Computerlinguistik II

Vorlesung im SoSe 2019
(M-GSW-10)

Prof. Dr. Udo Hahn

Lehrstuhl für Computerlinguistik
Institut für Germanistische Sprachwissenschaft
Friedrich-Schiller-Universität Jena

http://www.julielab.de

Allgemeine Hinweise

- Vorlesung: Mi, 10-12h (Humboldt 8, SR 3)
- Übung zV: Di 10-12h (Fürstengraben 27, CoKa)
  - Hat schon begonnen
- Vorlesungsmaterialien im Netz
  - **http://www.julielab.de/** ⇨ „Students“
- M-GSW-10 besteht aus VL+ÜB und Seminar!
- Sprechstunde: Mi, 12-13h (bA) (FG 30, R 004)
- Email: **udo.hahn@uni-jena.de**
- URL: **http://www.julielab.de**
- Fachliteratur ist überwiegend in Englisch

Veranstaltungen im SS 2019

- Seminar „Toxische Sprache im Internet“
  - Do, 16-18 Uhr
- Software-Praktikum: „Softwaretechnologien für Natürlichsprachliche Systeme“
  - Di, 16-18 Uhr
- Theoreticum: Methoden der Computerlinguistik,
- Technicum: Praxis sprachtechnologischer Systeme
  - Alterierend: Fr, 9-11 Uhr

der folgende Teil der Vorlesung
ist einer Ausarbeitung entnommen von

Dr. Christel Kemke

Department of Computer Science
University of Manitoba

Sample Grammar

Grammar (S, NT, T, P) – Sentence Symbol $S \in NT$, Part-of-Speech $\subseteq NT$, syntactic Constituents $\subseteq NT$, Grammar Rules $P \subseteq NT \times (NT \cup T)^*$

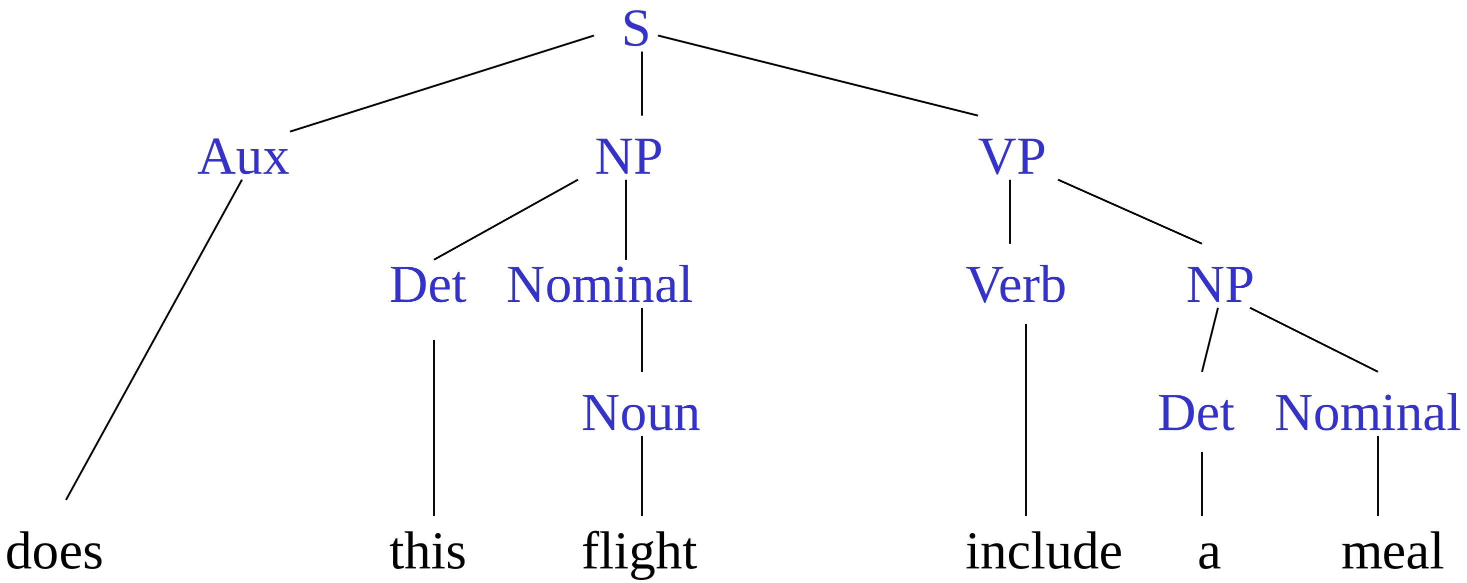
S → NP VP statement
S → Aux NP VP question
S → VP command
NP → Det Nominal
NP → Proper-Noun
Nominal → Noun | Noun Nominal | Nominal PP
VP → Verb | Verb NP | Verb PP | Verb NP PP
PP → Prep NP

Det → that | this | a
Noun → book | flight | meal | money
Proper-Noun→ Houston | American Airlines | TWA
Verb → book | include | prefer
Aux → does
Prep → from | to | on

Task: Parse "Does this flight include a meal?"

Sample Parse Tree

Task: Parse "Does this flight include a meal?"



Problems with Bottom-up and Top-down Parsing

Problems with left-recursive rules like NP → NP PP: don’t know how many times recursion is needed (top-down)

Pure Bottom-up or Top-down Parsing is inefficient because it generates and explores too many structures which in the end turn out to be invalid (several grammar rules applicable → ‘interim’ ambiguity).

Combine top-down and bottom-up approach: Start with sentence; use rules top-down (look-ahead); read input; try to find shortest path from input to highest unparsed constituent (from left to right).

→ Chart-Parsing / Earley-Parser

Chart Parsing / Earley Algorithm

Earley-Parser based on Chart-Parsing

Essence: Integrate top-down and bottom-up parsing. Keep recognized sub-structures (sub-trees) for shared use during parsing.

Top-down: Start with S-symbol. Generate all applicable rules for S. Go further down with left-most constituent in rules and add rules for these constituents until you encounter a left-most node on the RHS (Right Hand Side of a production) which is a word category (POS).

Bottom-up: Read input word and compare. If word matches, mark as recognized and move parsing on to the next category in the rule(s).

Chart – a Graph-Based Data Structure for Parsing

Chart

Sequence of n input words; n+1 nodes marked 0 to n.

Arcs indicate recognized part of RHS of rule.

The • indicates recognized constituents in rules.

