Intro to Natural Language Processing

A dream of AI,

Always a little out of reach.

If a computer could understand language...

- voice control of all machinery
  - UBER, Drive me to the nearest pub
- no more programming (jobs) - just tell computers what to do
  - OK Google, factor 1652376
- Machines could translate between languages allowing world peace!
  - Has google Translate lead to peace?
- Machines could sort through the news for the kind you like, and summarize stories.
  - Facebook tried and we got "fake news"

Computers need NLP to pass "the Turing Test"

well, twitterbots...

ELIZA (Weizenbaum CACM 1966)

- Used pattern recognition and pattern substitution in model for computer natural language processing
- Very easy to write!
- Fooled a lot of people!
- Does not really understand anything!

How Eliza Works

- Large database of "patterns" and possible response actions
  - "I feel bad about $x" --> "why do you feel bad about $X"
  - "yes" --> "Please go on..."
  - "$ mother $" --> Tell me more about your family
- Set of state variables to drive initiation:
  - FAMILYFLAG, DEPRESSEDFLAG, WORKFLAG, SEXFLAG
    - these can be turned on and off by different actions, and gate others

Example of Eliza Dialog

- I am the psychotherapist. Please, describe your problems.
  - I have been having trouble sleeping
- Why do you say you have been having trouble sleeping?
  - because I wake at 3:30am almost every day
- Is it because you wake at 3 30am almost every day that you came to me?
  - yes

What's wrong with pattern recognition for NLP?

- You can get 90% of almost any NLP problem using simple linear techniques
  - The remaining 10% seem to require exponential increases in AI
- Its not really "understanding"
- Doesn't Scale up
  - Can we just add millions of Rules?
  - What happens when they start to collide?

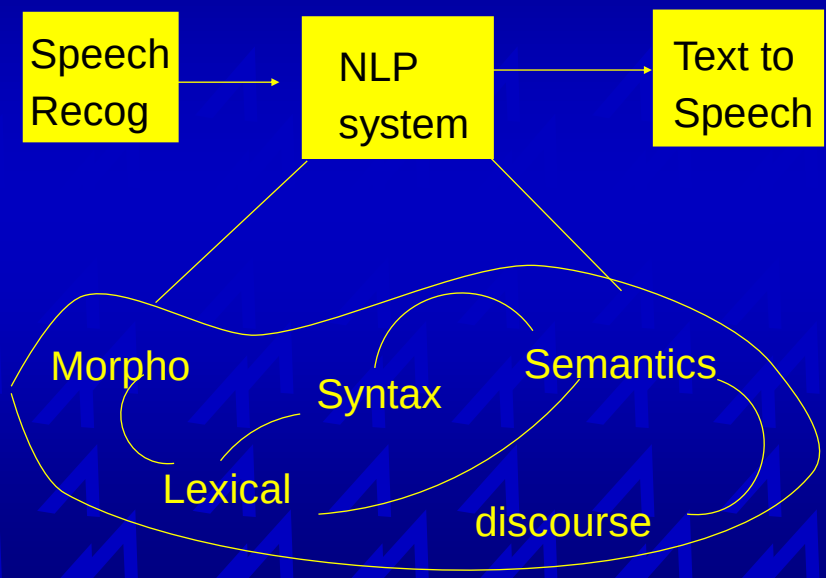
Machine Translation

- Big Defense Project in the 60's
- Use English/Russian Dictionary
- Use syntactic Rules to order words
- Small Scale Successes but global Failure. Why? Bar Hillel's Bailout:
  - The box is in the pen
  - The pen is in the box.

Natural Language DB Interface

- Allow executives access to databases
- Language is very restricted
  - to domain of database
  - to certain kinds of complex queries
- Language is
  - converted into formal query
  - Query is sent to database
  - Report is presented back to user
- But learning SQL is better!

Generic Block Diagram



Knowledge and Processing Components for NLP

- Morphological Analysis
  - affix and punctuation segmentation
- Lexical Selection
  - category type and meaning of word
- Syntactic Analysis
  - linear sequence -> grouping structure
- Semantic Analysis
  - tree structures -> Knowledge Rep (e.g logic or semantic net)

Knowledge and Processing Components

- Discourse Integration
  - context across multiple sentences
- Pragmatics
  - Resolving speech acts

Syntactic Parsing is at heart of many NLP systems

- Historic Issues
  - Algorithmic Efficiency
  - Psychological Plausibility
  - Top down/bottom up
  - Integration with other forms of knowledge?
  - All parses or Best First

Parsing is Interdisciplinary Work

- Linguistics:
  - interested in the principles and theories which govern natural language
- Theoretical Computer Science:
  - foundation of work was on language hierarchy, and complexity of recognition algorithms.
- Computational Linguistics
  - Born from puzzles of parsing
- AI/NLP
  - Concerned with effective use of knowledge in NLP systems

Grammar

- The term means the overall knowledge of a language, and includes all levels of knowledge
- Universal Grammar
  - refers to those components and constraints which operate in all human languages
- Phrase Structure Grammar
  - The main element of syntactic level, accounts for hierarchical/sequential structure

Autonomy of Syntax

- Judgements of grammaticality can be separated from judgements of meaningfulness
  - the what girl school little red
  - someone accidentally swallowed a truck
  - colorless green ideas sleep furiously
  - twas brillig and the slithy toves did gyre and gimble in the wabe.

