9 Giugno 2006

Sintassi

Parsing

Marco Pennacchiotti

pennacchiotti@info.uniroma2.it

Tel. 06 7259 7717

Ing.dell'Informazione, *stanza P1B-03* (nuova ala Ing.Inf, primo piano)

Programma

- **Breve introduzione all'NLP**
  - Linguaggi Naturali e Linguaggi Formali
  - Complessità
- **Morfologia**
  - *Teoria:* Morfologia del Linguaggio Naturale
  - *Strumenti:* Automi e Trasduttori
  - *Analisi Morfologica:* con automi e trasduttori
- **Part of Speech Tagging**
  - *Teoria:* Le classi morfologiche
  - *Strumenti a Analisi:* modelli a regole e statistici
- **Sintassi**
  - *Teoria:* Sintassi del Linguaggio Naturale
  - *Strumenti:* CFG
  - *Analisi Sintattica:* parsing top-down, bottom-up, Early
- **Semantica**
  - Lexical Semantics
  - Sentence Semantics

Info

Analisi Sintattica: *definizioni*

- *Tre concetti fondamentali*
  - **Costituenti**
    - Come le parole si raggruppano in unità *(sintagmi)* e come queste unità si comportano
    - ES: *[il cane affamato] [ha rincorso] [il gatto] [nel giardino]*
  - **Relazioni Grammaticali**
    - Quali flessioni le parole devono avere per relazionarsi
    - ES: *il cane/SGMasc affamato/SGMasc*
  - **Sottocategorizzazione**
    - Quali costruzioni sintattiche caratterizzano un verbo
    - ES: [*il treno] [arriva] [a Roma]* (SOG vb PREP(a))

Constituency

Costituente: Gruppo di parole **consecutive** che si comportano come una singola unità sintattica. Generalmente costituenti dello stesso tipo occorrono in contesti sintattici simili (ad es. *Noun Phrases* precedono un verbo)

- Noun Phrases (Sintagmi Nominali) NP
  [la mamma] compra [il gelato]
- Prepositioanl Phrases (Sintagmi Preposizionali) PP
  la mamma [di Mario]
- Verb Phrases (Sintagmi Verbali) VP
  la mamma [compra il gelato] al bambino
- Adjectival Phrases (Sintagmi Aggettivali) AP
  lo strumento [utile a rivoltare la terra] è esposto al museo
- Sentences (Frasi) S
  [la mamma [che compra il gelato] ha un bambino]

Constituency Test

Come riconoscere un costituente ? *(Alcune regole)*

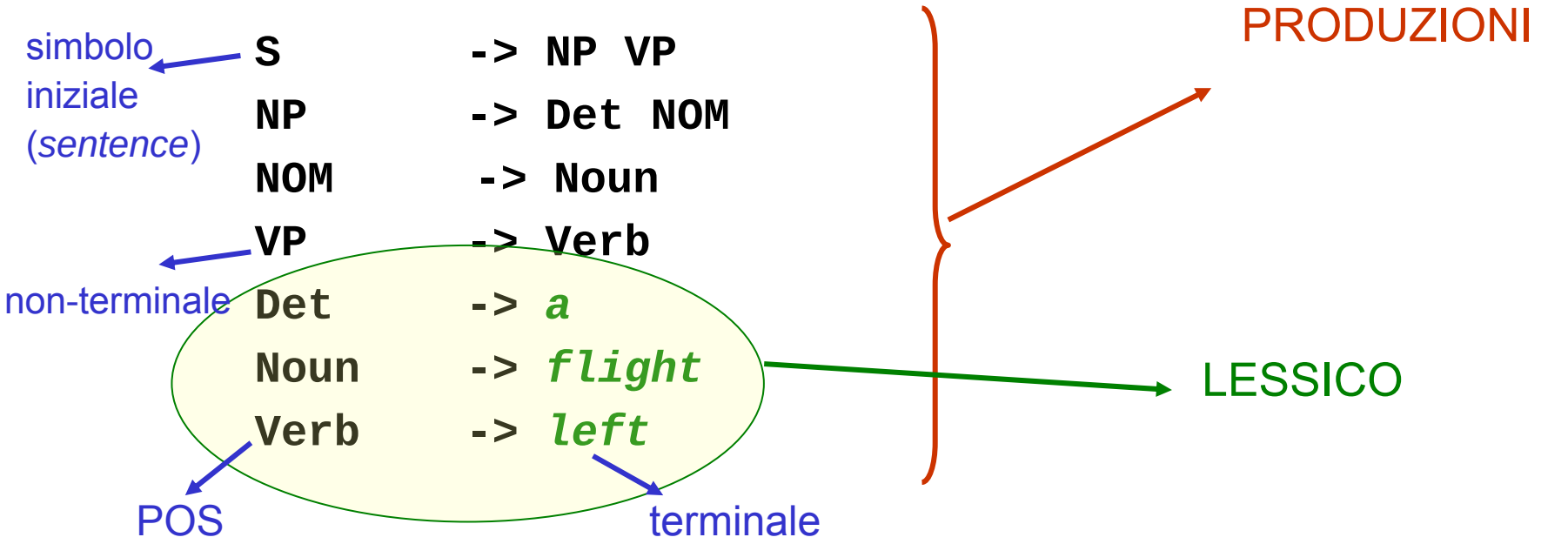
- Movimento: se è possibile effettuare un *movimento* del gruppo di parole nella frase, allora il gruppo è un costituente
  - preposing: un costituente si può spesso preporre, un non-costituente no
    EN: *I can't stand [your new friend] → [Your new friend], I can't stand*
    EN: *I can't stand [your new] friend → [Your new], I can't stand brother*
    IT: *non sopporto [il tuo nuovo amico] → **è** [il tuo nuovo amico] **che** non sopporto*
  - postposing: un costituente si può spesso postporre, un non-costituente no
    EN: *He tells [all the terrible problems] to her → He tells to her [all the terrible problems]*
    IT: *ho preso [il caffè] al bar → ho preso al bar [il caffè]*

- Isolabilità: se è possibile porre una domanda su un gruppo di parole nella frase, allora il gruppo è un costituente
  EN *You were reading [a nice book] → What were you reading? [A nice book]*
  EN *You were reading [a nice] book → What book were you reading? [A nice]*
  IT: *non sopporto [il tuo nuovo amico] → Chi non sopporti? [il tuo nuovo amico]*

- Coordinazione: Solo i costituenti si possono coordinare
  EN *You were reading [a nice book] and [the beautifol Black's novel]*
  IT: *non sopporto [il tuo nuovo amico] e [la signora dell'apparamento di fronte]*

CFG per la sintassi

- *Regole sintattiche* : modellate dalle produzioni
- *Elementi del Lessico* : alfabeto **Σ**
- *Costituenti* : alfabeto **N**



Σ = {a, flight, left}

N = {S,NP,VP,NOM,Det,Noun,Verb}

L = {*a flight left* }

CFG per la sintassi

-Generalmente le derivazioni vengono rappresentate con un **parse-tree** *(albero sintattico)*

-Un albero può rappresentare più derivazioni

Grammatica

```
S      -> NP VP
NP     -> Det NOM
NOM    -> Noun
VP     -> Verb
Det    -> a
Noun   -> flight
Verb   -> left
```

ESEMPIO

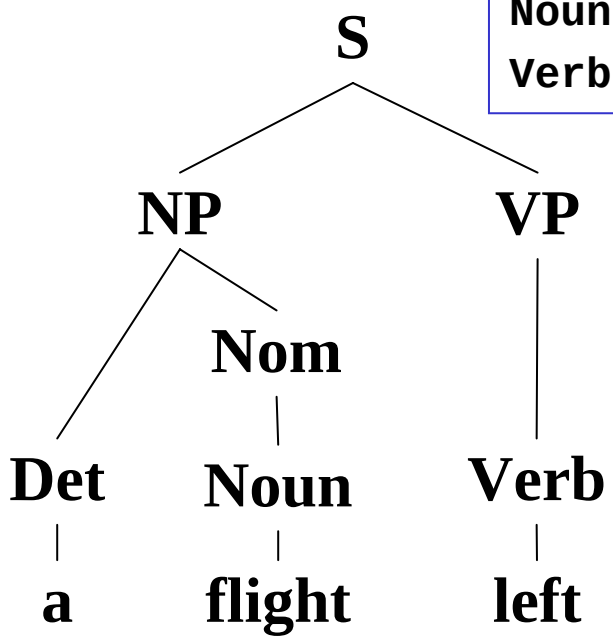
"a flight left"

DERIVAZIONE 1

```
S
NP VP
Det Nom VP
Det Nom Verb
a Nom Verb
a Noun Verb
a flight Verb
a flight left
```

DERIVAZIONE 2

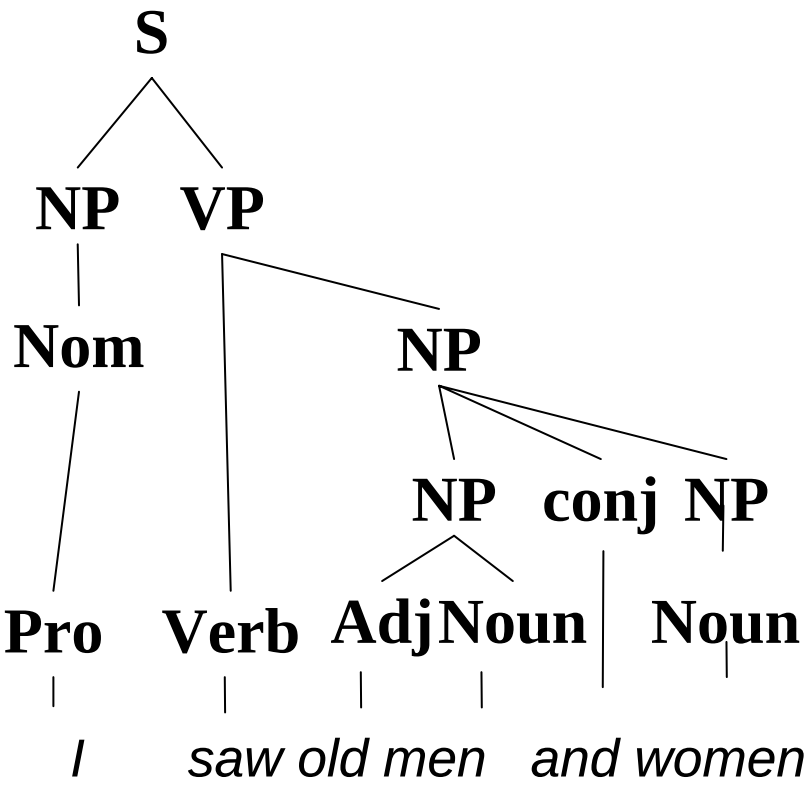
```
S
NP VP
NP Verb
NP left
Det Nom left
Det Noun left
Det flight left
a flight left
```

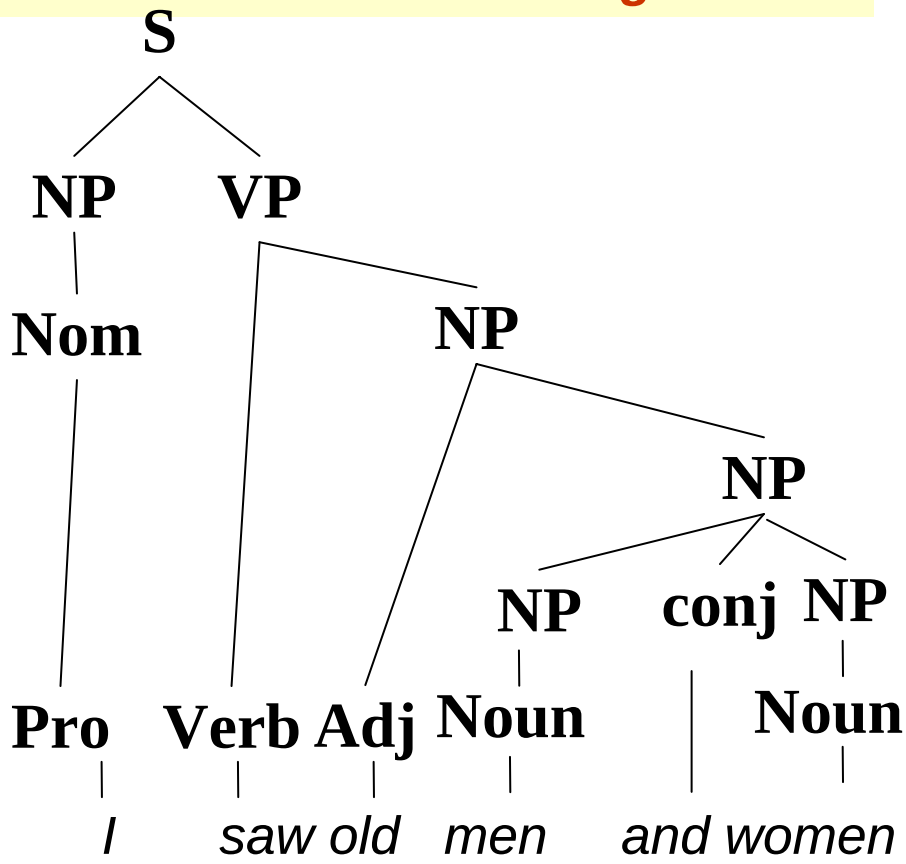


CFG: ambiguità

AMBIGUITA'

Se è possibile costruire due alberi diversi a partire dalla stessa frase e dalla stessa grammatica, la frase è *sintatticamente ambigua*





Sommario

- **Strumenti per la Sintassi**

- Introduzione
- Context-Free Grammar (*CFG*)
  - Definizione
  - CFG per la sintassi
  - Limiti e problemi
- **Parsing**
  - Parsing a costituenti
    - Parsing Top-Down
    - Parsing Bottom-Up
    - Parsing misto (*left-corner*)
    - Chart parsing
    - Programmazione dinamica
    - Algoritmo di Earley
  - Parsing a dipendenze
    - Cenni
    - Conversione

  - Chaos

Parsing

Cos'è il parsing?