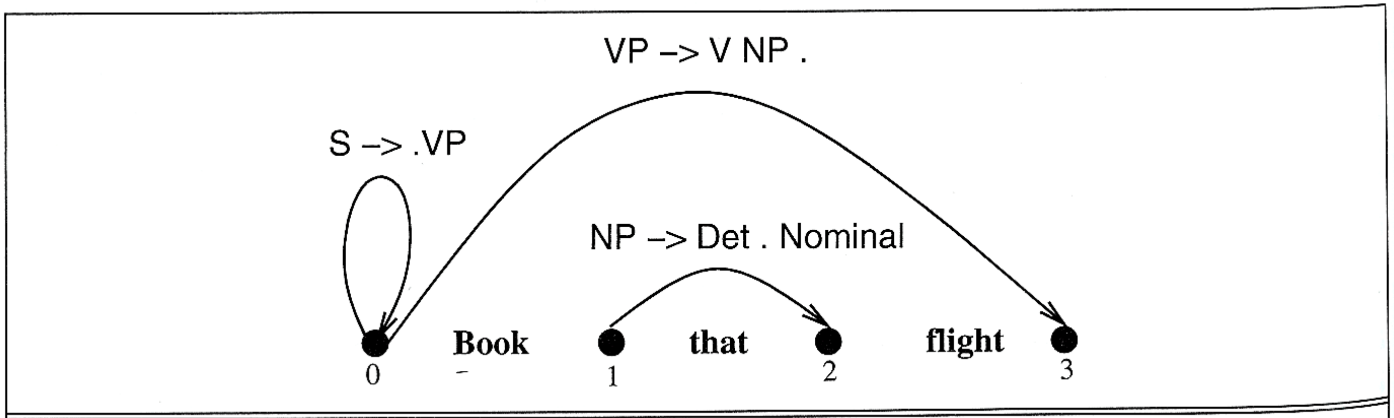


Figure 10.15 A directed acyclic graph representation of the three dotted rules above.

Jurafsky & Martin, Figure 10.15, p. 380

Chart Parsing / Earley Parser 1

Chart

Sequence of input words; n+1 nodes marked 0 to n.

States in chart represent possible rules and recognized constituents, with arcs.

Interim state

S → • VP, [0,0]

- top-down look at rule S → VP
- nothing of RHS of rule yet recognized (• is far left)
- arc at beginning, no coverage (covers no input word; beginning of arc at 0 and end of arc at 0)

Chart Parsing / Earley Parser 2

Interim states

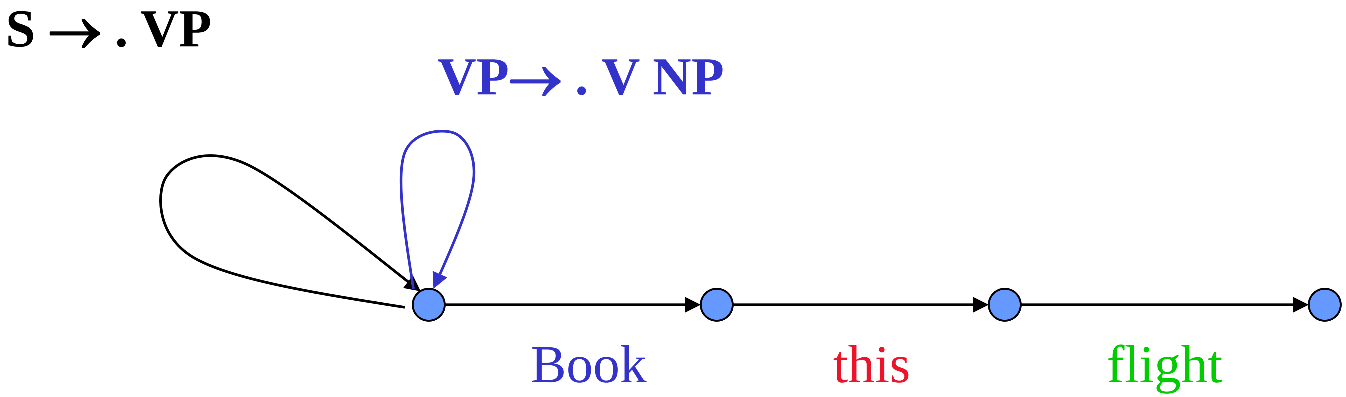
NP → Det • Nominal, [1,2]

- ➢ top-down look with rule NP → Det • Nominal
- ➢ Det recognized (• after Det)
- ➢ arc covers one input word which is between node 1 and node 2
- ➢ look next for Nominal

NP → Det Nominal • , [1,3]

- ➢ Nominal was recognized, move • after Nominal
- ➢ move end of arc to cover Nominal (change 2 to 3)
- ➢ structure is completely recognized; arc is inactive; mark NP as recognized in other rules (move •).

