Quick Speech Synthesis

CMSC 35100

Natural Language Processing

April 29, 2003

Speech Synthesis

- Text to speech produces
  - Sequence of phones, phone duration, phone pitch
- Most common approach:
  - Concatentative synthesis
    - Glue waveforms together
- Issue: Phones depend heavily on context
  - Diphone models: mid-point to mid-point
    - Captures transitions, few enough contexts to collect (1-2K)

Speech Synthesis: Prosody

- Concatenation intelligible but unnatural
- Model duration and pitch variation
  - Could extract pitch contour directly
  - Common approach: TD-PSOLA
    - Time-domain pitch synchronous overlap and add
      - Center frames around pitchmarks to next pitch period
      - Adjust prosody by combining frames at pitchmarks for desired pitch and duration
      - Increase pitch by shrinking distance b/t pitchmarks
      - Can be squeaky
- Higher-level stress, accents, boundaries
  - ToBI model: align with synthetic TTS content

Parsing I:
CFGs & the Earley Parser

CMSC 35100

Natural Language Processing

April 29, 2003

Roadmap

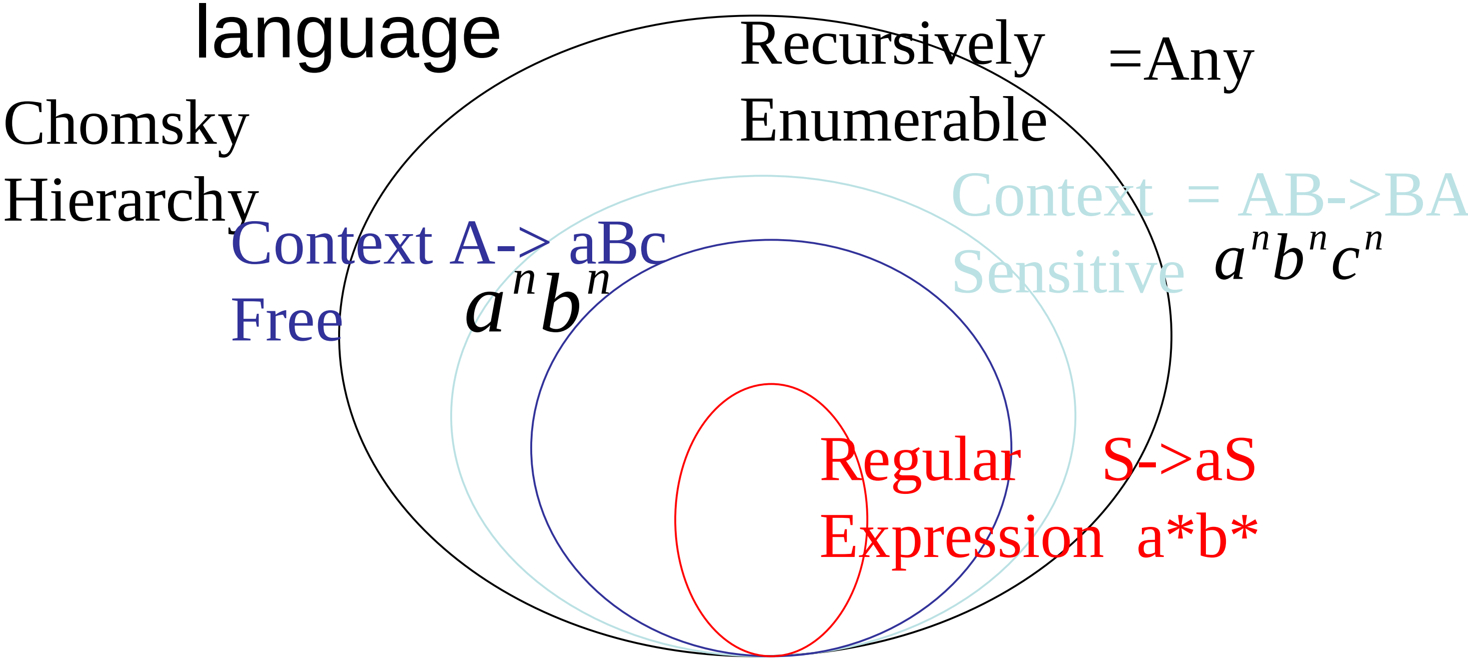
- Sentence Structure
  - Motivation: More than a bag of words
- Representation:
  - Context-free grammars
    - Chomsky hierarchy
- Parsing:
  - Accepting & analyzing
  - Combining top-down & bottom-up constraints
    - Efficiency
  - Earley parsers