QUALCHE DEFINZIONE:

- **in informatica:**

  Assegnazione di una descrizione strutturale a una stringa di caratteri

- **in linguistica:**

  Assegnazione di una descrizione sintattica ad una fras

- **in sistemi basati su conoscenza:**

  Assegnazione di un valore conoscitivo ad una espressione

PARSING SINTATTICO

- **Parsing effettuato su *frasi* e che produce una *struttura sintattica.***
- ***NOTA:* D'ora in avanti per *"parsing"* intenderemo *"parsing sintattico"***

Parsing

PARSING: *definizione computazionale*

Processo di riconoscimento di una stringa e di assegnazione ad essa di una struttura sintattica corretta, in base ad una specifica grammatica.

Il parsing riconosce tutte le stringhe del linguaggio definito dalla grammatica.

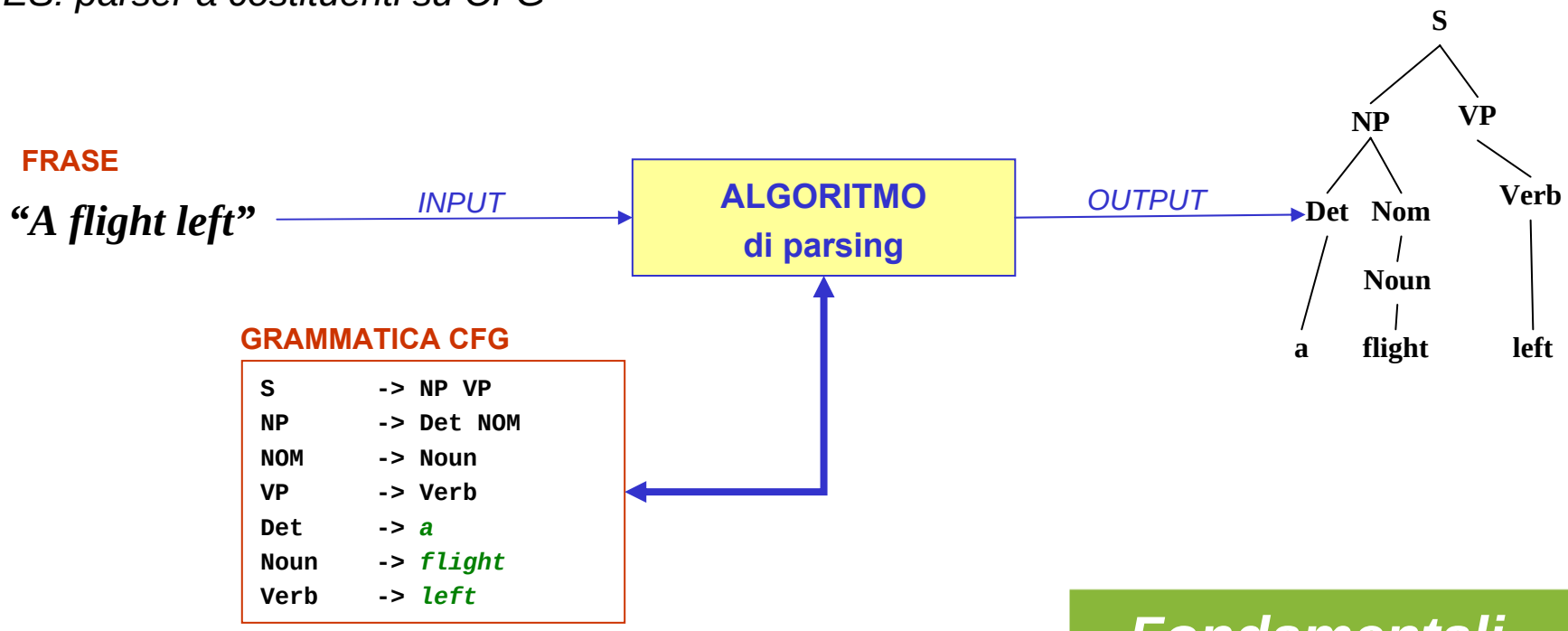
PRE-REQUISITI:

- **Formalismo grammaticale** (es. CFG)
  - Un formalismo deve essere abbastanza espressivo per modellare il linguaggio (tipo di espressività nella gerarchia di Chomsky)
- **Grammatica** (es. S→NP VP ……)
  - Una grammatica deve essere adeguata al liguaggio (deve rappresentare i fenomeni sintattici del linguaggio)
- **Algoritmo** (es. top-down)
  - Un algoritmo deve essere efficiente nel riconoscere la stringa in input e assegnargli una struttura

Parsing e grammatica

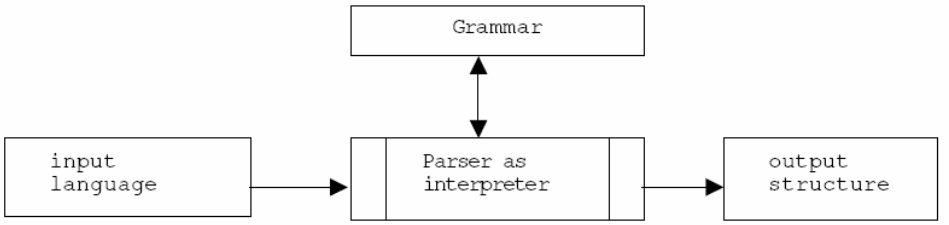
Parsing e grammatica sono due cose diverse!

- Una **grammatica** è un *modello* dichiarativo che definisce un linguaggio
- Il **parsing** è un *processo* di assegnazione di una struttura ad una stringa in base ad una grammatica
- *ES: parser a costituenti su CFG*



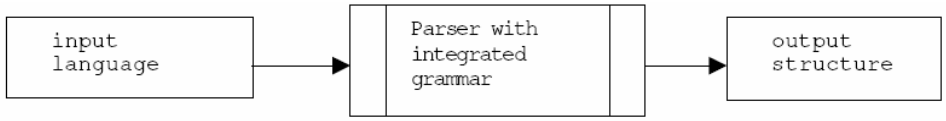
Parsing e grammatica

Implementazioni possibili tra *grammatica* e *parsing* (*da Hellwig,2003*):



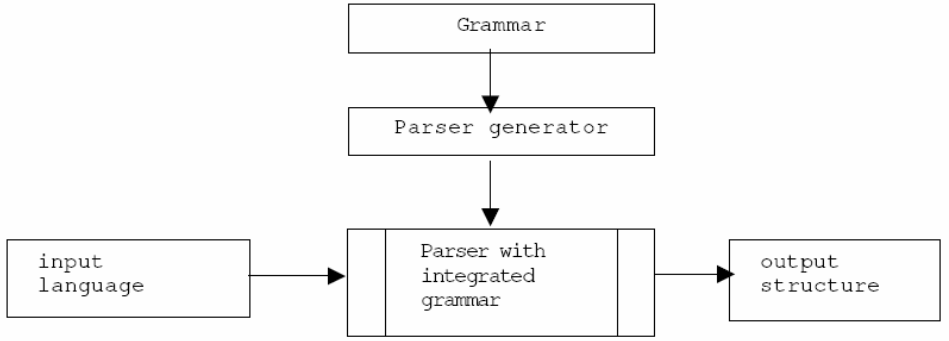
Interpreting parser

- Grammatica e algoritmo di parsing sono completamente separati



Procedural parser

- La grammatica è integrata nell'algoritmo di parsing (es. proceduralmente con un FST, ATN, ecc.)
- La grammatica definisce la procedura di parsing



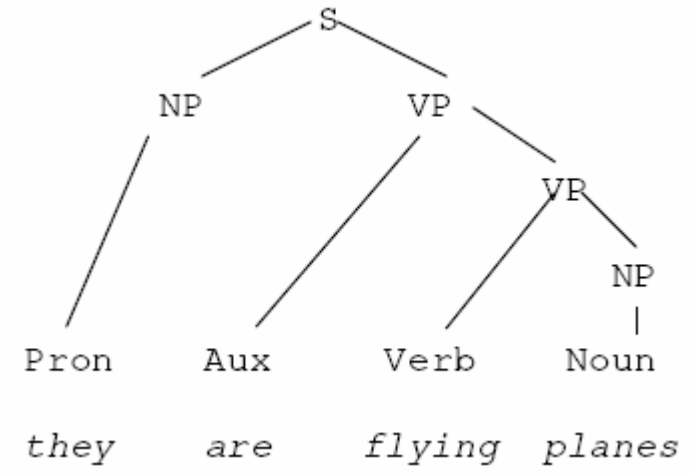
Compiled parser

- Grammatica e algoritmo sono completamente separati
- Prima dell'esecuzione la grammatica è integrata nell'algoritmo di parsing

Costituenti VS Dipendenze

Esistono due classi principali di parser, in base alla struttura prodotta in output:

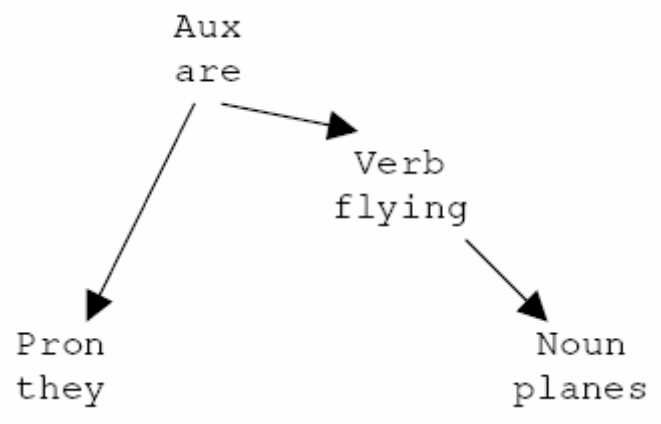
parser a costituenti



NODI: costituenti, parole

ARCHI: composizione di costituenti

parser a dipendenze



NODI: parole

ARCHI: dipendenze sintagmatiche tra una parola e un sintagma dominato da una parola

Costituenti VS Dipendenze

Esistono due classi principali di parser, in base alla struttura prodotta in output:

parser a costituenti

ORIGINI: stoicismo (logica)

OGGI:

Noam Chomsky (1950)

Charniak parser (1997)

Collins parser (1999)

- Il focus è sui costituenti facenti parte di una frase

- La struttura prodotta è ricorsiva

parser a dipendenze

ORIGINI: grammatici Latina e Arabica (linguaggi basati fortemente sulle dipendenze!)

OGGI:

Desnière (1959) Hays (1960)

Link Grammar (Sleator Temperley,1993)

Constraint Grammar (Karlsson et al.,1995)

- Il focus è sulle relazioni grammaticali tra parole, e sul ruolo di *head* e *dependant*

- La struttura prodotta è una rete di relazioni

Gran parte dei parser oggi disponibili sono a costituenti

Fondamentali

Sommario

- **Strumenti per la Sintassi**

- Introduzione
- Context-Free Grammar (*CFG*)
  - Definizione
  - CFG per la sintassi
  - Limiti e problemi
- **Parsing**
  - **Parsing a costituenti**
    - **Parsing Top-Down**
    - **Parsing Bottom-Up**
    - **Parsing misto (*left-corner*)**
    - **Chart parsing**
    - **Programmazione dinamica**
    - **Algoritmo di Earley**
  - Parsing a dipendenze
    - Cenni
    - Conversione
  - Chaos

Parsing a costituenti

PARSING A COSTITUENTI

Processo di riconoscimento di una stringa e di assegnazione ad essa di una struttura sintattica corretta, ovvero un parse-tree:

- che abbia *S* come radice
- che abbia tutti e soli gli elementi della stringa come foglie

PUNTI FONDAMENTALI:

- Le CFG sono un modello dichiarativo
  - quindi non specificano come effettuare il parsing (creare l'albero)
  - al contrario, nella morfologia, gli FST sono modelli procedurali
- Le DCG possono implementare limitatamente il parsing
  - Solamente ricerca top-down depth-first
- Il parsing sintattico può essere visto come ***problema di ricerca di alberi corretti***
  - Come già avviene in morfologia con gli FST
  - Possono essere quindi usate diverse strategie …

