Syntactic Parsing

Le Anh Cuong

Reading

- Chapter 5 [1]
- Chapter 10 [2]

Contents

- Syntax?
  - Context Free Grammar
- Parsing Algorithms
  - Topdown
  - Bottom-up
  - CYK
  - Earley - Chart parsing
- Probabilistic CFG & Statistical Parsing

Syntax

- By syntax (or grammar) I mean the kind of implicit knowledge of your native language that you had mastered by the time you were 2 or 3 years old without explicit instruction
- Not the kind of stuff you were later taught in school.

Syntax

- Why should you care?
  - Grammar checkers
  - Question answering
  - Information extraction
  - Machine translation

Context-Free Grammars

- Capture constituency and ordering
  - Ordering is easy

    What are the rules that govern the ordering of words and bigger units in the language
  - What's constituency?

    How words group into units and how the various kinds of units behave with regard to (wrt) one another

CFG Examples

- S -> NP VP
- NP -> Det NOMINAL
- NOMINAL -> Noun
- VP -> Verb
- Det -> *a*
- Noun -> *flight*
- Verb -> *left*

CFGs

- S -> NP VP
  - This says that there are units called S, NP, and VP in this language
  - That an S consists of an NP followed immediately by a VP
  - Doesn't say that that's the only kind of S
  - Nor does it say that this is the only place that NPs and VPs occur

A Context-Free Grammar for English

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}

S = S

Σ = {sleeps, saw, man, woman, telescope, the, with, in}

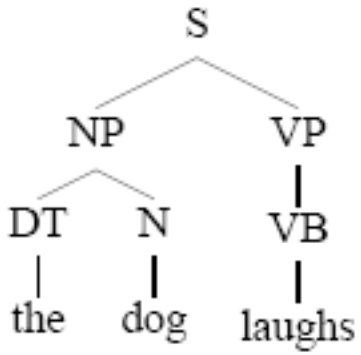
R =

S	⇒	NP	VP
VP	⇒	Vi	
VP	⇒	Vt	NP
VP	⇒	VP	PP
NP	⇒	DT	NN
NP	⇒	NP	PP
PP	⇒	IN	NP

Vi	⇒	sleeps
Vt	⇒	saw
NN	⇒	man
NN	⇒	woman
NN	⇒	telescope
DT	⇒	the
IN	⇒	with
IN	⇒	in

Note: S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase, DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition

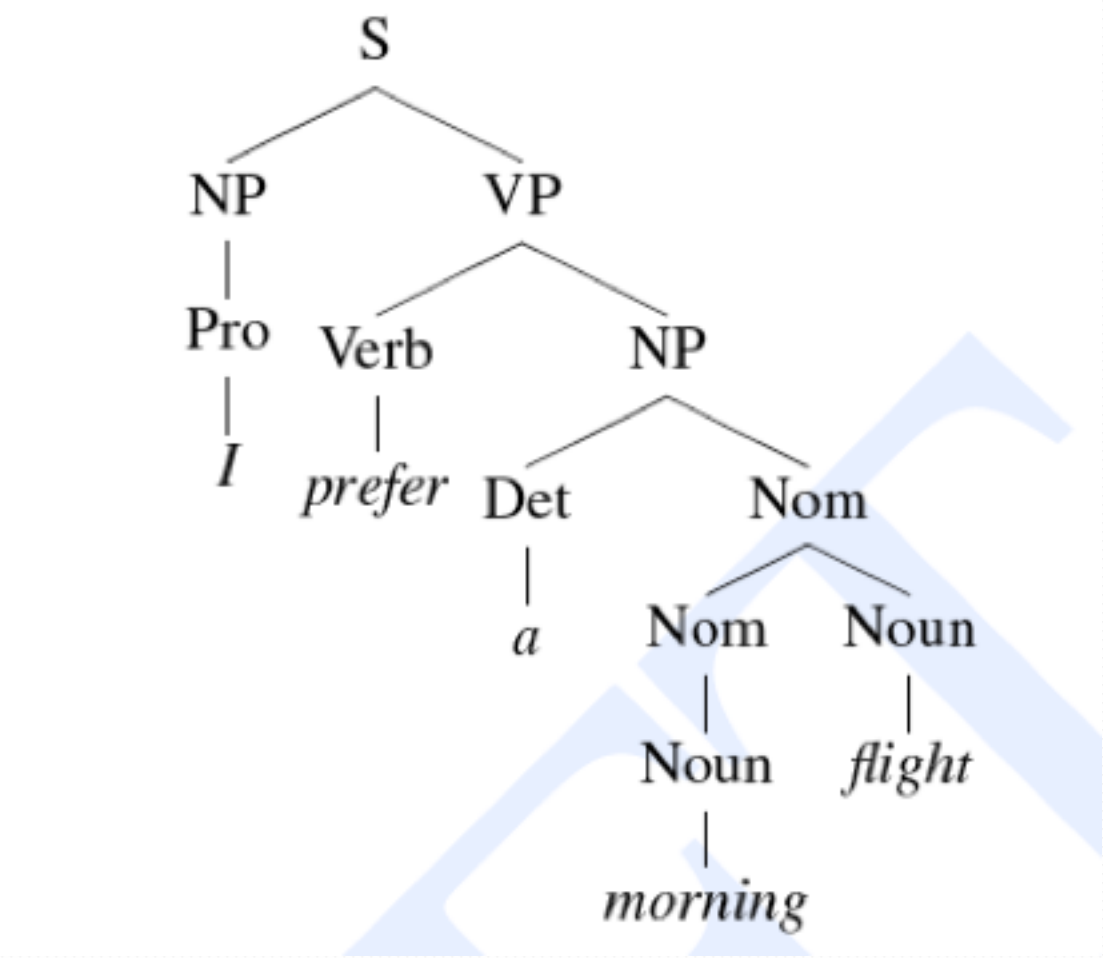
DERIVATION	RULES USED
S	S → NP VP
NP VP	NP → DT N
DT N VP	DT → the
the N VP	N → dog
the dog VP	VP → VB
the dog VB	VB → laughs
the dog laughs	



Derivations

- A derivation is a sequence of rules applied to a string that accounts for that string
  - Covers all the elements in the string
  - Covers only the elements in the string

Derivations as Trees



Parsing

- Parsing is the process of taking a string and a grammar and returning a (many?) parse tree(s) for that string
- It is completely analogous to running a finite-state transducer with a tape
  - It's just more powerful
    - Remember this means that there are languages we can capture with CFGs that we can't capture with finite-state methods

Context?

- The notion of context in CFGs has nothing to do with the ordinary meaning of the word context in language.
- All it really means is that the non-terminal on the left-hand side of a rule is out there all by itself (free of context)

  A -> B C

  Means that

  - I can rewrite an A as a B followed by a C regardless of the context in which A is found
  - Or when I see a B followed by a C I can infer an A regardless of the surrounding context

Key Constituents (English)

- Sentences
- Noun phrases
- Verb phrases
- Prepositional phrases

Sentence-Types

- Declaratives: A plane left

  S -> NP VP

- Imperatives: Leave!

