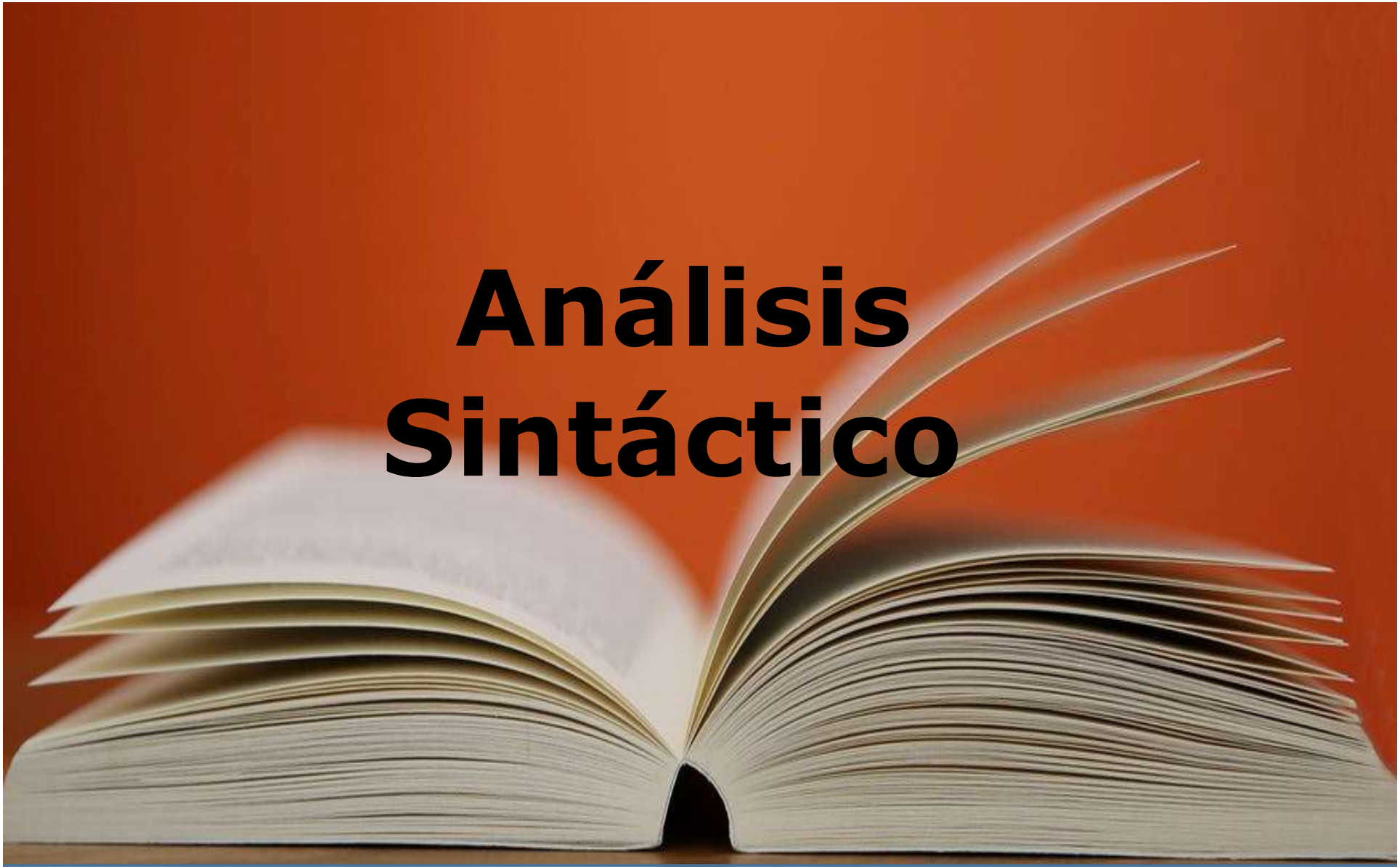
Procesamiento del Lenguaje Natural

Dr. John Atkinson

iUAI
UNIVERSIDAD ADOLFO IBÁÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS



Análisis Sintáctico

Procesamiento de Lenguaje Natural

iUAI
UNIVERSIDAD ADOLFO IBÁÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS

Análisis Sintáctico (Parsing)

Parsing es el análisis automático de una oración respecto a la *estructura sintáctica* existente entre sus palabras.

O sea, dado una gramática, esto significa "*producir*" (*derivar*) una *estructura de frase jerárquica (árbol) para* cada oración.

Sintaxis

- La sintaxis se refiere a la forma en que las palabras se "*ordenan*" entre sí, y la "*relación*" entre ellas.

- El objetivo de la sintaxis es *modelar el conocimiento* de lo que la gente inconcientemente tiene acerca de la gramática en su lenguaje nativo.

¿Porqué es de Interés?

- *Paso previo para intentar "entender" una oración en cualquier aplicación.*
- *Sistemas de pregunta-respuesta.*
- *Extracción de Información.*
- *Traducción automática.*
- *Revisores gramaticales.*
- *etc*

Ambigüedad

Pero, cada oración podría tener muchas estructuras u *árboles válidos* → gramática ambigüa !!

¿Deberíamos recuperar todos ó solo uno?

Si es uno, ¿Cómo sabemos cuál?

Ambigüedad

Esta oración es ambigüa, ¿En qué forma?

`I booked a flight from LA`

¿Qué debería pasar si analizamos la oración?

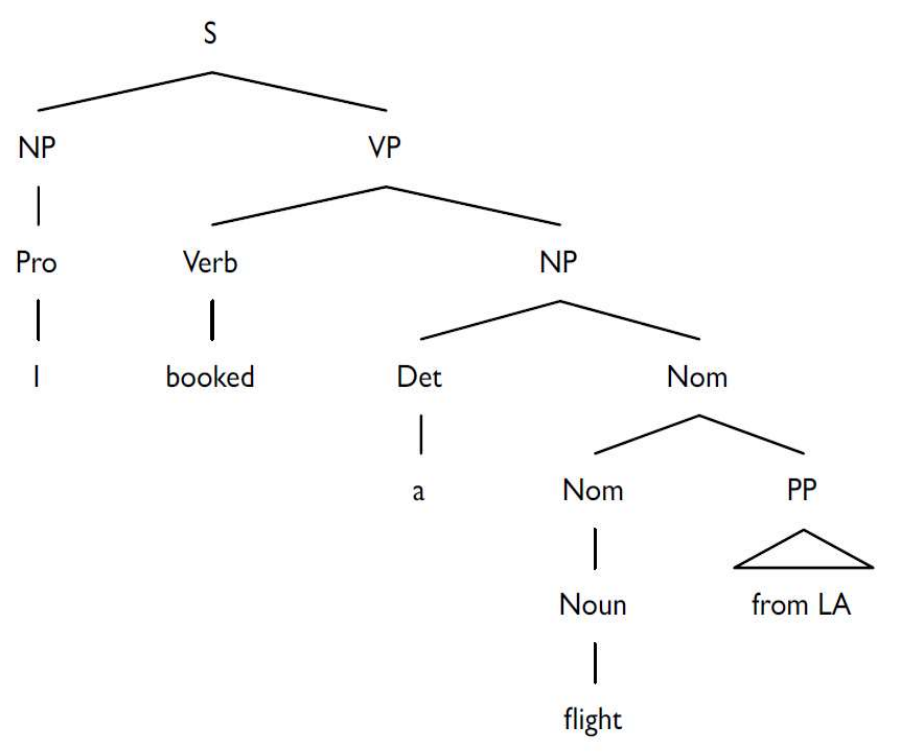
Ambigüedad

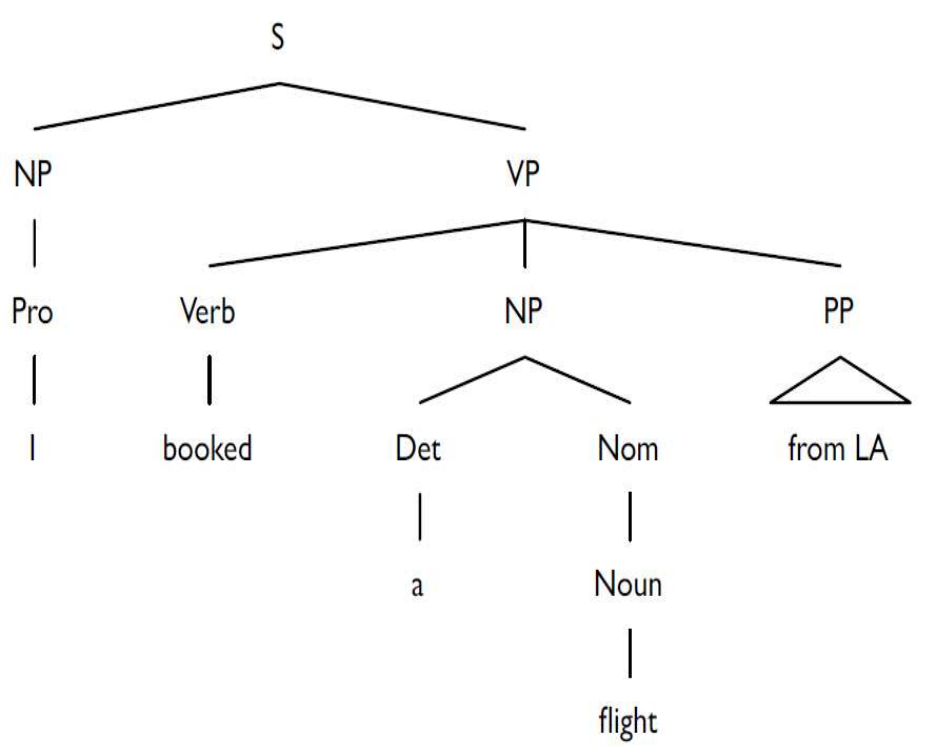
¿Qué debería pasar si analizamos la oración?