More than a Bag of Words

- Sentences are structured:
  – Impacts meaning:
    - Dog bites man vs man bites dog
  – Impacts acceptability:
    - Dog man bites
- Composed of constituents
  – E.g. The dog bit the man on Saturday.
    - On Saturday, the dog bit the man.

Sentence-level Knowledge: Syntax

- Language models
  - More than just words: “banana a flies time like”
  - Formal vs natural: Grammar defines language



Representing Sentence Structure

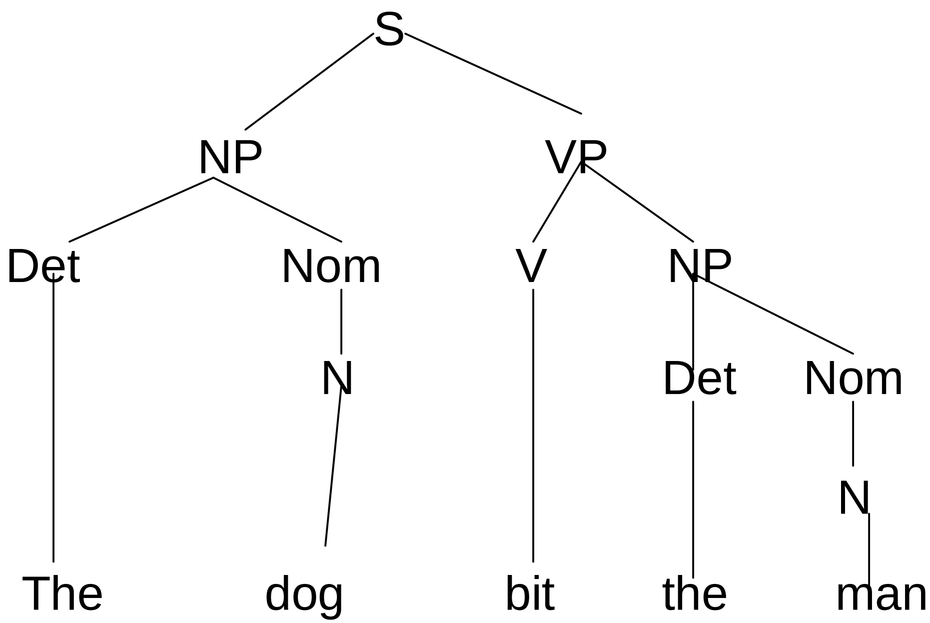
- Not just FSTs!
  - Issue: Recursion
    - Potentially infinite: It's very, very, very,…..
- Capture constituent structure
  - Basic units
  - Subcategorization (aka argument structure)
  - Hierarchical

Representation:
Context-free Grammars

- CFGs: 4-tuple
  - A set of terminal symbols: Σ
  - A set of non-terminal symbols: N
  - A set of productions P: of the form A -> α
    - Where A is a non-terminal and α in (Σ U N)*
  - A designated start symbol S
- L = W|w in Σ* and S=>*w
  - Where S=>*w means S derives w by some seq

Representation: Context-free Grammars

- Partial example
  - Σ: the, cat, dog, bit, bites, man
  - N: NP, VP, AdjP, Nominal
  - P: S-> NP VP; NP -> Det Nom; Nom-> N Nom|N
  - S



Grammar Equivalence and Form

- Grammar equivalence
  - Weak: Accept the same language, May produce different analyses
  - Strong: Accept same language, Produce same structure
- Canonical form:
  - Chomsky Normal Form (CNF)
    - All CFGs have a weakly equivalent CNF
    - All productions of the form:
      - A-> B C where B,C in N, or
      - A->a where a in Σ

