Syntax-based decoding

Tuesday, March 27, 2012
Administrative issues

• Class moving to Shaffer 202 starting this Thursday
• Assignment 2 due a week from today
  • Extra TA office hours on Monday
  • Leaderboards...
• Get your revised proposals in!
Review (1)

• 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 → NP₁ VP₂</td>
<td>NP₁ VP₂</td>
</tr>
<tr>
<td>VP → PP₁ VP₂</td>
<td>VP₂ PP₁</td>
</tr>
<tr>
<td>VP → V₁ AUX₂</td>
<td>AUX₂ V₁</td>
</tr>
<tr>
<td>PP → NP₁ P₂</td>
<td>P₂ NP₁</td>
</tr>
<tr>
<td>NP → hamd ansary</td>
<td>Hamid Ansari</td>
</tr>
<tr>
<td>NP → najb sdr</td>
<td>Vice President</td>
</tr>
<tr>
<td>V → namzd</td>
<td>nominated</td>
</tr>
<tr>
<td>P → klyye</td>
<td>for</td>
</tr>
<tr>
<td>AUX → taa</td>
<td>was</td>
</tr>
</tbody>
</table>
...and how we could extract those rules automatically from text.

\[
X_1 \\
\text{与} \\
\text{北韩}
\]

\[
X_1 \\
X_2 \\
\text{有} \\
\text{邦交}
\]

\[
X_1 \\
\text{have} \\
X_3 \\
\text{with} \\
\text{North Korea}
\]

\[
\text{diplomatic relations}
\]
Today

• 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 \( a \)s
  
  • 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:

\[
\begin{align*}
S & \rightarrow \quad \text{NP \ VP} \\
VP & \rightarrow \quad \text{VBN ~ PRT} \\
PRT & \rightarrow \quad \text{RP} \\
VP & \rightarrow \quad \text{VBD ~ VP} \\
NP & \rightarrow \quad \text{NNP ~ NNP} \\
NNP & \rightarrow \quad \text{Fred | Jones} \\
VBD & \rightarrow \quad \text{was} \\
VBN & \rightarrow \quad \text{worn} \\
RP & \rightarrow \quad \text{out}
\end{align*}
\]
Parsing with CKY

sentence

grammar

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

Fred Jones was worn out
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

```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

```
    nodes.append(item(S,1,N))
    while nodes.size() > 0:
        item = nodes.pop()
        print item
        nodes.append(item.rhsr)
        nodes.append(item.rhsl)
```

- nodes

```
(S,1,5)
(NP,3,5)
(NNP,2,2)
(NP,1,2)
(NNP,1,1)
(VBD,3,3)
(VP,4,5)
(RP,5,5)
(VBN,4,4)
(PRT,5,5)
```

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)
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 $w[1..N]$

<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>$X \rightarrow w[i]$</td>
<td>$A \rightarrow BC$</td>
</tr>
<tr>
<td></td>
<td>$(X, i, i)$</td>
<td>$(B, i, j)$, $(C, j, k)$, $(A, i, k)$</td>
</tr>
<tr>
<td></td>
<td>in bottom-up order (smaller spans first)</td>
<td></td>
</tr>
<tr>
<td>Goal</td>
<td>$(S, 1, n)$</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:** \( \text{words}[1..N] \)
for \( i \) in 1..N
for each unary rule \( X \rightarrow \text{words}[i] \)
    add \((X,i,i)\) to the chart
for span in 1..N
    for \( i \) in 1..(N-span)
        \( j = i + \text{span} \)
        for \( k_1 \) in \( i...j-1 \)
            for \( k_2 \) in \( k_1...j \)
                for rule \( X \rightarrow W Y Z \)
                    if \((W,i,k_1)\) and \((Y,k_1,k_2)\) and \((Z,k_2,j)\)
                        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

![Diagram showing binarization process]

two split points: $O(n^4)$

new nonterminal uniquely identifies subtree

only one split point
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:

  \[
  \begin{align*}
  &A \rightarrow \text{fat, guapos} \\
  &A \rightarrow \text{thin, delgados} \\
  &N \rightarrow \text{cats, gatos} \\
  &VP \rightarrow \text{eat, comen} \\
  &NP \rightarrow A^{(1)} N^{(2)}, N^{(2)} A^{(1)} \\
  &S \rightarrow NP^{(1)} VP^{(2)}, NP^{(1)} VP^{(2)}
  \end{align*}
  \]

  (lexical, inverted, straight)

• and the following sentence pair:

  fat cats eat / gatos guapos comen
Synchronous parsing

• 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

\[
\begin{array}{c|c|c}
\text{comen} & (3,3,3,3) & (3,3,3,3) \\
\text{guapos} & (1,1,2,2) & (1,2,1,2) \\
\text{gatos} & (2,2,1,1) & & \\
& \text{fat} & \text{cats} & \text{eat} \\
\end{array}
\]
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 span\(_1\) in 1..N
  for i in 1..(N–span\(_1\))
    j = i + span\(_1\)
    for k in i..j
      for span\(_2\) in 1..M
        for s in 1..(M–span\(_2\))
          t = s + span\(_2\)
          for u in s..t
            for rule X \(\rightarrow [Y \ Z]\)
              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

A → fat, guapos
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>(1,1,2,2)</td>
<td>(1,2,1,2)</td>
</tr>
<tr>
<td>guapos</td>
<td>(2,2,1,1)</td>
<td></td>
</tr>
<tr>
<td>gatos</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Visualization of O(GN^6) complexity

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