I booked (a flight from LA)

(I booked a flight) from LA

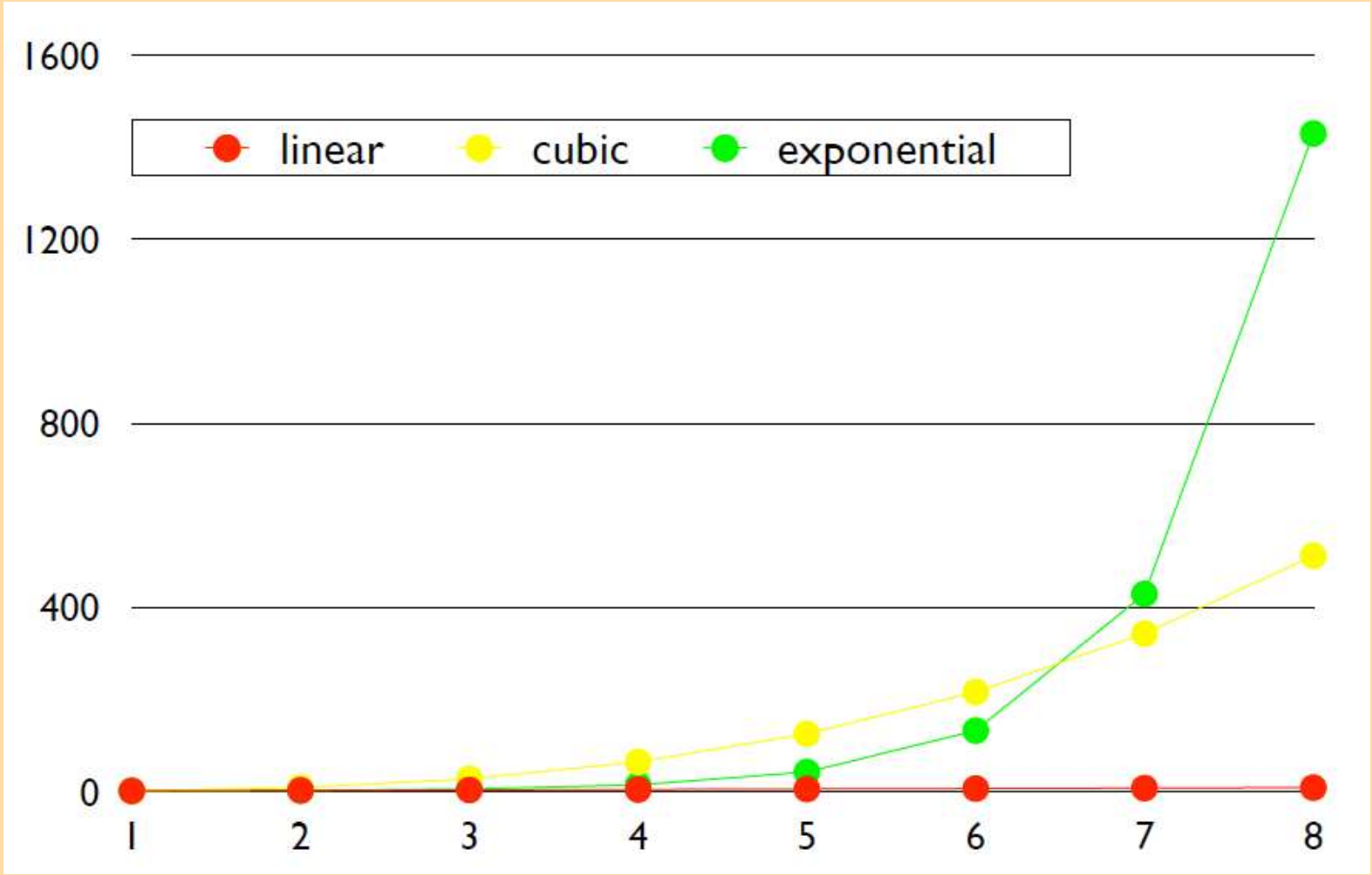
Ambigüedad: *Parse trees*



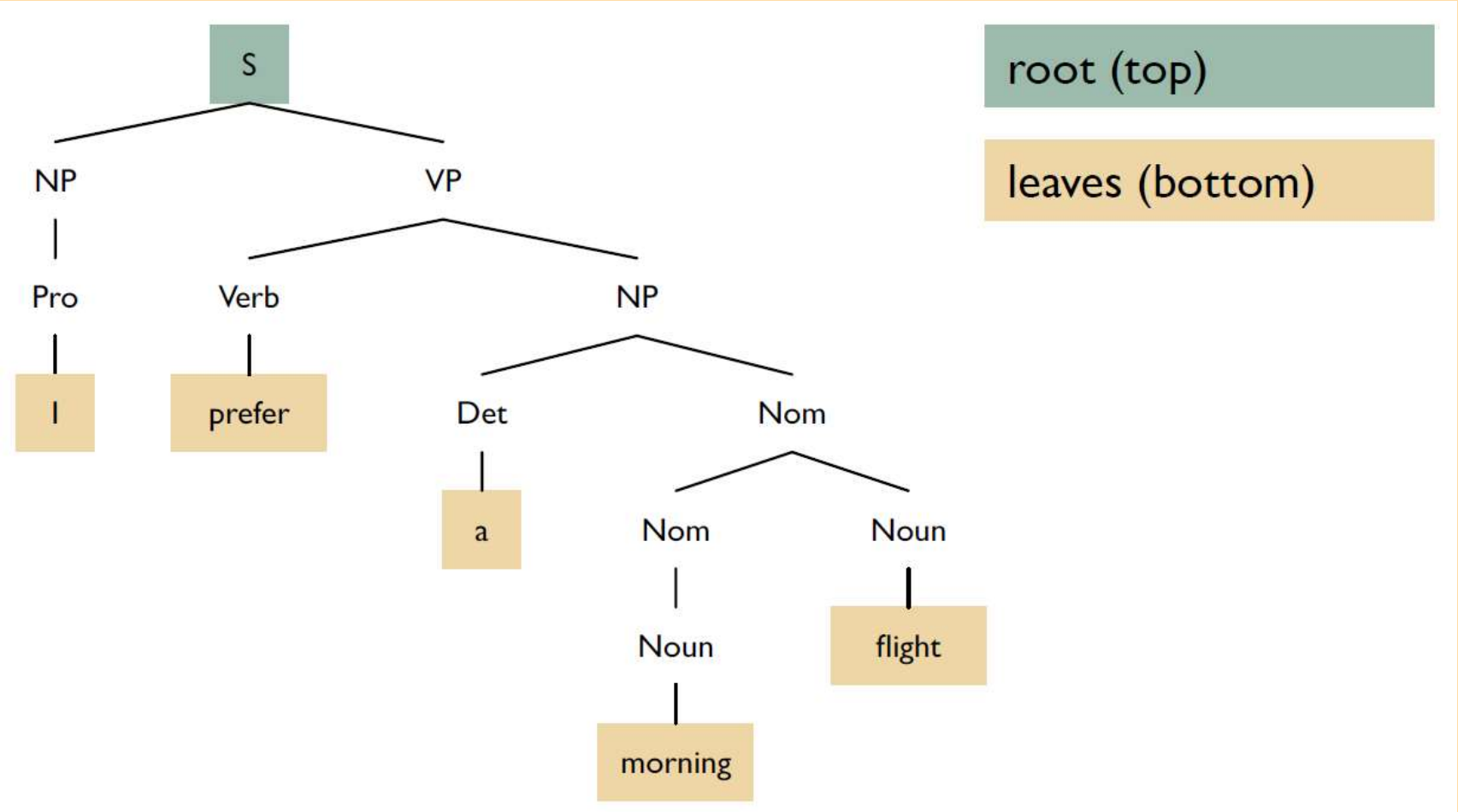


2 posibles estructuras sintáticas (parse trees) !!

Búsqueda
Explosión Combinatoria



Árboles (de *Estructura de Frase)*



Gramática

El tipo de gramática denominada "*Gramática Independiente del Contexto*" (CFG) captura los *constituyentes* y el *ordenamiento* de un lenguaje*:*

- *Ordenamiento*:
  - ¿Cuáles son las reglas que controlan el ordenamiento de las palabras y unidades mayores en el lenguaje?
- *Constituyentes*:
  - ¿Cómo las palabras se agrupan en unidades y cómo se comportan estas unidades?

