CKY & Earley Parsing

Ling 571
Deep Processing Techniques for NLP
January 14, 2015

Roadmap

- CKY Parsing:
  - Recognizer → Parser
- Earley parsing
  - Motivation:
    - CKY Strengths and Limitations
  - Earley model:
    - Efficient parsing with arbitrary grammars
    - Procedures:
      - Predictor, Scanner , Completer

0 Book 1 the 2 flight 3 through 4 Houston 5

Book	the	Flight	Through	Houston
NN, VB, Nominal, VP, S [0,1]	[0,2]	S, VP, X2 [0,3]	[0,4]	S, VP, X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		NN, Nominal [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NNP, NP [4,5]

From Recognition to Parsing

- Limitations of current recognition algorithm:

From Recognition to Parsing

- Limitations of current recognition algorithm:
  - Only stores non-terminals in cell
    - Not rules or cells corresponding to RHS

From Recognition to Parsing

- Limitations of current recognition algorithm:
  - Only stores non-terminals in cell
    - Not rules or cells corresponding to RHS
  - Stores SETS of non-terminals
    - Can't store multiple rules with same LHS

From Recognition to Parsing

- Limitations of current recognition algorithm:
  - Only stores non-terminals in cell
    - Not rules or cells corresponding to RHS
  - Stores SETS of non-terminals
    - Can't store multiple rules with same LHS
- Parsing solution:
  - All repeated versions of non-terminals

From Recognition to Parsing

- Limitations of current recognition algorithm:
  - Only stores non-terminals in cell
    - Not rules or cells corresponding to RHS
  - Stores SETS of non-terminals
    - Can't store multiple rules with same LHS

- Parsing solution:
  - All repeated versions of non-terminals
  - Pair each non-terminal with pointers to cells
    - Backpointers

