thead>
<tr>
<th>Arabic</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>فالتضييب لا يزال يمارس على نطاق واسع</td>
<td>Torture is still being practised on a wide scale.</td>
</tr>
<tr>
<td>وتتم عمليات الاعتقال والاحتجاز دون سبب بصورة روتينية</td>
<td>Arrest and detention without cause take place routinely.</td>
</tr>
<tr>
<td>وحان وقت التحلى بالبصيرة والشجاعة السياسية.</td>
<td>This is a time for vision and political courage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chinese</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>我国 能源 原材料 工业 生产 大幅度 增长.</td>
<td>China's energy and raw materials production up.</td>
</tr>
<tr>
<td>非国大 要求 阻止 更 多 被 拘留 人员 死亡.</td>
<td>ANC calls for steps to prevent deaths in police custody.</td>
</tr>
</tbody>
</table>

...
• Consider the redundancy in this phrase table

<table>
<thead>
<tr>
<th>Spanish</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>la bruja verde</td>
<td>the green witch</td>
</tr>
<tr>
<td>la bruja roja</td>
<td>the red witch</td>
</tr>
<tr>
<td>la bruja azúl</td>
<td>the blue witch</td>
</tr>
</tbody>
</table>

• What generalization is missing?
• Synchronous grammar rules

\[ X \rightarrow \text{la bruja } X_{(1)} \mid \mid | \text{ the } X_{(1)} \text{ witch} \]
\[ X \rightarrow \text{verde } \mid \mid | \text{ green} \]

• As a tree

```
    X
   / \   / \  
  X  X  X  X  
 / \ / \ / \  
la bruja verde the witch green
```
Hiero-style SCFG rules

• Most common type of SCFG in SMT is Hiero which has rules w/one non-terminal symbol
• Not as nice as linguistically motivated rules, does not capture the reordering in Urdu

```
X₁
/    \    /
/     \   /
 与  X₂  有  X₃
(北韩)  (邦交)
```

```
X₁
/    \    /
/     \   /
have  X₃  with  X₂
(diplomatic relations  North Korea)
```
Consider the redundancy in this phrase table:

<table>
<thead>
<tr>
<th>Spanish</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>la bruja verde</td>
<td>the green witch</td>
</tr>
<tr>
<td>la bruja roja</td>
<td>the red witch</td>
</tr>
<tr>
<td>la bruja azul</td>
<td>the blue witch</td>
</tr>
</tbody>
</table>

What generalization is missing?

Hiero abandons conventional English syntax.

Relies instead on evidence-based phrasal “subtractions”
Australia is one of the few countries that have diplomatic relations with North Korea.
Decoding

- We now have a way to obtain a synchronous grammar
- Last week, we sketched the decoding algorithm, which was based on parsing
- Today, we’ll cover it in more detail, and correct a crucial omission (ngram language models)
• We’ve discussed how syntactic differences between languages motivated reordering as a preprocessing step

Ich werde Ihnen den Report aushaendigen, damit Sie den eventuell uebernehmen koennen.

Ich werde aushaendigen Ihnen den Report, damit Sie koennen uebernehmen den eventuell.
• We’ve also discussed **synchronous grammar** rules, which describe the generation of sentences in pairs

<table>
<thead>
<tr>
<th>Urdu</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>S $\rightarrow$ NP① VP②</td>
<td>NP① VP②</td>
</tr>
<tr>
<td>VP $\rightarrow$ PP① VP②</td>
<td>VP② PP①</td>
</tr>
<tr>
<td>VP $\rightarrow$ V① AUX②</td>
<td>AUX② V①</td>
</tr>
<tr>
<td>PP $\rightarrow$ NP① P②</td>
<td>P② NP①</td>
</tr>
<tr>
<td>NP $\rightarrow$ hamd ansary</td>
<td>Hamid Ansari</td>
</tr>
<tr>
<td>NP $\rightarrow$ na\text{\text`} b sdr</td>
<td>Vice President</td>
</tr>
<tr>
<td>V $\rightarrow$ namzd</td>
<td>nominated</td>
</tr>
<tr>
<td>P $\rightarrow$ kylye</td>
<td>for</td>
</tr>
<tr>
<td>AUX $\rightarrow$ taa</td>
<td>was</td>
</tr>
</tbody>
</table>
...and how we could extract those rules automatically from text