Constituyentes
(*Grupos Sintácticos*)

- ✓ *Frases Nominales* (NP): tienen como "*cabeza*" (*head*) un *nombre* ó *sustantivo*:
  - *Minera Escondida..*
  - *Los mejores alumnos de la clase..*
  - *Rodrigo Alvarez..*

¿Cómo sabemos que estos forman un *constituyente*?

- Todas pueden aparecer antes que un verbo:
  - *Minera Escondida obtuvo…*
  - *Los mejores alumnos de la clase fueron…*
  - *Rodrigo Alvarez jubiló...*

Problemas

- Pero las palabras individuales NO siempre aparecen antes que los *verbos*!!!.

- Debe existir generalización de estados (reglas gramaticales):
  - Ej: "*Las Frases Nominales (NP) ocurren antes que los Verbos*"

¿Qué es una *Gramática* (CFG)?

Una CFG es una 4-tupla definida como:

1) Un conjunto de símbolos no-terminales (variables) **N**
2) Un conjunto de símbolos terminals (alfabeto) Σ
3) Un conjunto de producciones **P** (reglas) de la forma:

   $\mathbf{A} \rightarrow \alpha$

   Donde **A** es un no-terminal y α es un string de símbolos del conjunto infinito de strings $(\Sigma \cup \mathbf{N})^*$
4) Un símbolo de comienzo (raíz) **S**

Componentes de una CFG

- ✓ *Terminales:*
  - Sólo aparecen como “hojas” del árbol.
  - Aparecen en lado derecho de las reglas (RHS).
  - Corresponden a *palabras* del lenguaje:
    - **Cliente, queja, realiza, buzón**
- ✓ *No-terminales:*
  - No aparecen como “hojas” del *parse tree.*
  - Aparecen en lado izquierdo (LHS) y derecho (RHS) de las reglas.
  - Corresponden a *constituyentes* del lenguaje:
    - **NP, VP, etc**

Ejemplo de CFG:

S → NP VP

NP → Art NOMINAL

NOMINAL → Nombre

VP → Verbo

Art → un

Art → el

Nombre → vuelo

Verbo → despegó

Verbo → aterrizó

Interpretación

"**A** → α" se puede interpretar como "**A** *se re-escribe como* α" ó "**A** *produce ó deriva* α"

Por tanto,

S → NP VP

- Dice que existen unidades llamadas **S**, **NP**, y **VP** en este lenguaje
- Que un **S** consiste de (ó *produce*) un **NP** seguido de un **VP**

Análisis Sintáctico

Dados una gramática ***G*** y un string ***w***, existen dos problemas que podemos resolver:

- ✓ Reconocimiento: determinar si ***G*** acepta ***w***
- ✓ Parsing: recuperar (todos ó algunos) *parse trees* asignados a ***w*** por ***G***

Derivaciones

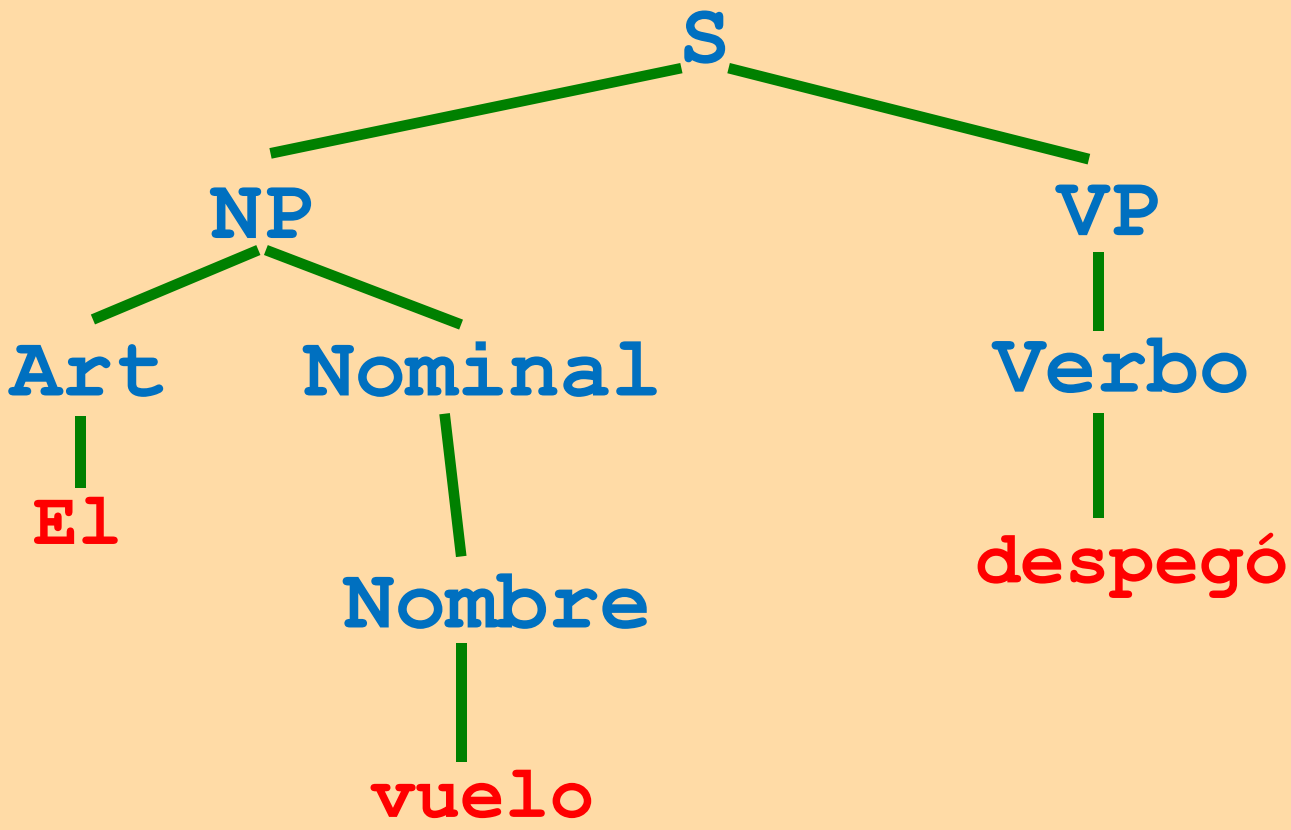
Una derivación es una secuencia de reglas aplicadas a un string tal que:

- *Cubre todos los elementos en un string.*
- *Cubre sólo los elementos en un string.*

Derivaciones como *Árboles*

¿Como derivamos el árbol para

"**El vuelo despegó**"?



Algunas Características

Tenemos que manejar reglas donde el no-terminal de la *izquierda* también aparece en alguna posición de la *derecha*:

NP → NP PP [[The flight] [to Boston]]

VP → VP PP [[departed Miami] [at noon]]

Recursión

Los constituyentes anteriores pueden producen estructuras sintácticas interesantes repetitivas:

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

Recursión

Si tenemos la regla

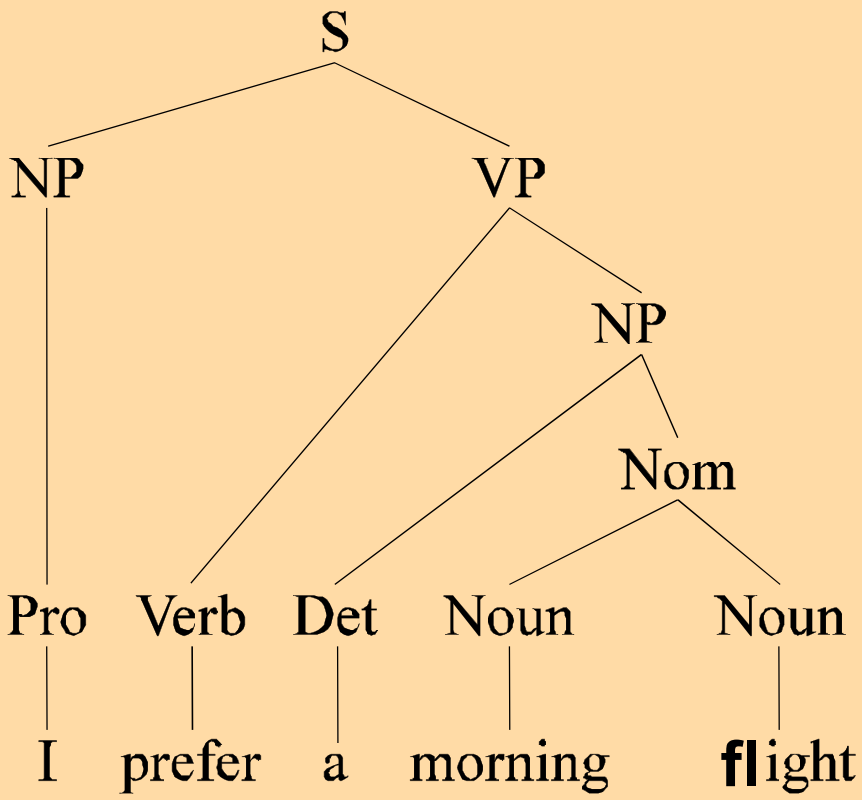
VP → V NP

Un analizador (*parser*) se preocupa que el elemento después de **V** sea un NP.