Generative Systems

- Take as a given that
  - the sentences of a language form an infinite set
  - the knowledge structures of cognition are finite
- Therefore
  - knowledge of language must be a finite system which can GENERATE an infinite system of meaningful utterances

Three Models for Language Description

- Chomsky (1957) began the modern revolution in linguistics and cognitive science with an analysis of the requirements on grammar in order to situate it as an object of scientific study.
- He showed three kinds of Infinitely Generative systems
  - Finite State Machines
  - Phrase Structure grammar
  - Transformational Grammar
- And proved that the first two were inadequate

Background:Formal Theories of Grammar

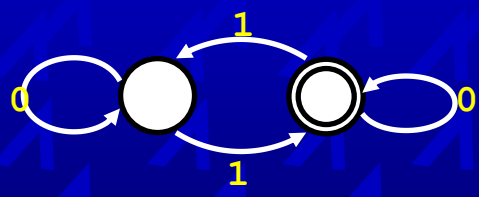
- Σ is a finite set of tokens (terminals/words/letters)
- A language is a subset of Σ*
  - the infinite set of all finite strings of tokens

The Chomsky Hierarchy

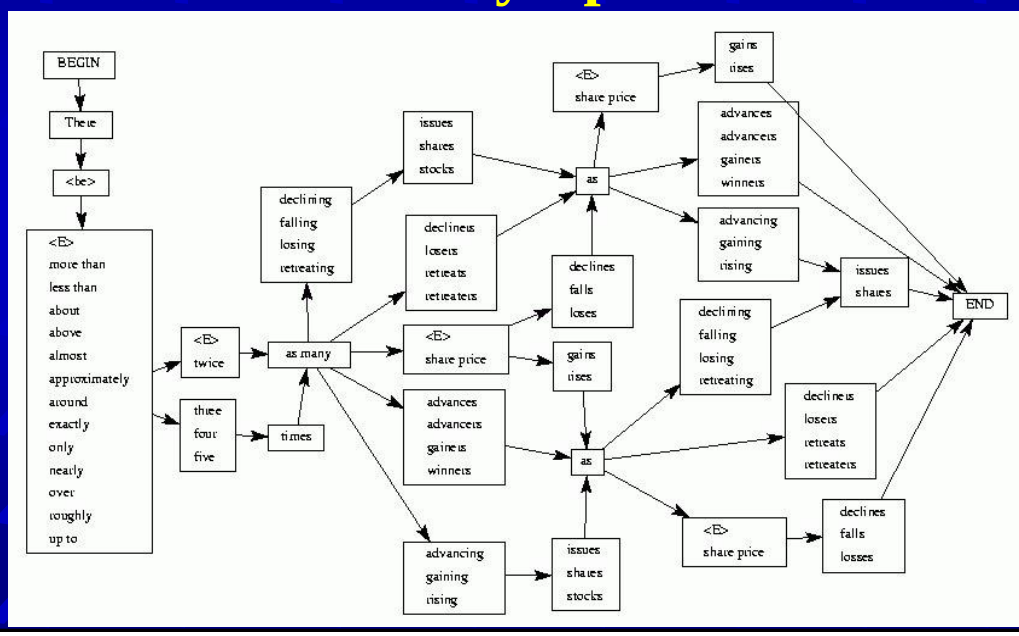
- Languages form a hierarchy of types, and the types correspond to computation models:
  - regular Language /Finite State Machine
    - aaaaaab ababababab
  - Context-Free/Push Down Automata
  - ab, aabb, aaabbb,aaaabbbb
  - Context-Sensitive/Linear Bounded Automata
    - He shot herself* (coordination across)
  - Recursively Enumerable languages/Turing Machines
    - strings which are "prime number" in length

Finite State Machines

- nodes Q are states
  - Start State Q0
  - Final States F subset of Q
- links signify tokens from set Σ
- language is set is all strings which end in final states



FSM's get big and sloppy Chomsky's point?



Why not Finite State machines?

- English has embedded constructions where arbitrary amounts of stuff can be inserted between connected parts:
  - the rat [the cat ate] died
  - the rat [the cat the dog chased ate] died
  - john called the guy up
  - john called the guy who smashed his car's window with a crowbar purchased at sears up
- the FSM graph would have to be infinite to account for a simple embedded construction.

PHRASE STRUCTURE GRAMMAR

- Σ is a finite set of tokens (terminals/words/letters)
- A language is a subset of Σ^*
  - the infinite set of all finite strings of tokens
- a set of non-terminals N
- A set of Rules of the form N -> $\{N \times \Sigma\}^*$
- A starting symbol, $s \subset N$
  - A string (of terminals) is in the language of the grammar if there exists a derivation from s by the rules to the string.