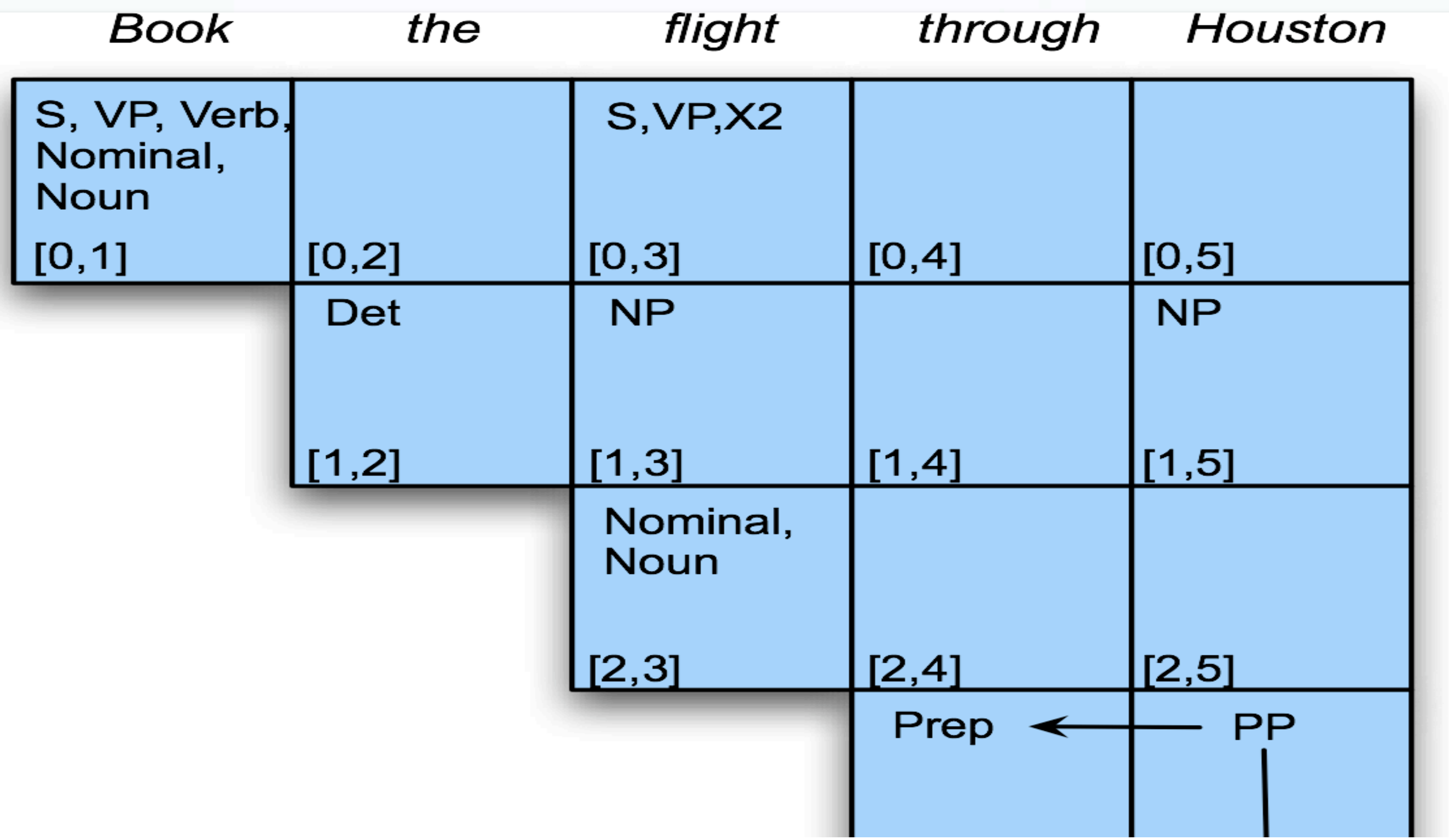
From Recognition to Parsing

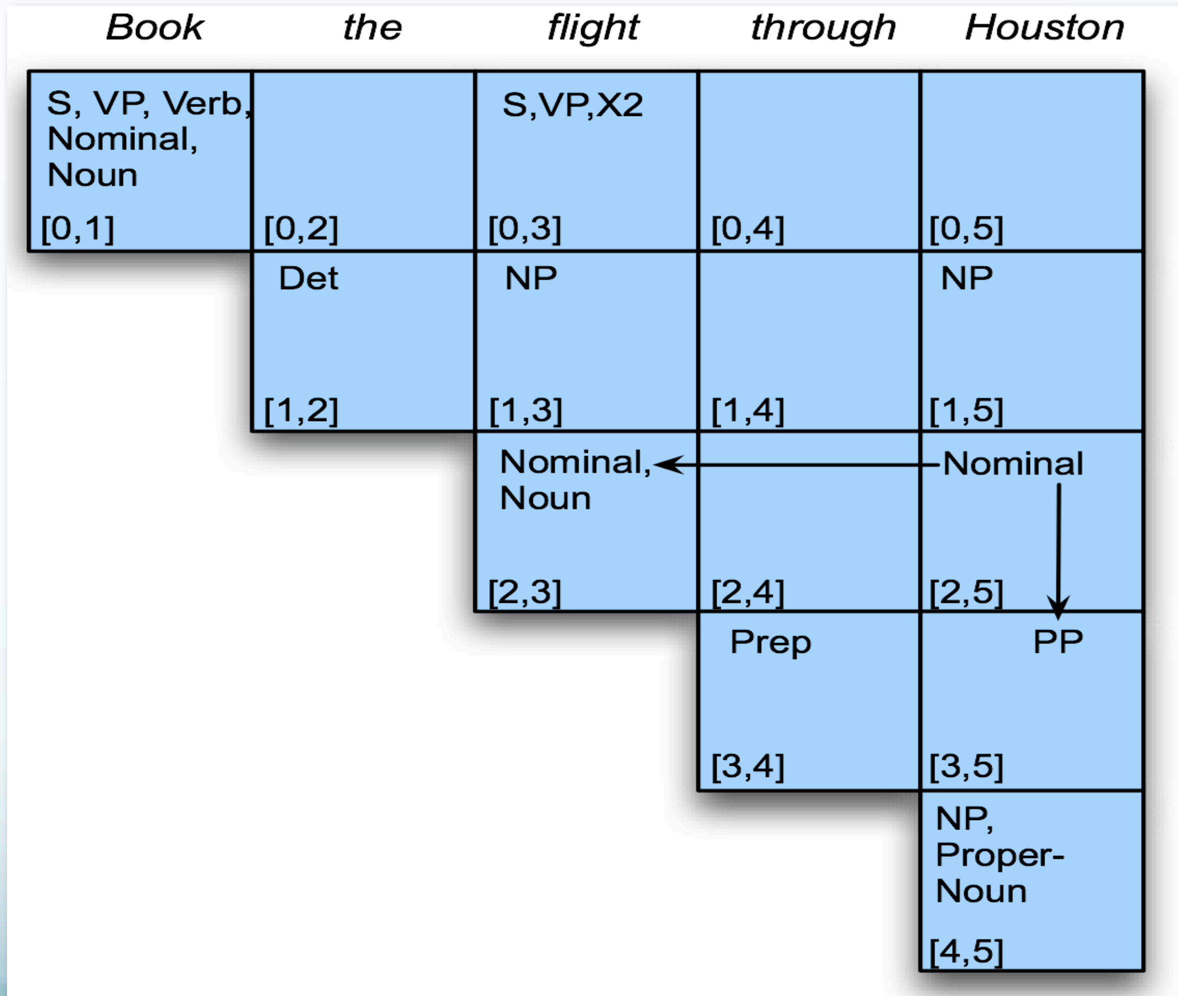
- Limitations of current recognition algorithm:
  - Only stores non-terminals in cell
    - Not rules or cells corresponding to RHS
  - Stores SETS of non-terminals
    - Can't store multiple rules with same LHS