![Diagram of X1 with relationships to X2, X3, 北韩, 有, 邦交 in Chinese, and have, X3, with, diplomatic relations, North Korea in English]
• How do we actually decode with these grammars?
• The solution is the CKY / CYK algorithm

Outline

• Parsing in one language
• Parsing in two languages with inversion transduction grammar (ITG)
• Decoding as parsing with synchronous context-free grammars (SCFG) and integrated language models
• Time-permitting: advanced topics
Review: monolingual parsing

Using the CKY algorithm to find (the best) structure for a sentence given a grammar
Formal definitions

- **Formal languages** are (possibly infinite) sets of strings that are generated by a grammar
  - e.g., \{a+\} is a language of all strings with one or more `a`
  - Its grammar could be written as
    \[
    A \rightarrow Aa \\
    A \rightarrow a
    \]
- We can view **natural languages** in this manner, too
  - e.g., the **English language** is the set of word sequences that constitute valid English sentences
  - We believe there to be a grammar that generates those sentences
  - We don’t know what it is, but we have some guesses and approximations
Given a sentence and a grammar, how do we find its structure?

We’ll use the CKY algorithm (Cocke-Kasami-Younger)

Basic idea: build small items before larger ones

**Sentence:**
```
Fred Jones was worn out
```

**Grammar:**
```
S → NP VP
VP → VBN PRT
PRT → RP
VP → VBD VP
NP → NNP NNP
NNP → Fred | Jones
VBD → was
VBN → worn
RP → out
```
Parsing with CKY

```
S → NP VP
VP → VBN  PRT
PRT → RP
VP → VBD  VP
NP → NNP  NNP
NNP → Fred | Jones
VBD → was
VBN → worn
RP → out

sentence
grammar
```
Implementation details

• Dynamic programming maintains a chart of items
  • Each cell item represents the dynamic programming state
    • (NNP,1,1), (S,1,5)
  • The chart is the collection of all items

• The score resolves alternate ways of constructing an item

• We also store backpointers: the items and rule used to construct each item
  a.k.a. “predecessor”

```c
struct item {
    // d.p. state
    string nt;
    int i, j;
    // backpointer
    float score;
    Rule* rule;
    item* rhs1,
    rhs2;
};
```
CKY algorithm

**input:** words[1..N]
for i in 1..N
  for each unary rule X → words[i]
    add (X,i,i) to the chart
for span in 1..N
  for i in 1..(N-span)
    j = i + span
    for k in i..j
      for rule X → Y Z
        if (Y,i,k) and (Z,k,j)
          add (X,i,j) to the chart

**output:** (S,1,N)
Parsing with CKY

item

nt = "S";
i = 1, j = 5;
score = -42.5;
Rule = &rule("S → NP VP")
rhs1 = &item(NP,1,2);
rhs2 = &item(VP,3,5);
Reconstructing the best parse

- We can reconstruct the best parse by following backpointers

```
while nodes.size() > 0:
    item = nodes.pop()
    print item
    nodes.append(item.rhsr)
    nodes.append(item.rhsl)
```

- We can reconstruct the best parse by following backpointers

```
nodes.append(item(S, 1, N))
```

- We can reconstruct the best parse by following backpointers

```
S → NP VP (1, 5)
NP → NNP NNP (1, 2)
NNP → Fred (1, 1)
NNP → Jones (2, 2)
VP → VBD VP (3, 5)
VBD → was (3, 3)
VP → VBN PRT (4, 5)
VBN → worn (4, 4)
PRT → RP (5, 5)
RP → out (5, 5)
```

- We can reconstruct the best parse by following backpointers

```
Fred NNP
Jones NP NNP
was VBD
worn VBN
out RP

Fred Jones was worn out
```
Fred Jones was worn out from caring for his often screaming and crying wife during the day but he couldn’t sleep at night for she in a stupor from the drugs that didn’t ease the pain would set the house ablaze with a cigarette.
• Deductive reasoning:
  • **axioms**: statements that are true or false (“it is raining”)
  • **inference rules**: statements that are conditionally true (“If it is raining and I am outside, I’ll get wet”)
  • **goals**: statements that are licensed by combinations of axioms, inference rules, and other conclusions (“I am wet”)
## Parsing as (weighted) deduction