Sample CF Grammar

- RULES

```
S -> NP VP
NP -> nom | det NP1 | NP PP
NP1 -> Adj NP1 | n
PP -> prep NP
VP -> v NP | VP PP
```

- Lexicon (as Rules)

```
det -> a the          n -> bat hair ball
prep -> with          v -> hit
nom -> john mary bill
adj -> little big long
```

How to represent a Grammar?

```
(defvar *gram* '((s (np vp))
                 (np (nom)(det NP1)(np pp))
                (np1 (adj np1)(n))
                 (pp (prep np))
                 (vp (v np) (v))
                 (det a the some)
                 (adj little big)
                 (nom john mary bill)
                 (n bat hair ball)
                 (prep with up in)
                 (v ate hit saw loved)))
```

Applying rules generates all legal strings

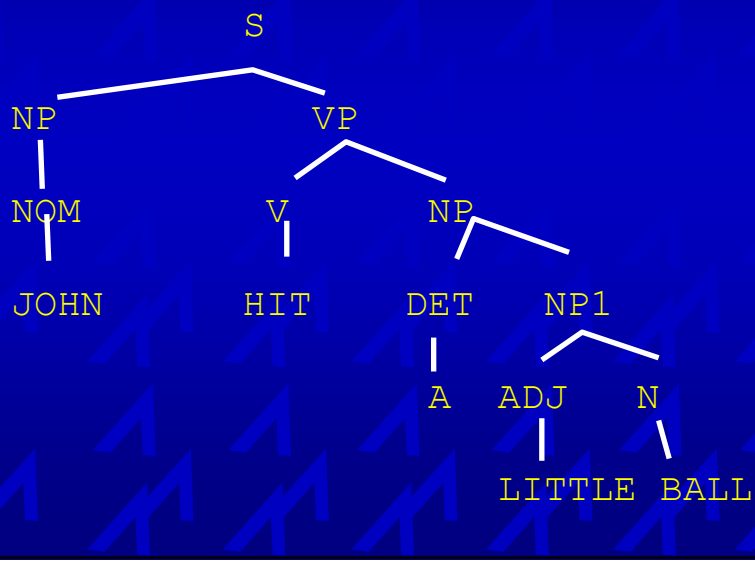
```
S -> NP VP
  -> NOM VP
  -> JOHN VP
  -> JOHN V NP
  -> JOHN HIT NP
  -> JOHN HIT DET NP1
  -> JOHN HIT A NP1
  -> JOHN HIT A ADJ NP1
  -> JOHN HIT A LITTLE NP1
  -> JOHN HIT A LITTLE N
  -> JOHN HIT A LITTLE BALL
```

Generation is simple

```
(defun rewrites (rule)
  (cdr (assoc rule *gram*)))

(defun pick (l) (nth (random (length l)) l))

(defun mappend (fn list)
  (apply #'append (mapcar fn list)))

(defun generate (phrase)
  (cond ((listp phrase)
         (mappend #'generate phrase))
        ((rewrites phrase)
         (generate (pick (rewrites phrase))))
        (t (list phrase))))
```

Parse tree of derivation

(S (NP (NOM JOHN))(VP (V HIT)(NP (DET A)(NP1 (ADJ LITTLE)(N BALL))))



Ambiguity

- John hit the ball with a bat
- John liked the girl with long hair

- VP -> VP PP
- NP -> NP PP

- So both sentences have two parse trees...

What to do with grammars?

- GENERATE
  - {random, all} sequences in the langauge
- RECOGNIZE
  - determine if a sequence of symbols is in the language, e.g. return T or Nil.
- PARSE
  - determine {ONE, SOME, ALL} Parse-tree (phrase marker derivation) for a sequence of symbols.

Parsing Problem

- Given a sequence of tokens
- find a legal "tree" of rules which expand from 'S to the sequence
- Find one, find some, or find all?

Top Down Parser

- Search depth-first among alternative constructions for matches
- Bottoms out and backs up when lack of match or out of input

Principle of Top Down

- to determine if (john hit the little ball) is an 'S
- see if (john hit the little ball) is one of the expansions of s...
  - Is (John hit the little ball) (NP VP)
    - Is (john NP) and (Hit the little ball VP)
    - or is (John HIT) NP and (The little ball VP)
    - or ...
- It will eventually work, like a tile puzzle!
- Needs a lot of pruning.

Top Down Recognizer

```
TOPDOWN(SENT: LIST TOKEN:LIST)
  If SENT=TOKENS=NIL, RETURN T
  if TOKEN IS NIL or SENT IS NIL,
    OR |SENT| < |TOKEN| RETURN NIL

IF FIRST(TOKEN) IS LEXCAT THEN
  IF FIRST(SENT)MEMB TOKEN THEN
     TOPDOWN REST(SENT),REST(TOKEN)
  ELSE RETURN NIL
ELSE FOR EACH RHS OF FIRST(TOKEN)
  UNTIL ONE RETURNS T
  TOPDOWN (SENT, RHS+REST(TOKEN))
```