Chart - 0



S → . VP

VP→ . V NP

Book this flight

Chart - 1

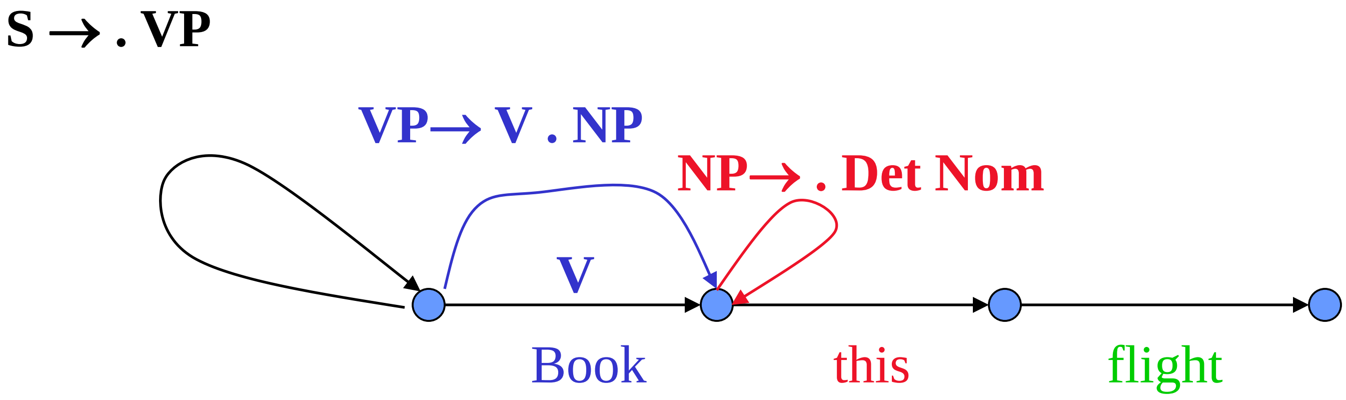


Chart - 2

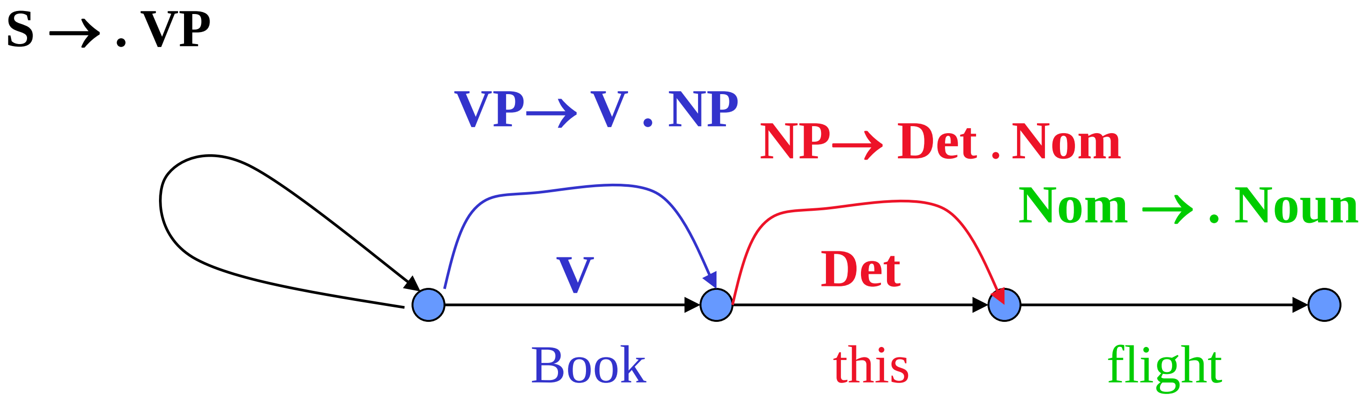
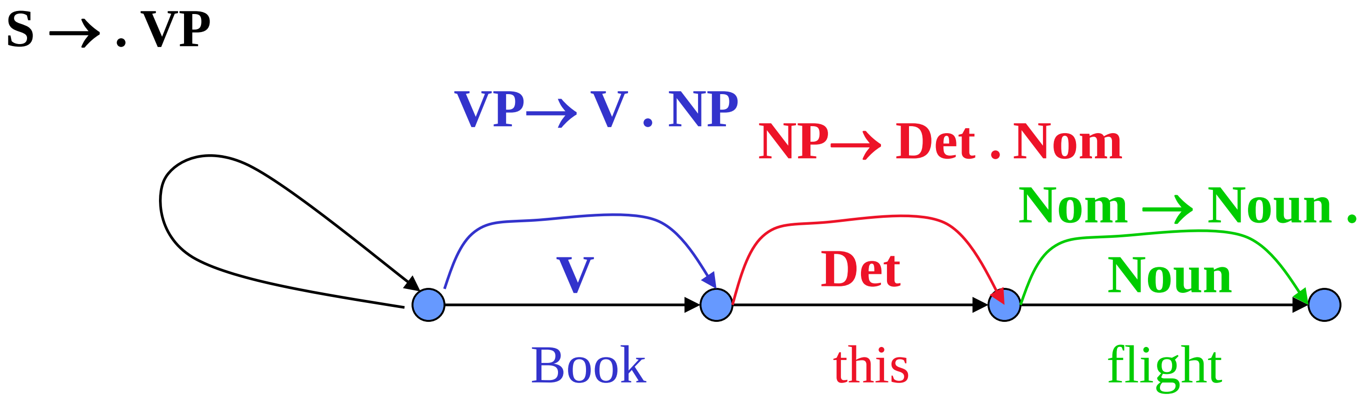


Chart - 3a



S → . VP

VP→ V . NP

NP→ Det . Nom

Nom → Noun .