Fondamentali

Parsing a costituenti

STRATEGIE DI PARSING

- **Top-Down VS Bottom-up**
  - La ricerca può essere effettuata partendo da S o dalle parole
  - Razionalisti VS Empiristi
- **Depth-First VS Breadth-First**
  - Si può andare in profondità su un ramo dell'albero (finchè non si raggiunge una foglia) o in ampiezza (sviluppo di tutti i rami di un livello)
  - Depth-First è generalmente utilizzato nei Top-Down, mentre Breadth-First nei Bottom-up
- **Left-To-Right VS Right-To-Left**
  - La RHS della produzione può essere letta da sinistra verso destra o viceversa
- **Agenda**
  - Come per gli FST, si può utilizzare una agenda per memorizzare gli stati della ricerca già esaminati
- **Dynamic Programming**
  - Suddividere il problema della ricerca in sotto-problemi e rircordare le soluzioni parziali

Parsing

PUNTO DI PARTENZA

- La stringa in input è memorizzata in un buffer
- Non è stato effettuato nessun POS-tagging
- Non interessa la struttura morfologica (agreement)
- Tutte le parole della stringa sono nel lessico della grammatica
- La ricerca :
  - procede nello spazio di tutti i possibili parse-tree
  - termina quando sono stati trovati tutti i parse-tree corretti
  - Un parse-tree corretto ha radice S e come foglie gli elementi in input

Parsing

PUNTO DI PARTENZA

Input: *Book that flight*

```
S        -> NP VP
S        -> Aux NP VP
S        -> VP
NP       -> Det Nom
NP       -> PropN
Nom      -> Noun
Nom      -> Noun Nom
VP       -> Verb
VP       -> Verb NP

Det      -> that | this | a
Noun     -> book | flight | meal | money
Verb     -> book |include | prefer
Aux     -> does
PropN    -> Houston | TWA
```

Strategia Top-Down

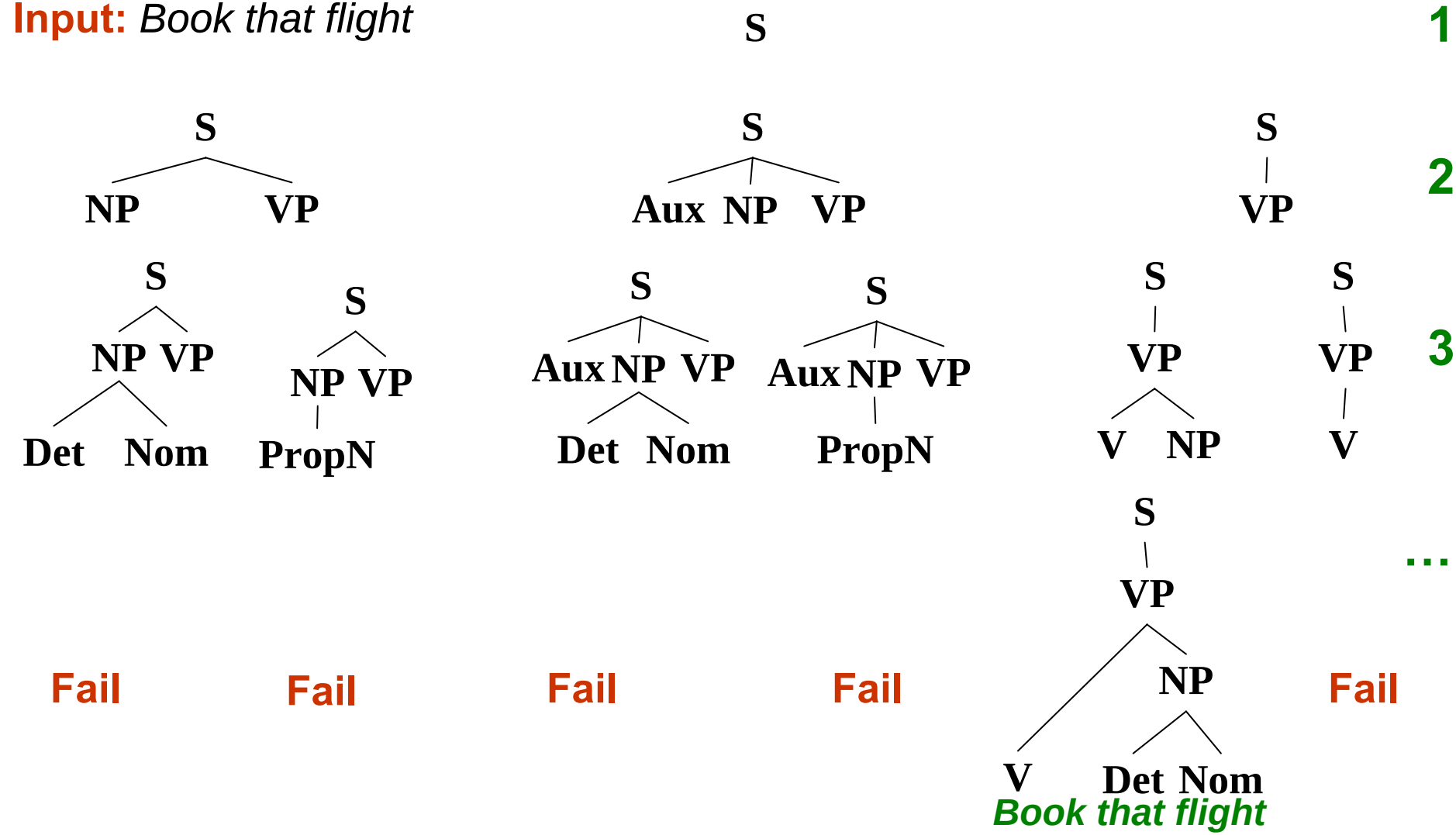
STRATEGIA DI RICERCA

- Cerca gli alberi partendo dalla radice *S*
- Applica le regole dall'alto verso il basso (e da sinistra a destra, *espansione*) sino a raggiungere le categoria POS (ultimi rami non-terminali)
- Ha successo se espandendo le POS nel lessico si ottiene la stringa
- Generalmente utilizza strategia *Depth-First*
- Possibili ricerche:
  - Depth-First
  - Breadth-First
  - **Parallelo**
- Possibili direzioni di lettura:
  - **Left-To-Right**
  - Right-To-Left

Strategia Top-Down, parall., Left-to-Right

Input: *Book that flight*

1



Strategia Top-Down

VANTAGGI

- La ricerca genera sempre alberi consistenti con la radice
  - Tutti gli alberi partono dalla *S*
  - Non vengono mai generati alberi che non terminano in *S*

SVANTAGGI

- La ricerca è *cieca in basso*: non è guidata dalla frase in input
  - La frase in input viene esaminata solo alla fine
  - Vengono espansi fino alla fine tutti gli alberi, anche quelli che non hanno speranza di catturare la stringa

Strategia Bottom-Up

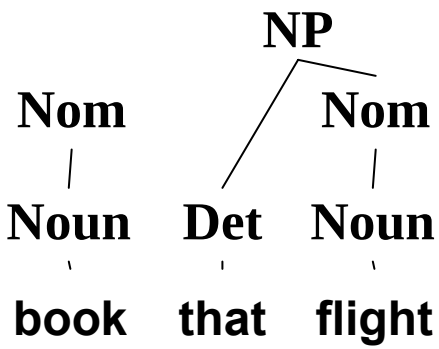
STRATEGIA DI RICERCA

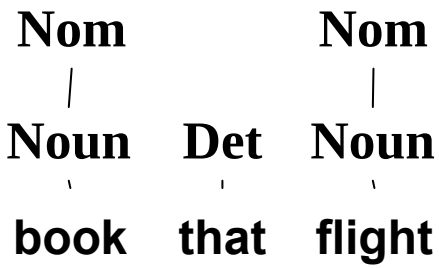
- Cerca gli alberi partendo dalle foglie (frase in input)
- Applica il lessico , e quindi le altre regole dal basso verso l'alto (e da destra a sinistra, *riduzione*) sino a raggiungere le categorie più generali
- Ha successo se espandendo ottiene come unico nodo finale la *S*, che copra la frase in input
- Generalmente utilizza strategia *Bottom-Up*
- Possibili ricerche:
  - Depth-First
  - Breadth-First
  - **Parallelo**
- Possibili direzioni di lettura:
  - **Left-To-Right**
  - Right-To-Left

Strategia Bottom-Up

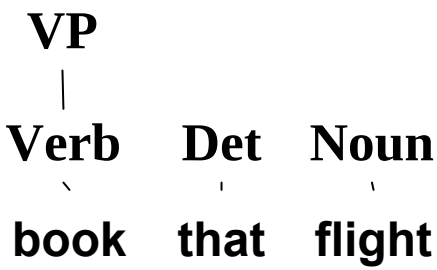
Input: *Book that flight*

Fail

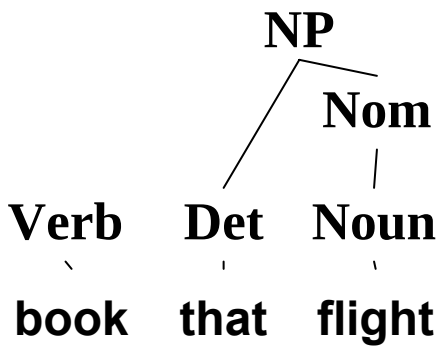




Fail

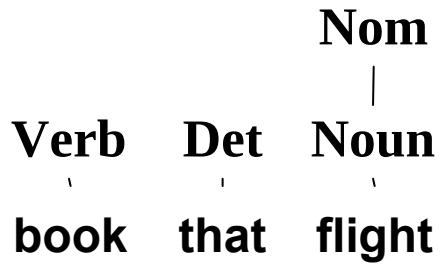


OK!



…

3



2

1

Ricerca

Strategia Bottom-Up

VANTAGGI

- La ricerca genera sempre alberi consistenti con la frase in input
  - Tutti gli alberi partono dalle parole della frase
  - Non vengono mai generati alberi che non terminano con la frase

SVANTAGGI

- La ricerca è *cieca in alto*: non è guidata dalla radice *S*
  - Solo alla fine verifica se l'albero creato termina in *S*
  - Vengono espansi fino alla fine tutti gli alberi, anche quelli che non hanno speranza di avere come radice *S*

Parsing: ricerca

Data la seguente grammatica presentata nella lezione, scrivere lo sviluppo dell'albero di parsing per le seguenti frasi:

"Book that flight"

"Does the TWA flight include a meal ?"

"Does the flight from Houston include a meal ?"

Utilizzando un parser top-down, depth-first, left-to-right

ESERCIZIO PARS 1

Strategia mista (LEFT-CORNER)

Top-Down + Bottom-Up

- Utilizzare i vantaggi di entrambe le strategie
  - *Top-Down:* tutti gli alberi partono dalla *S*
  - *Bottom-Up:* alberi consistenti con la frase in input
- Quindi, un parser misto dovrebbe:
  - evitare di espandere alberi che non convergono su S
  - evitare di espandere alberi che non terminano sulla frase

STRATEGIA DI RICERCA

- Utilizzare la strategia Top-Down come controllo è Bottom-Up come filtro
- Ricerca: *depth-first*
- Direzione di lettura: *Left-to-Right*
- Ordine di lettura delle regole: *dall'alto verso il basso*

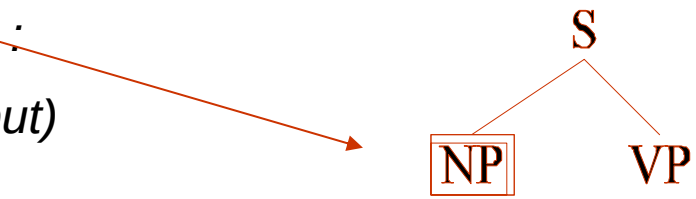
Parser top-down,depth-first,left-to-right

- E’ un parser basilare che può essere utilizzato come controllo in un parser left-corner
- A livello implementativo gli algoritmi di parsing utilizzano un’*agenda:*

Agenda

- Stack utilizzata per memorizzare i ***search-states*** :

Search-state = (partial-tree , next-input)



- Ogni search-state memorizza il parse-tree che si sta analizzando e l’indice della posizione della frase in input cui è giunta l’analisi
- Strategia **LIFO** per implementare la ricerca Depth-First

[Does]

Parser top-down,depth-first,left-to-right

ALGORITMO DI RICERCA

Agenda ← (S-tree,begin-of-input)

currentSearchState ← *POP* (agenda)

Loop (until success)

if *CAT*(*NODE-TO-EXAPAND*(currentSearchState) is POS)

if *CAT*(node-to-expand) *is_in POS*(*CURRENT-INPUT*(current-search-state))

PUSH(*APPLY-LEXICAL-RULE*(current-sarch-state), agenda)

else ***return*** REJECT

else PUSH(APPLY-RULES(currentSearcState,grammar),agenda)

if (agenda is_empty)

return REJECT

else current-search-state← NEXT(agenda)

Parser top-down,depth-first,left-to-right

ESEMPIO

GRAMMATICA:

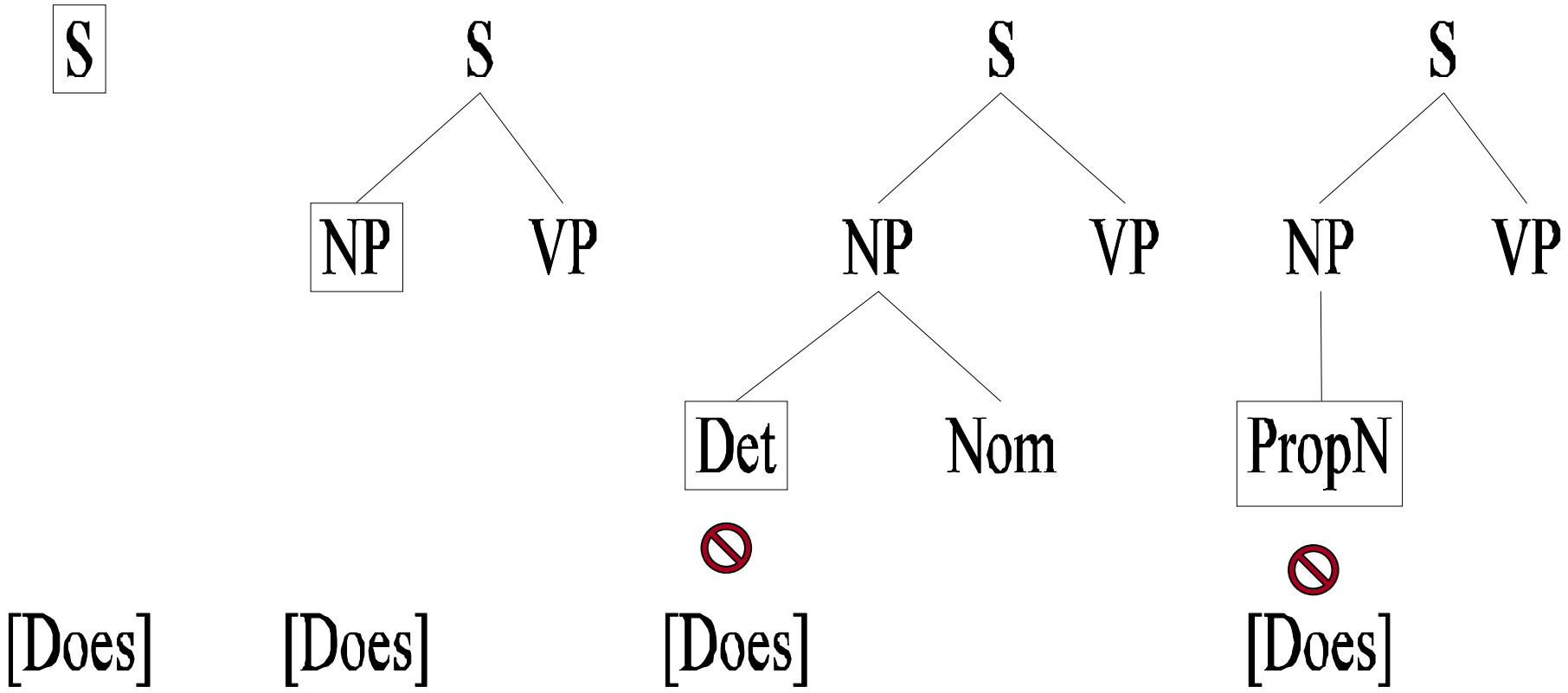
```
S       -> NP VP
S       -> Aux NP VP
S       -> VP
NP      -> Det Nom
NP      -> PropN
Nom     -> Noun
Nom     -> Noun Nom
VP      -> Verb
VP      -> Verb NP

Det     -> that | this | a
Noun    -> book | flight | meal | money
Verb    -> book |include | prefer
Aux    -> does
PropN   -> Houston | TWA
```

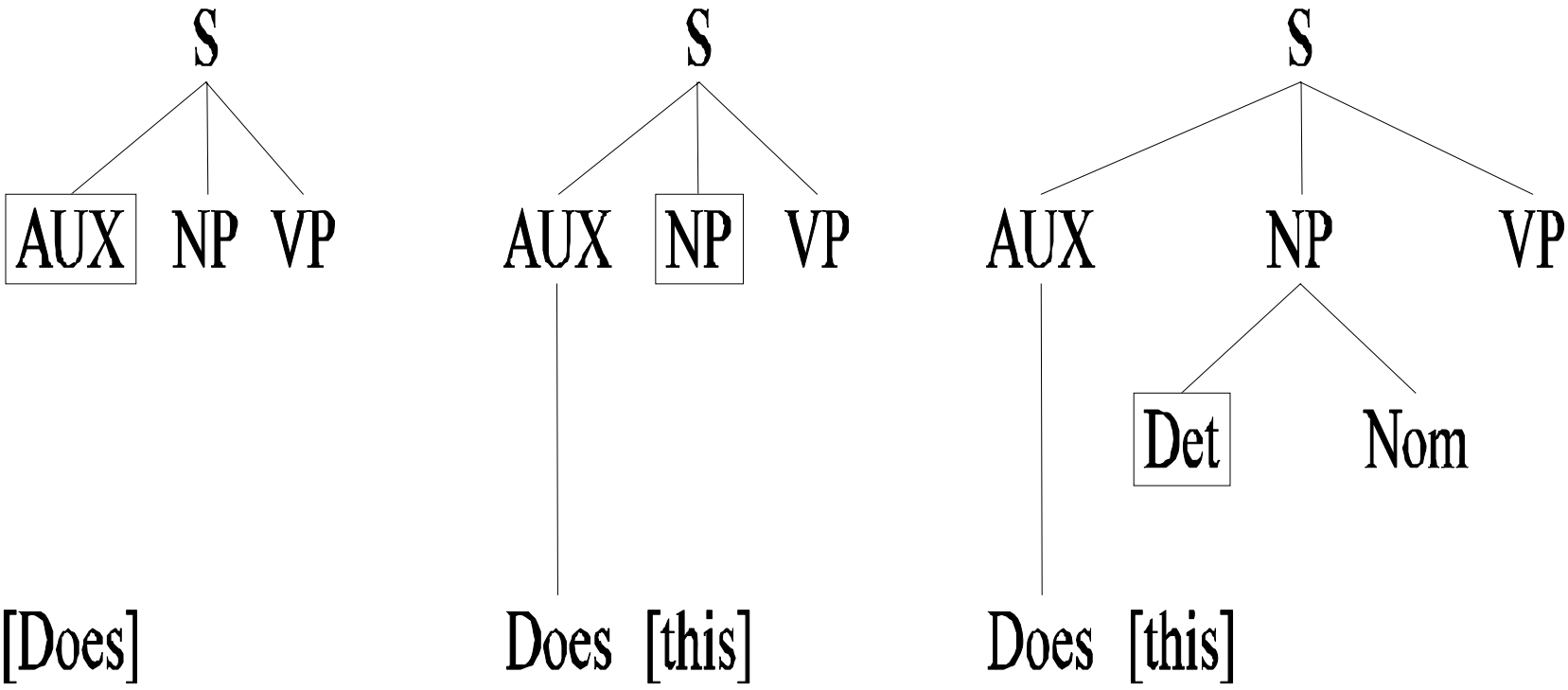
FRASE:

“Does this flight include a meal?”

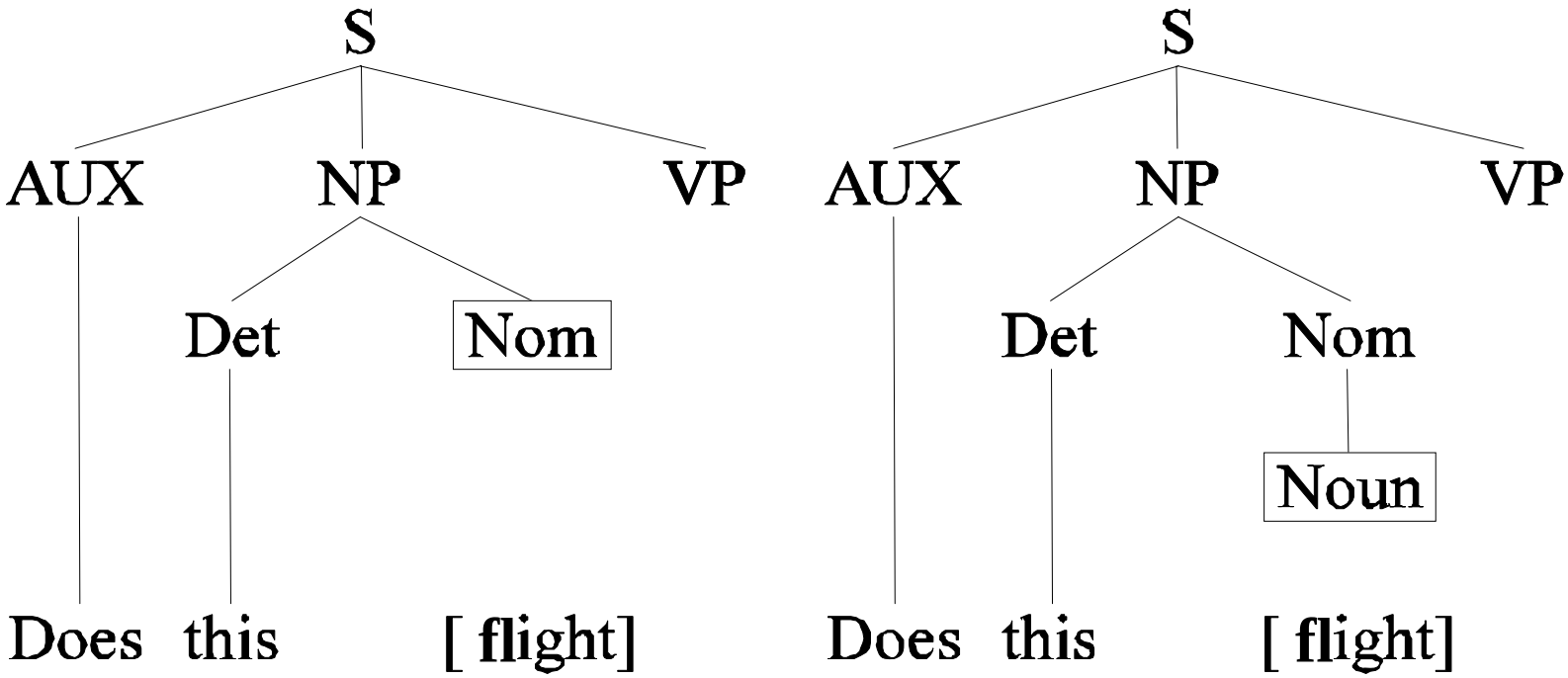
Parser top-down,depth-first,left-to-right



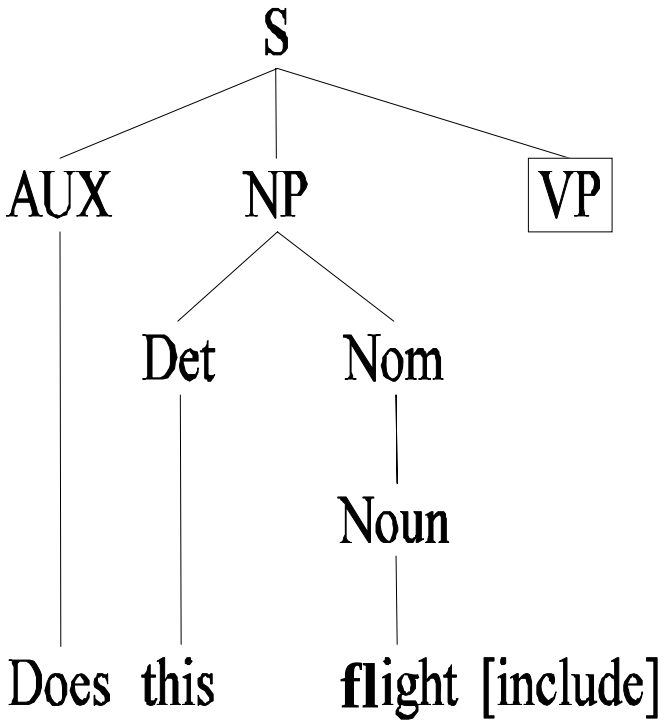
Parser top-down,depth-first,left-to-right