Parsing Goals

- Accepting:
  - Legal string in language?
    - Formally: rigid
    - Practically: degrees of acceptability
- Analysis
  - What structure produced the string?
    - Produce one (or all) parse trees for the string

Parsing Search Strategies

- Top-down constraints:
  - All analyses must start with start symbol: S
  - Successively expand non-terminals with RHS
  - Must match surface string
- Bottom-up constraints:
  - Analyses start from surface string
  - Identify POS
  - Match substring of ply with RHS to LHS
  - Must ultimately reach S

Integrating Strategies

- Left-corner parsing:
  - Top-down parsing with bottom-up constraints
  - Begin at start symbol
  - Apply depth-first search strategy
    - Expand leftmost non-terminal
    - Parser can not consider rule if current input can not be first word on left edge of some derivation
    - Tabulate all left-corners for a non-terminal

Issues

- Left recursion
  - If the first non-terminal of RHS is recursive ->
    - Infinite path to terminal node
    - Could rewrite
- Ambiguity: pervasive (costly)
  - Lexical (POS) & structural
    - Attachment, coordination, np bracketing
- Repeated subtree parsing
  - Duplicate subtrees with other failures

Earley Parsing

- Avoid repeated work/recursion problem
  - Dynamic programming
    - Store partial parses in “chart”
      - Compactly encodes ambiguity
    - O(N^3)
- Chart entries:
  - Subtree for a single grammar rule
  - Progress in completing subtree
  - Position of subtree wrt input

Earley Algorithm

- Uses dynamic programming to do parallel top-down search in (worst case) $O(N^3)$ time
- First, left-to-right pass fills out a chart with N+1 states
  - Think of chart entries as sitting between words in the input string keeping track of states of the parse at these positions
  - For each word position, chart contains set of states representing all partial parse trees generated to date. E.g. chart[0] contains all partial parse trees generated at the beginning of the sentence

Chart Entries

Represent three types of constituents:

- predicted constituents
- in-progress constituents
- completed constituents

Progress in parse represented by Dotted Rules

- Position of • indicates type of constituent
- $_0$ Book $_1$ that $_2$ flight $_3$
  - S → • VP, [0,0] (predicted)
  - NP → Det • Nom, [1,2] (in progress)
  - VP →V NP •, [0,3] (completed)
- [x,y] tells us what portion of the input is spanned so far by this rule
- **Each State s_i:**
  <dotted rule>, [<back pointer>,<current position>]