- **Input:** words \( w[l..N] \)

<table>
<thead>
<tr>
<th>Axioms</th>
<th>( X \rightarrow w[i] )</th>
<th>for all ( (X \rightarrow w[i]) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inference rules</td>
<td>( X \rightarrow w[i] ) ( (X, i, i) ) ( (B, i, j) ) ( (C, j, k) ) ( A \rightarrow BC ) ( (A, i, k) )</td>
<td>in bottom-up order (smaller spans first)</td>
</tr>
<tr>
<td>Goal</td>
<td>( (S, l, n) )</td>
<td></td>
</tr>
</tbody>
</table>
Complexity

• Complexity of parsing is $O(Gn^3)$
  • $G$ - number of (binarized) rules in the grammar
  • $n$ - length of the sentence

• All those rules were binary; what about longer rules?
  • e.g.,

```
NP
  /\  \\
DT JJ NN
```

• We have to enumerate every split point!
CKY algorithm

input: words[1..N]
for i in 1..N
  for each unary rule X → words[i]
    add (X,i,i) to the chart
for span in 1..N
  for i in 1..(N-span)
    j = i + span
    for k₁ in i..j-1
      for k₂ in k₁..j
        for rule X → W Y Z
          if (W,i,k₁) and (Y,k₁,k₂) and (Z,k₂,j)
            add (X,i,j) to the chart
output: (S,1,N)
Binarization into Chomsky Normal Form

• In general, for a rule with $k$ RHS items, complexity is $O(n^{k+1})$ (and cumbersome, since you have to explicitly add inner loops to enumerate them)

• Fortunately, we can binarize rules to make them all have a rank of 2

```
NP
  /\    /
 |  DT JJ
NN
```

```
NP
  /\    /
| DT JJ:NN
|  JJ NN
```

Only one split point

`two split point: O(n^4)`

New nonterminal uniquely identifies subtree
CKY algorithm

- In summary, monolingual parsing:
  - finds the best structure
  - works bottom-up, enumerating all spans, from small to large, building searching for applicable rules and building new chart items
  - works with the binarized form of a grammars (easily unbinarized afterward) for a complexity of $O(Gn^3)$
  - all grammars are binarizable
Synchronous parsing
Synchronous parsing

• We can extend CKY to parse two languages at once!

• Consider the following grammar:

  \[
  \begin{aligned}
  A & \rightarrow \text{fat, gordos} \quad \text{(lexical)} \\
  A & \rightarrow \text{thin, delgados} \\
  N & \rightarrow \text{cats, gatos} \\
  VP & \rightarrow \text{eat, comen} \\
  NP & \rightarrow A^{(1)} N^{(2)}, N^{(2)} A^{(1)} \quad \text{(inverted)} \\
  S & \rightarrow NP^{(1)} VP^{(2)}, NP^{(1)} VP^{(2)} \quad \text{(straight)}
  \end{aligned}
  \]

• and the following sentence pair:

  fat cats eat / gatos gordos comen
• We now have to enumerate *pairs* of spans
  • instead of (i,j)...
  • ...we have (i,j) and (s,t)

• For each of the bilingual blocks, we attempt to match both **straight** and **inverted** rules

```
Synchronous parsing

A → fat, gordos
N → cats, gatos
VP → eat, comen
VP → eat, como
NP → A(1) N(2), N(2) A(1)
S → NP(1) VP(2), NP(1) VP(2)
```

<table>
<thead>
<tr>
<th></th>
<th>(3,3,3,3)</th>
<th>(3,3,3,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>comen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gordos</td>
<td>(1,1,2,2)</td>
<td>(1,2,1,2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gatos</td>
<td>(2,2,1,1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cats</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32
Relation to monolingual parsing

- Why do we combine like this?
  - Think about monolingual CKY: combine adjacent spans
  - These pieces are adjacent in both languages; it’s only when we consider them together that reordering comes into play

- Why can’t we do this?
  - It doesn’t make sense!

