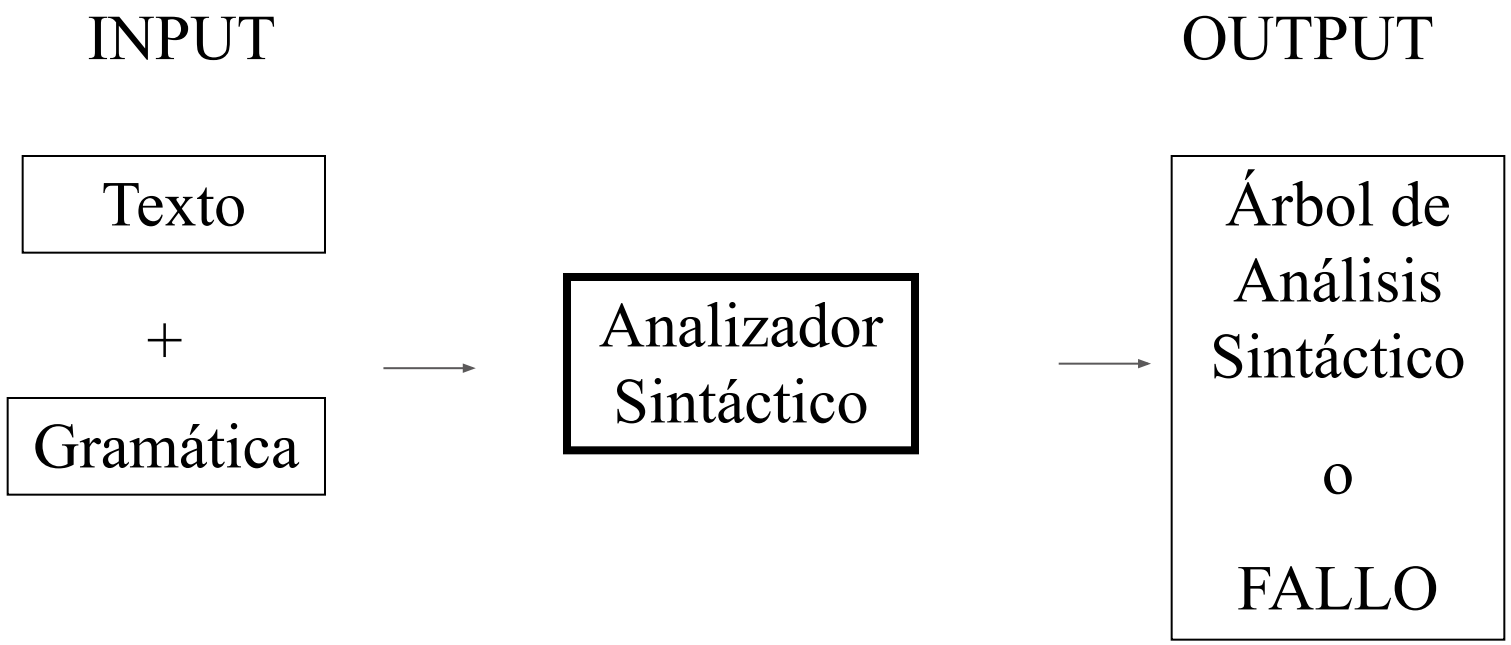
Introducción al Procesamiento de Lenguaje Natural

Grupo PLN – InCo

Análisis Sintáctico

Análisis Sintáctico



Esquema general de un analizador sintáctico

There are and can exist but two ways of investigating and discovering truth. The one hurries on rapidly from the senses and particulars to the most general axioms, and from them. . . derives and discovers the intermediate axioms. The other constructs its axioms from the senses and particulars, by ascending continually and gradually, till it finally arrives at the most general axioms.