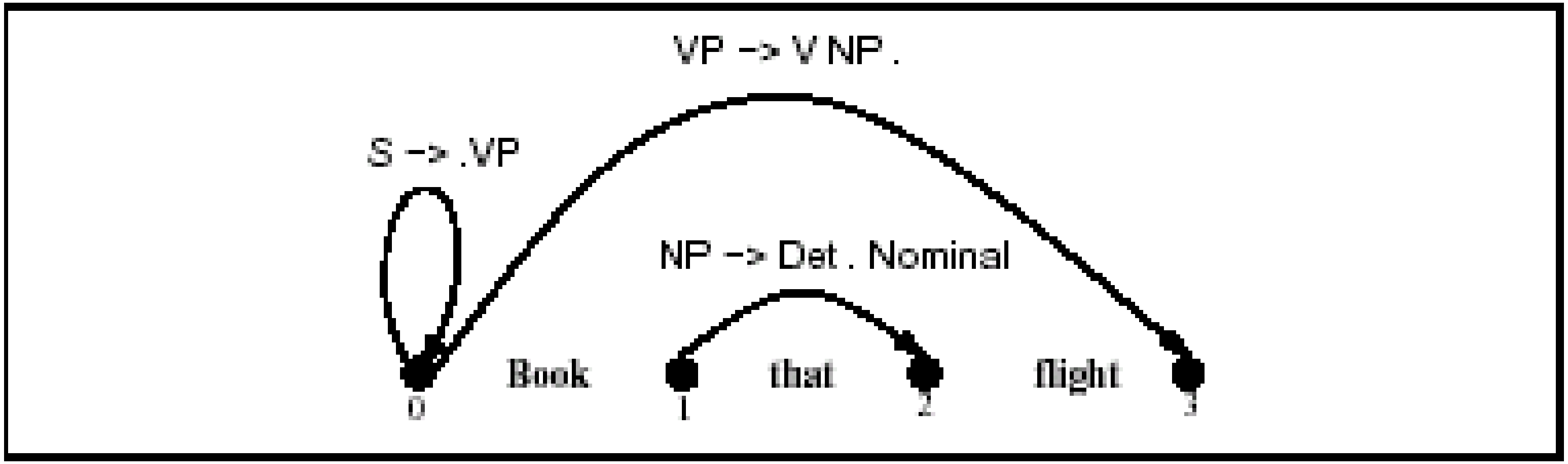
$_0$ Book $_1$ that $_2$ flight $_3$

- S → • VP, [0,0]
  - First 0 means S constituent begins at the start of input
  - Second 0 means the dot here too
  - So, this is a top-down prediction
- NP → Det • Nom, [1,2]
  - the NP begins at position 1
  - the dot is at position 2
  - so, Det has been successfully parsed
  - Nom predicted next

$_0$ Book $_1$ that $_2$ flight $_3$ (continued)

VP → V NP •, [0,3]

– Successful VP parse of entire input



Successful Parse

- Final answer found by looking at last entry in chart
- If entry resembles $S \rightarrow \alpha \bullet [nil,N]$ then input parsed successfully
- Chart will also contain record of all possible parses of input string, given the grammar

Parsing Procedure for the Earley Algorithm

- Move through each set of states in order, applying one of three operators to each state:
  - **predictor:** add predictions to the chart
  - **scanner:** read input and add corresponding state to chart
  - **completer:** move dot to right when new constituent found
- Results (new states) added to current or next set of states in chart
- No backtracking and no states removed: keep complete history of parse

States and State Sets

- **Dotted Rule s_i** represented as <dotted rule>, [<back pointer>, <current position>]

- **State Set S_j** to be a collection of states s_i with the same <current position>.

Earley Algorithm from Book

```
function EARLEY-PARSE(words, grammar) returns chart

  ENQUEUE((γ → • S, [0, 0]), chart[0])
  for i ← from 0 to LENGTH(words) do
   for each state in chart[i] do
     if INCOMPLETE?(state) and
               NEXT-CAT(state) is not a part of speech then
        PREDICTOR(state)
     elseif INCOMPLETE?(state) and
               NEXT-CAT(state) is a part of speech then
         SCANNER(state)
     else
        COMPLETER(state)
   end
  end
  return(chart)

 procedure PREDICTOR((A → α • B β, [i, j]))
    for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
         ENQUEUE((B → • γ, [j, j]), chart[j])
    end

 procedure SCANNER((A → α • B β, [i, j]))
    if B ⊂ PARTS-OF-SPEECH(word[j]) then
        ENQUEUE((B → word[j], [j, j + 1]), chart[j+1])

 procedure COMPLETER((B → γ •, [j, k]))
    for each (A → α • B β, [i, j]) in chart[j] do
         ENQUEUE((A → α B • β, [i, k]), chart[k])
    end

 procedure ENQUEUE(state, chart-entry)
    if state is not already in chart-entry then
        PUSH(state, chart-entry)
    end
```

Earley Algorithm (simpler!)

1. Add Start → · S, [0,0] to state set 0
   Let i=1

2. **Predict** all states you can, adding new predictions to state set 0

3. **Scan** input word i—add all matched states to state set S_i.
   Add all new states produced by **Complete** to state set S_i
   Add all new states produced by **Predict** to state set S_i
   Let i = i + 1
   Unless i=*n*, repeat step 3.

4. At the end, see if state set *n* contains Start → S ·, [nil,n]

3 Main Sub-Routines of Earley Algorithm

- **Predictor**: Adds predictions into the chart.
- **Completer**: Moves the dot to the right when new constituents are found.
- **Scanner**: Reads the input words and enters states representing those words into the chart.