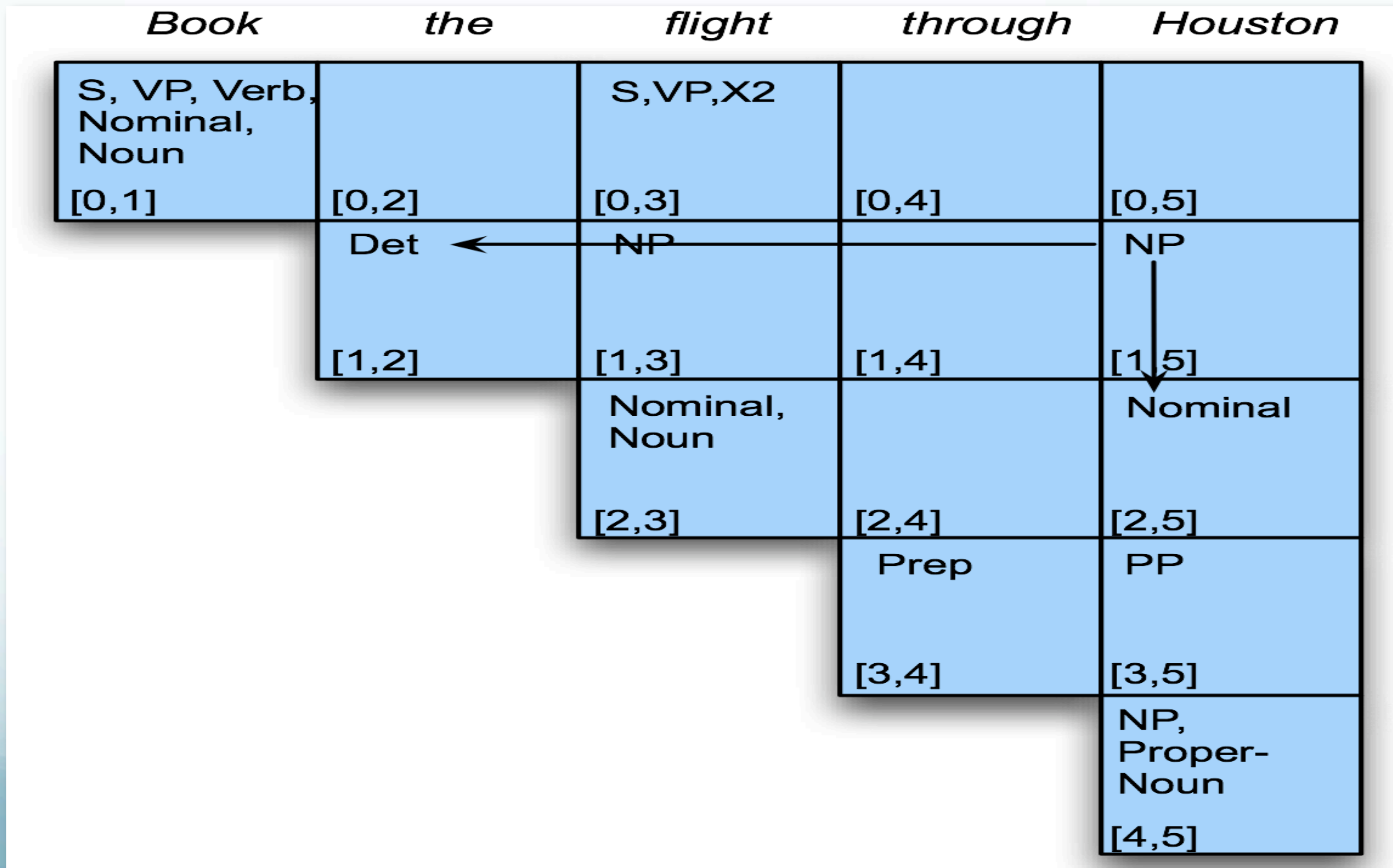
- Parsing solution:
  - All repeated versions of non-terminals
  - Pair each non-terminal with pointers to cells
    - Backpointers
  - Last step: construct trees from back-pointers in [0,n]