  S -> VP

- Yes-No Questions: Did the plane leave?

  S -> Aux NP VP

- WH Questions: When did the plane leave?

  S -> WH Aux NP VP

Recursion

- We'll have to deal with rules such as the following where the non-terminal on the left also appears somewhere on the right (directly).

  Nominal -> Nominal PP [[flight] [to Boston]]

  VP -> VP PP [[departed Miami] [at noon]]

Recursion

- Of course, this is what makes syntax interesting

  flights from Denver

  Flights from Denver to Miami

  Flights from Denver to Miami in February

  Flights from Denver to Miami in February on a Friday

  Flights from Denver to Miami in February on a Friday under $300

  Flights from Denver to Miami in February on a Friday under $300 with lunch

Recursion

- Of course, this is what makes syntax interesting

  [[flights] [from Denver]]

  [[[Flights] [from Denver]] [to Miami]]

  [[[[Flights] [from Denver]] [to Miami]] [in February]]

  [[[[[Flights] [from Denver]] [to Miami]] [in February]] [on a Friday]]

  Etc.

The Point

- If you have a rule like
  - VP -> V NP
  - It only cares that the thing after the verb is an NP. It doesn't have to know about the internal affairs of that NP

The Point

Conjunctive Constructions

- S -> S and S
  - John went to NY and Mary followed him
- NP -> NP and NP
- VP -> VP and VP
- …
- In fact the right rule for English is

  X -> X and X

Problems

- Agreement
- Subcategorization
- Movement (for want of a better term)

Agreement

- *This dogs
- *Those dog
- This dog
- Those dogs

- *This dog eat
- *Those dogs eats
- This dog eats
- Those dogs eat

Agreement

- In English,
  - subjects and verbs have to agree in person and number
  - Determiners and nouns have to agree in number
- Many languages have agreement systems that are far more complex than this.

Subcategorization

- Sneeze: John sneezed
- Find: Please find [a flight to NY]$_{NP}$
- Give: Give [me]$_{NP}$[a cheaper fare]$_{NP}$
- Help: Can you help [me]$_{NP}$[with a flight]$_{PP}$
- Prefer: I prefer [to leave earlier]$_{TO\text{-}VP}$
- Told: I was told [United has a flight]$_{S}$
- …

Subcategorization

- *John sneezed the book
- *I prefer United has a flight
- *Give with a flight

- Subcat expresses the constraints that a predicate (verb for now) places on the number and syntactic types of arguments it wants to take (occur with).

So?

- So the various rules for VPs *overgenerate*.
  - They permit the presence of strings containing verbs and arguments that don't go together
  - For example
  - VP -> V NP

  therefore

  Sneezed the book is a VP since "sneeze" is a verb and "the book" is a valid NP

So What?

- Now *overgeneration* is a problem for a generative approach.
  - The grammar is supposed to account for all and only the strings in a language

Possible CFG Solution

- S -> NP VP
- NP -> Det Nominal
- VP -> V NP
- SgS -> SgNP SgVP
- PlS -> PlNp PlVP
- SgNP -> SgDet SgNom
- PlNP -> PlDet PlNom
- PlVP -> PlV NP
- SgVP ->SgV Np
- …

CFG Solution for Agreement

- It works and stays within the power of CFGs
- But its ugly
- And it doesn't scale all that well

Forward Pointer

- It turns out that verb subcategorization facts will provide a key element for semantic analysis (determining who did what to who in an event).

Movement

- Core (canonical) example
  - My travel agent booked the flight

Movement

- Core example
  - $[[\text{My travel agent}]_{NP}\ [\text{booked}\ [\text{the flight}]_{NP}]_{VP}]_{S}$

- I.e. “book” is a straightforward transitive verb. It expects a single NP arg within the VP as an argument, and a single NP arg as the subject.

Movement

- What about?
  - Which flight do you want me to have the travel agent book?
- The direct object argument to “book” isn’t appearing in the right place. It is in fact a long way from where its supposed to appear.
- And note that its separated from its verb by 2 other verbs.

The Point

- CFGs appear to be just about what we need to account for a lot of basic syntactic structure in English.
- But there are problems
  - That can be dealt with adequately, although not elegantly, by staying within the CFG framework.
- There are simpler, more elegant, solutions that take us out of the CFG framework (beyond its formal power)

Parsing

- Parsing with CFGs refers to the task of assigning correct trees to input strings
- Correct here means a tree that covers all and only the elements of the input and has an S at the top
- It doesn't actually mean that the system can select the correct tree from among all the possible trees

Parsing

- As with everything of interest, parsing involves a search which involves the making of choices
- We'll start with some basic (meaning bad) methods before moving on to the one or two that you need to know

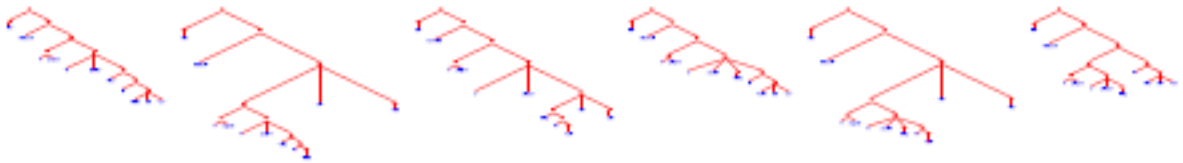
The Problem with Parsing: Ambiguity

INPUT:

She announced a program to promote safety in trucks and vans

⇓

POSSIBLE OUTPUTS:



And there are more...

A Probabilistic Context-Free Grammar (PCFG)

S	⇒	NP	VP	1.0
VP	⇒	Vi		0.4
VP	⇒	Vt	NP	0.4
VP	⇒	VP	PP	0.2
NP	⇒	DT	NN	0.3
NP	⇒	NP	PP	0.7
PP	⇒	P	NP	1.0

Vi	⇒	sleeps	1.0
Vt	⇒	saw	1.0
NN	⇒	man	0.7
NN	⇒	woman	0.2
NN	⇒	telescope	0.1
DT	⇒	the	1.0
IN	⇒	with	0.5
IN	⇒	in	0.5

- Probability of a tree with rules $\alpha_i \rightarrow \beta_i$ is $\prod_i P(\alpha_i \rightarrow \beta_i | \alpha_i)$