Francis Bacon, *Novum Organum* Book I.19 (1620)

Métodos

Dos grandes clases de análisis sintáctico:

- Descendentes (top-down)

  Para una secuencia de palabras, construye un árbol de análisis sintáctico desde la raíz a las hojas

Métodos

Dos grandes clases de análisis sintáctico:

- Descendentes (top-down)

  Para una secuencia de palabras, construye un árbol de análisis sintáctico desde la raíz a las hojas

- Ascendentes (bottom-up)

  Para una secuencia de palabras, construye un árbol de análisis sintáctico desde las hojas a la raíz

Métodos

Dos grandes clases de análisis sintáctico:

- Descendentes (top-down)
  Para una secuencia de palabras, construye un árbol de análisis sintáctico desde la raíz a las hojas
- Ascendentes (bottom-up)
  Para una secuencia de palabras, construye un árbol de análisis sintáctico desde las hojas a la raíz
- Existen combinaciones

Comparación

Top-Down (Racionalista):

- Sólo genera árboles que comienzan en S
- Pierde tiempo con árboles que no son consistentes con la entrada

Comparación

Bottom-up (Empirista):

- Explora árboles que no llegarán a S
- Nunca sugieren árboles que no sean al menos localmente consistentes con la entrada

Ambigüedad

- Estructural
  - “Vi a un conejo con un telescopio” (PP attachment)
  - “Hombres y mujeres de 60 años pueden acceder al beneficio” (Coordinación)

- Local (POS-tags)

Ejemplo

Gramática que genera la oración: "*tomo un vuelo a Paris*"

O → GV
| GN GV
GN → Det Nom
| Det Nom Adj
| NomProp
| GN GP
| GN GP GP
GV → V
| V GN
| V GN GP
GP → Prep GN

Det → un
Nom → tomo | examen | vuelo
NomProp → París | Montevideo
Prep → a | de
V → tomo
Adj → directo

Parsing con programación dinámica

Tres métodos clásicos:

- Cocke-Kasami-Younger (CKY, 1965-1967)
- Earley (1970)
- Chart Parsing (Kaplan, 1973; Key, 1986)

Algoritmo CKY

- La gramática debe estar escrita acorde a la Forma Normal de Chomsky (FNC):
  - Todas las producciones son de la forma:
    - $X \rightarrow Y_1 Y_2$
    - $X \rightarrow a$

- Algoritmo para llevar a FNC:
  - eliminación de reglas que mezclan terminales y no terminales
  - eliminación de reglas unitarias
  - eliminación de reglas de largo mayor a dos

Algoritmo CKY

- Técnica bottom-up. A partir de un par de categorías (lado derecho de las reglas) generamos el lado izquierdo
- Como la gramática está en FNC, las combinaciones posibles son binarias
- Se puede utilizar una matriz bidimensional
- A partir de ella se puede construir el árbol sintáctico

Algoritmo CKY

Sea la siguiente gramática:

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

y consideremos w = baaba

queremos ver si $S \Rightarrow^* w$

Algoritmo CKY

Determina para cada variable $A \in V$ y $\forall i,j$ si $A \Rightarrow^* w_{ij}$ / w_{ij} es un substring de w de largo j que comienza en la posición i