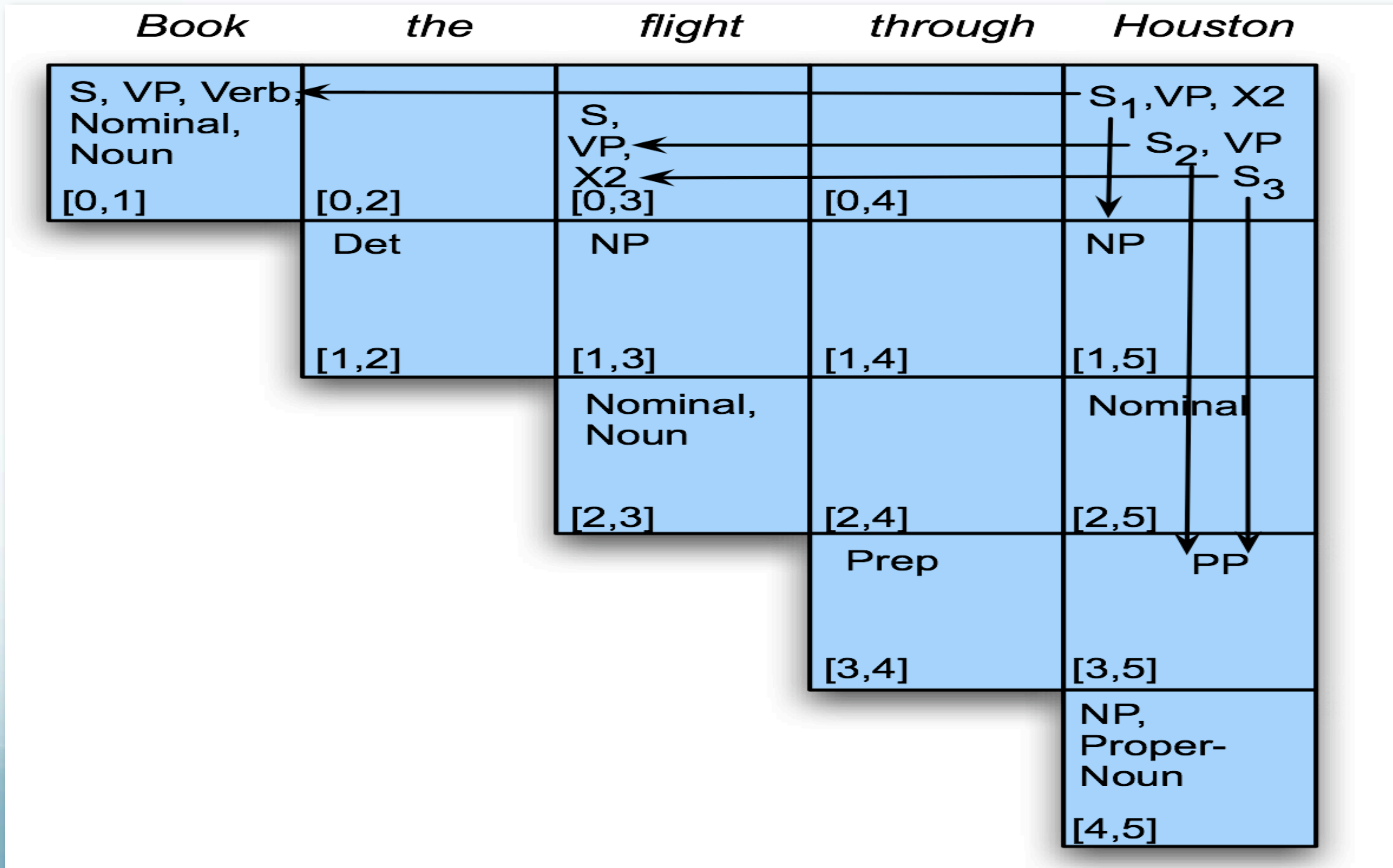
Filling column 5

	Book	*the*	*flight*	*through*	*Houston*
	S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
		Det [1,2]	NP [1,3]	[1,4]	[1,5]
			Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
				Prep [3,4]	[3,5]
					NP, Proper-Noun [4,5]


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
[2,5]
Prep
PP


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]


Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S,VP,X2
[0,3]
[0,4]
[0,5]
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]



Book
the
flight
through
Houston
S, VP, Verb, Nominal, Noun
[0,1]
[0,2]
S, VP, X2
[0,3]
[0,4]
S1,VP, X2
S2, VP
S3
Det
[1,2]
NP
[1,3]
[1,4]
NP
[1,5]
Nominal, Noun
[2,3]
[2,4]
Nominal
[2,5]
Prep
[3,4]
PP
[3,5]
NP, Proper-Noun
[4,5]

CKY Discussion

- Running time:

  $O(n^3)$

CKY Discussion

- Running time:
  - $O(n^3)$ where *n* is the length of the input string

CKY Discussion

- Running time:
  - $O(n^3)$ where *n* is the length of the input string
  - Inner loop grows as square of # of non-terminals
- Expressiveness:

CKY Discussion

- Running time:
  - $O(n^3)$ where *n* is the length of the input string
  - Inner loop grows as square of # of non-terminals
- Expressiveness:
  - As implemented, requires CNF
    - Weakly equivalent to original grammar
    - Doesn't capture full original structure
      - Back-conversion?

CKY Discussion

- Running time:
  - $O(n^3)$ where *n* is the length of the input string
  - Inner loop grows as square of # of non-terminals

- Expressiveness:
  - As implemented, requires CNF
    - Weakly equivalent to original grammar
    - Doesn't capture full original structure
      - Back-conversion?
        - Can do binarization, terminal conversion
        - Unit non-terminals require change in CKY

Parsing Efficiently

- With arbitrary grammars
  - Earley algorithm
    - Top-down search
    - Dynamic programming
      - Tabulated partial solutions
    - Some bottom-up constraints

Earley Parsing

- Avoid repeated work/recursion problem
  - Dynamic programming
    - Store partial parses in “chart”
      - Compactly encodes ambiguity
    - $O(N^3)$

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

- 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

Parse Progress

- Represented by Dotted Rules
- Position of • indicates type of constituent
- $_0$ Book $_1$ that $_2$ flight $_3$
  - S → • VP, [0,0] (predicted)

Parse Progress

- 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)

Parse Progress

- 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)

Parse Progress

- 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

Parse Progress

- 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

$_0$ 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

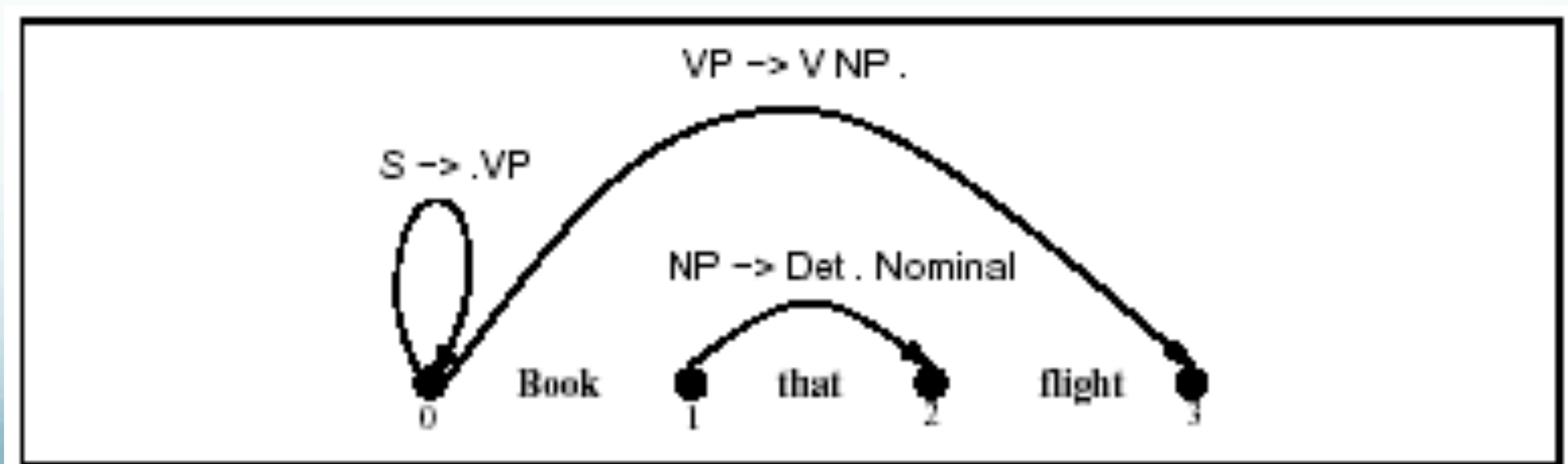
NP → Det • Nom, [1,2]