Chart - 3b

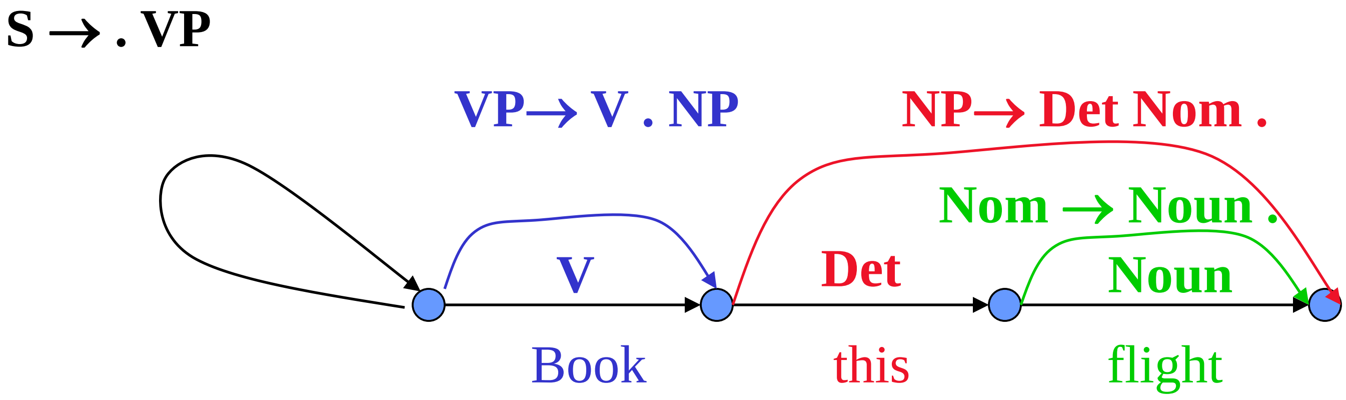


Chart - 3c

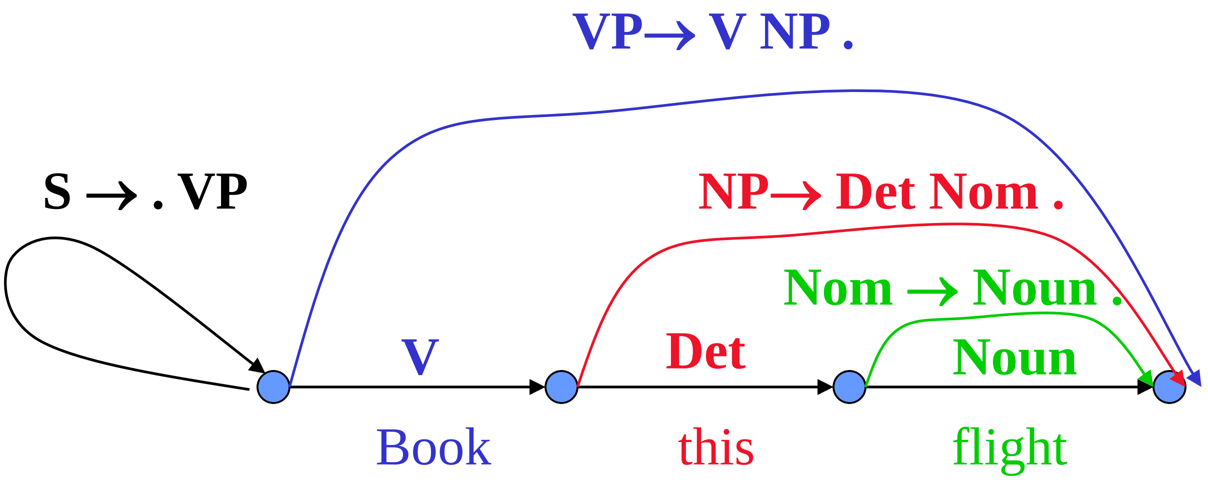
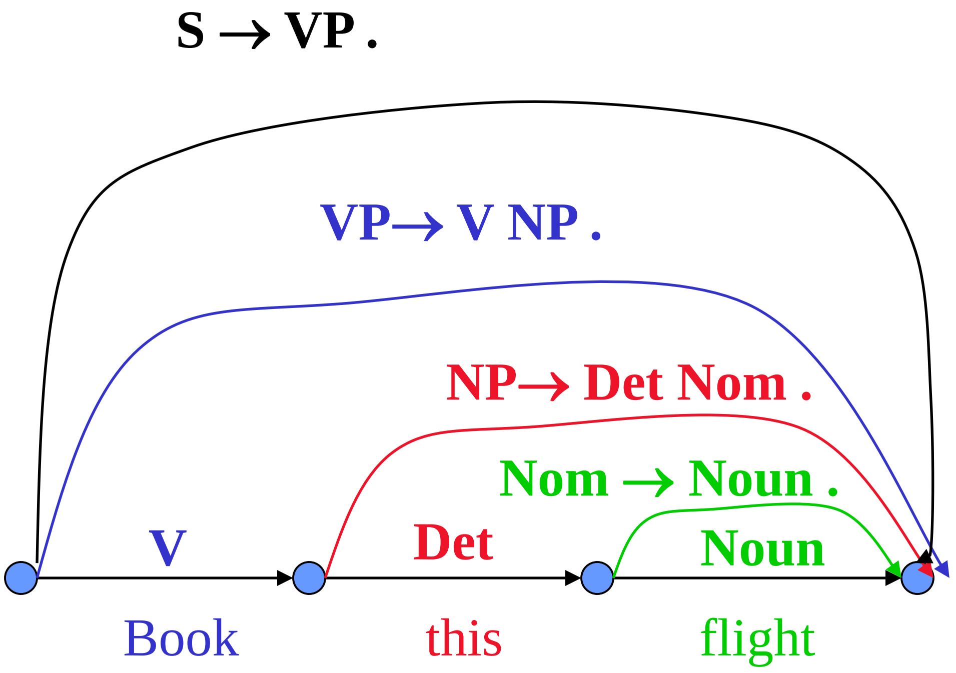


Chart - 3d



S → VP .

VP→ V NP .

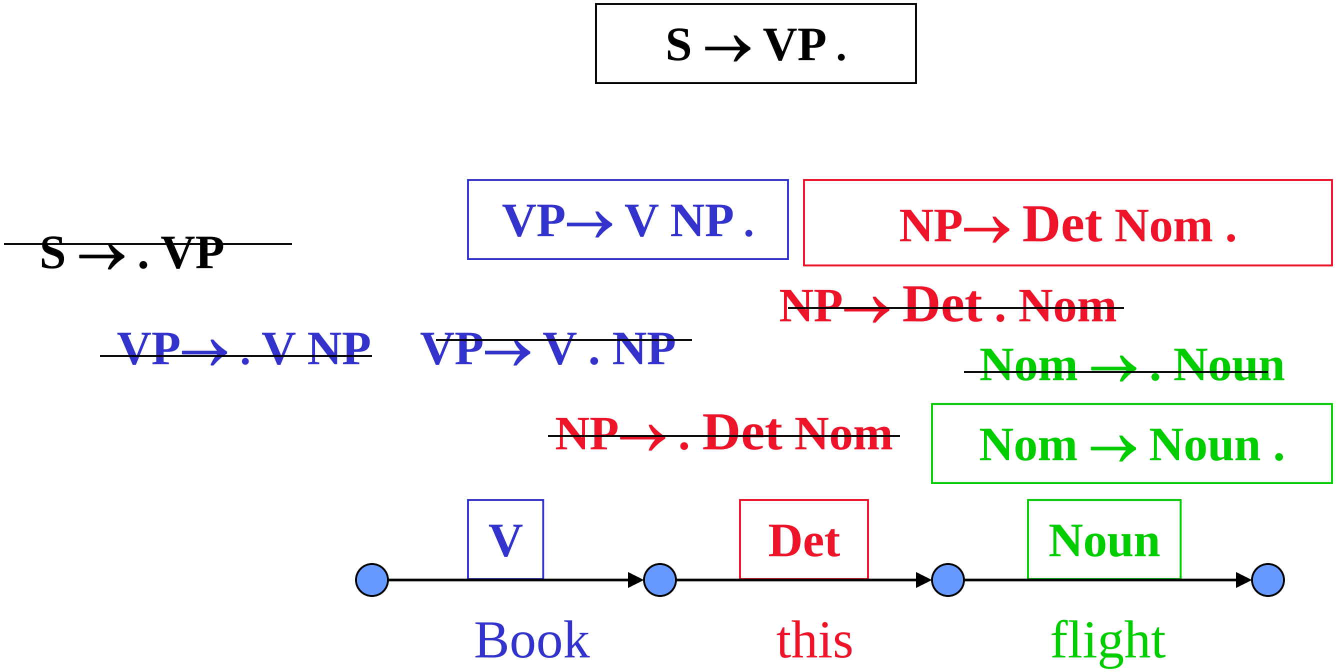
NP→ Det Nom .