Norvig Top Down Parser

```
(defun parse (tokens start-symbol)
  (if (eq (first tokens) start-symbol)
    (list (make-parse :tree (first tokens) :rem (rest tokens)))
    (mapcan #' (lambda (rule)
                        (extend-parse (lhs rule) nil tokens (rhs rule)))
                (rules-for start-symbol) ) ) )

(defun extend-parse (lhs rhs rem needed)
  (if (null needed)
     (list (make-parse :tree (cons lhs rhs) :rem rem))
    (mapcan
     #'(lambda (p)
            (extend-parse lhs (append rhs (list (parse-tree p)))
                          (parse-rem p) (rest needed)))
     (parse rem (first needed)))))
```

support functions

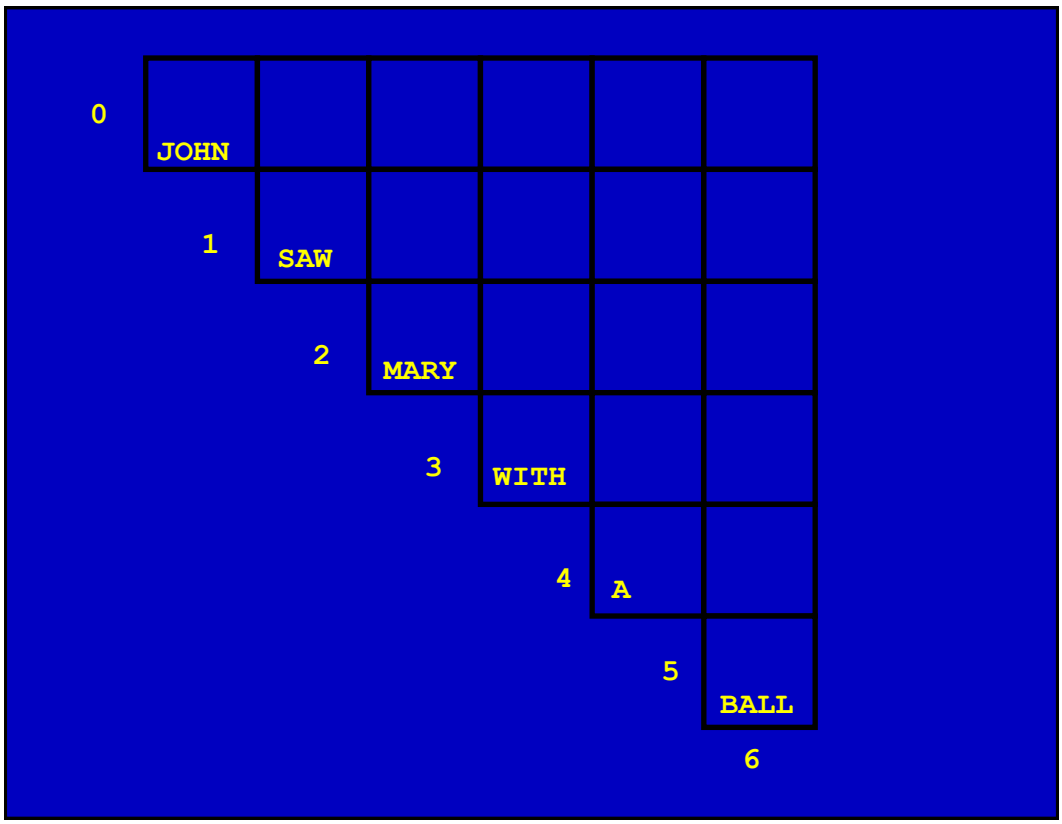
```
(defstruct (parse) "A parse tree and a remainder." tree rem)

;; Trees (and rules) are of the form: (lhs . rhs)
(defun lhs (tree) (first tree))

(defun rhs (tree) (rest tree))

(defun rules-for (symbol)
  "Return a list of the rules with symbol on the left hand side."
  (remove symbol *grammar* :key #'lhs :test-not #'eql)).
```