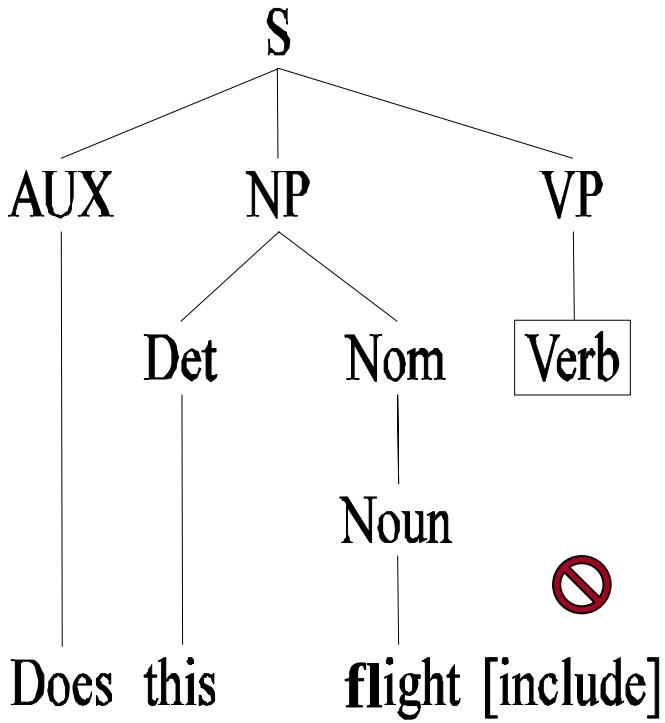


Parser top-down,depth-first,left-to-right

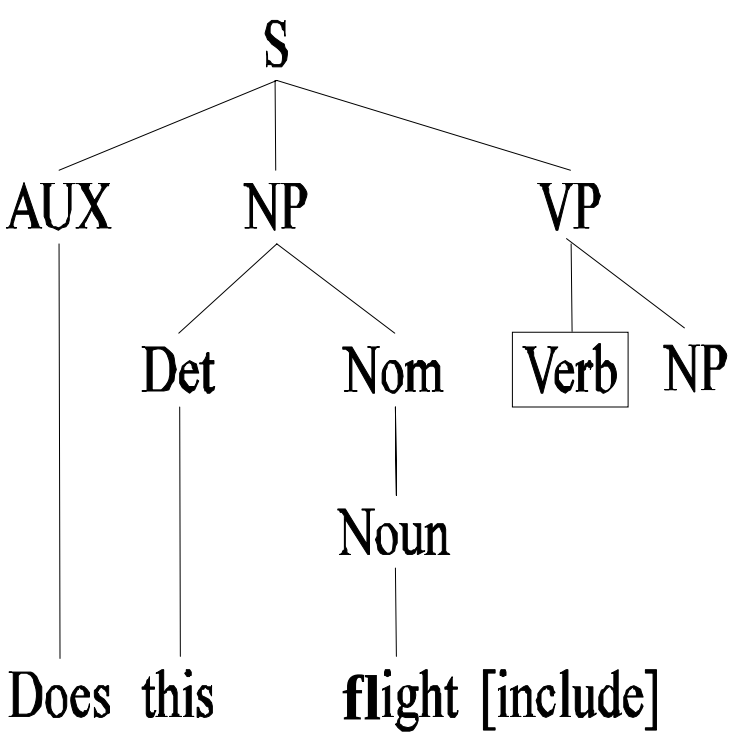


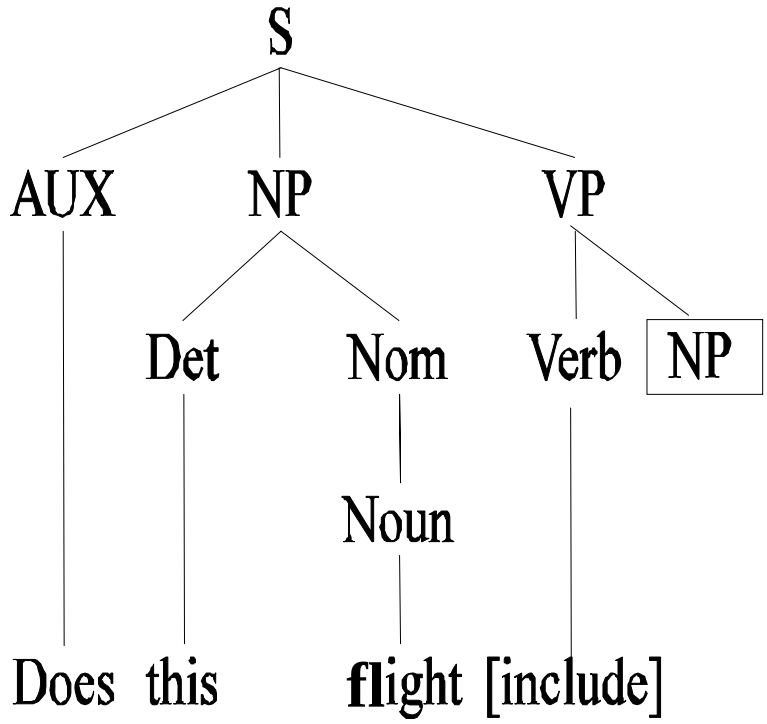
Parser top-down,depth-first,left-to-right



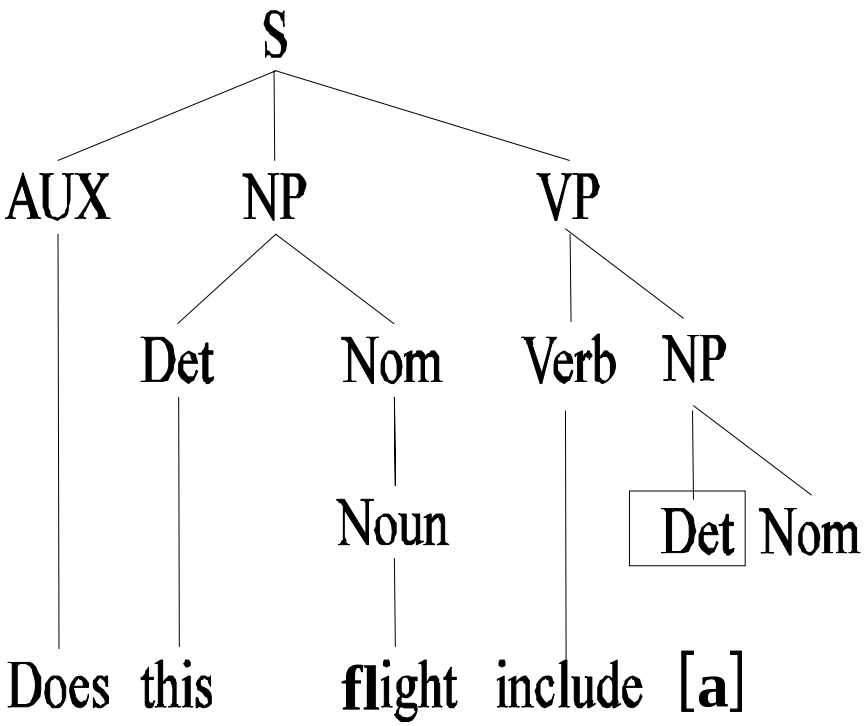


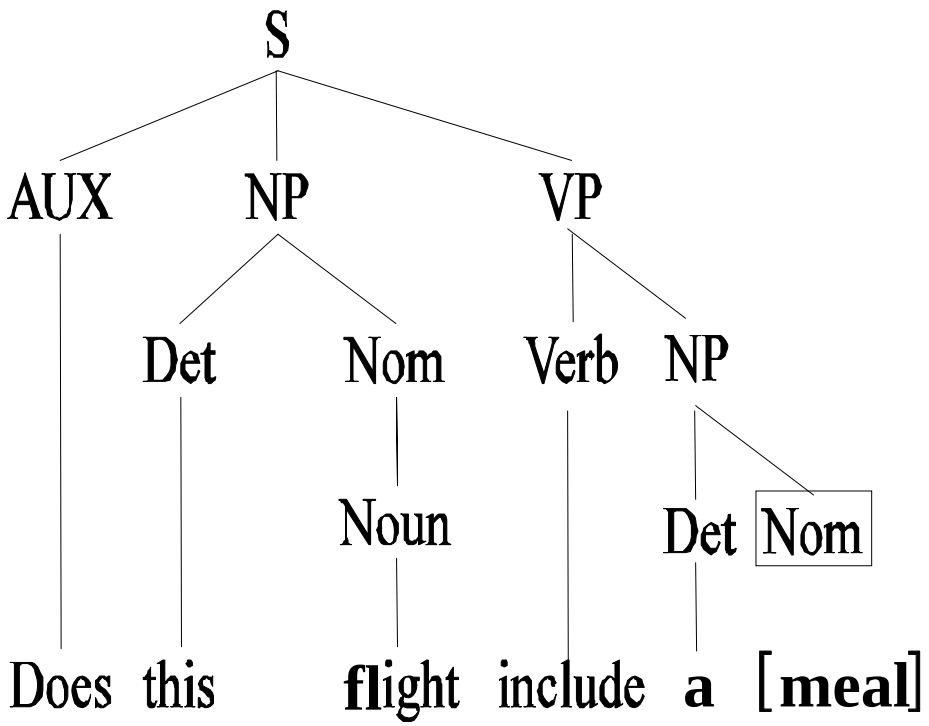
Parser top-down,depth-first,left-to-right



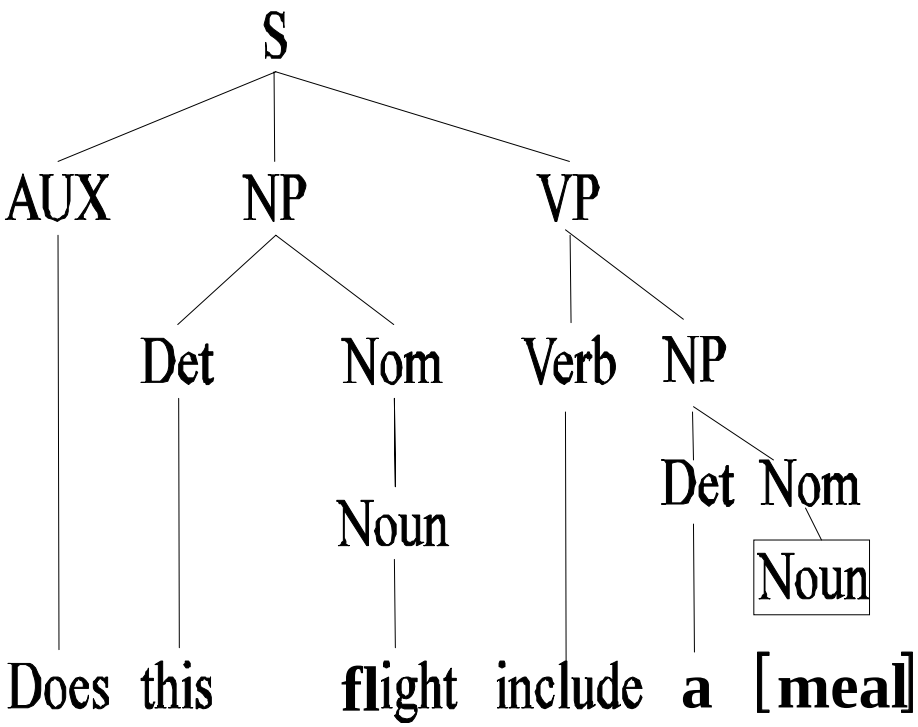


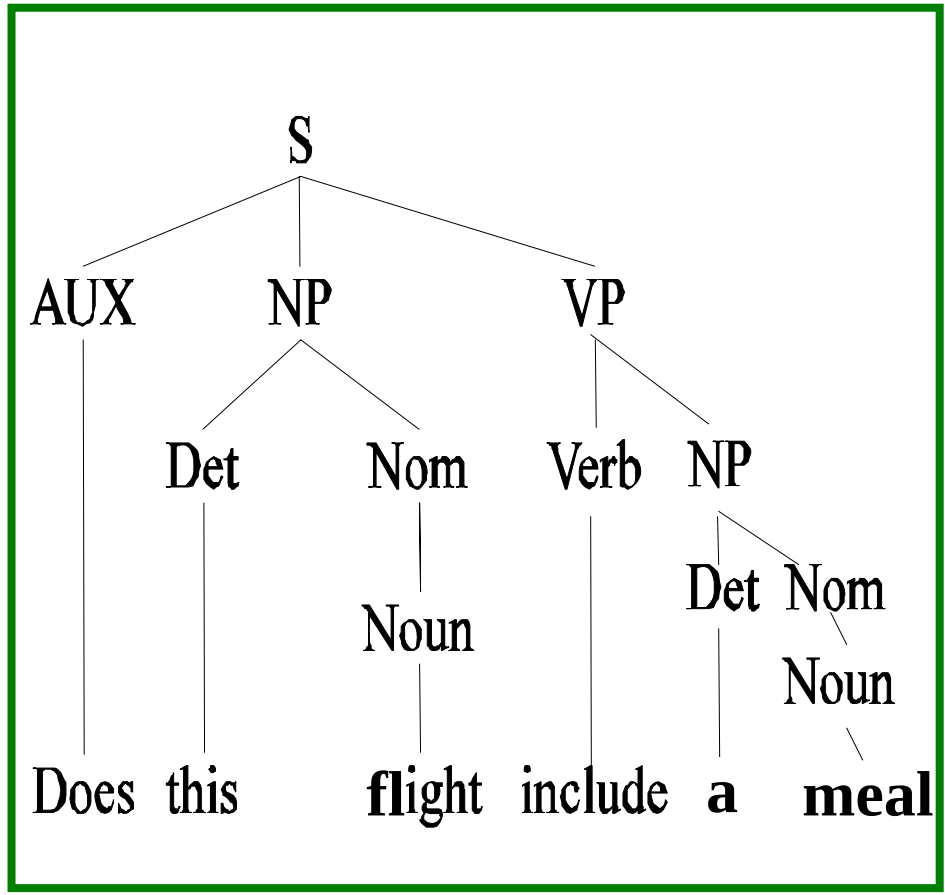
Parser top-down,depth-first,left-to-right