Nom → Noun .

V Det Noun

Book this flight

Chart - All States



S → VP .

~~S → . VP~~

VP→ V NP .

NP→ Det Nom .

~~NP→ Det . Nom~~

~~VP→ . V NP~~

~~VP→ V . NP~~

~~Nom → . Noun~~

~~NP→ . Det Nom~~

Nom → Noun .

V

Det

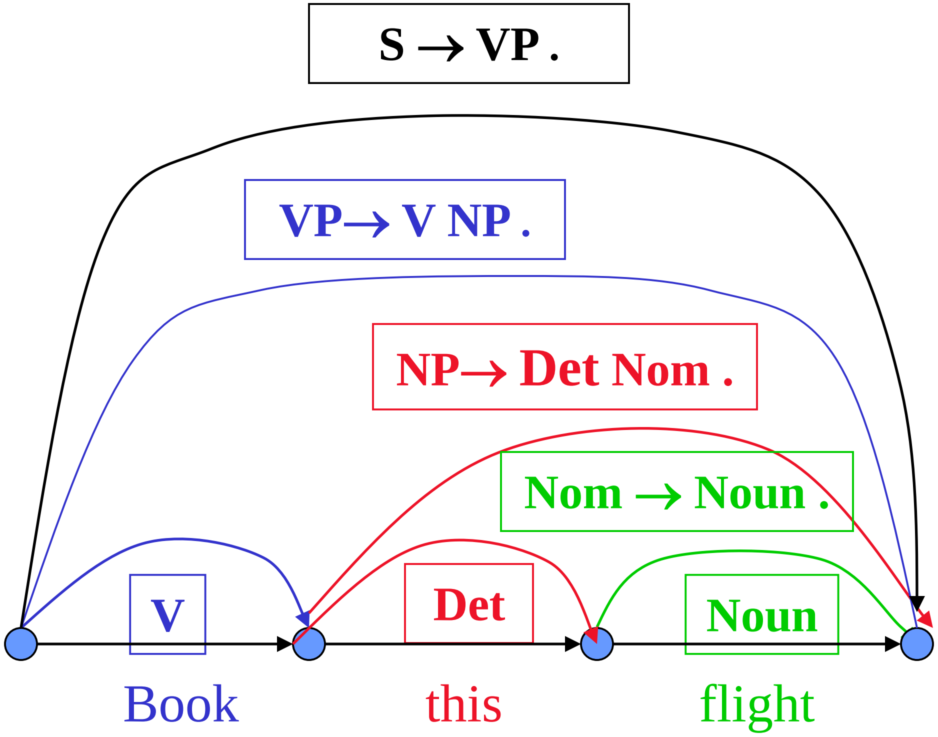
Noun

Book

this

flight

Chart - Final States



S → VP .

VP→ V NP .

NP→ Det Nom .

Nom → Noun .