La tira que se pretende reconocer es la w_{1n}

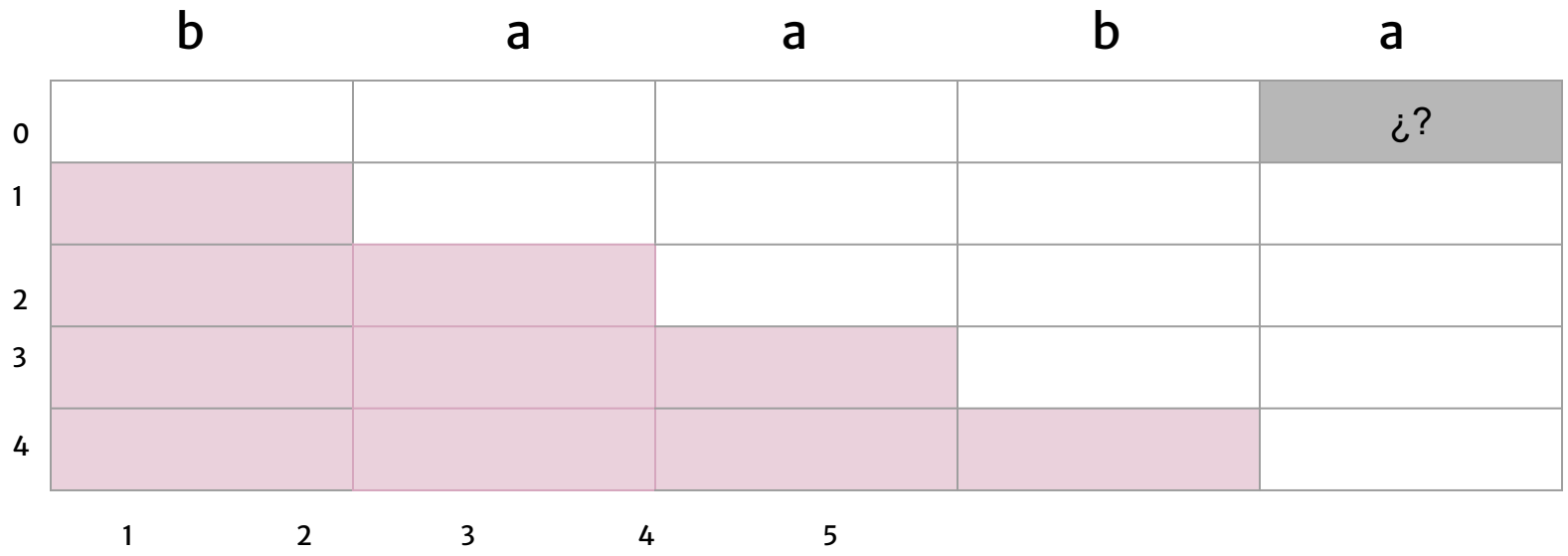
$V_{ij} = \{ A / A \Rightarrow^* w_{ij} \}$

ver que $w_{ij} = w_{ik} w_{kj}$

$V_{ij} = \{ A / A \rightarrow BC \in P \wedge B \in V_{ik}, C \in V_{kj} \}$ $k=i+1..j-1$

Si $S \in V_{0n}$ entonces $S \Rightarrow^* w$ y entonces $w \in L(G)$

Algoritmo CKY



Por lo tanto, si $S \in V_{05}$, entonces *baaba* $\in L(G)$

Algoritmo CKY

comienzo

para j = 1 to n hacer

$V_{j-1\,j} = \{ A / A \rightarrow a \in P \wedge a \text{ es el } j\text{-ésimo símbolo de } w \}$

para i = j−2 down 0 hacer

para k = i+1 to j−1 hacer

$V_{ij} = V_{ij} \cup \{ A / A \rightarrow BC \in P \wedge B \in V_{ik}, C \in V_{kj} \}$

fin

Algoritmo CKY

	b	a	a	b	a
0	B	S,A	∅	∅	¿?
1		A,C	B	B	S,C,A
2			A,C	S,C	B
3				B	S,A
4					A,C
	1	2	3	4	5

Por lo tanto, si $S \in V_{05}$, entonces *baaba* $\in L(G)$

Algoritmo CKY

Gramática que genera la oración: "*tomo un vuelo a Paris*"

O → GV
| GN GV
GN → Det Nom
| Det Nom Adj
| NomProp
| GN GP
| GN GP GP
GV → V
| V GN
| V GN GP
GP → Prep GN

Det → un
Nom → tomo | examen | vuelo
NomProp → París | Montevideo
Prep → a | de
V → tomo
Adj → directo

Algoritmo CKY

O → **tomo** |
| V GN
| **Aux1** GP
| GN GV

GN → Det Nom
| **Aux2** Adj
| **París** | **Montevideo**
| GN GP
| **Aux3** GP

GV → **tomo**
| V GN
| **Aux1** GP

GP → Prep GN

Analizar con CKY: "***tomo un vuelo a París***"