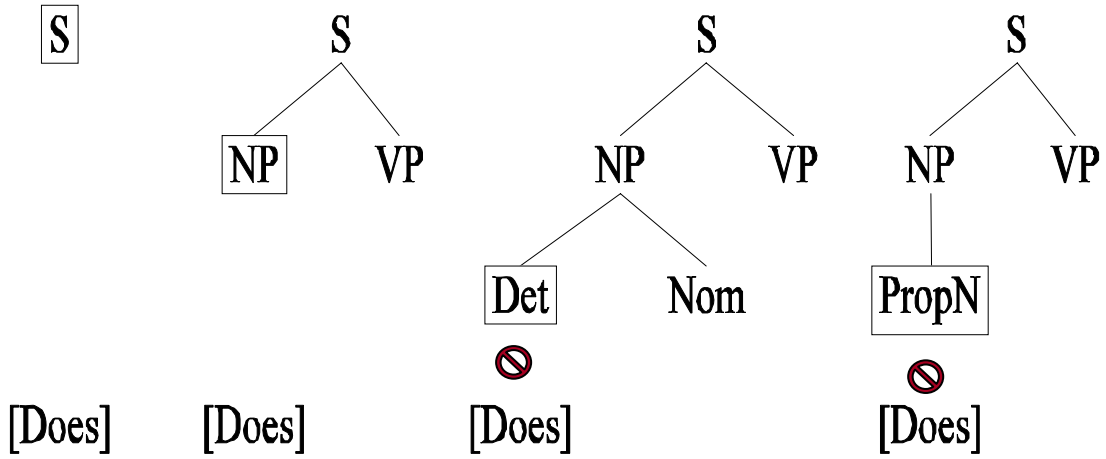
Parser top-down,depth-first,left-to-right





Parsing misto : aggiunta Bottom-Up

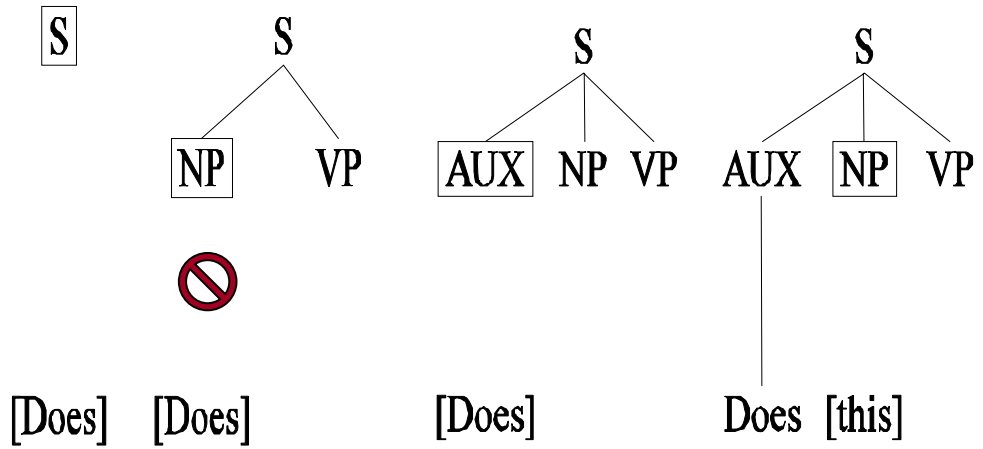
- Per ottenere un parser misto (o left-corner) è necessario inserire informazioni derivanti dalla strategia Bottom-Up:
  - La ricerca è infatti ancora cieca verso il basso:



- I due fallimenti potrebbero essere evitati sapendo che:
  - *does* è la parola che deve essere catturata
  - un *NP* non attiva mai la POS di *does*, ovvero *Aux*

Parsing misto : aggiunta Bottom-Up

- **SOLUZIONE (parser misto)**
  - Aggiungiamo la strategia Bottom-Up:
    - espandere i nodi solo con regole consistenti con la POS della parola in analisi:



- Devono quindi essere considerate solo le regole il cui primo non-terminale in RHS ha un POS uguale a quello della parola in input, oppure lo ha una sua derivazione sinistra (*left-corner*) , ovvero:

se **A** $\Rightarrow^*$ **B**α *B è left-corner di A*

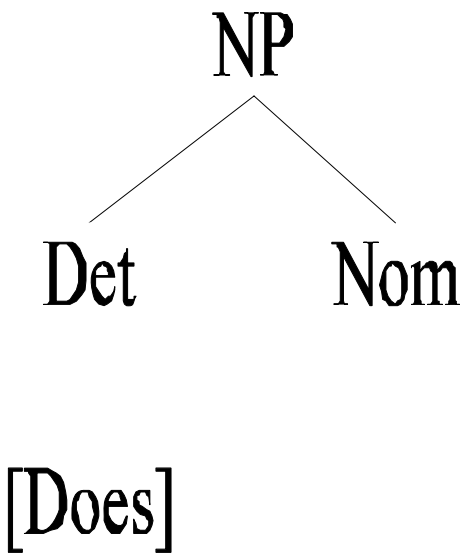
Parsing misto : aggiunta Bottom-Up

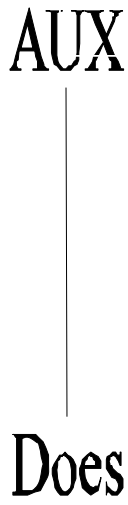
ESEMPIO

```
S -> NP VP          NO!
S -> Aux, NP, VP    SI!
```

NP non ha POS(does) come left corner

AUX ha POS(does) come left corner





Det ≠ Aux

PropN ≠ Aux

Aux = Aux

Parsing misto : aggiunta Bottom-Up

- **FILTRO BOTTOM-UP:**
  - Ogni volta che viene esaminata una regola da espandere:
    - Controllare il primo non-terminale in RHS
    - Controllare in una tabella di **left-corners** (tabella che contiene tutti i left-corners che sono POS, ottenuti applicando tutte le regole della grammatica a un dato non terminale) se esso espande in una POS compatibile con quella dell'elemento in input

```
S       -> NP VP
S       -> Aux NP VP
S       -> VP
NP      -> Det Nom
NP      -> PropN
……
Det     -> that | this | a
Aux     -> does
PropN   -> Houston | TWA
```

Category	Left-corner
S	Det,PropN,**Aux**,Verb
NP	Det,PropN
Nom	Noun
VP	Verb

Parsing: left-corners

1. **Determinare la tabella dei Left-Corners della seguente grammatica**

```
S  -> VP
S  -> NP VP
NP -> Det Noun
NP -> Poss Noun
NP -> Pron
NP -> NP PP
PP -> Prep NP
VP -> Verb
VP -> Verb PP NP
```

```
Det  -> an
Noun -> elephant|tiger |pajamas|
Pron -> I
Poss -> my
Verb -> shot
Prep -> in
Cong -> and
```

2. **Data la frase *"I shot a tiger"*, scrivere gli stati prodotti da un parser :**
   1. **Bottom-up, depth-first, left-to-right**
   2. **Left Corner (ovvero con l'aggiunta del filtro bottom-up).**

Quali miglioramenti ci sono ?

ESERCIZIO PARS 2

Parsing misto: problemi

LEFT-RECURSION

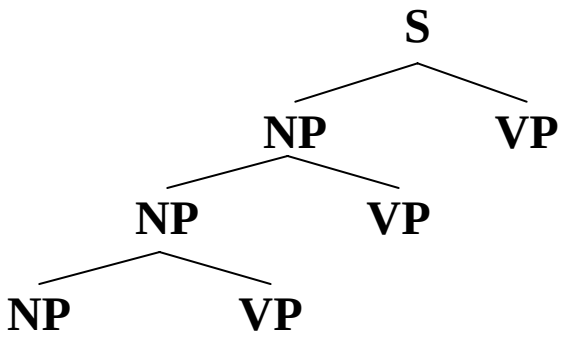
- Poiché la ricerca è Top-Down, Depth-First, Left-to-Right, c'è il problema della *left-recursion:*

Se nella grammatica è presente una regola left-recursive l'algoritmo può entrare in un loop infinito

ESEMPIO

The flight leaves

```
S  -> NP VP
NP -> NP PP
NP -> Nom
VP -> Verb
……
```



Parsing misto: problemi

AMBIGUITA'

Una frase che abbia più interpretazioni possibili (più di un parse-tree) è detta sintatticamente ambigua (o *strutturalmente ambigua*)

"Mario guarda Laura col cannocchiale"

Tre tipi principali di ambiguità strutturale:

- **attachment ambiguity**
  - *"We saw the Eiffel Tower flying to Paris"*
- **coordination ambiguity**
  - *"I saw old men and women"*
- **noun-phrase bracketing ambiguity**
  - *"Can you book TWA flights ?"*

Parsing misto: problemi

AMBIGUITA'

Per **disambiguare** frasi ambigue sintatticamente è necessario di disporre di informazione semantica o statistica:

- Unica soluzione a livello sintattico: far restituire dal parser tutti i possibili corretti ed ambigui parse-tree

- Quanti parse-tree in media vengono restituiti ?

  Una grammatica con le regole:

  NP → NP PP

  VP → VP PP

  può produrre moltissime alternative !

Num. di PP	Num. Di alberi
2	2
3	5
4	14
5	132
6	469
7	1430
8	4867

Parsing: ambiguità

Data la seguente grammatica:

```
S  -> NP VP
S  -> VP
NP -> Det Noun
NP -> Poss Noun
NP -> NP Conj NP
NP -> Pron
NP -> NP PP
PP -> Prep NP
VP -> Verb
VP -> Verb NP
VP -> Verb NP PP
```

```
Det  -> an
Noun -> elephant|tiger |pajamas|
Pron -> I
Poss -> my
Verb -> shot
Prep -> in
Cong -> and
```

1. **Verificare se le seguenti frasi sono sintatticamente ambigue:**
   - **“I shot an elephant in my pajamas”**
   - **“I shot an elephant and a tiger”**
2. **In caso di ambiguità costruire i relativi parse-tree e dire qual è il tipo di ambiguità riscontrata**

ESERCIZIO PARS 3

Parsing: ambiguità

3. Data la seguente grammatica:

```
S → NP VP
VP → Verb NP
VP → VP PP
VP → VP Conj VP
NP → NP PP
NP → NP CONJ NP
PP → Prep NP
PP → PP Conj PP
```

```
NP → Sally
NP → London
NP → Paris
NP →April
Verb →loves
P →in
Conj → and
```

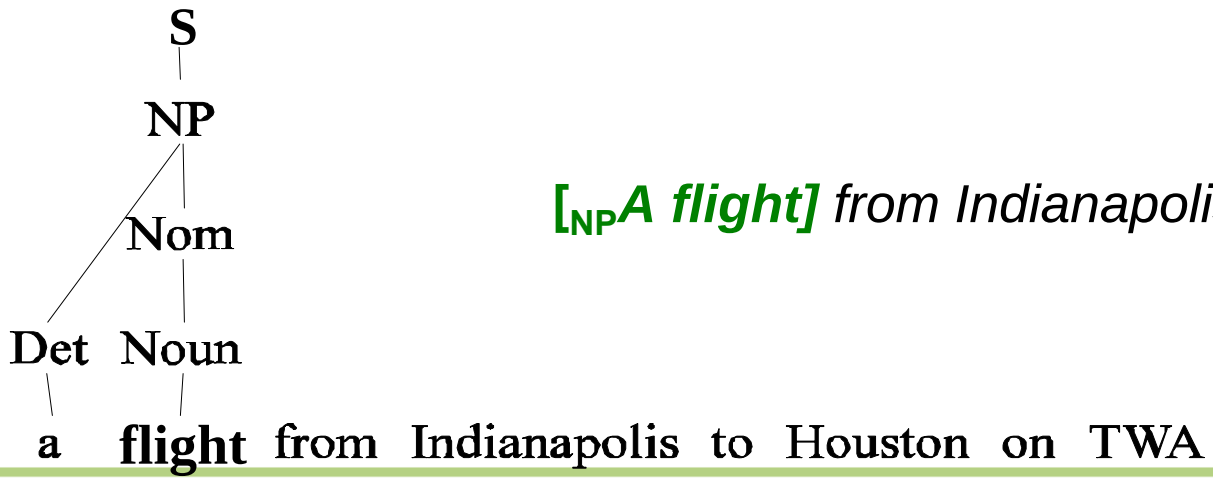
verificare se la frase "*Sally loves London and Paris in April*" è ambigua. In caso positivo, scrivere tutti i possibili parse-tree corretti e indicare la tipologia di ambiguità.