- 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

Successful Parse

- Final answer found by looking at last entry in chart
- If entry resembles $S \rightarrow \alpha \bullet [0,N]$ 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

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

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)
```

Earley Algorithm from Book

procedure PREDICTOR($(A \rightarrow \alpha \bullet B \beta, [i,j])$)
  for each $(B \rightarrow \gamma)$ **in** GRAMMAR-RULES-FOR(B, *grammar*) **do**
    ENQUEUE($(B \rightarrow \bullet \gamma, [j,j])$, *chart*$[j]$)
  end

procedure SCANNER($(A \rightarrow \alpha \bullet B \beta, [i,j])$)
  if B $\subset$ PARTS-OF-SPEECH(*word[j]*) **then**
    ENQUEUE($(B \rightarrow word[j], [j, j+1])$, *chart[j+1]*)

procedure COMPLETER($(B \rightarrow \gamma \bullet, [j,k])$)
  for each $(A \rightarrow \alpha \bullet B \beta, [i,j])$ **in** *chart*$[j]$ **do**
    ENQUEUE($(A \rightarrow \alpha B \bullet \beta, [i,k])$, *chart[k]*)
  end

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.

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]**

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]

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]

Chart[0]

S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded.

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

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded.

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

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded.

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

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded.

Scanner

- Intuition: Create new states for rules matching part of speech of next word.

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…’

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
  V → Book •[0,1]

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
  V → Book •[0,1]
- Formally:
  S_j: A → α • B β, [i,j], B in POS(words[j])
- S_{j+1}: B → words[j]•, [j,j+1]

Completer

- Intuition: parser has finished a new phrase, so must find and advance all states that were waiting for this

Completer

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

  NP → Det Nom • [1,3]

Completer

- Intuition: parser has finished a new phrase, so must find and advance all states 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]

Completer

- Intuition: parser has finished a new phrase, so must find and advance all states 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].

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

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded.

Chart[1]

Chart[1]

S12	$Verb \rightarrow book \bullet$	[0,1]	Scanner

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

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

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

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 few rules in Chart[1] are added by **Predictor** when VP → V • NP is processed
- And so on….

Charts[2] and [3]

Charts[2] and [3]

S23	$Det \rightarrow that \bullet$	[1,2]	Scanner

Charts[2] and [3]

S23	$Det \rightarrow that \bullet$	[1,2]	Scanner
S24	$NP \rightarrow Det \bullet Nominal$	[1,2]	Completer

Charts[2] and [3]

S23	$Det \rightarrow that \bullet$	[1,2]	Scanner
S24	$NP \rightarrow Det \bullet Nominal$	[1,2]	Completer
S25	$Nominal \rightarrow \bullet Noun$	[2,2]	Predictor
S26	$Nominal \rightarrow \bullet Nominal\ Noun$	[2,2]	Predictor
S27	$Nominal \rightarrow \bullet Nominal\ PP$	[2,2]	Predictor

Charts[2] and [3]

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
S28	*Noun* → *flight* •	[2,3]	Scanner

Charts[2] and [3]

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
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

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

- What about ambiguity?

- What about ambiguity?
- CKY/Earley can represent it

- What about ambiguity?
- CKY/Earley can represent it
- Can’t resolve it