Aux1 → V GN
Aux2 → Det Nom
Aux3 → GN GP

Det → un
Nom → tomo | examen | vuelo
NomProp → París | Montevideo
Prep → a | de
V → tomo
Adj → directo

Algoritmo CKY

	tomo (1)	un (2)	vuelo (3)	a (4)	París (5)
0	V, GV,O,Nom	∅	GV, Aux1, O	∅	O, GV, Aux1
1		Det	GN, Aux2	∅	GN, Aux3
2			Nom	∅	∅
3				Prep	GP
4					GN, NomProp

(0,1) tomo	O GV Nom V
(1,2) un	Det
(0,2) tomo un	
(2,3) vuelo	Nom
(1,3) un vuelo	GN Aux2
(0,3) tomo un vuelo	O GV Aux1
(3,4) a	Prep
(2,4) vuelo a	
(1,4) un vuelo a	
(0,4) tomo un vuelo a	
(4,5) Paris	NomProp GN
(3,5) a París	GP
(2,5) vuelo a París	
(1,5) un vuelo a París	GN Aux3
(0,5) tomo un vuelo a París	O GV Aux1

Como $O \in V_{05}$ la frase es generada

Problemas

- Se necesita reescribir la gramática
- ¿Reconocedor o parser?
- Si bien reduce el costo de reconocer a tiempo polinomial, calcular todos los árboles puede llevar un tiempo exponencial

Algoritmo de Earley

- No requiere gramáticas en FNC
- Enfoque Top-Down
- Llena una tabla (chart) en una sola recorrida sobre las N palabras a reconocer:
  - Largo de la tabla es N+1
  - Las entradas (celdas) en la tabla representan:
    - Constituyentes completos y sus posiciones
    - Constituyentes en reconocimiento
    - Constituyentes predichos

Algoritmo de Earley

En cada celda, se van colocando los denominados **estados** y se representan con **reglas con punto** (*dotted rules*)

Las posiciones son los "huecos" entre las palabras

Tres casos posibles:

O → • GV	predecimos un GV
GN → Det • Nom	un GN está siendo reconocido
GV → V GN •	un GV fue encontrado

Algoritmo de Earley

Es decir:

- en cada entrada en la tabla se coloca para cada posición en la oración, la lista de estados con los árboles parciales hasta el momento

- a su vez estos estados contienen información de 3 tipos:
  - un subárbol (regla)
  - el progreso de aplicación de la regla para completar el subárbol (indicado por un punto (***dot***) •)
  - la posición del subárbol en relación a la entrada ([i,j])

Algoritmo de Earley

Necesitamos relacionar los estados con la entrada:

O → • GV [0,0] predecimos un GV al comienzo de la oración

GN → Det • Nom [1,2] un GN está siendo reconocido; el Det va de 1 a 2 y ya fue reconocido

GV → V GN • [0,3] un GV fue encontrado y va de 0 a 3

Algoritmo de Earley

Para iniciar el proceso se agrega a la primera celda de la tabla un estado "comodín":

$\gamma \rightarrow \bullet O \ [0,0]$ (O: símbolo inicial)

Cuando se termina de procesar la lista de estados de la última celda de la tabla, si en esa celda tenemos un estado:

$O \rightarrow \alpha \bullet \ [0,N]$

y por lo tanto

$\gamma \rightarrow O \bullet \ [0,N]$

entonces la entrada **α** fue reconocida.