Predictor

- Intuition: create new state for top-down prediction of new phrase.
- Applied when non part-of-speech non-terminals are to the right of a dot: **S → • VP [0,0]**
- Adds new states to ***current*** chart
  - One new state for each expansion of the non-terminal in the grammar
    VP → • V [0,0]
    VP → • V NP [0,0]
- Formally:
  S_j: A → α · B β, [i,j]
  S_j: B → · γ, [j,j]

Scanner

- Intuition: Create new states for rules matching part of speech of next word.
- Applicable when part of speech is to the right of a dot: VP → • V NP [0,0] ‘Book…’
- Looks at current word in input
- If match, adds state(s) to ***next*** chart
  VP → V • NP [0,1]
- Formally:
  S_j: $A \rightarrow \alpha \cdot B\ \beta$, [i,j]
  S_{j+1}: $A \rightarrow \alpha\ B \cdot \beta$, [i,j+1]

Completer

- Intuition: parser has finished a new phrase, so must find and advance states all that were waiting for this
- Applied when dot has reached right end of rule

  NP → Det Nom • [1,3]
- Find all states w/dot at 1 and expecting an NP: VP → V • NP [0,1]
- Adds new (completed) state(s) to ***current*** chart : VP → V NP • [0,3]
- Formally: S_k: $B \rightarrow \delta \cdot$, [j,k]

  S_k: $A \rightarrow \alpha\ B \cdot \beta$, [i,k],

  where: S_j: $A \rightarrow \alpha \cdot B\ \beta$, [i,j].

Example: State Set S_0 for Parsing "Book that flight" using Grammar G_0

$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
$NP \rightarrow \bullet Det\ NOMINAL$	[0,0]	Predictor
$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
$S \rightarrow \bullet VP$	[0,0]	Predictor
$VP \rightarrow \bullet Verb$	[0,0]	Predictor
$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor

Example: State Set S_1 for Parsing "Book that flight"

~~$Verb \rightarrow book \bullet$~~	~~[0,1]~~	~~Scanner~~	
$VP \rightarrow Verb \bullet$	~~[0,1]~~	~~Completer~~	Scanner
$S \rightarrow VP \bullet$	~~[0,1]~~	~~Completer~~	Scanner
$VP \rightarrow Verb \bullet NP$	[0,1]	Completer	
$NP \rightarrow \bullet Det\ NOMINAL$	[1,1]	Predictor	
$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor	

VP→ • V and VP → • V NP are both passed to **Scanner**, which adds them to Chart[1], moving dots to right

Prediction of Next Rule

- When VP → V • is itself processed by the Completer, S → VP • is added to Chart[1] since VP is a left corner of S
- Last 2 rules in Chart[1] are added by **Predictor** when VP → V • NP is processed
- And so on….

Last Two States

Chart[2]

~~*Det* → *that*•~~	~~[1,2]~~	~~Scanner~~	
NP → *Det*•*NOMINAL*	[1,2]	~~Completer~~	Scanner
NOMINAL → • *Noun*	[2,2]	Predictor	
NOMINAL → • *Noun NOMINAL*	[2,2]	Predictor	

Chart[3]

~~*Noun* → *flight*•~~	~~[2,3]~~	~~Scanner~~	
NOMINAL → *Noun*•	[2,3]	~~Completer~~	Scanner
NOMINAL → *Noun*• *NOMINAL*	[2,3]	~~Completer~~	Scanner
NP → *Det NOMINAL* •	[1,3]	Completer	
VP → *Verb NP* •	[0,3]	Completer	
S → *VP*•	[0,3]	Completer	
NOMINAL → • *Noun*	[3,3]	Predictor	
NOMINAL → • *Noun NOMINAL*	[3,3]	Predictor	

How do we retrieve the parses at the end?

- Augment the Completer to add pointers to prior states it advances as a field in the current state
  - i.e. what state did we advance here?
  - Read the pointers back from the final state