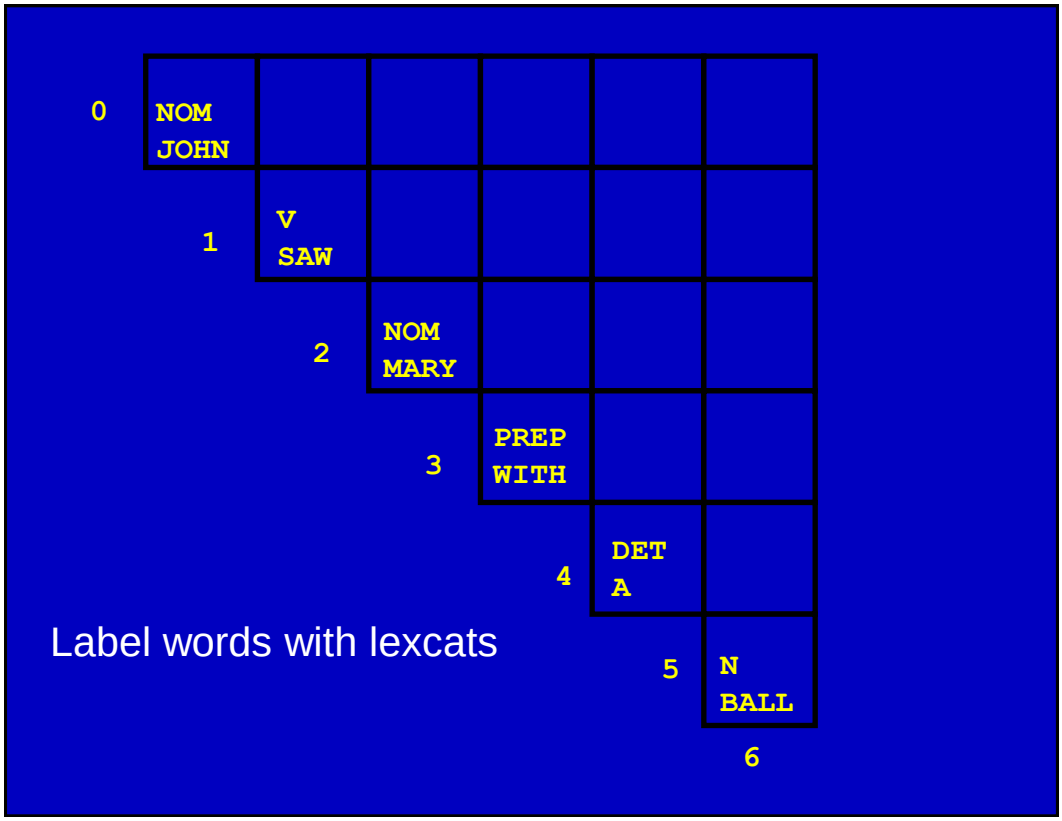
Grammar for Top Down Parser

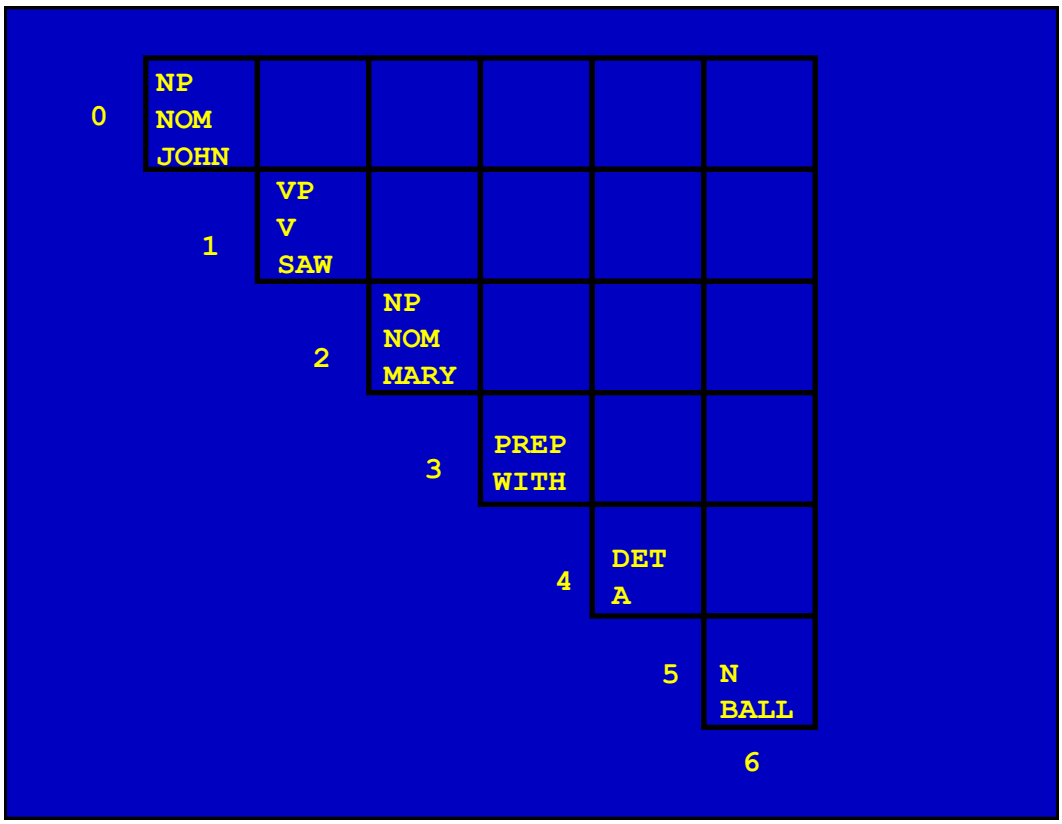
```
• (defparameter *grammar*
•   '((s np vp)
•     (np nom)  (np det NP1) (np np pp)
•     (np1 adj np1) (np1 n)
•     (pp prep np)
•     (vp v np) (vp v)
•     (det a)(det the)(det some)
•     (adj little)(adj long)(adj big)
•     (nom john)(nom  mary)(nom  bill)
•     (n girl)(n bat)(n hair)(n ball)
•     (prep with) (prep up) (prep in)
•     (v ate)(v hit)(v saw)(v loved)))
```