Parsing misto: problemi

RIPETIZIONI

- Alcune volte l'algoritmo produce sotto-alberi corretti, ma successivamente li scarta durante il backtracking:
  - Porzioni di frase sono ben interpretate , ma poi scartate poiché ottenute da regole sbagliate

ESEMPIO



*[$_{NP}$**A flight**] from Indianapolis to Houston on TWA*

Parsing misto: problemi

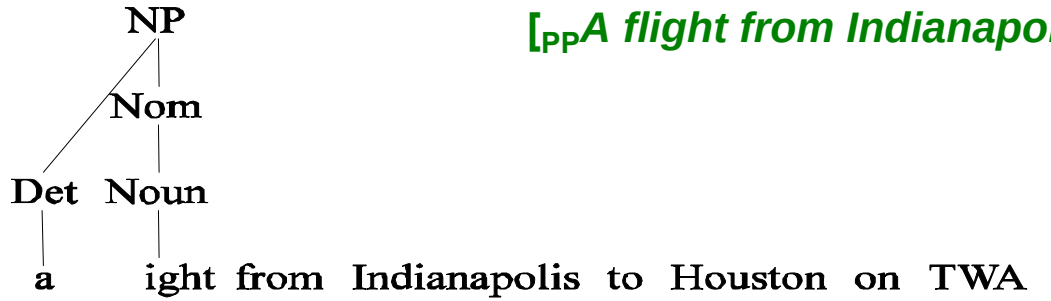
RIPETIZIONI

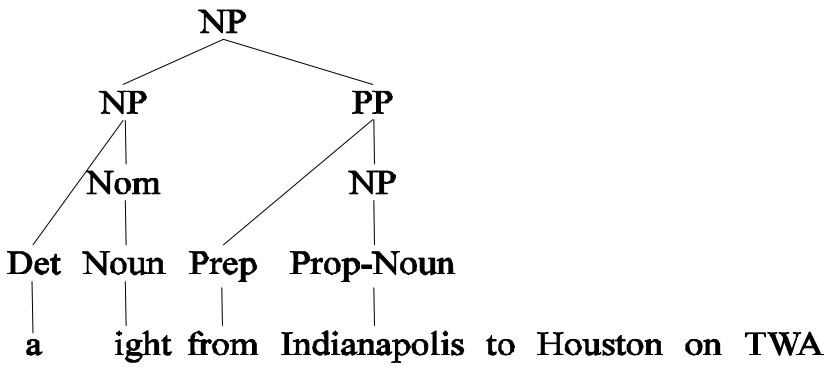
- Alberi errati prodotti in successione, ma che catturano correttamente:

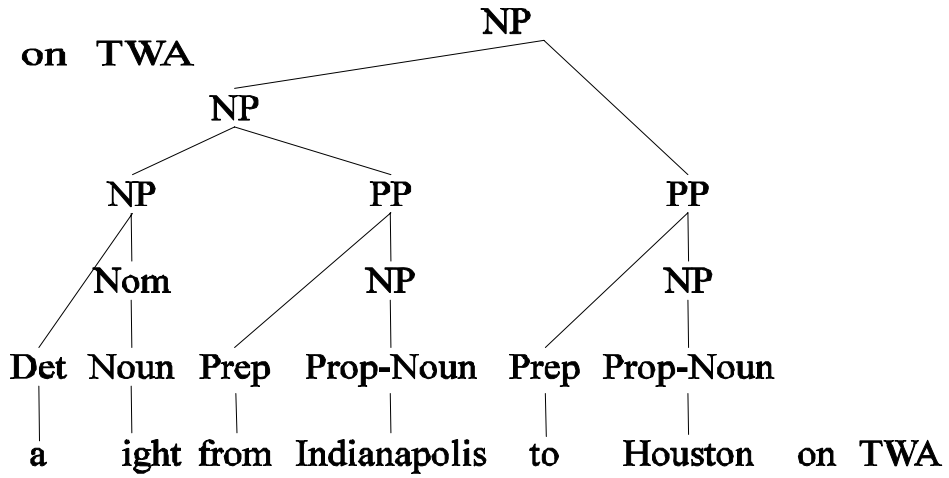
*[$_{NP}$**A flight]** from Indianapolis to Houston on TWA*

*[$_{PP}$**A flight from Indianapolis]** to Houston on TWA*

*[$_{PP}$**A flight from Indianapolis to Houston]** on TWA*



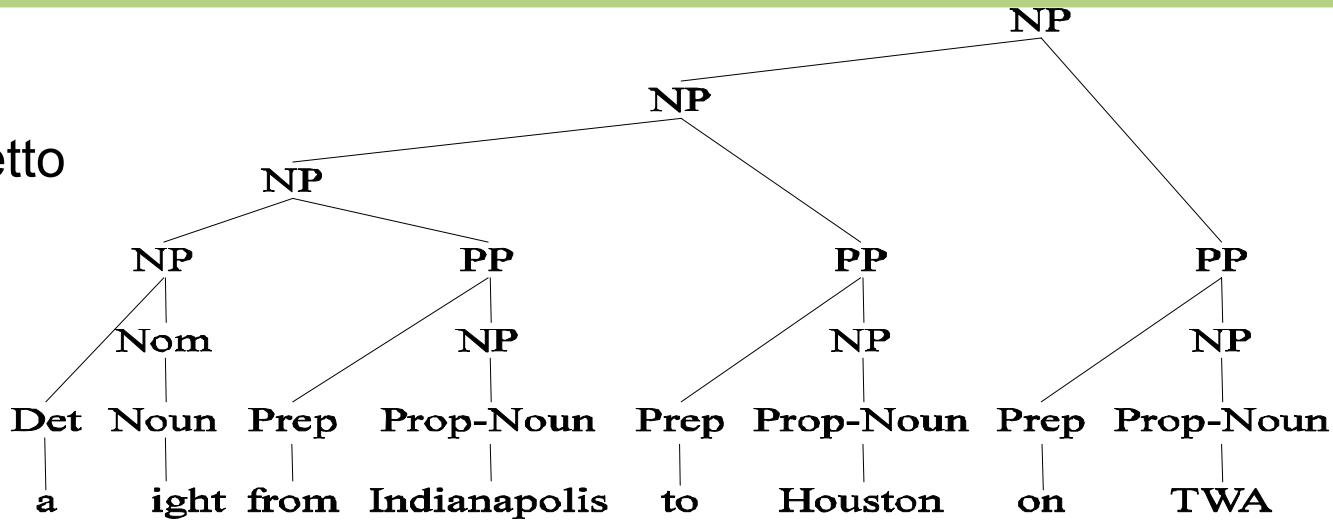




Parsing misto: problemi

RIPETIZIONI

- Albero finale corretto



- Quante ripetizioni ?

A flight	4
From Indianapoli	3
To Houston	2
On TWA	1
A flight from Indianapolis	3
A flight from Indianapolis to Houston	2
A flight from Indianapolis to Houston on TWA	1

- SOLUZIONE ? Adottare una strategia per *ricordare* le strutture corrette create → **Chart Parsing**

Chart Parsing e Programmazione Dinamica

OBIETTIVO: risolvere i problemi del pasing standard

- Evitare le ripetizioni
- Evitare i problemi della *left-recursion*
- Gestire l'ambiguità
- Risolvere un problema esponenziale in tempo polinominale

SOLUZIONE

- Utilizzare delle tabelle *(chart)* per ricordare ciò che è stato già scoperto
- Dividere il problema generale in sotto-problemi

Earley

Programmazione Dinamica

DIVIDI ET IMPERA…

- Un problema complesso viene diviso in sotto-problemi
- Per ogni sottoproblema crea una tabella in cui memorizzare le soluzioni
- Al termine del processo le soluzioni nelle tabelle risolvono il problema generale

APPLICAZIONE AL PARSING

- Ogni tabella contiene alberi parziali relativi ai diversi costituenti in input
- Gli alberi parziali vengono quindi scoperti una sola volta:
  - ogni *strada* diversa presa dal parser non deve riscoprirli
  - elimina il problema delle ripetizioni
  - elimina il problema dell'ambiguità (le tabelle rappresentano implicitamente tutte le soluzioni)

Earley

Programmazione Dinamica

PARSER BASATI SU QUESTA STRATEGIA

- **Earley** (1970)
  - Strategia Top-Down
  - Nessuna restrizione sul tipo di grammatica CFG in input
- **CYK** (1960)
  - Strategia Top-Down
  - Restrizione a forme normali (CNF)
- **GHR** (1980)

Algoritmo di Earley

IDEA DI BASE

- La stringa in input, di lunghezza *N* viene analizzato da sinistra a destra
- Ogni posizione nella stringa in input è numerata
- L'algoritmo procede quindi da una posizione alla successiva
- Un ***chart*** memorizza tutti gli alberi parziali creati fino alla posizione corrente
- Il *chart* è composto da *N+1* ***tabelle***
- Una tabella contiene una lista di ***stati***
- Ad ogni posizione:
  - Viene aggiunto un nuovo stato al chart
  - Ogni stato rappresenta un albero parziale generato sino a quel punto
- Al termine il chart rappresenta tutti i possibili albero di parsing per la frase

Algoritmo di Earley: stati

Uno stato contiene:

1. Un albero parziale corrispondente ad una regola della grammatica
2. La porzione dell'albero parziale che è stata già analizzata
3. La posizione dell'albero nella frase di ingresso

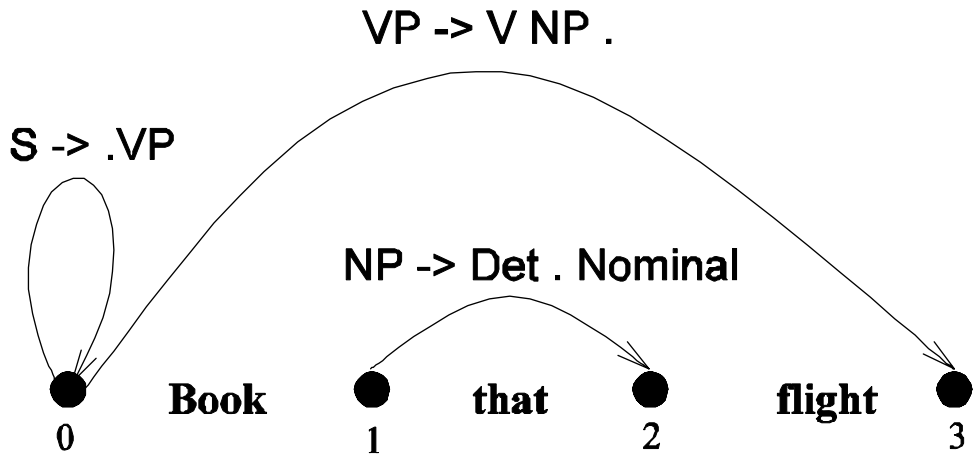
ESEMPIO

" book that flight"

DOTTED RULES

```
S  → • VP          [0,0]
NP → Det • Nom     [1,2]
VP → V NP •        [0,3]
```

1 2 3



Algoritmo di Earley: stati

ESEMPIO

`S → • VP [0,0]`

- Sono all'inizio della frase, in 0.
- Sto predicendo che ci sia una regola *S → VP*
- …ma non ho ancora analizzato nulla

`NP → Det • Nom [1,2]`

- Sto applicando la regola da 1 a 2
- Ho scoperto un *Det* da 1 a 2
- …ma non ho ancora finito

`VP → V NP • [0,3]`

- Ho applicato la regola da 0 a 3
- Ho scoperto un VP da 0 a 3
- …ho finito di applicare la regola

Algoritmo di Earley: algoritmo

ALGORITMO

- Parti dall'inizio della frase (posizione 0)
- Inserisci in `chart[0]` (tabella della posizione 0) lo stato $\gamma \rightarrow \bullet$ `S, [0,0]`
- For `i` from `0` to `N+1`

    For each `chart[i]`

        For each stato in `chart[i]`

            applica analisi

NOTA

- Le tabelle sono analizzate progressivamente da 0 a N+1
- Gli stati nelle tabelle sono letti in ordine dall'alto verso il basso
- *Applica analisi* cerca di far progredire l'analisi di un singolo stato

Algoritmo di Earley: operatori

Applica analisi applica ad uno stato uno dei tre seguenti operatori, a seconda di quale è il loro status:

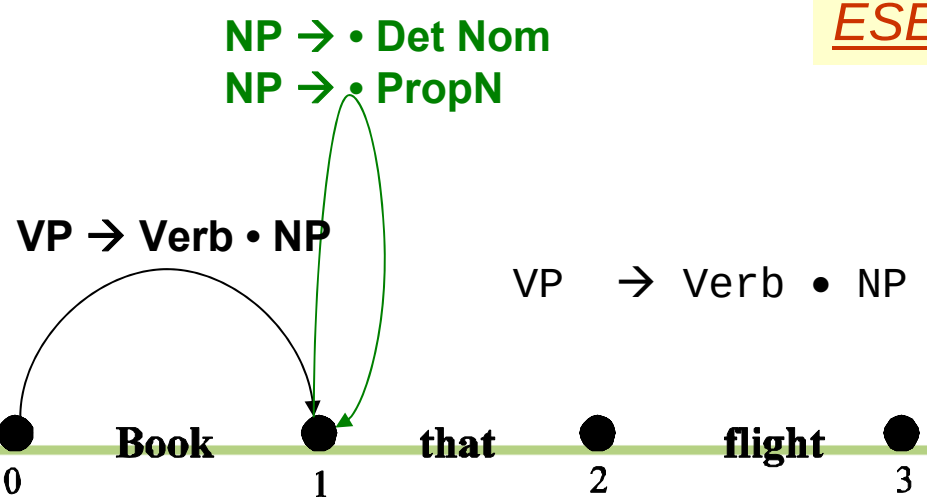
- PREDICTOR
  - *Crea un nuovo stato sulla base di quello correte in strategia Top-Down*
- SCANNER
  - *Applica le regole del lessico quando è possibile*
- COMPLETER
  - *Quando uno stato è completo* (dot *a destra*) *cerca gli stati che erano in sua attesa e li completa*

Algoritmo di Earley: operatori

PREDICTOR

- Applicato ad uno stato **s** che sia:
  - *Incompleto*
  - *Che abbia un non-terminale* **NT** *alla destra del* dot
  - *In cui* **NT** *non sia Part Of Speech*
- COSA FA ?
  - *Crea un nuovo stato per ogni regola che si può applicare ad* **NT**
  - *Tale stato viene messo nella stessa tabella di* **s**
  - *L'inizio e la fine del nuovo stato sono nella posizione in cui finisce* **s**