- What about these?
  - Possible, but complex
CKY for synchronous parsing

**input**: source[1..N], target[1..M]

for span₁ in 1..N
  for i in 1..(N−span₁)
    j = i + span₁
    for k in i..j
      for span₂ in 1..M
        for s in 1..(M−span₂)
          t = s + span₂
          for u in s..t
            for rule X → [Y Z]
              if (Y,i,k,s,u) and (Z,k,j,u,v) then
                add (X,i,j,s,t) to chart

**output**: (S,1,N,1,M)
Synchronous parsing

• Complexity:
  \[ O(GN^3M^3) \approx O(GN^6) \]

• Why?
  • We have to enumerate all valid combinations of six variables
  • This can be seen in the six nested loops of the algorithm

\[
\begin{array}{c}
A \rightarrow \text{fat, gordos} \\
N \rightarrow \text{cats, gatos} \\
VP \rightarrow \text{eat, comen} \\
VP \rightarrow \text{eat, como} \\
NP \rightarrow A^{(1)} N^{(2)}, N^{(2)} A^{(1)} \\
S \rightarrow NP^{(1)} VP^{(2)}, NP^{(1)} VP^{(2)}
\end{array}
\]
Visualization of $O(GN^6)$ complexity

**input:** source[1..N], target[1..M]

for span₁ in 1..N
    for i in 1..(N–span₁)
        j = i + span₁
        for k in i..j
            for span₂ in 1..M
                for s in 1..(M–span₂)
                    t = s + span₂
                    for u in s..t
                        for rule X → [Y Z]
                            if (Y,i,k,s,u) and (Z,k,j,u,v) then
                                add (X,i,j,s,t) to chart

**output:** (S,1,N,1,M)
Synchronous binarization

• In the above, we considered two nonterminals (per side)

• What if we want more (Zhang et al., 2006)?

\[
S \rightarrow \text{NP}^{(1)} \text{VP}^{(2)} \text{PP}^{(3)} , \text{NP}^{(1)} \text{PP}^{(3)} \text{VP}^{(2)} \\
\text{NP} \rightarrow \text{Powell, Baoweier} \\
\text{VP} \rightarrow \text{held a meeting, juxing le huitan} \\
\text{PP} \rightarrow \text{with Sharon, yu Shalong}
\]

• Three nonterminals? No problem:

\[
S \rightarrow V_{\text{NP-PP}} \text{ VP} \quad \text{or} \quad S \rightarrow \text{NP} \ V_{\text{PP-VP}} \\
V_{\text{NP-PP}} \rightarrow \text{NP} \ \text{PP} \\
V_{\text{PP-VP}} \rightarrow \text{PP} \ \text{VP}
\]

• More?
Permutations

• The nonterminals in the right-hand side of a rule define a permutation between the languages

  • we assume the source language nonterminals are in order (wlog)

  • intermingled terminal symbols do not affect binarization ability

• Example:

  \[ S \rightarrow \text{NP}^{(1)} \text{VP}^{(2)} \text{PP}^{(3)}, \text{NP}^{(1)} \text{PP}^{(3)} \text{VP}^{(2)} \]

  • permutation: 1 3 2
Bad news: synchronous grammars can’t be binarized in the general case (Shapiro & Stephens, 1991; Wu, 1997)

Famous examples: the (2,4,1,3) and (3,1,4,2) permutations

What makes these unbinarizable?

- Crucial: parsing works by combining adjacent elements
- No pair of alignments here is adjacent in both languages simultaneously

(*) Technically, you can binarize any synchronous grammar, but you may increase the fan-out, which mitigates the potential gains.
Synchronous binarization

• As the rank of a rule grows, the percentage of binarizable rules approaches 0

• In summary:
  • We can’t binarize all rules
  • The first unbinarizable rule has rank 4
Empirically, we don’t observe that many non-binarizable rules (Zhang et al., 2006):

- 99.7% of rules extracted were binarizable
- Many not were due to alignment errors

...and we can safely throw out the ones we do find

- 99.7% of rules extracted were binarizable
- Many not were due to alignment errors
Decoding as parsing
Synchronous decoding

• Enough parsing; what we care about is decoding

• Parsing is relevant, though, because we can view decoding as a task where we are doing synchronous parsing but we don’t happen to know the target side text