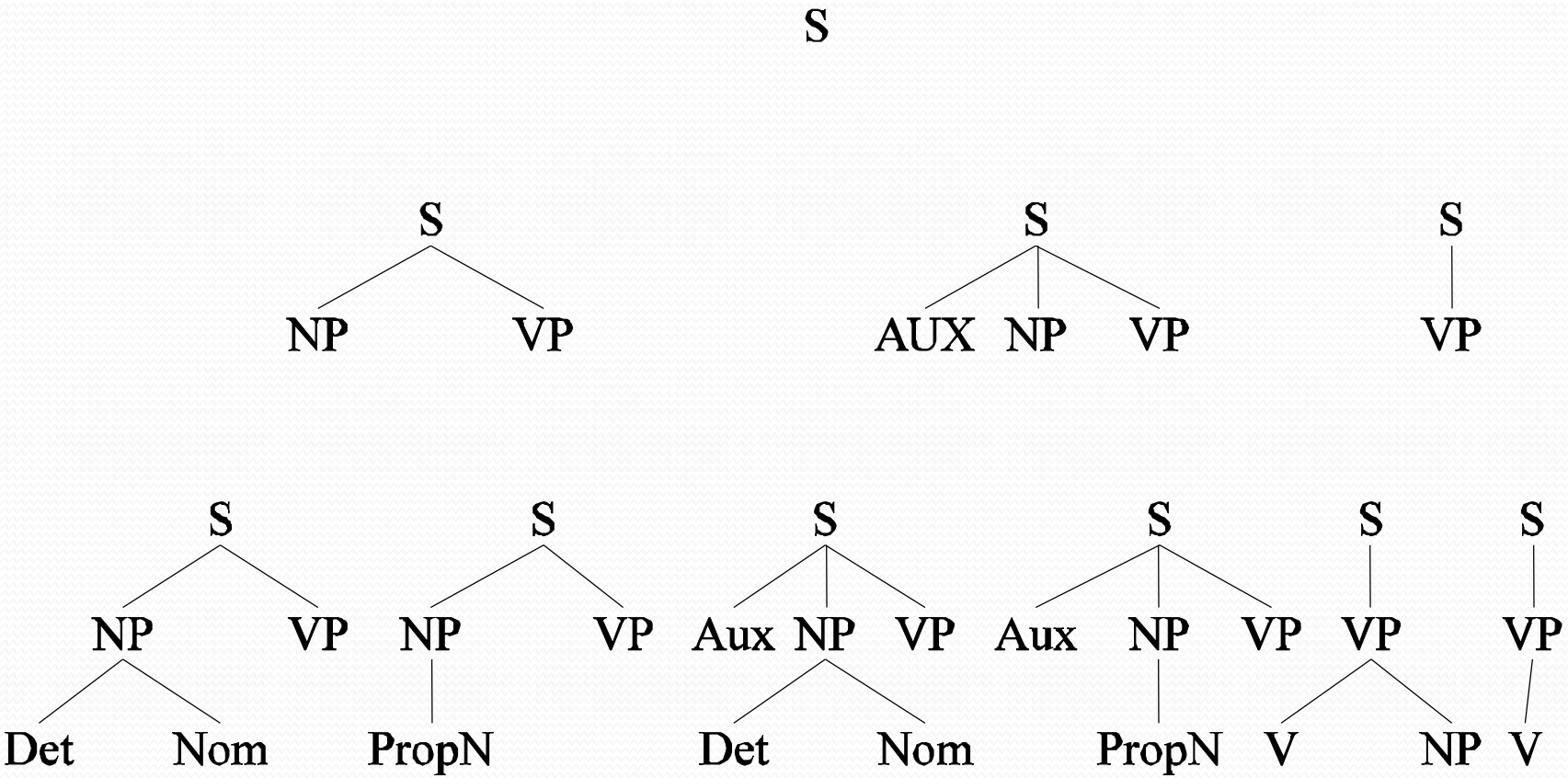
DERIVATION	RULES USED	PROBABILITY
S	S → NP VP	1.0
NP VP	NP → DT N	0.3
DT N VP	DT → the	1.0
the N VP	N → dog	0.1
the dog VP	VP → VB	0.4
the dog VB	VB → laughs	0.5
the dog laughs		

TOTAL PROBABILITY = $1.0 \times 0.3 \times 1.0 \times 0.1 \times 0.4 \times 0.5$

Top-Down Parsing

- Since we're trying to find trees rooted with an S (Sentences) start with the rules that give us an S.
- Then work your way down from there to the words.

Top Down Space



Bottom-Up Parsing

- Of course, we also want trees that cover the input words. So start with trees that link up with the words in the right way.
- Then work your way up from there.

Control

- Of course, in both cases we left out how to keep track of the search space and how to make choices
  - Which node to try to expand next
  - Which grammar rule to use to expand a node

Top-Down and Bottom-Up

- Top-down
  - Only searches for trees that can be answers (i.e. S's)
  - But also suggests trees that are not consistent with any of the words
- Bottom-up
  - Only forms trees consistent with the words
  - But suggest trees that make no sense globally

Problems

- Even with the best filtering, backtracking methods are doomed if they don't address certain problems
  - Ambiguity
  - Shared subproblems

Shared Sub-Problems

- No matter what kind of search (top-down or bottom-up or mixed) that we choose.
  - We don't want to unnecessarily redo work we've already done.

Shared Sub-Problems

- Assume a top-down parse making bad initial choices on the Nominal rule.
- In particular…
  - Nominal -> Nominal Noun
  - Nominal -> Nominal PP

Parsing strategy

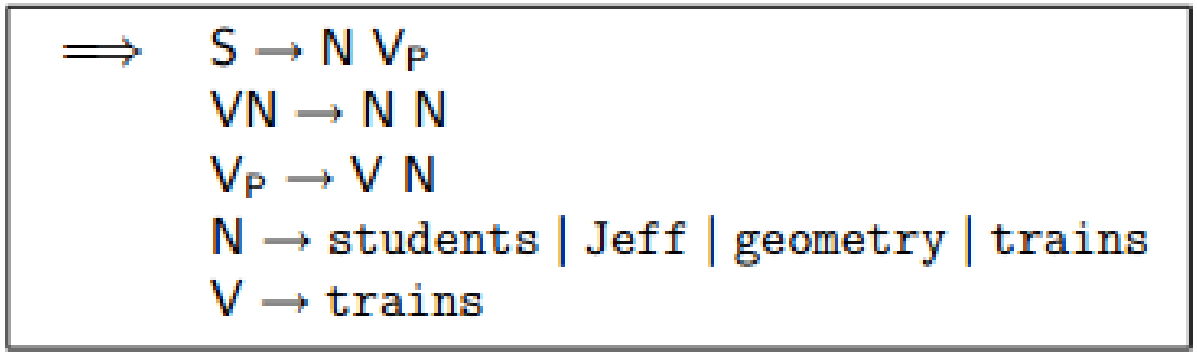
- Topdown
- Bottom-up

S -> X Y
X -> X A | a | b
Y -> A Y | a
A -> a

- Parsing “baaa”?