El analizador no tiene que saber los aspectos internos de NP.

Notación: Paréntesis vs Árbol

[$_S$ [$_{NP}$ [$_{PRO}$ I] [$_{VP}$ [$_V$ prefer [$_{NP}$ [$_{NP}$ [$_{Det}$ a] [$_{Nom}$ [$_N$ morning] [$_N$ flight]]]]]]]



Bancos de Árboles (*Treebanks*)

- ✓ Usualmente, debemos construir nuestra propia *gramática* para el lenguaje que deseamos reconocer.

- ✓ Sin embargo, en algunos casos, especialistas se han dado de trabajo de crear ó anotar "*bancos de árboles*" ó *treebanks*, que contienen las estructuras sintácticas resultantes:
  - Treebanks usuales: *Brown* y *WSJ*
  - Útil para tareas de evaluación en NLP.

Conceptos de Parsing

- ✓ Parsing: tarea de asignar *árboles válidos* a un string de entrada (*oración*). Esta tarea es usualmente realizada por un *parser* ó *analizador sintáctico*.

- ✓ Esto se puede ver como una tarea de *búsqueda* de árboles válidos (“caminos” ó estructuras posibles).

Conceptos de Parsing

Un *árbol válido* (*correcto*) es aquel que cubre todos y sólo los elementos del string de entrada y tiene un símbolo inicial no-terminal S en la raíz.

¿Podríamos buscar/enumerar todos los posibles árboles?

- Problemas de ambigüedad!!!. Se debe elegir el *árbol correcto* entre MUCHOS posibles.

Parsing como Búsqueda

- ✓ Dos estrategias básicas de *búsqueda*:
  - Top-down (*descendente*): comienza *buscando* en la raíz del árbol e intenta llegar a las hojas (string de entrada).
  - Bottom-up (*ascendente*): comienza en las hojas del árbol (string de entrada) e intenta buscar para llegar a la raíz.

Supuestos por ahora …

- Todas las palabras se encuentran en algún “buffer” temporal.
- No se realiza POS tagging a la entrada.
- No interesa el análisis morfológico.
- Todas las palabras son conocidas.

Parsing *Top-Down*

Idea básica:

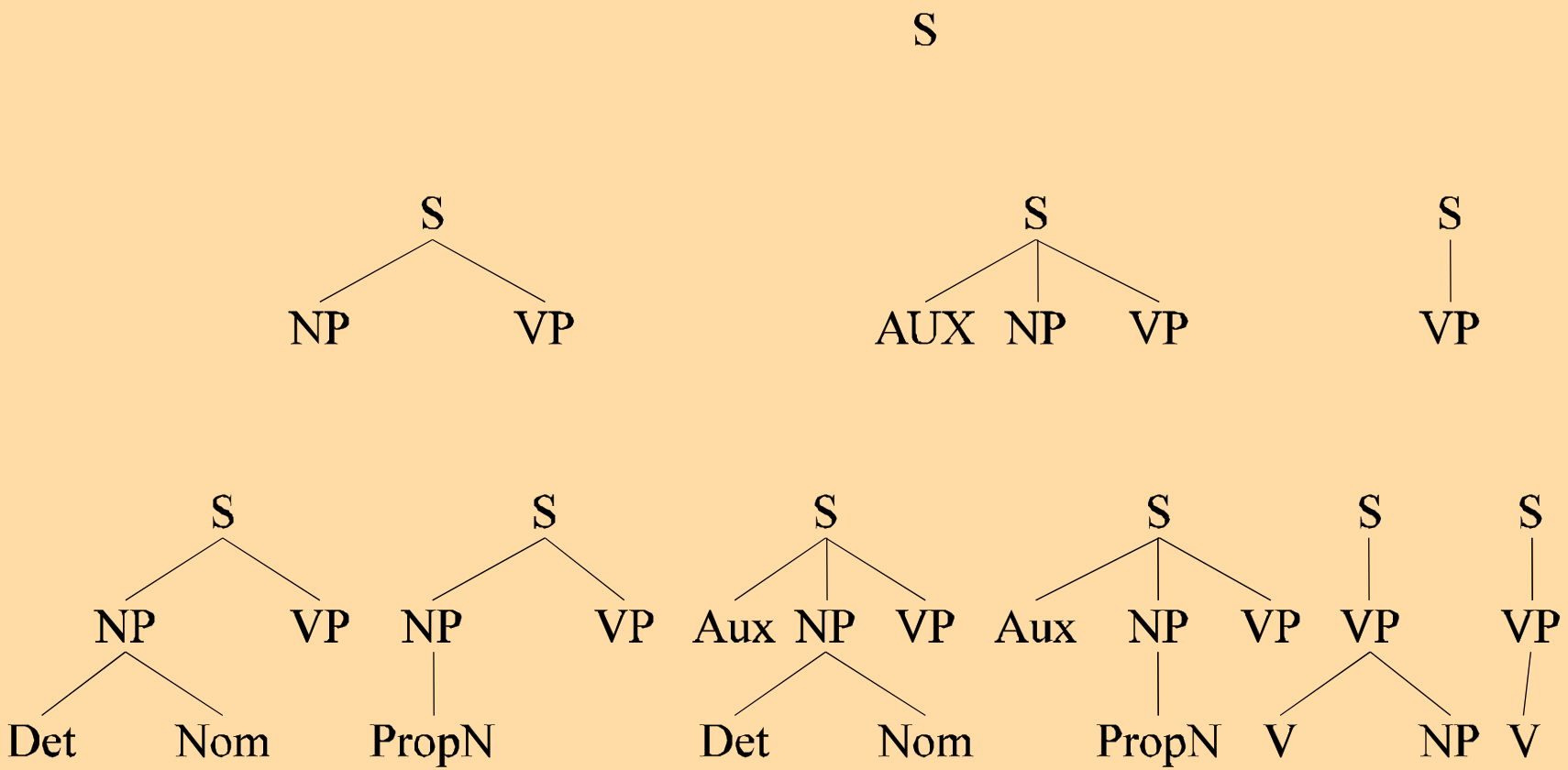
- Comenzar en nodo raíz, y expandir el árbol “hacienda coincidir” (*matching*) el lado izquierdo de las reglas.
- Derivar un árbol cuyas hojas “*coincidan*” con el string de entrada (oración).

Problemas potenciales:

- Uso de reglas que nunca podrían “producir” la entrada.
- Podría entrar en ciclos con reglas recursivas:

VP → VP PP

Espacio *Top Down*



Parsing *Bottom-Up*

Idea básica:

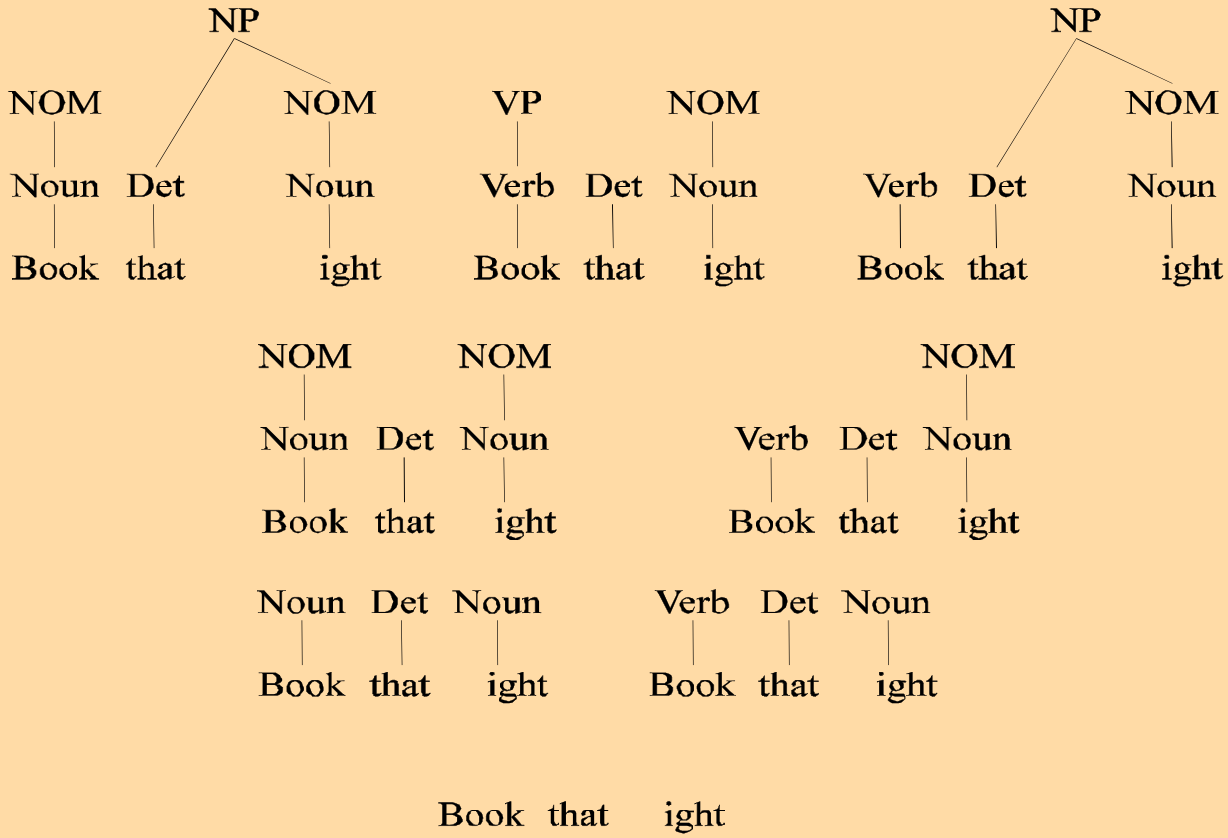
- Comenzar con las hojas, construya el árbol "calzando" el lado derecho de las reglas.
- Construir un árbol con S en la raíz.