Algoritmo de Earley

Gramática que genera la oración: "*tomo un vuelo a Paris*"

O → GV
| GN GV

GN → Det Nom
| NomProp
| GN GP
| GN GP GP

GV → V
| V GN
| V GN GP

GP → Prep GN

Det → un

Nom → tomo | examen | vuelo

NomProp → París | Montevideo

Prep → a | de

V → tomo

Adj → directo

Algoritmo de Earley

- Se recorre la tira de entrada de izquierda a derecha.
- En cada celda, se procesa cada estado de la lista de estados de la celda.
- El procesamiento de un estado consiste en la aplicación de uno de los tres operadores disponibles.
- El procesamiento de un estado genera la incorporación de estados nuevos (si estos no existen) al final de la lista de estados de la celda actual o de la celda siguiente dependiendo de la operación.
- El algoritmo siempre avanza agregando estados, nunca se eliminan estados ni se retrocede a una celda anterior.
- En cada paso, aplicar uno de los siguientes operadores:

Algoritmo de Earley

Predecir *(predictor)*

- Aplica cuando a la derecha del ***dot*** hay un **no terminal**, que no sea una categoría léxica (POS).
- Crea – en la misma entrada de la tabla – estados nuevos al final, que representan las posibles expansiones del no terminal según la gramática.
- Los nuevos estados comienzan y terminan en la posición en la que termina el estado que los generó.

Ejemplo, para el estado: O → . GV [0,0]

se agregan:

GV → . V [0,0]

GV → . V GN [0,0]

GV → . V GN GP [0,0]

en la misma celda de la tabla

Algoritmo de Earley

Buscar *(scanner)*

- Aplica cuando a la derecha del ***dot*** hay un **POS** y *matchea* las predicciones de palabras para el POS con las palabras de la entrada.
- Crea un nuevo estado que avanza el ***dot*** una posición pero en la siguiente entrada (celda) de la tabla.
- Este nuevo estado comienza donde termina el estado que lo generó y termina una posición después.

Ejemplo para el estado: GV → . V GN [0 , 0]

y la entrada: *tomo un examen* (en la posición 0 hay un V)

se agrega:

V → tomo . [0 , 1]

en la celda siguiente de la tabla

Algoritmo de Earley

Completar *(completer)*

- Se aplica a todo estado que tenga el ***dot*** al final de la regla (estado completo). Es decir, cuando un estado está completo, busca cuáles reglas estaban esperando el constituyente que fue completado (y en la posición en la que apareció).
- El nuevo estado, con el ***dot*** después de la categoría creada, es agregado en la entrada actual.
- Los nuevos estados comienzan en la misma posición que los estados previos y terminan en donde termina el estado que los generó.

Ejemplo para el estado: GN → Det Nom . [1 , 3]

a partir de los estados (generados previamente): GV → V . GN [0,1]

GV → V . GN GP [0,1]

se agregan: GV → V GN . [0,3]

GV → V GN . GP [0,3]