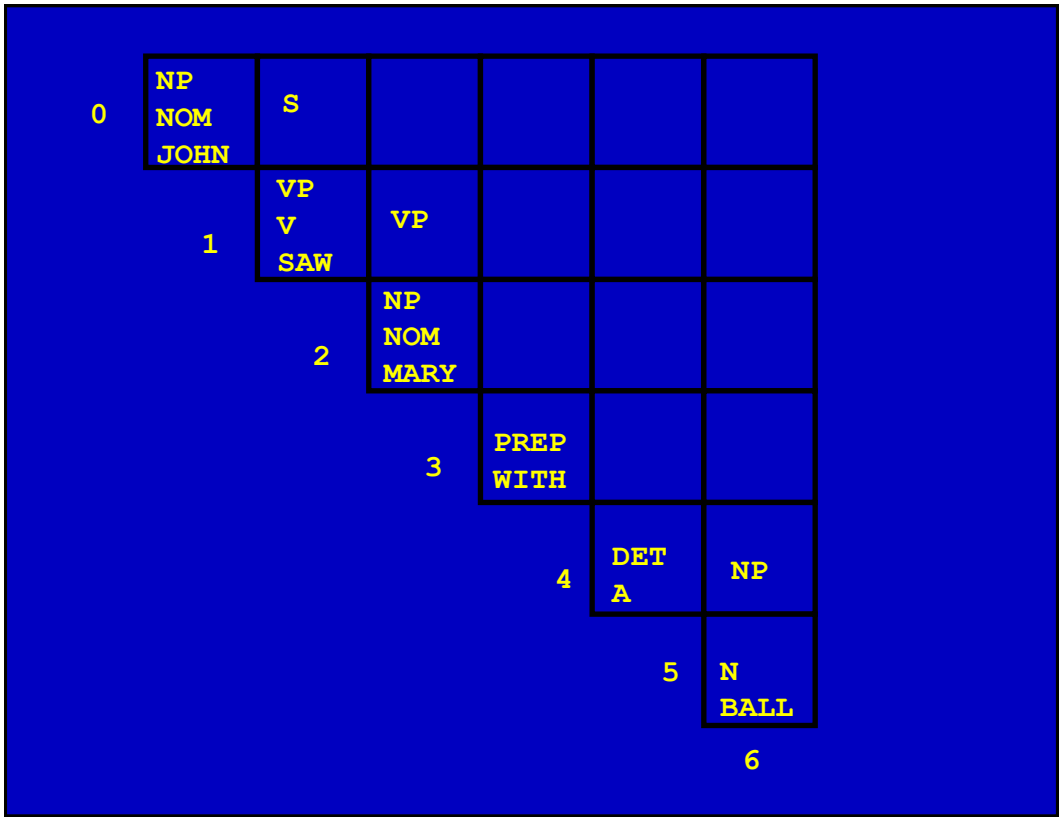
Chart Parser O (n^3) (all ways bottom up)

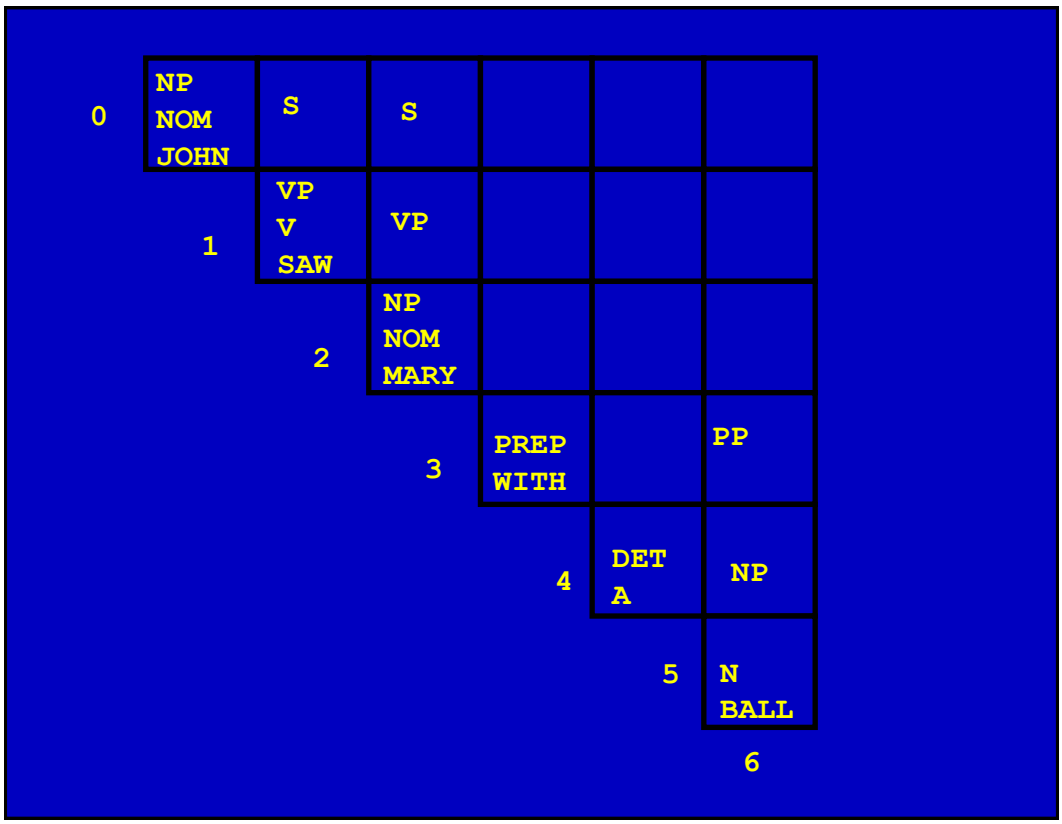
- First label "spaces" from 0 to n
- Chart is an upper triangular +diagonal array
  - Chart(i,j) contains partial parses for positions i thru j
- For J = 1 to N
  - chart(j-1,j) = lexical categories of word(j)
- For J = 2 to N do
  - for I = j-2 down to 0 do
    - chart(i,j) = Union RULES(Chart (i,k) + Chart (k.j)
- See if "S" is in Chart (0,n)