V

Det

Noun

Book

this

flight

Chart 0 with two S- and two VP-Rules

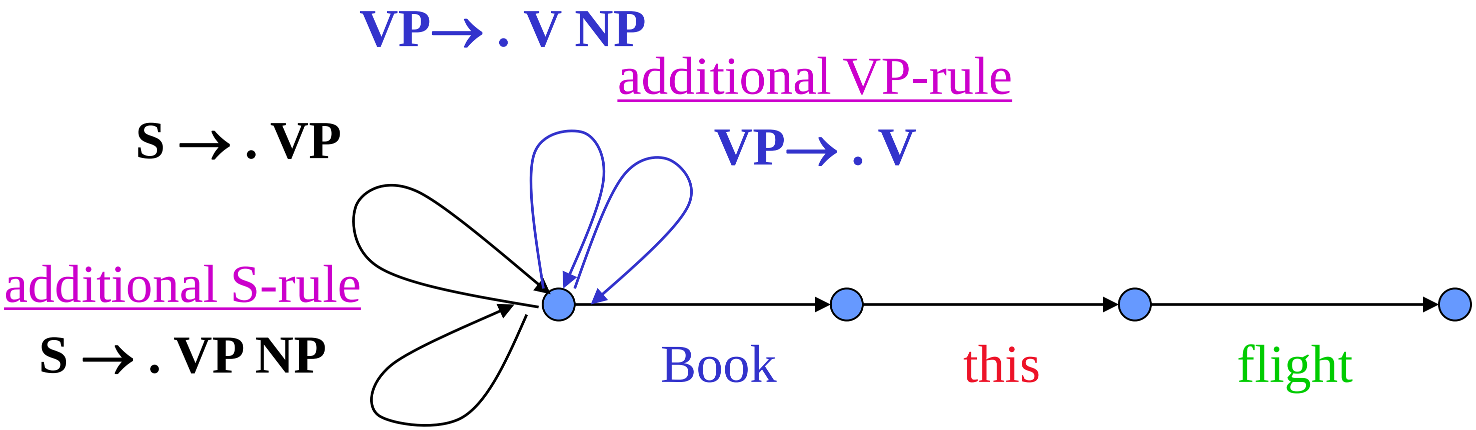


Chart 1a with two S- and two VP-Rules

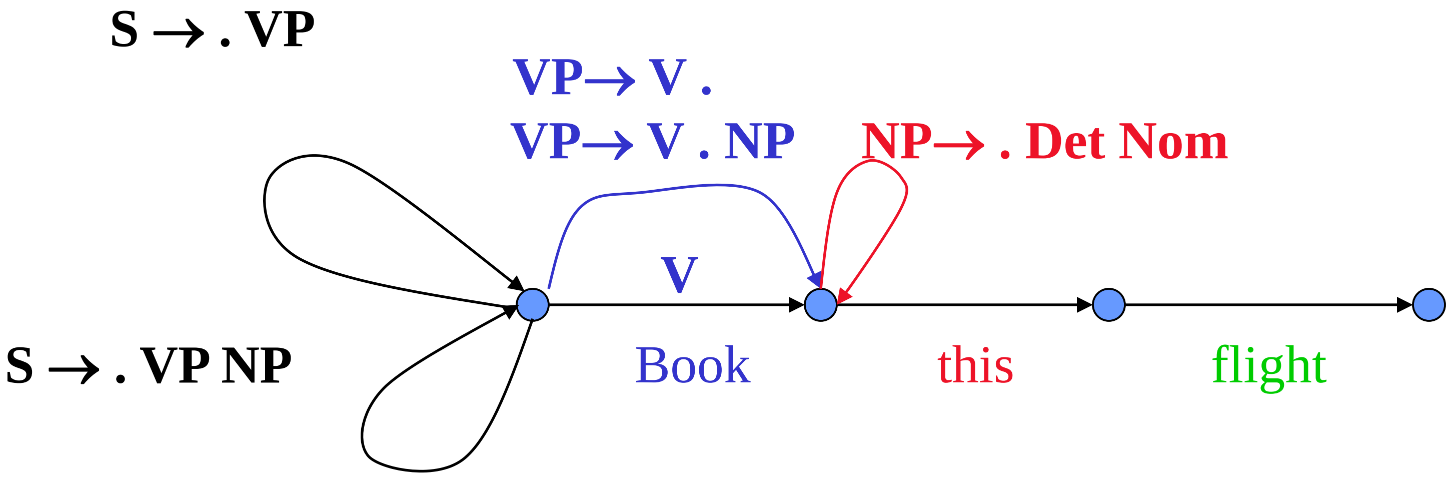


Chart 1b with two S- and two VP-Rules

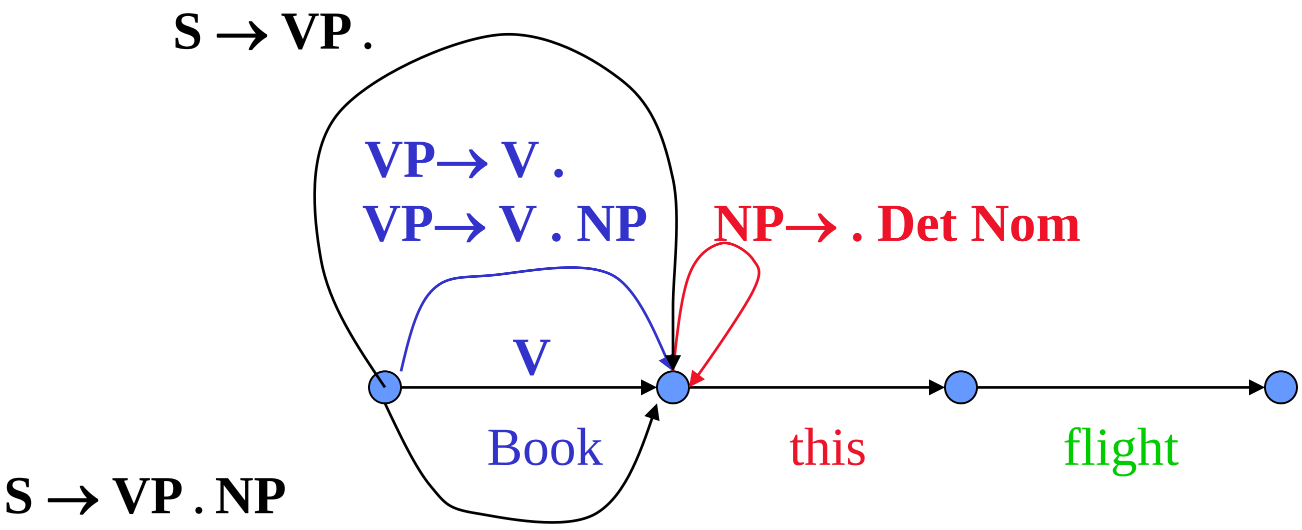
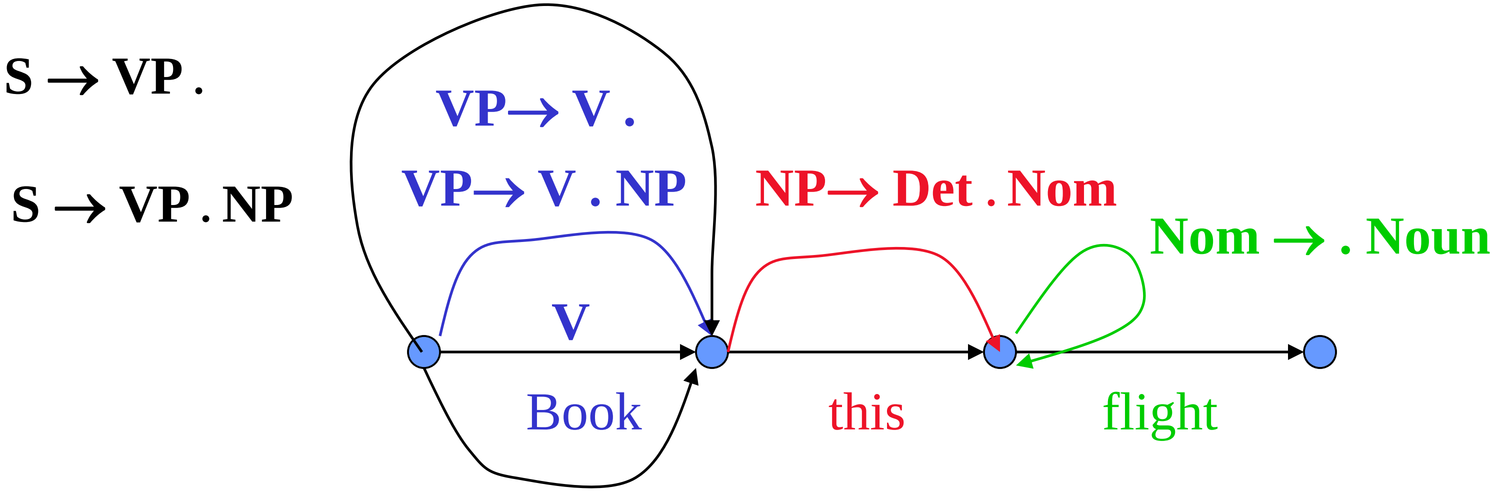


Chart 2 with two S- and two VP-Rules



S → VP .

S → VP . NP

VP→ V .