CYK (Cocke-Younger-Kasami)

- Chomsky Normal Form(Văn phạm dạng chuẩn Chomsky)
  Rule Forms:
  A -> B C
  A -> a

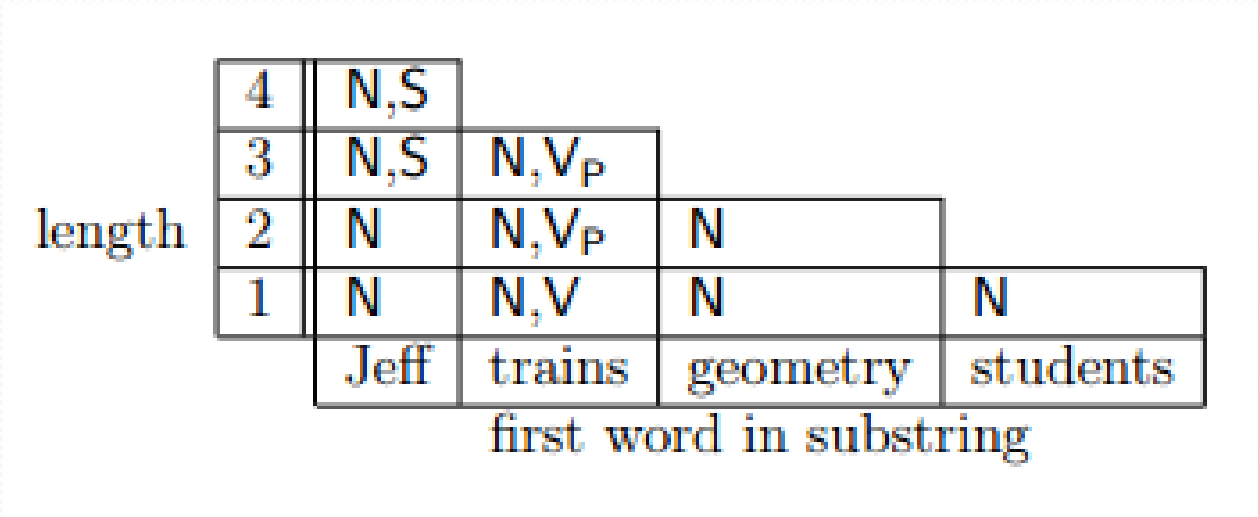


length					
	4				
	3				
	2				
	1				
		Jeff	trains	geometry	students

first word in substring

length					
	4				
	3				
	2				
	1	N	N,V	N	N
		Jeff	trains	geometry	students

first word in substring


length
4
3
2
1
N,S
N,S
N
N
N,VP
N,VP
N,V
N
N
Jeff
trains
geometry
students
first word in substring

CYK

Parsing the sentence:

"book that flight"

"book the flight through Houston"

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

Figure 13.1 The $\mathcal{L}_1$ miniature English grammar and lexicon.

CYK (cont.)

- CFG -> Chomsky Normal Form

1) A -> B C D

   A -> X D

   X -> B C

2) Remove? A -> B

B -> α genarate A -> α

$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

Figure 13.8 $\mathscr{L}_1$ Grammar and its conversion to CNF. Note that although they aren't shown here all the original lexical entries from $\mathscr{L}_1$ carry over unchanged as well.

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

Figure 13.1 The $\mathscr{L}_1$ miniature English grammar and lexicon.

CYK (cont.)

S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	V, VP	Det	N	P	Det	N
she	eats	a	fish	with	a	fork

Grammar?

CYK algorithm

```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar }
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                           {A | A → BC ∈ grammar,
                                B ∈ table[i,k],
                                C ∈ table[k,j] }
```

Figure 13.10 The CKY algorithm

Parsing problems?

- Inefficiency of backtracking parsers;
- Inadequacy of trees as representations for parsing when there are local ambiguities;
- Inadequacy of trees as representations for parsing when there are incomplete structures.

1.1 Parsing inefficiency

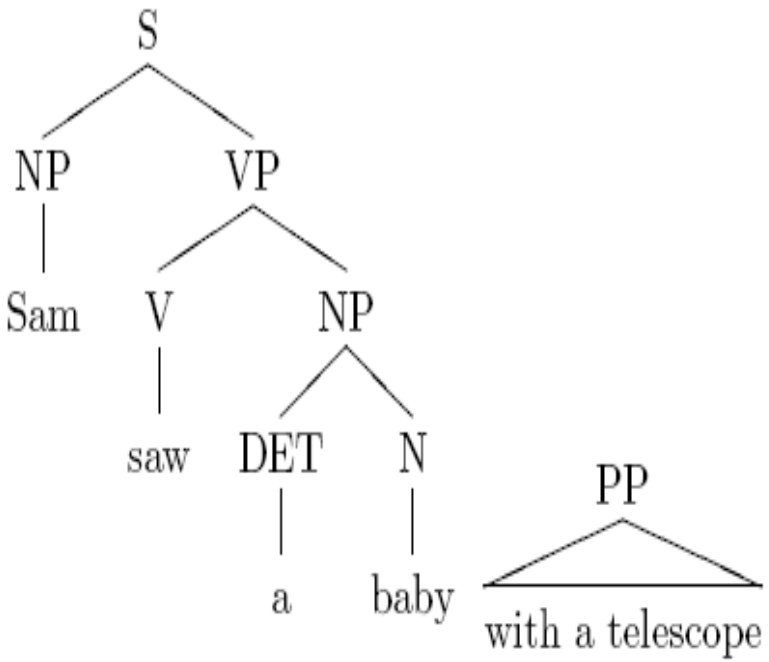
A backtracking parser may perform many redundant actions:

- rediscovering phrases it has already successfully found; and
- re–exploring hypotheses that have already failed.

(1) Have the [$_{NP}$ students that we talked about] take the exam.
(2) Have the [$_{NP}$ students that we talked about] taken the exam?
(3) The cherry blossoms [$_{PP}$ in the garden],
are lovely.
(4) The cherry blossoms [$_{PP}$ in the garden],
and a rose blooms by the house.
(5) Several cars raced [$_{PP}$ at the the most dangerous race track in the world]
were tested for problems.