ESEMPIO



`VP → Verb • NP [0,1]`

`NP → • Det Nom [1,1]`

`NP → • PropN [1,1]`

Algoritmo di Earley: scanner

SCANNER

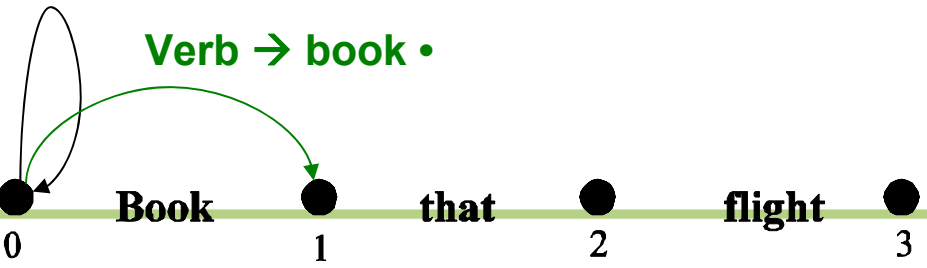
- Applicato ad uno stato **s** che sia:
  - *Incompleto*
  - *Che abbia un non-terminale* **NT** *alla destra del* dot
  - *In cui* **NT** *sia Part Of Speech*
- COSA FA ?
  - *Se la prossima parola in input ha POS uguale a* **NT**
    - *Crea un nuovo stato nella tabella successiva che inizi nella tabella precedente e finisca in quella successiva*

ESEMPIO

Verb → book (nel lessico)

VP → • Verb NP `VP → • Verb NP [0,0]` ⇒ `Verb → book • [0,1]`

Verb → book •

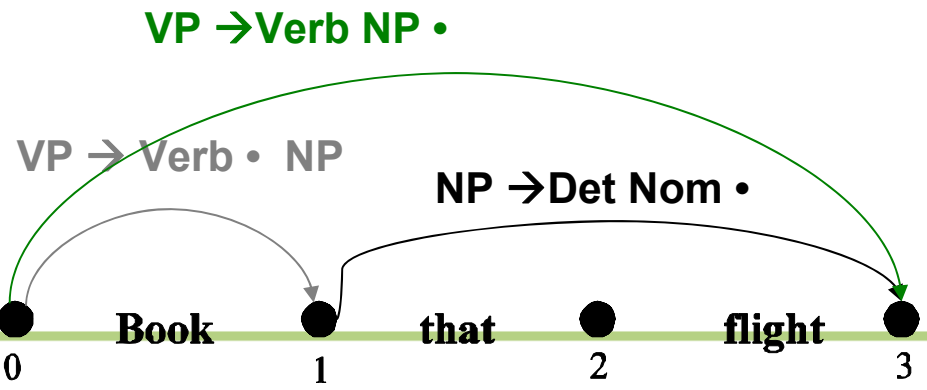


Algoritmo di Earley: scanner

COMPLETER

- Applicato ad uno stato **s** che sia:
  - *Completo (*dot *alla fine della regola), cioè:*
    - *il parser ha scoperto un costituente nella frase*
- COSA FA ?
  - *In ogni tabella precedente:*
    - *Cerca gli stati che erano in attesa di quel costituente in quella posizione*
    - *Fai avanzare tali stati: copiali nella tabella corrente, sposta in avanti il* dot, *aggiorna i loro inizio e fine*

ESEMPIO



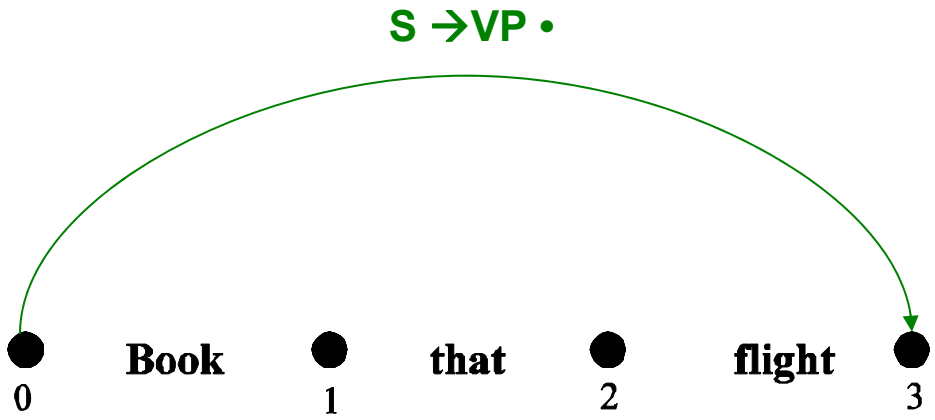

```
VP  → Verb • NP [0,1]
NP  → Det Nom • [1,3]
```

```
VP → Verb NP • [0,3]
```

Algoritmo di Earley: terminazione

- L'algoritmo **termina** quando una frase è stata riconosciuta, a partire dall'inizio dell'input fino al suo ultimo elemento, ovvero quando viene creato uno stato:

$$S \rightarrow \alpha \bullet \quad [0,N]$$



S → VP • [0,3]

Algoritmo di Earley: Esempio

ESEMPIO

- Frase in input: *"Book that flight"*

- Grammatica:

```
S    → NP VP          Noun → flight
S    → Aux NP VP      Det  → that
S    → VP             Verb → book
NP   → Det Nom
NP   → PropN
VP   → Verb
VP   → Verb NP
Nom  → Noun
Nom  → Noun Nom
```

- Condizione di terminazione: $S \rightarrow \alpha \bullet [0,3]$

Algoritmo di Earley: Esempio

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

(NB: nel libro questo chart è errato!)

Algoritmo di Earley: Esempio

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

Chart[1]		
Verb ■ *book* ▪	[0,1]	Scanner
VP ■ *Verb* ▪	[0,1]	Completer
S ■ *VP* ▪	[0,1]	Completer
VP ■ *Verb* ▪ *NP*	[0,1]	Completer
NP ■ ▪ *Det NOMINAL*	[1,1]	Predictor
NP ■ ▪ *Proper-Noun*	[1,1]	Predictor

Algoritmo di Earley: Esempio

Chart[1]

Verb ■ *book* ■	[0,1]	Scanner
VP ■ *Verb* ■	[0,1]	Completer
S ■ *VP* ■	[0,1]	Completer
VP ■ *Verb* ■ *NP*	[0,1]	Completer
NP ■ ■ *Det NOMINAL*	[1,1]	Predictor
NP ■ ■ *Proper-Noun*	[1,1]	Predictor

Chart[2]

Det ■ *that* ■	[1,2]	Scanner
NP ■ *Det* ■ *NOMINAL*	[1,2]	Completer
NOMINAL ■ ■ *Noun*	[2,2]	Predictor
NOMINAL ■ ■ *Noun NOMINAL*	[2,2]	Predictor

Algoritmo di Earley: Esempio

Chart[3]

Noun → flight •	**[2,3]**	**Scanner**
NOMINAL → Noun •	**[2,3]**	**Completer**
NOMINAL → Noun • NOMINAL	**[2,3]**	**Completer**
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**

Algoritmo di Earley: Parser

- Così strutturato l'algoritmo è un **riconoscitore**, non un parser !
  - Alla fine rimane solo lo stato finale **S → α • [0,N+1]**
  - Non c'è nessun modo di rintracciare la struttura di S, cioè…
  - … il *percorso* che ha portato alla formazione di S, cioè…
  - … il *parse-tree* di S

SOLUZIONE

- Estrarre gli alberi parziali per ogni costituente riconosciuto
- Ovvero, per ogni stato nella tabella:
  - memorizzare gli stati completi che lo hanno generato
- Basta modificare il *Completer*:
  - ad ogni stato che crea aggiungere un puntatore alla regola che ha permesso la creazione

Algoritmo di Earley: Esempio

ESEMPIO

Chart[1]

S8	*Verb* ■ *book* ■	[0,1]	Scanner	[]
S9	*VP* ■ *Verb* ■	[0,1]	Completer	[S8]
S10	*S* ■ *VP* ■	[0,1]	Completer	[S9]
S11	*VP* ■ *Verb* ■ *NP*	[0,1]	Completer	[S8]
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]	Completer	[S14]
S16	*NOMINAL* ■ ■ *Noun*	[2,2]	Predictor	[]
S17	*NOMINAL* ■ ■ *Noun NOMINAL*	[2,2]	Predictor	[]

Algoritmo di Earley: Esempio

Chart[3]

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

Algoritmo di Earley: Parser

CREAZIONE DELL'ALBERO

- Partire dallo stato completo della *S* nell'ultima tabella (*S23*)
- Leggere ricorsivamente i puntatori all'indietro creati dal Completer
- Fermarsi agli stati che non hanno più puntatori

AMBIGUITA'

- Ambiguità sintattica: più alberi corretti per la stessa frase
- In Earley vuol dire: più stati completi di S nell'ultima tabella
- Per avere tutte le interpretazione si risalgono i percorsi di tutte le *S*
- Fermarsi agli stati che non hanno più puntatori

Algoritmo di Earley: Esempio

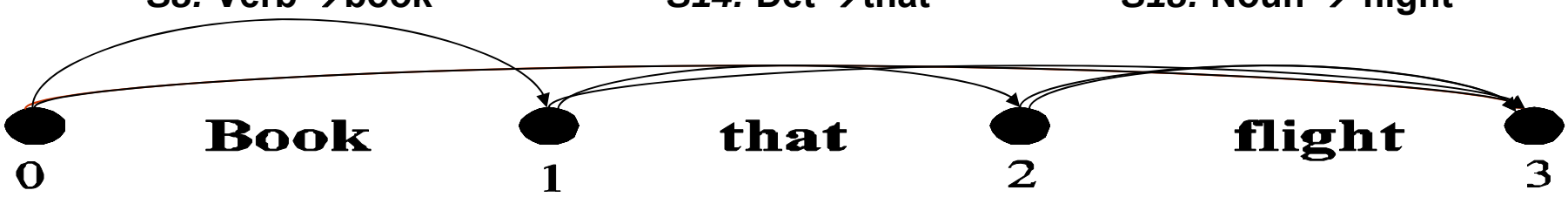
S23: **S →VP •**

S22: **VP →Verb NP •**

S21: **NP →Det NOM •**

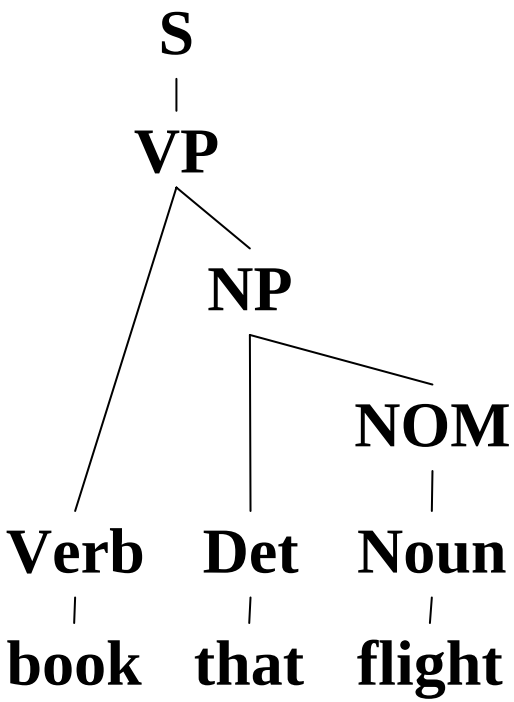
S19: **NOM → Noun•**

S8: **Verb →book •** ***S14:*** **Det →that •** ***S18:*** **Noun → flight •**



Algoritmo di Earley: Esempio

E quindi …



Algoritmo di Earley: Riassumendo

COSA MIGLIORA EARLEY RISPETTO AI PARSER STANDARD ?

- **Evita le ripetizioni!**
  - *Grazie alla memorizzazione in tabelle*
- **Gestire l'ambiguità**
  - *Soluzioni ambigue vengono semplicemente rappresentate*
- **Risolvere un problema esponenziale in tempo polinominale**
  - *Grazie alla memorizzazione non si devono ricreare alberi parziali*
  - *L'ambiguità è rappresentata implicitamente nel chart*
- **Left Recursion ???**

Algoritmo di Earley: Riassumendo

LEFT-RECURSION

- Per eliminare la *left-recursion*è sufficiente imporre nel *Predictor*:

Non inserire in una tabella uno stato già presente

ESEMPIO

Grammatica con:

S → NP VP

NP → NP PP

S → • NP VP	[0,0]	
NP → • NP PP	[0,0]	Predictor
~~NP → • NP PP~~	~~[0,0]~~	~~Predictor~~

Parsing: Earley

Data la seguente grammatica:

```
S    → NP VP
S    → VP
NP   → Det Nom
NP   → PropN
VP   → Verb
VP   → Verb NP
Nom  → Noun
Nom  → Noun Nom
```

```
Noun → elephant | flight
Det  → an |the
Verb → takes
```

Scrivere il processo di parsing completo secondo l'algoritmo di Earley, per la seguente frase:

"An elephant takes the flight"

ESERCIZIO PARS 4