VP→ V . NP

NP→ Det . Nom

Nom → . Noun

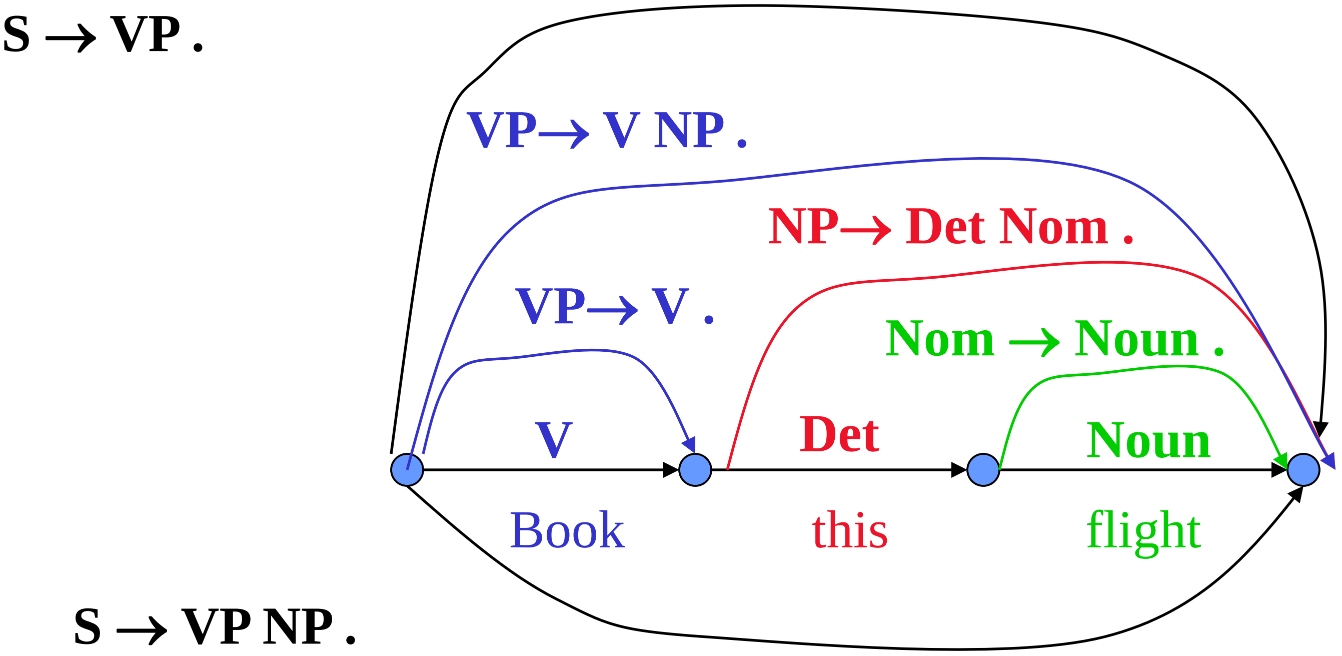
V

Book

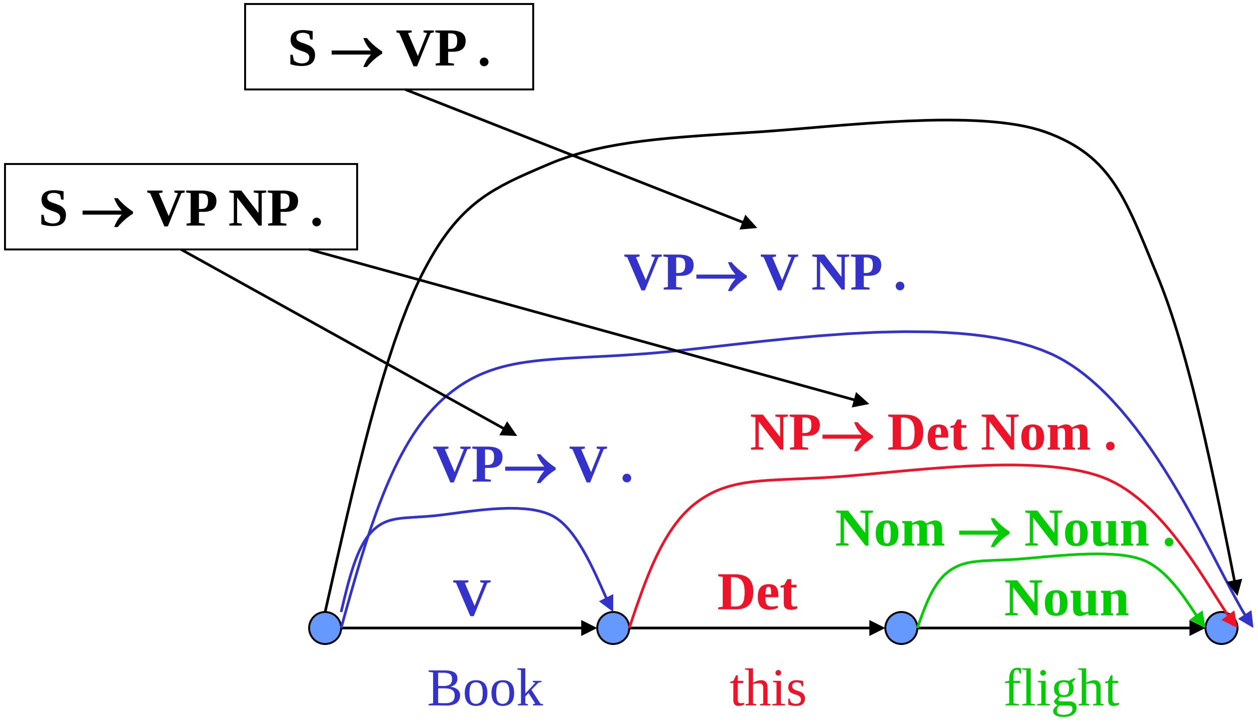
this

flight

Chart 3 with two S- and two VP-Rules



Final Chart - with two S-and two VP-Rules



S → VP .

S → VP NP .

VP→ V NP .

NP→ Det Nom .

VP→ V .

Nom → Noun .

V

Det

Noun

Book

this

flight

Earley Algorithm - Functions

predictor

generates new rules for partly recognized RHS with constituent right of • (top-down generation)

scanner

if word category (POS) is found right of the • , the Scanner reads the next input word and adds a rule for it to the chart (bottom-up mode)

completer

if rule is completely recognized (the • is far right), the recognition state of earlier rules in the chart advances: the • is moved over the recognized constituent (bottom-up recognition).