1.2 Inadequacy of Trees (1)

Trees do not provide a neat way of representing local ambiguity. The PP below may attach to the NP, the VP, or the S. Representing each possibility requires a separate tree. It is not possible to collapse the trees so as to share the common information:



Inadequacy of Trees (2)

- Trees do not provide a neat way of representing incomplete structures (which may arise because the
- input cannot be completely parsed). The following is not a tree (no root):

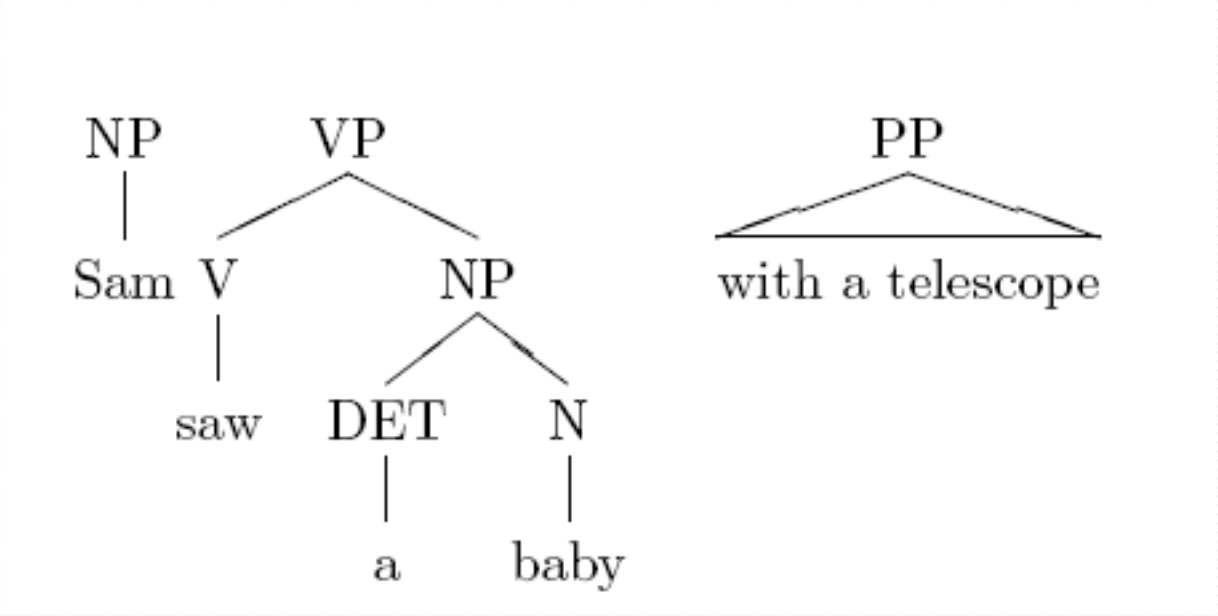


Chart parsing

The use of a chart offers three advantages:

- it avoids multiplication of effort;
- it provides a compact representation for 'local ambiguity';
- it provides a representation for 'partial parses'.

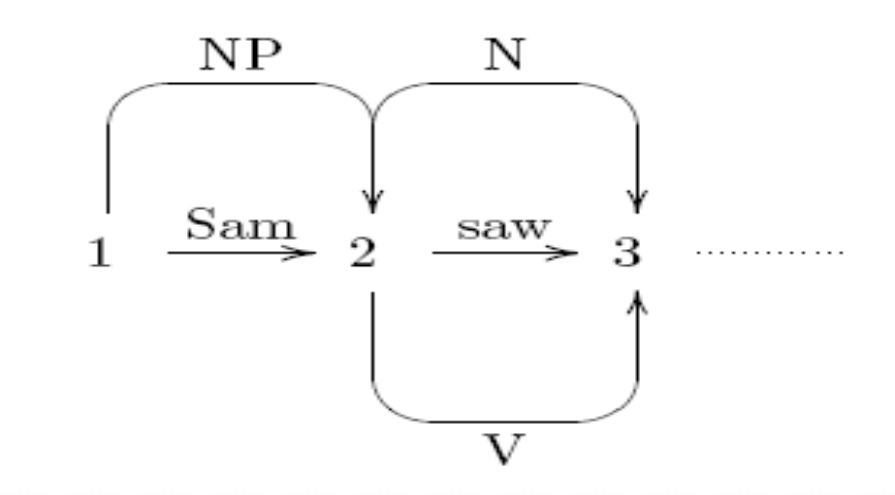
2.1 Avoiding duplication of work

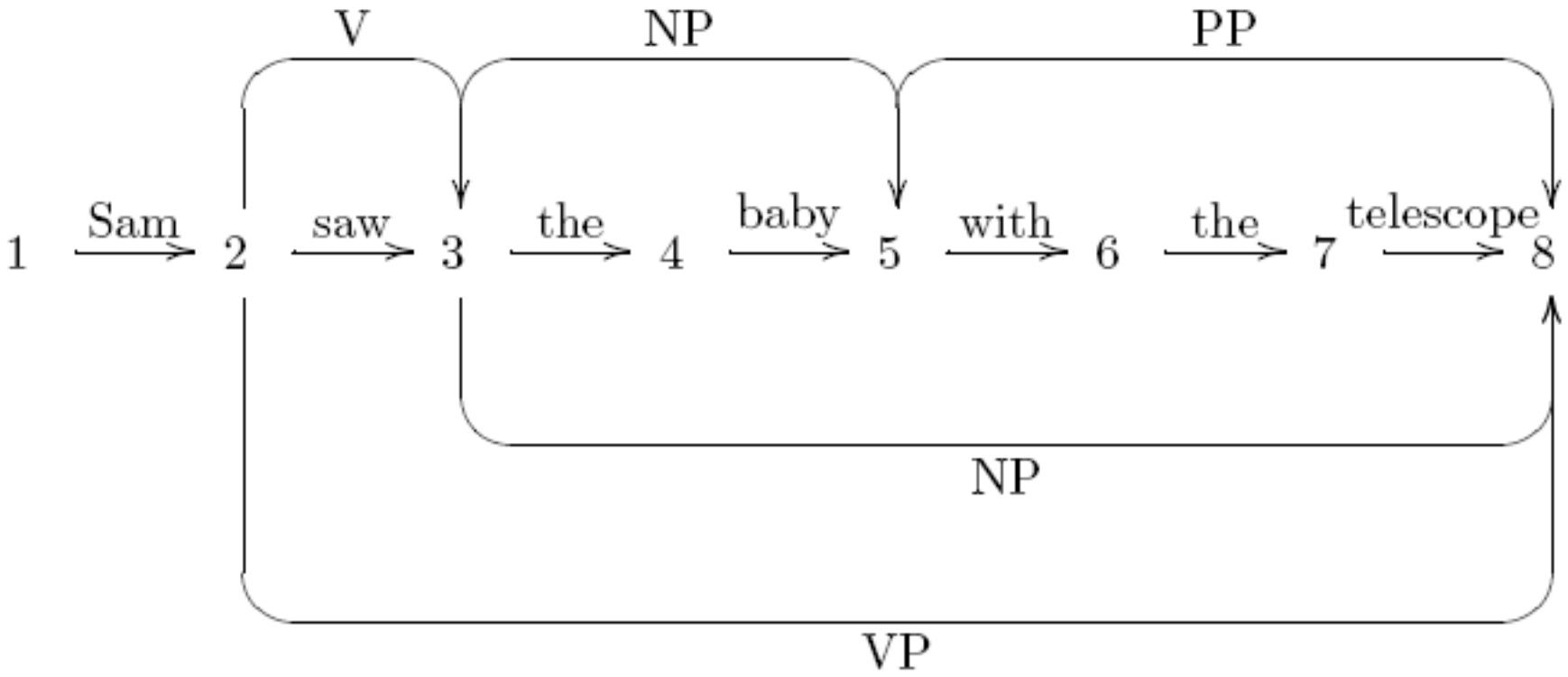
- As parts of the input are successfully parsed, they are entered in the chart. The idea is that before trying to parse any part of the input as (say) a PP, we look in the chart to see if we have parsed it as a PP before, (and if so, don't parse it again). Thus, the sub-string with a telescope is only parsed once as a PP.
- This technique is often called memoization.