• This works by parsing with a source-side projection of the synchronous grammar rules

  • At the end, we can follow backpointers to discover the most probable target side
Updated data structure

• Just like regular parsing, we combine items in pairs to produce new items over larger spans:

\[(A,1,1) \quad (N,2,2)\]

\[\underline{(NP,1,2)}\]

• However, we also have to maintain our guess of the target side

\[A \rightarrow \text{fat, gordos}\]
\[N \rightarrow \text{cats, gatos}\]
\[VP \rightarrow \text{eat, comen}\]
\[VP \rightarrow \text{eat, como}\]
\[NP \rightarrow A^{(1)} \quad N^{(2)} ,\quad N^{(2)} \quad A^{(1)}\]
\[S \rightarrow NP^{(1)} \quad VP^{(2)} ,\quad NP^{(1)} \quad VP^{(2)}\]
Decoding

- Again, a bottom-up process

**Legend**
- straight rule application
- inverted rule application

\[
\begin{align*}
A & \rightarrow \text{fat, gordos} & 1.0 \\
N & \rightarrow \text{cats, gatos} & 1.0 \\
VP & \rightarrow \text{eat, comen} & 0.1 \\
VP & \rightarrow \text{eat, como} & 0.9 \\
NP & \rightarrow A^{(1)} N^{(2)}, N^{(2)} A^{(1)} & 1.0 \\
S & \rightarrow NP^{(1)} VP^{(2)}, NP^{(1)} VP^{(2)} & 1.0
\end{align*}
\]
Getting the translation

- Follow the backpointers
  - (S,1,3)
    - (NP,1,2)
      - (N,2,2) → gatos
    - (A,1,1) → gordos
  - (VP,3,3) → como
- translation:
  gatos gordos como

* cats fat lps-eat
What happened?

• We forgot the language model
• We’re inventing the target side (which is what decoding does), so we need to incorporate it
• How?
  • Stack-based decoding: we maintained the last word
  • Integration was easy because hypotheses always extended to the right
  • Here, hypotheses are merged either straight or inverted
Language model integration

**phrase-based**

\[
<s> \text{I tengo} \rightarrow \text{am} = <s> \text{I am}
\]

**synchronous grammars**

\[
\begin{align*}
\text{NP} \rightarrow & \quad \text{A}^{(1)} \quad \text{N}^{(2)}, \quad \text{N}^{(2)} \quad \text{A}^{(1)} \\
& = \quad \text{N}^{(1,2)} \\
\text{gatos gordos}
\end{align*}
\]

\[
\begin{align*}
\text{NP} \rightarrow & \quad \text{A}^{(1)} \quad \text{N}^{(2)}, \quad \text{A}^{(1)} \quad \text{N}^{(2)} \\
& = \quad \text{N}^{(1,2)} \\
\text{gordos gatos}
\end{align*}
\]
Language model integration

• We still maintain a chart of items, but now the items have to contain the target side words

• Just like regular parsing, we combine items in pairs to produce new items over larger spans

• When items are merged, we can use these words to compute a language model probability

• Formally, we are intersecting a \textit{context-free grammar} (the translation model) with a \textit{regular grammar} (Bar-Hillel et al., 1964; Wu, 1996)
• With dynamic programming, we only need a word on either side
  • (for bigram LMs; for the general case, see Chiang (2007, §5.3.2))
  • Following Chiang, we represent the elided middle portion with a ★
  • The complete string can be reconstructed by following the backpointers

```c
struct item {
  // d.p. state
  string nt;
  int i, j;
  string left_words;
  string right_words;
  // backpointer
  float score;
  Rule* rule;
  item* rhs1,
   rhs2;
}
```
Decoding with an integrated LM

```
A → fat, gordos  1.0
N → cats, gatos  1.0
VP → eat, comen  0.1
VP → eat, como  0.9
NP → A^{(1)} N^{(2)}, N^{(2)} A^{(1)}  1.0
S → NP^{(1)} VP^{(2)}, NP^{(1)} VP^{(2)}  1.0
```