Earley-Algorithm

```
function EARLEY-PARSE(words, grammar) returns chart
    ENQUEUE(($\gamma \rightarrow \bullet S$, [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)
                else if INCOMPLETE?(state) and
                      NEXT-CAT(state) is a part of speech
                    then SCANNER(state)
                    else COMPLETER(state)
        end
    end
    return(chart)
```

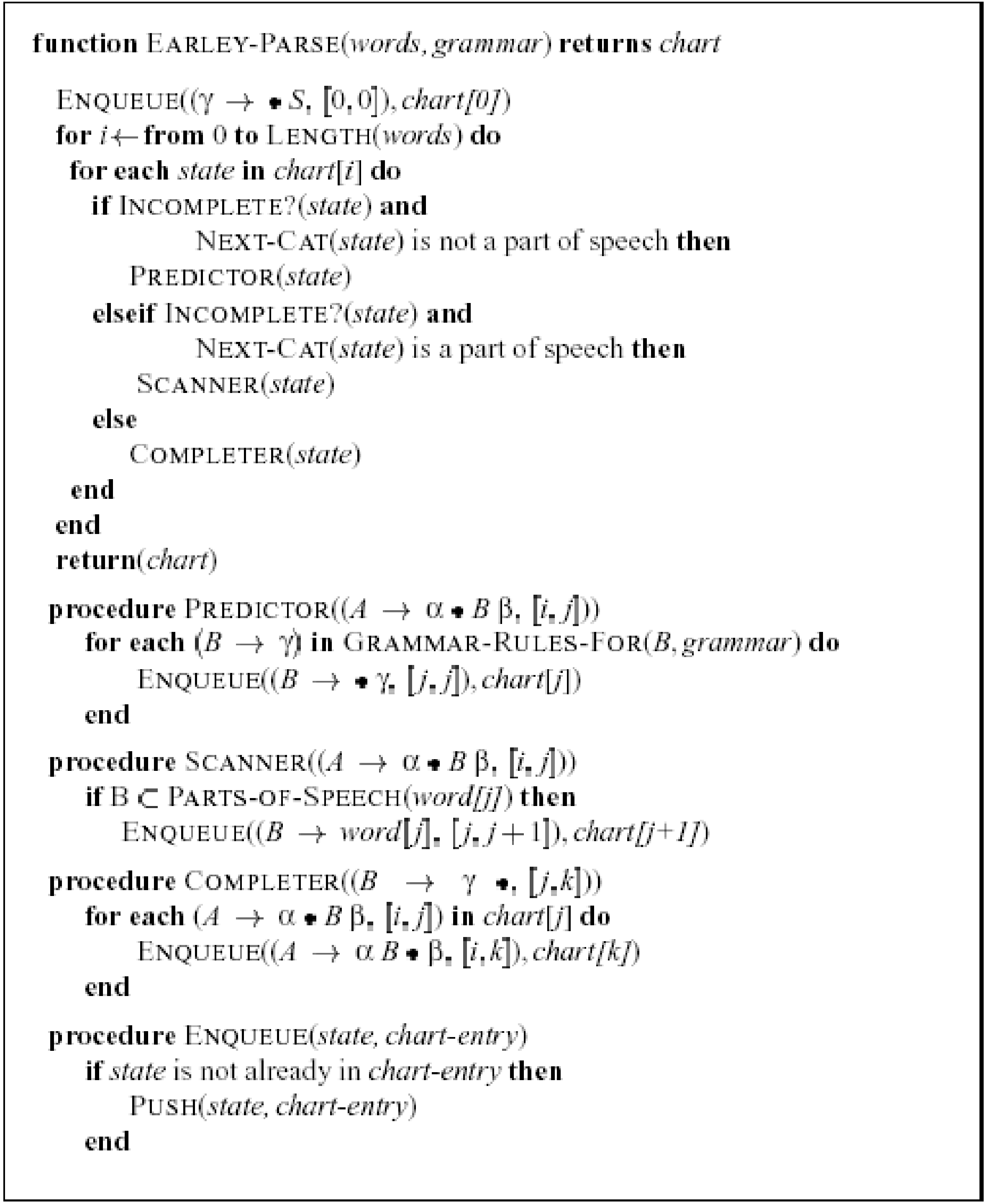
- continued on next slide -

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

procedure SCANNER (($A \rightarrow \alpha \bullet B \beta$ , [$i, j$]))
        if B $\in$ 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

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



```
function Earley-Parse(words, grammar) returns chart

  Enqueue(($\gamma \rightarrow \bullet S$, [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(($A \rightarrow \alpha \bullet B \beta$, [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 ⊂ 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

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

Chart[0]

S0	γ → • *S*	[0,0]	[]	Dummy start state
S1	*S* → • *NP VP*	[0,0]	[]	Predictor
S2	*NP* → • *Det NOMINAL*	[0,0]	[]	Predictor
S3	*NP* → • *Proper-Noun*	[0,0]	[]	Predictor
S4	*S* → • *Aux NP VP*	[0,0]	[]	Predictor
S5	*S* → • *VP*	[0,0]	[]	Predictor
S6	*VP* → • *Verb*	[0,0]	[]	Predictor
S7	*VP* → • *Verb NP*	[0,0]	[]	Predictor

Chart[1]

S8	*Verb* → *book* •	[0,1]	[]	Scanner
S9	*VP* → *Verb* •	[0,1]	[S8]	Completer
S10	*S* → *VP* •	[0,1]	[S9]	Completer
S11	*VP* → *Verb* • *NP*	[0,1]	[S8]	Completer
S12	*NP* → • *Det NOMINAL*	[1,1]	[]	Predictor
S13	*NP* → • *Proper-Noun*	[1,1]	[]	Predictor

Chart[2]

S14	*Det* → *that* •	[1,2]	[]	Scanner
S15	*NP* → *Det* • *NOMINAL*	[1,2]	[S14]	Completer
S16	*NOMINAL* → • *Noun*	[2,2]	[]	Predictor
S17	*NOMINAL* → • *Noun NOMINAL*	[2,2]	[]	Predictor

Chart[3]

S18	*Noun* → *flight* •	[2,3]	[]	Scanner
S19	*NOMINAL* → *Noun* •	[2,3]	[S18]	Completer
S20	*NOMINAL* → *Noun* • *NOMINAL*	[2,3]	[S18]	Completer
S21	*NP* → *Det NOMINAL* •	[1,3]	[S14,S19]	Completer
S22	*VP* → *Verb NP* •	[0,3]	[S8,S21]	Completer
S23	*S* → *VP* •	[0,3]	[S22]	Completer
S24	*NOMINAL* → • *Noun*	[3,3]	[]	Predictor
S25	*NOMINAL* → • *Noun NOMINAL*	[3,3]	[]	Predictor