2.2 Local Ambiguity

- The chart provides a compact representation of local ambiguity:
- The Basic Principle: Avoid duplication: Represent everything, but only represent it once.

- The N/V ambiguity of saw:




V
NP
PP
1
Sam
2
saw
3
the
4
baby
5
with
6
the
7
telescope
8
NP
VP

Earley Algorithm

- The **Earley parser** is a type of chart parser mainly used for parsing in computational linguistics, named after its inventor, Jay Earley. The algorithm uses dynamic programming.
- Earley parsers are appealing because they can parse all context-free languages. The Earley parser executes in cubic time (O(n3), where *n* is the length of the parsed string) in the general case, quadratic time (O(n2)) for unambiguous grammars, and linear time for almost all LR(k) grammars. It performs particularly well when the rules are written left-recursively.

Earley parsing

- Left to right
- Dynamic programming
- Using chart:
  - chart parser is a type of parser suitable for ambiguous grammars (including grammars of natural languages). It uses dynamic programming approach -- *partial hypothesized results are stored in a structure called a chart and can be re-used.* This eliminates backtracking and prevents a combinatorial explosion.

The Earley Parsing Algorithm

General Principles:

- A clever hybrid *Bottom-Up* and *Top-Down* approach
- *Bottom-Up* parsing completely guided by *Top-Down* predictions
- Maintains sets of "dotted" grammar rules that:
  - Reflect what the parser has "seen" so far
  - Explicitly predict the rules and constituents that will combine into a complete parse
- Similar to Chart Parsing - partial analyses can be shared
- Time Complexity $O(n^3)$, but better on particular sub-classes
- Developed prior to Chart Parsing, first efficient parsing algorithm for general context-free grammars.

The Earley Parsing Method

- Main Data Structure: The *"state"* (or *"item"*)
- A state is a "dotted" rule and starting position:
  $[A \rightarrow X_1 ... \bullet C ... X_m, p_i]$
- The algorithm maintains sets of "states", one set for each position in the input string (starting from 0)
- We denote the set for position i by S_i

The Earley Parsing Algorithm

Three Main Operations:

- **Predictor:** If state $[A \rightarrow X_1 ... \bullet C ... X_m, j] \in S_i$ then for every rule of the form $C \rightarrow Y_1 ... Y_k$, add to S_i the state $[C \rightarrow \bullet Y_1 ... Y_k, i]$
- **Completer:** If state $[A \rightarrow X_1 ... X_m \bullet, j] \in S_i$ then for every state in S_j of form $[B \rightarrow X_1 ... \bullet A ... X_k, l]$, add to S_i the state $[B \rightarrow X_1 ... A \bullet ... X_k, l]$
- **Scanner:** If state $[A \rightarrow X_1 ... \bullet a ... X_m, j] \in S_i$ and the next input word is $x_{i+1} = a$, then add to S_{i+1} the state $[A \rightarrow X_1 ... a \bullet ... X_m, j]$

The Earley Recognition Algorithm

The Main Algorithm: parsing input $x = x_1 ... x_n$

1. $S_0 = \{[S' \rightarrow \bullet S \ \$, 0]\}$
2. For $0 \leq i \leq n$ do:
   Process each item $s \in S_i$ in order by applying to it the *single* applicable operation among:
   (a) Predictor (adds new items to S_i)
   (b) Completer (adds new items to S_i)
   (c) Scanner (adds new items to S_{i+1})
3. If $S_{i+1} = \phi$, *Reject* the input
4. If $i = n$ and $S_{n+1} = \{[S' \rightarrow S \ \$\bullet, 0]\}$ then *Accept* the input

Earley Recognition - Example

The Grammar:

$$
\begin{aligned}
(1)\ S &\rightarrow NP\ VP \\
(2)\ NP &\rightarrow art\ adj\ n \\
(3)\ NP &\rightarrow art\ n \\
(4)\ NP &\rightarrow adj\ n \\
(5)\ VP &\rightarrow aux\ VP \\
(6)\ VP &\rightarrow v\ NP
\end{aligned}
$$

The original input: " $x =$ The large can can hold the water"

POS assigned input: " $x =$ art adj n aux v art n"

Parser input: " $x =$ art adj n aux v art n $"

Earley Recognition - Example

The input: "$x =$ **art** adj n aux v art n $"

S_0: $[S' \rightarrow \bullet S \$ \ , \ 0]$
$[S \rightarrow \bullet NP \ VP \ , \ 0]$
$[NP \rightarrow \bullet art \ adj \ n \ , \ 0]$
$[NP \rightarrow \bullet art \ n \ , \ 0]$
$[NP \rightarrow \bullet adj \ n \ , \ 0]$

S_1: $[NP \rightarrow art \ \bullet \ adj \ n \ , \ 0]$
$[NP \rightarrow art \ \bullet \ n \ , \ 0]$