Problemas Potenciales:

- Podría construir estructuras que nunca podrían estar en un árbol.
- Podría entrar en ciclos en las producciones nulas:

$$\mathbf{NP \rightarrow \varepsilon}$$

Espacio *Bottom-Up*



Top-Down vs Bottom-Up

- ✓ *Top-down*
  - Solo busca árboles que puedan ser respuesta.
  - Pero también sugiere árboles que no son consistentes con las palabras.

- ✓ *Bottom-up*
  - Solo forma árboles consistentes con las palabras.
  - Sugiere árboles que no tienen sentido globalmente.

Control en Parsing

- ✓ En ambos casos, no se registra ó "memoriza" el espacio de búsqueda y las decisiones (eficiencia?).

- ✓ Nos interesa saber:
  - *¿Qué nodo (del árbol) se debe expandir?*
  - *¿Qué regla gramatical se debe utilizar para expandir un nodo?*

Problemas con Parsing Top Down

1) *Recursión izquierda*
2) *Ambigüedad*
3) *Re-análisis de sub-árboles*

Problema (1): Recursion Izquierda

¿Qué pasa con la siguiente situación?

S → NP VP

S → Aux NP VP

NP → NP PP

NP → Det Nominal

…

Y con la oración que comienza con

Did the flight…

Problema (2): Ambigüedad

VP → VP PP

NP → NP PP

"Muéstreme la comida en el vuelo 286 desde Arica a Santiago"

Tenemos 14 árboles posibles de parsing!!

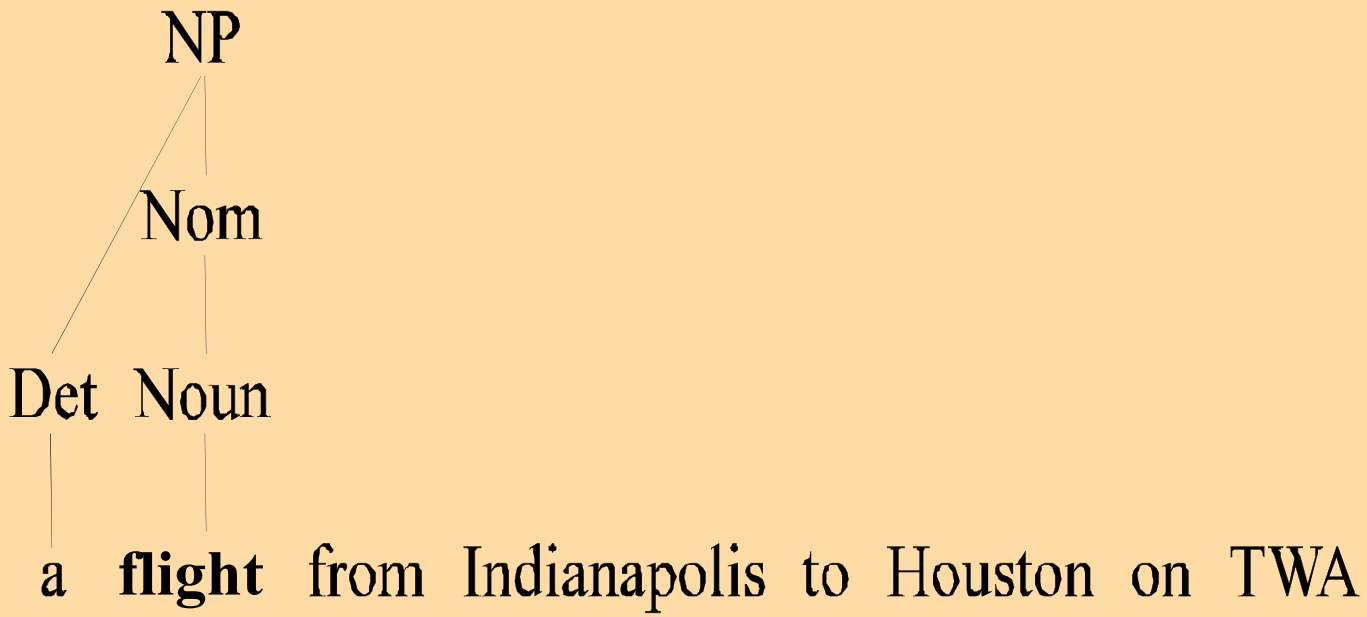
Manejo de Ambigüedad

- ✓ El número de posibles *parse trees* crece exponencialmente con el largo de la oración.
- ✓ Un enfoque del tipo *buscar-deshacer-reintentar* (*backtracking*) es demasiado ineficiente.
- ✓ Observación:
  - Los ***parse trees*** alternativos comparten sub-estructuras.
  - Podemos utilizar *programación dinámica*.

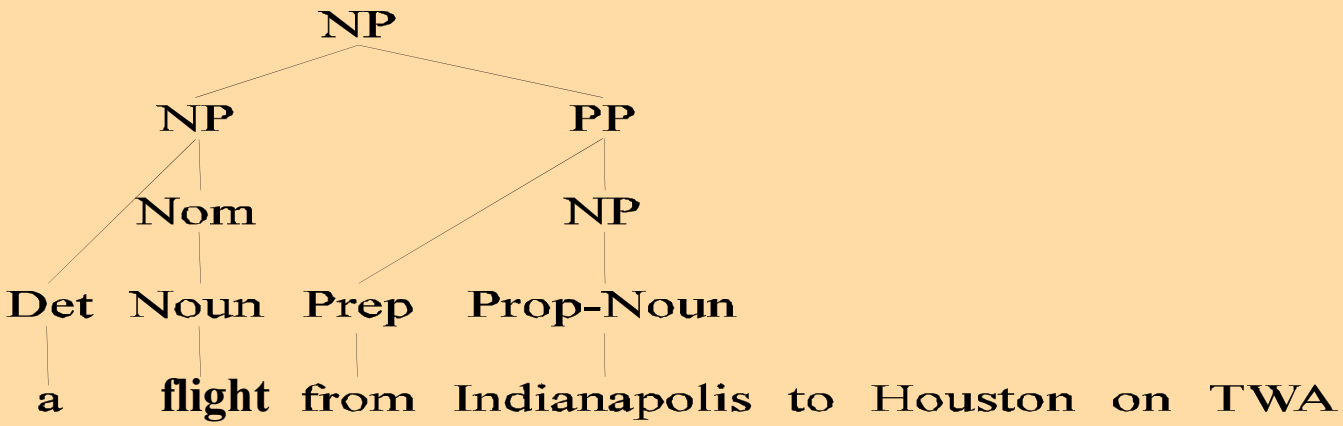
Problema (3): Re-análisis de sub-árboles

- ✓ Parsing es difícil y lento: es una pérdida de tiempo re-hacer los análisis una y otra vez.
- ✓ Considere el intento de *parsing top-down* para el siguiente NP:

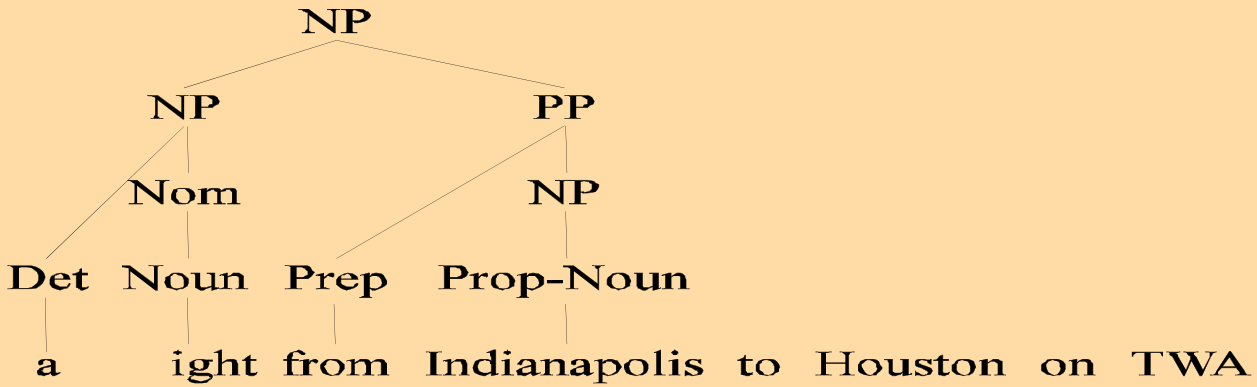
A flight from Indianapolis to Houston on TWA