en la misma celda de la tabla

Algoritmo de Earley

```
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)

  procedure PREDICTOR((A → α • B β, [i,j]))
     for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
         ENQUEUE((B → • γ, [j,j]), chart[j])
     end

  procedure SCANNER((A → α • B β, [i,j]))
     if B ⊂ PARTS-OF-SPEECH(word[j]) then
         ENQUEUE((B → word[j], [j,j+1]), chart[j+1])

  procedure COMPLETER((B → γ •, [j,k]))
     for each (A → α • B β, [i,j]) in chart[j] do
         ENQUEUE((A → α B • β, [i,k]), chart[k])
     end

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

Algoritmo de Earley

O → GV
| GN GV

GN → Det Nom
| NomProp
| GN GP
| GN GP GP

GV → V
| V GN
| V GN GP

GP → Prep GN

Det → un
Nom → tomo | examen | vuelo
NomProp → París | Montevideo
Prep → a | de
V → tomo
Adj → directo

Analizar con Earley: "***tomo un vuelo a Paris***"

$_0$tomo $_1$un $_2$vuelo $_3$a $_4$Paris $_5$

“tomo un vuelo a París”

chart[0]

γ –> • O	[0,0]	Dummy
O –> • GV	[0,0]	Predictor
O –> • GN GV	[0,0]	Predictor
GV –> • V	[0,0]	Predictor
GV –> • V GN	[0,0]	Predictor
GV –> • V GN GP	[0,0]	Predictor
GN –> • Det Nom	[0,0]	Predictor
GN –> • NomProp	[0,0]	Predictor
GN –> • GN GP	[0,0]	Predictor
GN –> • GN GP GP	[0,0]	Predictor

“tomo un vuelo a París”

chart[0]

γ –> · O	[0,0]	Dummy
O –> · GV	[0,0]	Predictor
O –> · GN GV	[0,0]	Predictor
GV –> · V	[0,0]	Predictor
GV –> · V GN	[0,0]	Predictor
GV –> · V GN GP	[0,0]	Predictor
GN –> · Det Nom	[0,0]	Predictor
GN –> · NomProp	[0,0]	Predictor
GN –> · GN GP	[0,0]	Predictor
GN –> · GN GP GP	[0,0]	Predictor

chart[1]

V –> tomo ·	[0,1]	Scanner
GV –> V ·	[0,1]	Completer
GV –> V · GN	[0,1]	Completer
GV –> V · GN GP	[0,1]	Completer
O –> GV ·	[0,1]	Completer
GN –> · Det Nom	[1,1]	Predictor
GN –> · NomProp	[1,1]	Predictor
GN –> · GN GP	[1,1]	Predictor
GN –> · GN GP GP	[1,1]	Predictor
γ –> O ·	[0,1]	Completer

“tomo un vuelo a París”

chart[2]

Det -> un ·	[1,2]	Scanner
GN -> Det · Nom	[1,2]	Completer

chart[3]

Nom -> vuelo ·	[2,3]	Scanner
GN -> Det Nom ·	[1,3]	Completer
GV -> V GN ·	[0,3]	Completer
GV -> V GN · GP	[0,3]	Completer
GN -> GN · GP	[1,3]	Completer
GN -> GN · GP GP	[1,3]	Completer
O -> GV ·	[0,3]	Completer
γ -> O ·	[0,3]	Completer
GP -> · Prep GN	[1,3]	Predictor

chart[4]

Prep -> a ·	[3,4]	Scanner
GP -> Prep · GN	[3,4]	Completer
GN -> · Det Nom	[4,4]	Predictor
GN -> · NomProp	[4,4]	Predictor
GN -> · GN GP	[4,4]	Predictor
GN -> · GN GP GP	[4,4]	Predictor

“tomo un vuelo a París”

chart[5]		
NomProp–> París ·	[4,5]	Scanner
GN –> NomProp ·	[4,5]	Completer
GN –> GN · GP	[4,5]	Completer
GN –> GN · GP GP	[4,5]	Completer
GP –> Prep GN ·	[3,5]	Completer
GP –> · Prep GN	[5,5]	Predictor
GN –> GN GP ·	[1,5]	Completer
GN –> GN GP · GP	[1,5]	Completer
GV –> V GN ·	[0,5]	Completer
GV –> V GN · GP	[0,5]	Completer
O –> GV ·	[0,5]	Completer
γ –> O ·	**[0,5]**	**Completer**
GV –> V GN GP ·	[0,5]	Completer

¿Reconocedor o parser?

chart[1]		
V -> tomo ·	[0,1]	Scanner
chart[2]		
Det -> un ·	[1,2]	Scanner
chart[3]		
Nom -> vuelo ·	[2,3]	Scanner
GN -> Det Nom ·	[1,3]	Completer
char[4]		
Prep -> a ·	[3,4]	Scanner
chart[5]		
NomProp -> París ·	[4,5]	Scanner
GN -> NomProp ·	[4,5]	Completer
GP -> Prep GN ·	[3,5]	Completer
GN -> GN GP ·	[1,5]	Completer
GV -> V GN ·	[0,5]	Completer
O -> GV ·	[0,5]	Completer
γ -> O ·	**[0,5]**	**Completer**

Referencias

- J.Martin & D.Jurafsky 2019. S*peech and Language Processing*. Third edition Capítulo 13