Earley

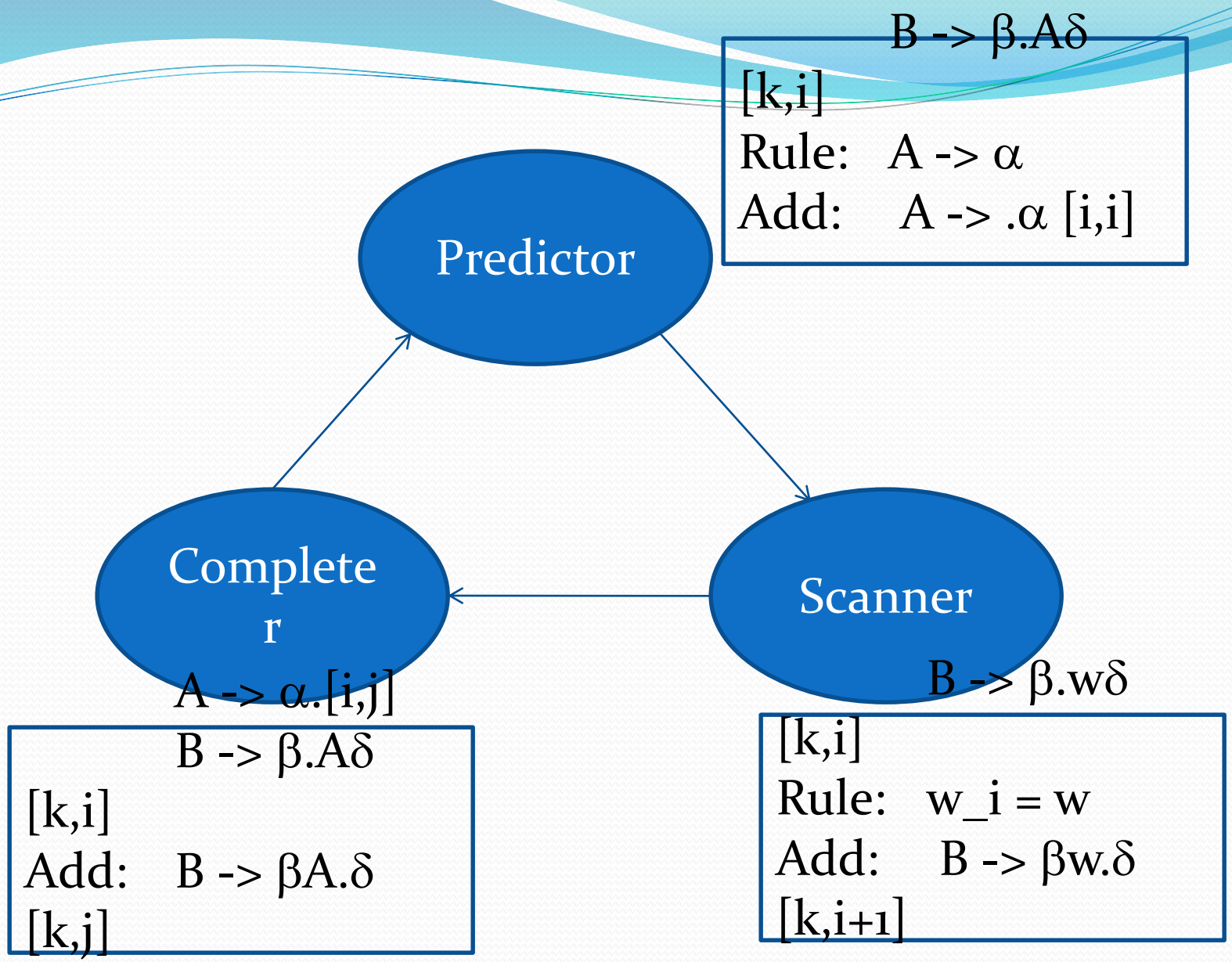
- Chart:

S -> .VP, [0,0]

NP -> Det . Nominal, [1,2]

VP -> V NP . , [0,3]

  - Dấu chấm: xác định tiến trình phân tích đối với luật đó. Luật có dấu chấm gọi là *dotted rule*
  - [*i,j*]: chart thứ *i,* ví trí word bắt đầu phân tích; *j* là vị trí word đang phân tích
  - Trạng thái kết thúc *S -> α., [0,N]*


B -> β.Aδ
[k,i]
Rule: A -> α
Add: A -> .α [i,i]
Predictor
Complete
r
A -> α.[i,j]
B -> β.Aδ
[k,i]
Add: B -> βA.δ
[k,j]
Scanner
B -> β.wδ
[k,i]
Rule: w_i = w
Add: B -> βw.δ
[k,i+1]

Earley

- Exercise
  - Parse the sentence: "book that flight"

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

Figure 13.1 The $\mathscr{L}_1$ miniature English grammar and lexicon.

Chart[0]	S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor
Chart[1]	S12	$Verb \rightarrow book \bullet$	[0,1]	Scanner
	S13	$VP \rightarrow Verb \bullet$	[0,1]	Completer
	S14	$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
	S15	$VP \rightarrow Verb \bullet NP\ PP$	[0,1]	Completer
	S16	$VP \rightarrow Verb \bullet PP$	[0,1]	Completer
	S17	$S \rightarrow VP \bullet$	[0,1]	Completer
	S18	$VP \rightarrow VP \bullet PP$	[0,1]	Completer
	S19	$NP \rightarrow \bullet Pronoun$	[1,1]	Predictor
	S20	$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor
	S21	$NP \rightarrow \bullet Det\ Nominal$	[1,1]	Predictor
	S22	$PP \rightarrow \bullet Prep\ NP$	[1,1]	Predictor

Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
	S24	*NP* → *Det* • *Nominal*	[1,2]	Completer
	S25	*Nominal* → • *Noun*	[2,2]	Predictor
	S26	*Nominal* → • *Nominal Noun*	[2,2]	Predictor
	S27	*Nominal* → • *Nominal PP*	[2,2]	Predictor
Chart[3]	S28	*Noun* → *flight* •	[2,3]	Scanner
	S29	*Nominal* → *Noun* •	[2,3]	Completer
	S30	*NP* → *Det Nominal* •	[1,3]	Completer
	S31	*Nominal* → *Nominal* • *Noun*	[2,3]	Completer
	S32	*Nominal* → *Nominal* • *PP*	[2,3]	Completer
	S33	*VP* → *Verb NP* •	[0,3]	Completer
	S34	*VP* → *Verb NP* • *PP*	[0,3]	Completer
	S35	*PP* → • *Prep NP*	[3,3]	Predictor
	S36	*S* → *VP* •	[0,3]	Completer
	S37	*VP* → *VP* • *PP*	[0,3]	Completer

Exercise 1

- $P \rightarrow S$
- $S \rightarrow S + M \mid M$
- $M \rightarrow M * T \mid T$
- $T \rightarrow$ number

- Input: “2 + 3 * 4”

Exercise 2

1. S -> NP VP
2. S -> VP
3. NP -> N
4. NP -> DT NP
5. VP -> V
6. VP -> V NP
7. N -> she | book
8. V -> likes
9. DT -> this

Input:

she likes this book

Compare between Earley and CYK?

- ???