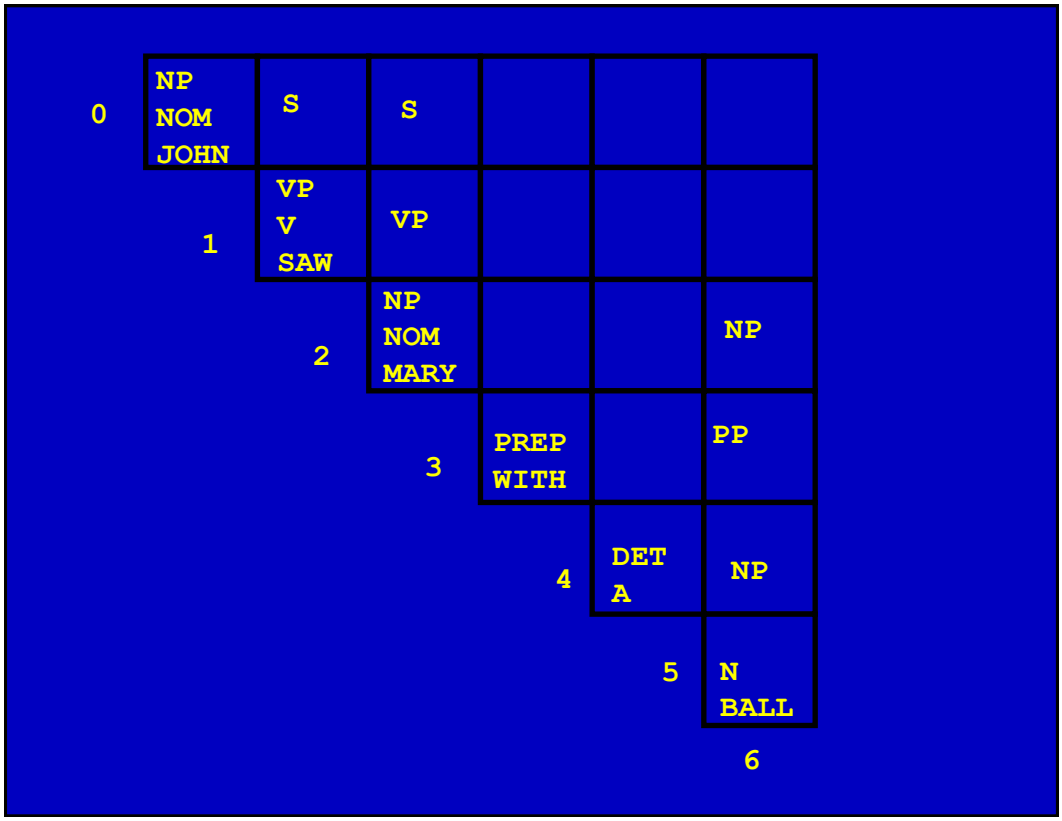


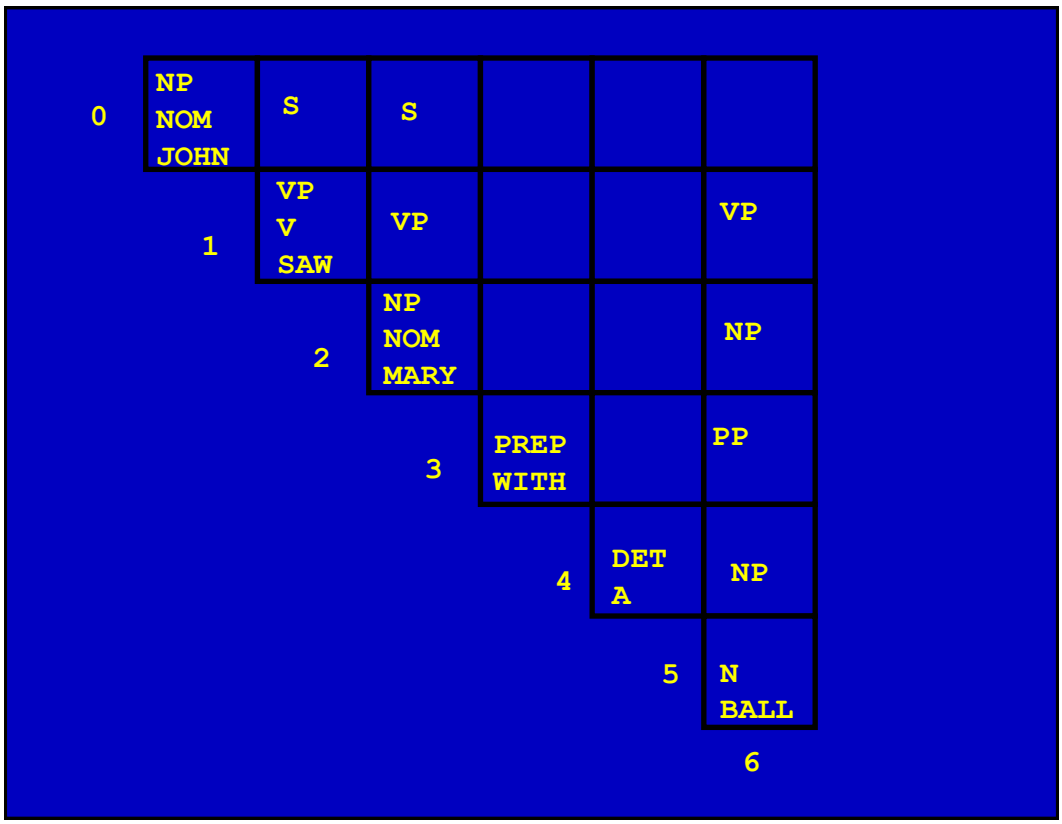

0
NOM
JOHN
1
V
SAW
2
NOM
MARY
3
PREP
WITH
4
DET
A
5
N
BALL
6
Label words with lexcats

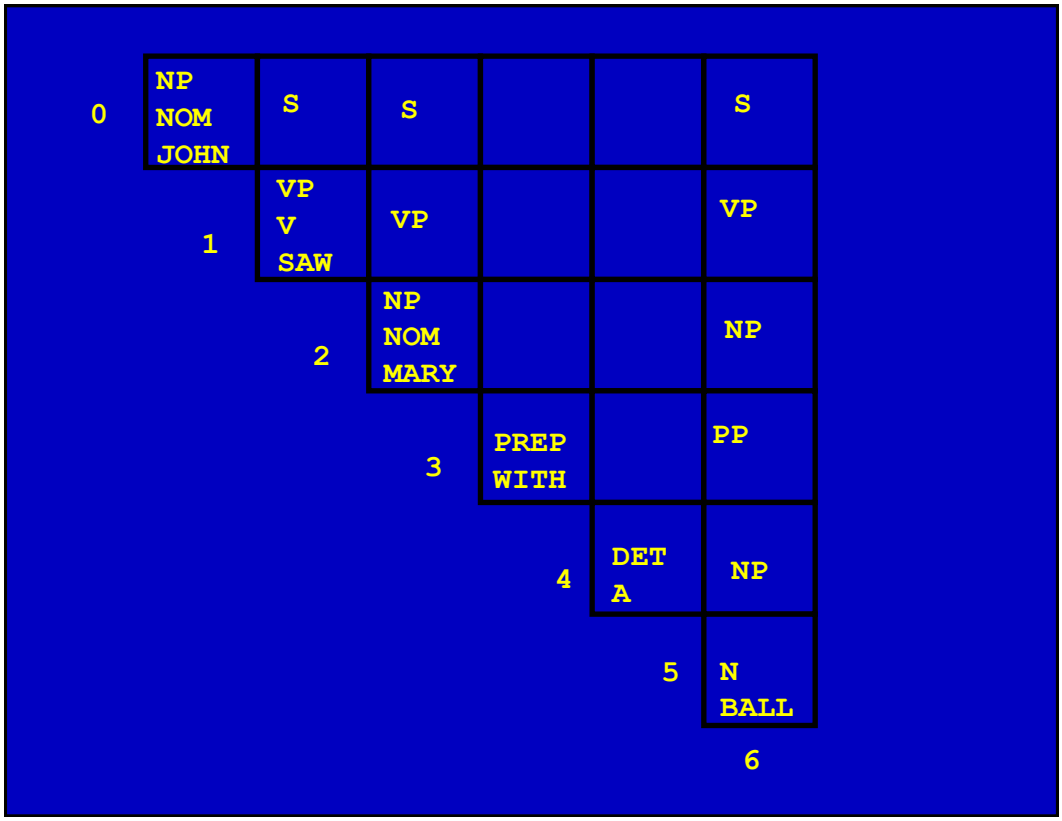

0
NP
NOM
JOHN
1
VP
V
SAW
2
NP
NOM
MARY
3
PREP
WITH
4
DET
A
5
N
BALL
6


0
NP
NOM
JOHN
S
1
VP
V
SAW
VP
2
NP
NOM
MARY
3
PREP
WITH
4
DET
A
NP
5
N
BALL
6


0
NP
NOM
JOHN
S
S
1
VP
V
SAW
VP
2
NP
NOM
MARY
3
PREP
WITH
PP
4
DET
A
NP
5
N
BALL
6


0
NP
NOM
JOHN
S
S
1
VP
V
SAW
VP
2
NP
NOM
MARY
NP
3
PREP
WITH
PP
4
DET
A
NP
5
N
BALL
6


0
NP
NOM
JOHN
S
S
1
VP
V
SAW
VP
VP
2
NP
NOM
MARY
NP
3
PREP
WITH
PP
4
DET
A
NP
5
N
BALL
6



Efficiency Issues

- Bottom Up Chart Parsing is O(n^3) for the length of the sentence, and is effective in practice.
- Sometimes generates too many trees to choose between (based on the ambiguity in the grammar itself)
- Norvig showed that applying memoization to top-down parser made it equal to chart parser!

Phrase Structure is not enough

- It cannot coordinate generated structures
- Conjunction
  - John likes turkey and bill likes chicken
  - john likes turkey and bill chicken
- Crossed Serial Dependencies
  - john and bill like turkey and chicken, respectively
- Duplication Phenomena
  - corporate interest rates? Schmorporate interests rates!

Old Code never dies: Elizas are still with us today.

- Doctor in Emacs
- Chatbots in IRC and Twitter
- Annual Turing Award Competitions
  - The Artificial Autistic Child...
- Talking head Avatars, e.g. Fake Kirk
- However, original use proposed by Ken Colby -- as first-line psychotherapist -- has never happened.