```plaintext
for span_1 in 1..N
    for i in 1..(N-span_1)
        j = i + span_1
        for k in i..j
            for span_2 in 1..M
                for s in 1..(M-span_2)
                    t = s + span_2
                    for u in s..t
                        times all rules...
                            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:

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

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

• More?
Permutations

- The nonterminals in the right-hand side of a rule define a permutation between the languages
  - wlog, we assume the source language nonterminals are in order
  - 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):

• ...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

Figure 6: The solid-line curve represents the distribution of all rules against permutation lengths. The dashed-line stairs indicate the percentage of non-binarizable rules in our initial rule set while the dotted-line denotes that percentage among all permutations.
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
• Just like regular parsing, we combine items in pairs to produce new items over larger spans:

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

\[
(NP,1,2)
\]

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

A \rightarrow \text{fat, guapos}
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)}
Decoding

• Again, a bottom-up process

A → fat, guapos
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)

Legend

- - - - inverted rule application

straight rule application
Getting the translation

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

- translation:
  gatos guapos 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**

\[
\langle s \rangle \text{ I} \quad + \quad \text{tengo} \rightarrow \text{am} \quad = \quad \langle s \rangle \text{ I am}
\]

**synchronous grammars**

\[
\begin{align*}
\text{NP} & \rightarrow \text{A}^{(1)} \quad \text{N}^{(2)}, \quad \text{N}^{(2)} \quad \text{A}^{(1)} \\
& = \\
\text{N}^{(1,2)} & \quad 1.0 \\
\text{gatos guapos}
\end{align*}
\]

\[
\begin{align*}
\text{NP} & \rightarrow \text{A}^{(1)} \quad \text{N}^{(2)}, \quad \text{A}^{(1)} \quad \text{N}^{(2)} \\
& = \\
\text{N}^{(1,2)} & \quad 1.0 \\
\text{guapos gatos}
\end{align*}
\]
• 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 *context-free grammar* (the translation model) with a *regular grammar* (Bar-Hillel et al., 1964; Wu, 1996)
Updated data structure

- 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

```
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 $\rightarrow$ fat, guapos
N $\rightarrow$ cats, gatos
VP $\rightarrow$ eat, comen
VP $\rightarrow$ eat, como
NP $\rightarrow$ A$^{(1)}$ N$^{(2)}$, N$^{(2)}$ A$^{(1)}$
S $\rightarrow$ NP$^{(1)}$ VP$^{(2)}$, NP$^{(1)}$ VP$^{(2)}$
Getting the translation

• Follow the backpointers
  • (S,1,3,gatos˒comen)
    • (NP,1,2,gatos˒guapos)
      • (N,2,2,gatos) → gatos
      • (A,1,1,guapos) → guapos
    • (VP,3,3,comen) → comen
  • translation:
    gatos guapos comen
    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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \rightarrow \langle \text{cong } X_{\square}, \text{from } X_{\square} \rangle$</td>
<td>2.1</td>
<td>5.1</td>
<td>8.2</td>
</tr>
<tr>
<td>$X \rightarrow \langle \text{cong } X_{\square}, \text{from the } X_{\square} \rangle$</td>
<td>5.5</td>
<td>8.5</td>
<td>11.5</td>
</tr>
<tr>
<td>$X \rightarrow \langle \text{cong } X_{\square}, \text{since } X_{\square} \rangle$</td>
<td>7.7</td>
<td>10.6</td>
<td>13.1</td>
</tr>
<tr>
<td>$X \rightarrow \langle \text{cong } X_{\square}, \text{through } X_{\square} \rangle$</td>
<td>11.1</td>
<td>14.3</td>
<td>17.3</td>
</tr>
</tbody>
</table>

$\Rightarrow$

- $[X, 5, 8; \text{from the } \ast \text{ the scheme}] : 2.1$
- $[X, 5, 8; \text{from the } \ast \text{ the plan}] : 5.1$
- $[X, 5, 8; \text{from the } \ast \text{ the scheme}] : 5.5$
- $[X, 5, 8; \text{since the } \ast \text{ the scheme}] : 7.7$
Cube pruning

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

Best item

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

2nd-best

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

3rd-best
Applying a unary rule

- The Huang & Chiang (2005) way:

<table>
<thead>
<tr>
<th>Rule Description</th>
<th>1</th>
<th>4</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \rightarrow \langle \text{cong } X_1, \text{ from } X_1 \rangle$</td>
<td>2.1</td>
<td>5.1</td>
<td>8.2</td>
</tr>
<tr>
<td>$X \rightarrow \langle \text{cong } X_1, \text{ from the } X_1 \rangle$</td>
<td>5.5</td>
<td></td>
<td>8.5</td>
</tr>
<tr>
<td>$X \rightarrow \langle \text{cong } X_1, \text{ since } X_1 \rangle$</td>
<td>7.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X \rightarrow \langle \text{cong } X_1, \text{ through } X_1 \rangle$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
Advanced topics: implicit binarization

• 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?

\[
X \rightarrow \text{the } X^{(1)} \text{ was } X^{(2)}, \text{ el } X^{(1)} \text{ era } X^{(2)}
\]
Advanced topics: implicit binarization

• One answer: binarize (terminals can always be binarized):

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

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

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

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

• 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