Earley: Ví dụ

- The algorithm sequentially constructs the sets S_i for $0 \leq i \leq n+1$
- We initialize the set S_0 with $S_0 = \{[S' \rightarrow \bullet S\ \$, 0]\}$

The Grammar:

$$
\begin{array}{rrcl}
(1) & S & \rightarrow & NP\ VP \\
(2) & NP & \rightarrow & art\ adj\ n \\
(3) & NP & \rightarrow & art\ n \\
(4) & NP & \rightarrow & adj\ n \\
(5) & VP & \rightarrow & aux\ VP \\
(6) & VP & \rightarrow & v\ NP
\end{array}
$$

The original input: " $x =$ The large can can hold the water"
POS assigned input: " $x =$ art adj n aux v art n"
Parser input: " $x =$ art adj n aux v art n $"

Parsing (lecture3)

- CYK
- Earley
- Chart-parsing …
- Lexicalized parsing …
- Probabilistic parsing …
- Unification Grammar …

Exercise

1. S -> NP VP
2. S -> VP
3. NP -> N
4. NP -> DT NP
5. VP -> V
6. VP -> V NP
7. N -> she | book
8. V -> likes
9. DT -> this

Input:

she likes this book

Limitations of Earley
-> sinh thừa trạng thái

1. S -> NP VP
2. S -> VP
3. NP -> N
4. NP -> DT NP
5. VP -> V
6. VP -> V NP
7. N -> she | book
8. V -> likes
9. DT -> this

Input:
she likes this book

Chart [0]
- S' -> .S [0,0]
- S -> . NP VP [0,0]
- S -> . VP [0,0]
- NP -> . N [0,0]
- NP -> . DT NP [0,0]
- VP -> . V [0,0]
- VP -> . V NP [0,0]
- N -> . she [0,0]
- N -> . book [0,0]
- V -> . likes [0,0]
- DT -> . this [0,0]

Chart[1]
- N -> she . [0,1]
- NP -> N . [0,1]
- S -> NP . VP [0,1]

Chart
- N -> she . [0,1]
- V -> likes . [1,2]
- DT -> this . [2,3]
- N -> book . [3,4]
- NP -> N . [0,1]
- NP -> N . [3,4]
- VP -> V . [1,2]
- VP -> V. NP [1,2]
- NP -> DT . NP [2,3]
- NP -> DT NP . [2,4]
- VP -> V NP . [1,4]
- S -> NP . VP [0,1]
- S -> NP VP . [0,4]

- S -> NP . VP [3,4]
- VP -> V NP . [0,2]
- S -> NP VP . []

Chart Parsing

General Principles:

- A *Bottom-Up* parsing method
  - Construct a parse starting from the input symbols
  - Build constituents from sub-constituents
  - When all constituents on the RHS of a rule are matched, create a constituent for the LHS of the rule
- The *Chart* allows storing partial analyses, so that they can be shared.
- Data structures used by the algorithm:
  - **The Key:** the current constituent we are attempting to "match"
  - **An Active Arc:** a grammar rule that has a partially matched RHS
  - **The Agenda:** Keeps track of newly found unprocessed constituents
  - **The Chart:** Records processed constituents (non-terminals) that span substrings of the input

Chart Parsing

Steps in the Process:

- Input is processed left-to-right, one word at a time

1. Find all POS of word (terminal-level)
2. Initialize Agenda with all POS of the word
3. Pick a Key from the Agenda
4. Add all grammar rules that start with the Key as active arcs
5. Extend any existing active arcs with the Key
6. Add LHS constituents of newly completed rules to the Agenda
7. Add the Key to the Chart
8. If Agenda not empty - goto (3), else goto (1)

The Chart Parsing Algorithm

Extending Active Arcs with a Key:

- Each **Active Arc** has the form: $[A \rightarrow X_1 ... \bullet C ... X_m](p_i, p_j)$
- A Key constituent has the form: $C(p_i, p_j)$
- When processing the Key $C(p_1, p_2)$, we search the active arc list for an arc $[A \rightarrow X_1 ... \bullet C ... X_m](p_0, p_1)$, and then create a new active arc $[A \rightarrow X_1 ... C \bullet ... X_m](p_0, p_2)$
- If the new active arc is a completed rule: $[A \rightarrow X_1 ... C \bullet](p_0, p_2)$, then we add $A(p_0, p_2)$ to the Agenda
- After "using" the key to extend all relevant arcs, it is entered into the Chart

```
Lexicon:

  I:          N
  can:        N, AUX, V
  see:        V
  the:        DET
  man:        N, V
  with:       P
  telescope:  N, V
```

```
Grammar:

  (1)   S --> NP VP
  (2)  NP --> DET N
  (3)  NP --> N
  (4)  NP --> NP PP
  (5)  VP --> AUX VP
  (6)  VP --> V NP
  (7)  VP --> V
  (8)  PP --> P NP
```

The Chart Parsing Algorithm

The Main Algorithm: parsing input $x = x_1 ... x_n$

1. $i = 0$
2. If Agenda is empty and $i < n$ then set $i = i + 1$, find all POS of x_i and add them as constituents $C(p_i, p_{i+1})$ to the Agenda
3. Pick a Key constituent $C(p_j, p_{i+1})$ from the Agenda
4. For each grammar rule of form $A \rightarrow CX_1 ... X_m$, add $[A \rightarrow \bullet CX_1 ... X_m](p_j, p_j)$ to the list of active arcs
5. Use Key to extend all relevant active arcs
6. Add LHS of any completed active arcs into the Agenda
7. Insert the Key into the Chart
8. If Key is $S(1, n)$ then Accept the input
   Else goto (2)

The Chart Parsing Algorithm

The Main Algorithm: parsing input $x = x_1 ... x_n$

1. $i = 0$
2. If Agenda is empty and $i < n$ then set $i = i + 1$, find all POS of x_i and add them as constituents $C(p_i, p_{i+1})$ to the Agenda
3. Pick a Key constituent $C(p_j, p_{i+1})$ from the Agenda
4. For each grammar rule of form $A \rightarrow CX_1 ... X_m$, add $[A \rightarrow \bullet CX_1 ... X_m](p_j, p_j)$ to the list of active arcs
5. Use Key to extend all relevant active arcs
6. Add LHS of any completed active arcs into the Agenda
7. Insert the Key into the Chart
8. If Key is $S(1, n)$ then Accept the input
   Else goto (2)

Agenda	$[\mathrm{NP}_2 \rightarrow \mathrm{N}_1](1, 2)$
Active Arcs	$[\mathrm{NP} \rightarrow \bullet\ \mathrm{N}](1, 1)$ $[\mathrm{S} \rightarrow \bullet\ \mathrm{NP\ VP}](1, 1)$ $[\mathrm{NP} \rightarrow \bullet\ \mathrm{NP\ PP}](1, 1)$ $[\mathrm{S} \rightarrow \mathrm{NP}_2 \bullet\ \mathrm{VP}](1, 2)$ $[\mathrm{NP} \rightarrow \mathrm{NP}_2 \bullet\ \mathrm{PP}](1, 2)$
Chart	$\mathrm{N}_1(1, 2)$

Exercise 1

The input: "x = The large can can hold the water"

POS of Input Words:

- the: *ART*
- large: *ADJ*
- can: *N, AUX, V*
- hold: *N, V*
- water: *N, V*

(1) $S \rightarrow NP\ VP$

2) $NP \rightarrow ART\ ADJ\ N$

3) $NP \rightarrow ART\ N$

4) $NP \rightarrow ADJ\ N$

5) $VP \rightarrow AUX\ VP$

6) $VP \rightarrow V\ NP$