- **S (1,3)**
  - ~ 0.1 \* P(comen | gordos)
  - gordos ★ comen

- **NP (1,2)**
  - ~ 1.0 \* P(gordos | gatos)
  - gatos ★ gordos

- **A (1,1)**
  - gordos  1.0

- **N (2,2)**
  - gatos  1.0

- **VP (3,3)**
  - comen  0.1

- **VP (3,3)**
  - como  0.9

- **fat**

- **cats**

- **eat**
Getting the translation

- Follow the backpointers
  - \((S, 1, 3, \text{gatos★comen})\)
    - \((\text{NP}, 1, 2, \text{gatos★gordos})\)
      - \((\text{N}, 2, 2, \text{gatos}) \rightarrow \text{gatos})
      - \((\text{A}, 1, 1, \text{gordos}) \rightarrow \text{gordos})
    - \((\text{VP}, 3, 3, \text{comen}) \rightarrow \text{comen})
  - translation:
    - \text{gatos gordos comen}
    - \text{cats fat 3pp-eat}
Pruning

• We have also not dealt much with ambiguity and competition amongst hypotheses

• In general, there are too many hypotheses to consider, so we keep only the top k of them (per input span (i,j))

• When considering a span (i,j) and a split point k, we have a large number of ways to combine items
  • there can be any number of applicable rules
  • there can be up to k items located at span (i,k)
  • there can be up to k items located at span (k,j)
Applying a unary rule

- The naive way is to consider the full cross product
• When considering a span \((i, j)\) of a length-\(N\) sentence:

  • **unary rules**: there are \(rk\) items to compute (\(r\) the number of rules, \(k\) the number of child items)

  • **binary rules**: there are \(Nrk^2\) items to compute (since there are \(O(N)\) split points)

• However, we’re only going to be keeping the top \(k\) of them!

  • this problem gets worse as \(k\) gets larger

• We’d like to avoid computing all of these new items, which we accomplish with **cube pruning**
Cube pruning

• We start with sorted lists of rules and the items they applied to

• Observation:
  • the best item comes from the best rule and the best cell
  • the next-best item uses either the 2nd best rule or the 2nd-best cell

<table>
<thead>
<tr>
<th>rule</th>
<th>rhs</th>
<th>rhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rule</th>
<th>rhs</th>
<th>rhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rule</th>
<th>rhs</th>
<th>rhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

best item  2nd-best  3rd-best
Applying a unary rule

- The Huang & Chiang (2005) way:
Cube pruning

- We haven’t discussed the language model, which complicates this procedure by making it *nonmonotonic*

- But that’s the basic idea
Today, we have reviewed

- Monolingual parsing
- Synchronous (bilingual) parsing
- Decoding as parsing with an intersected bigram language model

We have also briefly touched on efficiency considerations with cube pruning
Advanced topics
• We’d decoded in an ITG settings, where the rules all look like this:

\[
\begin{align*}
X & \rightarrow \text{boy, chico} \quad \text{(lexical)} \\
X & \rightarrow X^{(1)} X^{(2)}, X^{(2)} X^{(1)} \quad \text{(inverted)} \\
X & \rightarrow X^{(1)} X^{(2)}, X^{(1)} X^{(2)} \quad \text{(straight)}
\end{align*}
\]

• This is the closest thing to Chomsky Normal Form for synchronous grammars

• How do we decode with intermingled terminals and nonterminals?

\[
X \rightarrow \text{the } X^{(1)} \text{ was } X^{(2)}, \text{ el } X^{(1)} \text{ era } X^{(2)}
\]
• One answer: binarize (terminals can always be binarized):

\[ X \rightarrow \text{the } X \text{ was } X, \text{ el } X \text{ era } X \quad \text{X} \rightarrow \text{the } X_{174}, \text{ el } X_{174} \]

\[ X_{174} \rightarrow X \, X_{295}, \, X \, X_{295} \]

\[ X_{295} \rightarrow \text{was } X, \text{ era } X \]

• However, this is inefficient:
  
  • it leads to a huge blowup in the number of nonterminals
  
  • it introduces a split point that has to be searched over (avoidable in this case, but not always)
• Instead, we’d like to do implicit, **Earley-style binarization**
Advanced topics: spurious ambiguity

• **Spurious ambiguity** - multiple structures leading to the same interpretation

• Especially problematic in ITG with its weak grammar

• This can be addressed in various ways
  - Grammar canonical forms