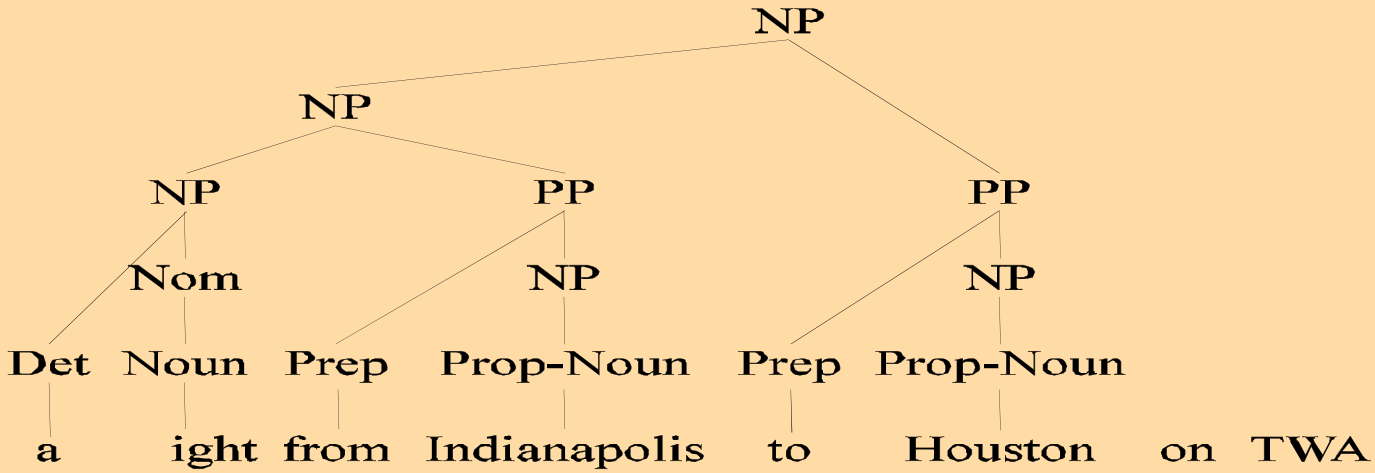

NP
Nom
Det Noun
a flight from Indianapolis to Houston on TWA


NP
Nom
Det
Noun
a
flight
from
Indianapolis
to
Houston
on
TWA

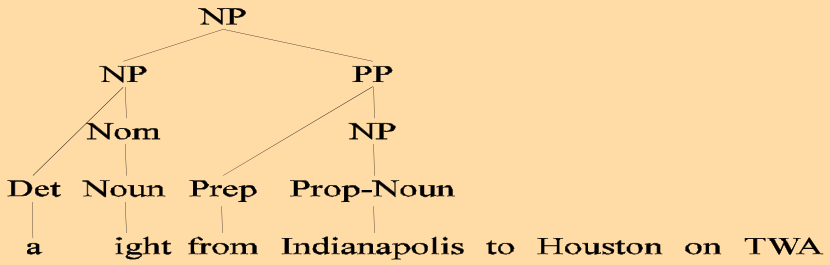

NP
NP
PP
Nom
NP
Det
Noun
Prep
Prop-Noun
a
flight
from
Indianapolis
to
Houston
on
TWA

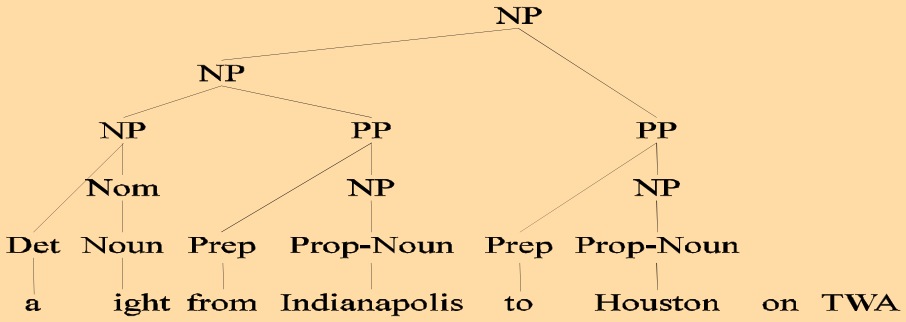

NP
Nom
Det Noun
a ight from Indianapolis to Houston on TWA

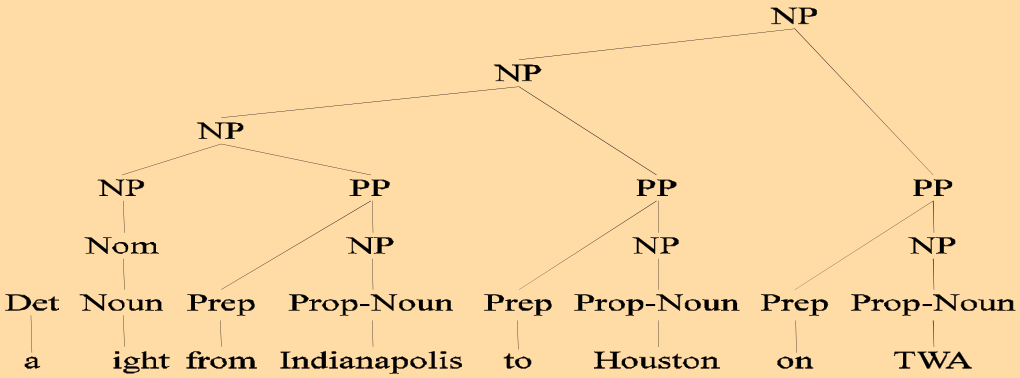

NP
NP PP
Nom NP
Det Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA


NP
NP
NP PP PP
Nom NP NP
Det Noun Prep Prop-Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA


NP
Nom
Det Noun
a ight from Indianapolis to Houston on TWA


NP
NP PP
Nom NP
Det Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA


NP
NP
NP PP PP
Nom NP NP
Det Noun Prep Prop-Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA


NP
NP
NP
NP PP PP PP
Nom NP NP NP
Det Noun Prep Prop-Noun Prep Prop-Noun Prep Prop-Noun
a ight from Indianapolis to Houston on TWA

Programación Dinámica

Necesitamos un método de parsing que llene alguna "*memoria*" (chart) con resultados parciales tal que:

- No repita trabajo.
- Evite la recursión izquierda.
- Encuentre todas las partes del árbol en un tiempo polinomial.

Programación Dinámica

- ✓ Existen muchos métodos de parsing top-down y bottom-up.
- ✓ Algunos métodos populares bottom-up basados en "*memoria*" (*chart parsing*):
  - Algoritmo CKY
  - Algoritmo de *Earley*
  - *etc*

Parsing CKY

- El algoritmo de *Cocke-Kasami-Younger (CKY)* es un tipo de ***chart parser***.
- Un ***chart*** es una tabla bi-dimensional donde se va "registrando" el trabajo realizado por el parser.
- El parser requiere que la gramática esté en forma normal (CNF), o sea, todas las reglas deben tener la forma (se limita el "ancho"):
  - **A → B C**
  - **A → w**

Parsing CKY

- ✓ Considere la regla **A → B C**
  - Si existe un A en la entrada entonces debe haber un B seguido por un C en la entrada.
  - Si A va desde i a j en la entrada entonces debe haber un k tal que i<k<j
    - Ej. B se separa de C en algún lugar.

Parsing CKY

- ✓ Construimos un *chart* de modo que un A vaya de i a j en la entrada y se registre en la celda [i,j] en la tabla.
- ✓ Así, un no-terminal que genera un string completo estará en la celda [0, n].
- ✓ Si construimos la tabla de forma *bottom-up*, sabremos que partes de A deben ir desde i a k y desde k a j.

Parsing CKY

- ✓ Para cada regla A → B C debemos buscar un B en [i,k] y un C en [k,j].
- ✓ En otras palabras,

  SI podría haber un A que abarca i,j

  y A → B C es una regla de la gramática

  ENTONCES

  debe existir un B en [i,k] y un C en [k,j] para algún i<k<j

- ✓ Luego, repetir los pasos sobre posibles valores de k

Algoritmo CKY

Dada una oración (string de palabras) y una gramática, el algoritmo se resume como:

```
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]}
```

Ejemplo:

Considere la siguiente CFG normalizada:

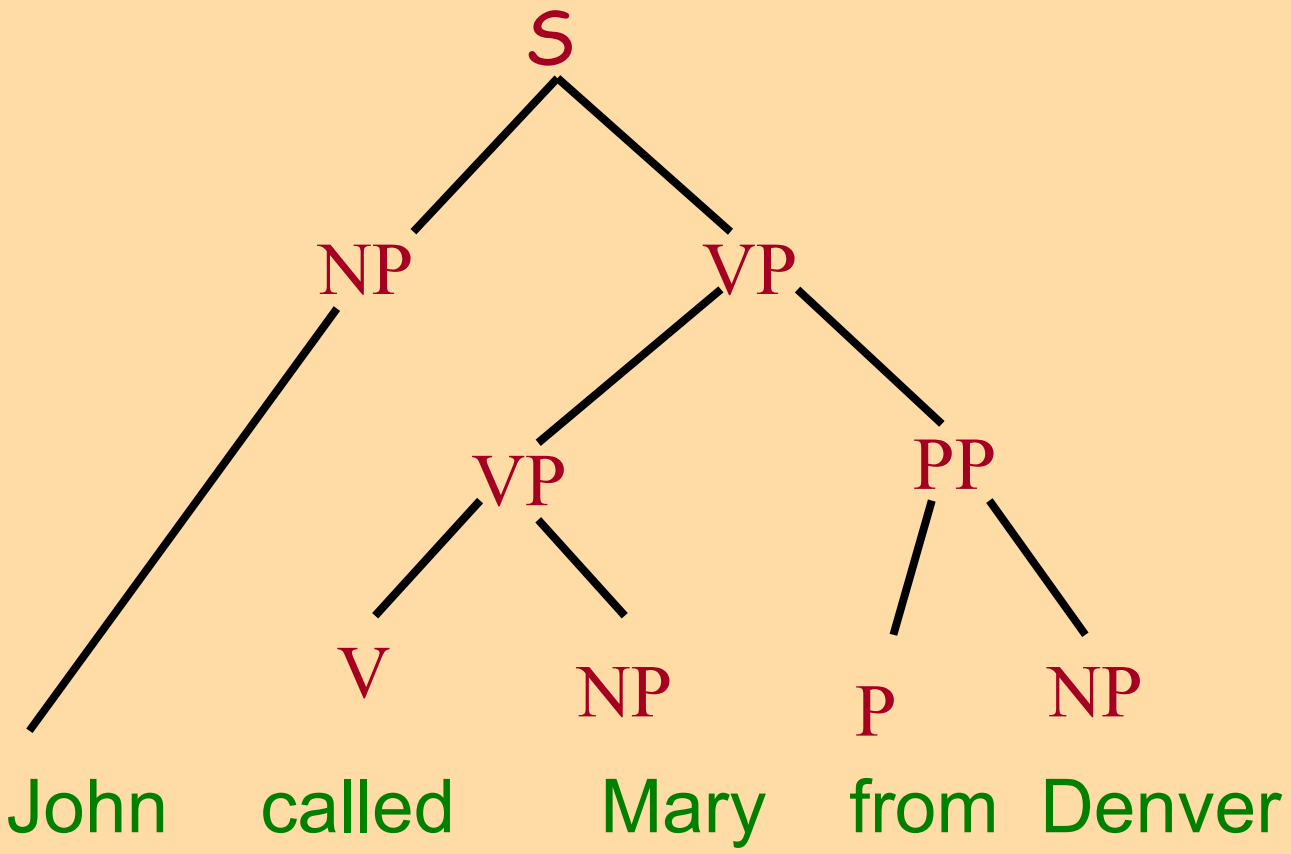
S → NP VP
VP → V NP
NP → NP PP
VP → VP PP
PP → P NP

NP → John
NP → Mary
NP → Denver
V → called
P → from

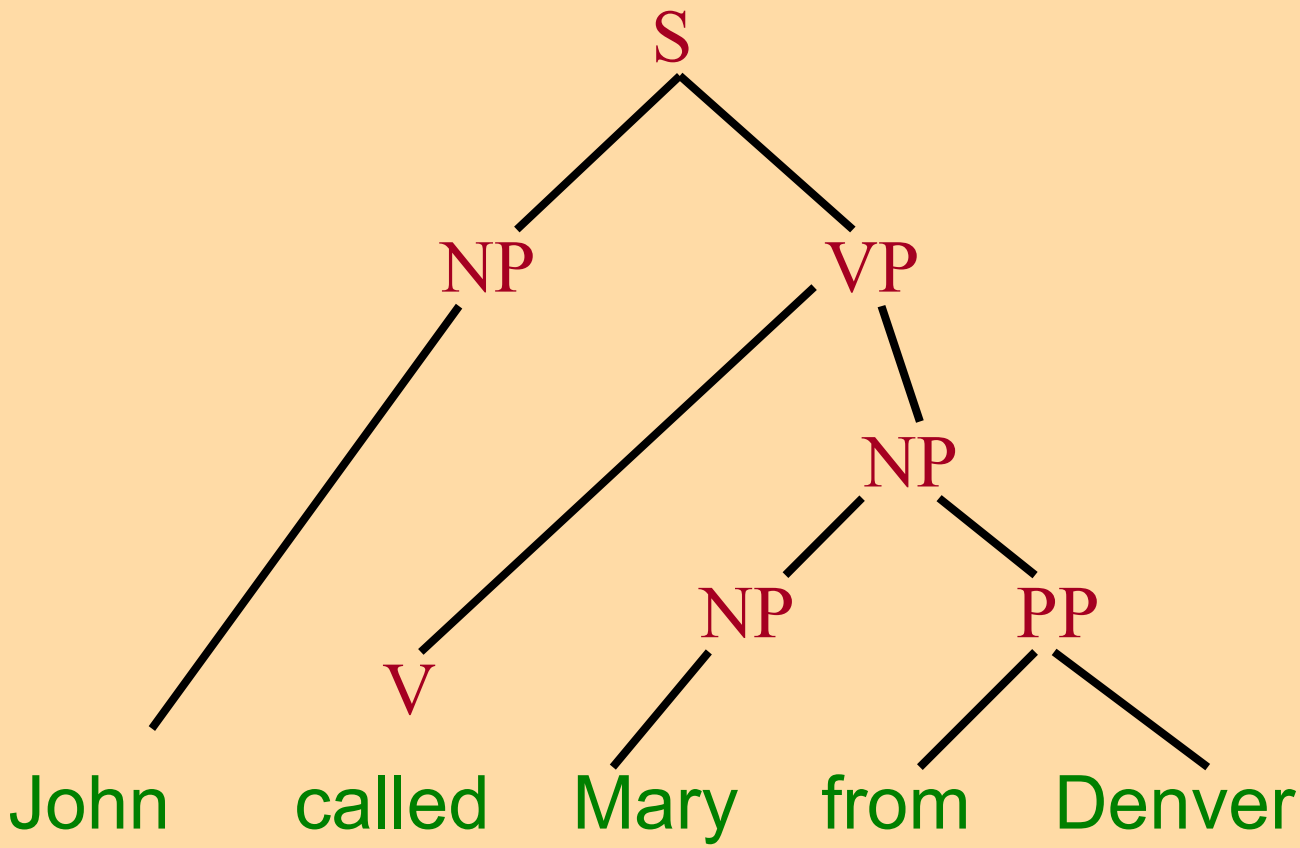
y la oración de entrada (string):

John called Mary from Denver

Parse Tree (1)



Parse Tree (2)



Analizando usando un *chart*

				NP
			P	Denver
		NP	from	
	V	Mary		
NP	called			
John				

Analizando usando un *chart*

				NP
			P	Denver
		NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

				NP
			P	Denver
	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

				NP
		X	P	Denver
	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

			PP →	NP
		X	P	Denver
	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

			PP	NP
		X	P	Denver
S →	VP	NP	from	
↓	V	Mary		
NP	called			
John				

Analizando usando un *chart*

			PP	NP
	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

		NP	PP	NP
	X		P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

		NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

	VP	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

	VP	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

	VP_1 VP_2	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Analizando usando un *chart*

S	VP_1 VP_2	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Finalmente…

S	VP	NP	PP	NP
X	X	X	P	Denver
S	VP	NP	from	
X	V	Mary		
NP	called			
John				

Ambigüedad

- ✓ Se resuelve ambigüedad: no necesariamente!
  - CKY genera múltiples estructuras S para la entrada [0,n].
  - Pero, el parser almacena eficientemente las sub-partes que son compartidas entre múltiples *parse trees*.
  - La presencia de un estado S (estado inicial) en el lugar correcto indica un reconocimiento exitoso.

Gramática Probabilística

- ✓ El número de *parse trees* posibles crece rápidamente con el largo de la entrada.
- ✓ Pero no todos los *parse trees* son igualmente útiles!!.
- ✓ En muchas aplicaciones, deseamos el "mejor" *parse tree*, ó los primeros mejores.
- ✓ Caso especial: "*mejor*" = "*más probable*"

CFG Probabilística

Una CFG probabilística (PCFG) es una CFG donde

- A cada regla ***r*** se le asigna una probabilidad ***p*(*r*)** entre **0** y **1**.
- Las probabilidades de las reglas con el mismo lado izquierdo suman **1**.

Ejemplo PCFG

Rule	Probability
S → NP VP	1
NP → Pronoun	1/3
NP → Proper-Noun	1/3
NP → Det Nominal	1/3
Nominal → Nominal PP	1/3
Nominal → Noun	2/3
VP → Verb NP	8/9
VP → Verb NP PP	1/9
PP → Preposition NP	1

Probabilidad de un *Parse Tree*

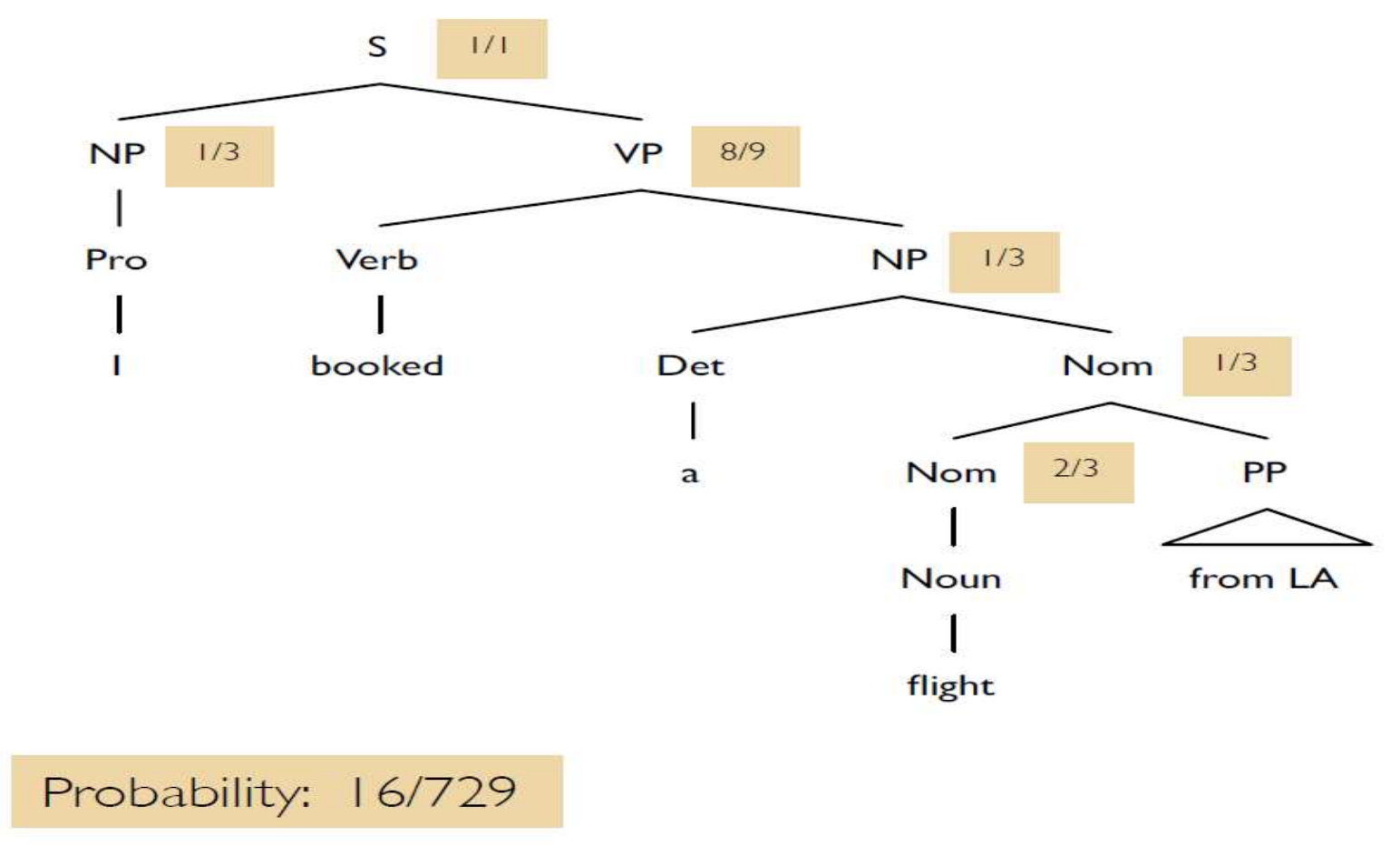
- ✓ Probabilidad de un *parse tree:*

  Producto de las probabilidades de las reglas que han sido usadas (*derivaciones*) para construir el *parse tree*.

- ✓ Las probabilidades se obtienen desde un *treebank.*
- ✓ Objetivo: Encontrar el *parse tree* de probabilidad máxima para una entrada.

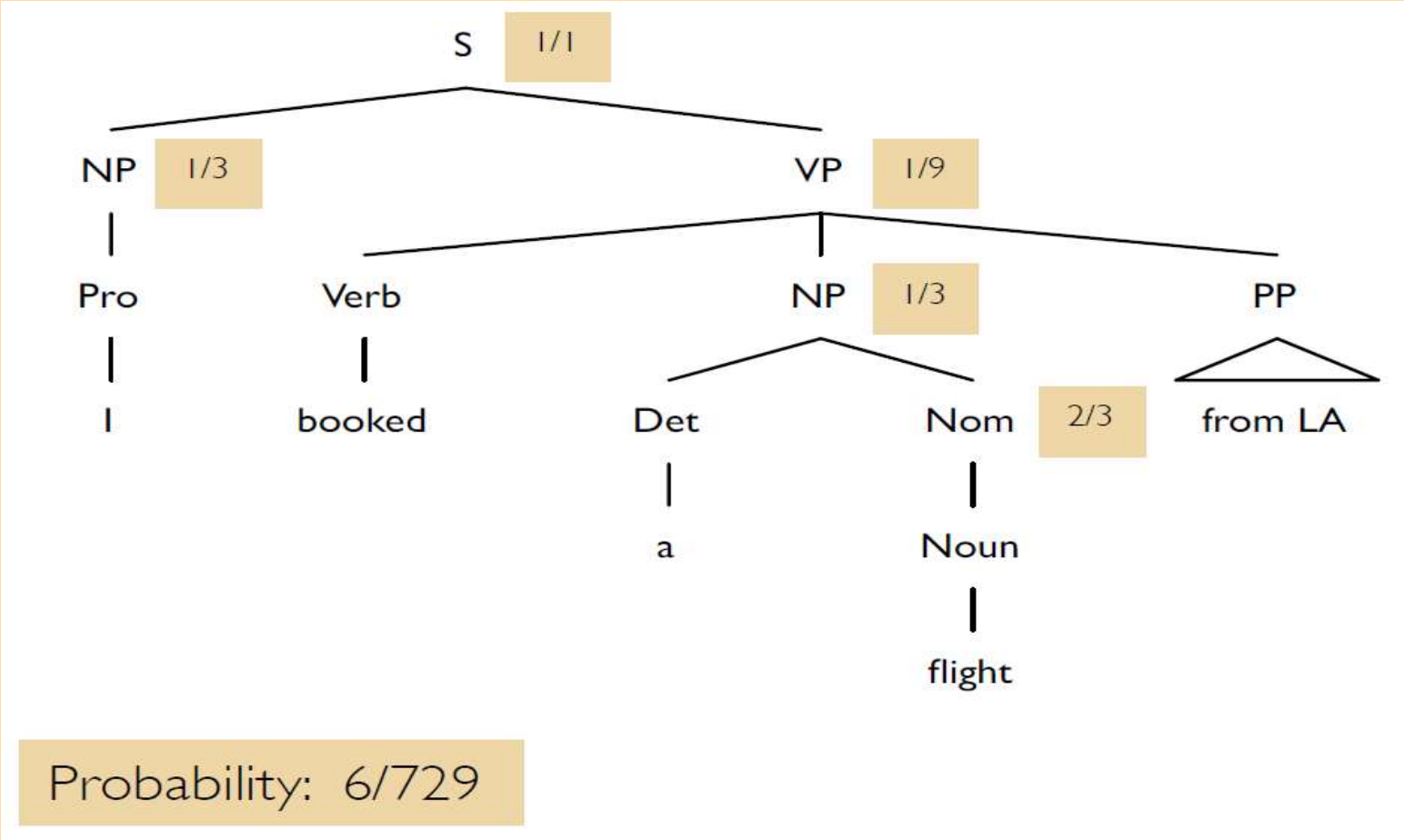
Probabilidad de *Parse Tree (1)*

Parse tree y probabilidad para la frase: "*I booked a flight from LA*"



Probabilidad de *Parse Tree (2)*

Parse tree y probabilidad para la frase: "*I booked a flight from LA*"



Resumen

- ✓ El análisis sintactico ó parsing es la tarea de construir automáticamente una estructura gramatical a partir de oraciones en lenguaje natural.
- ✓ Un parser utiliza conocimiento lingüístico en la forma de una gramática computacional para determinar la estructura de una oración en un lenguaje.
- ✓ Existen varias técnicas de parsing: top-down y bottom-up.
- ✓ Un parser debe dar cuenta de varios problemas lingüísticos: ambigüedad, recursión de reglas, búsqueda de parse trees, etc.


Tiempo de Ejercicios

Ejercicio Grupal

1. Cargue en *Python* (vía *Spyder*) lo siguiente:
   a) parsing1.py: funciones para parsing CFG.
   b) parsing.py: funciones para parsing probabilístico.
2. Revise la gramática de "aerolínea.cfg".
3. Ejecute (a) y vea que entrega al ingresar consultas como: "quiero un vuelo", luego "quiero un vuelo desde Santiago a Concepcion"
4. Revise la gramática de "aerolínea.pcfg".
5. Ejecute (a) y vea que entrega al ingresar consultas como: "quiero un vuelo" luego "quiero un vuelo desde Santiago a Concepcion"

¿Qué lenguaje reconoce la gramática aparte de los anteriores?