PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

FACULTAD DE LETRAS

Introducción a la lingüística computacional

César Antonio Aguilar

Facultad de Lenguas y Letras

26/10/2017

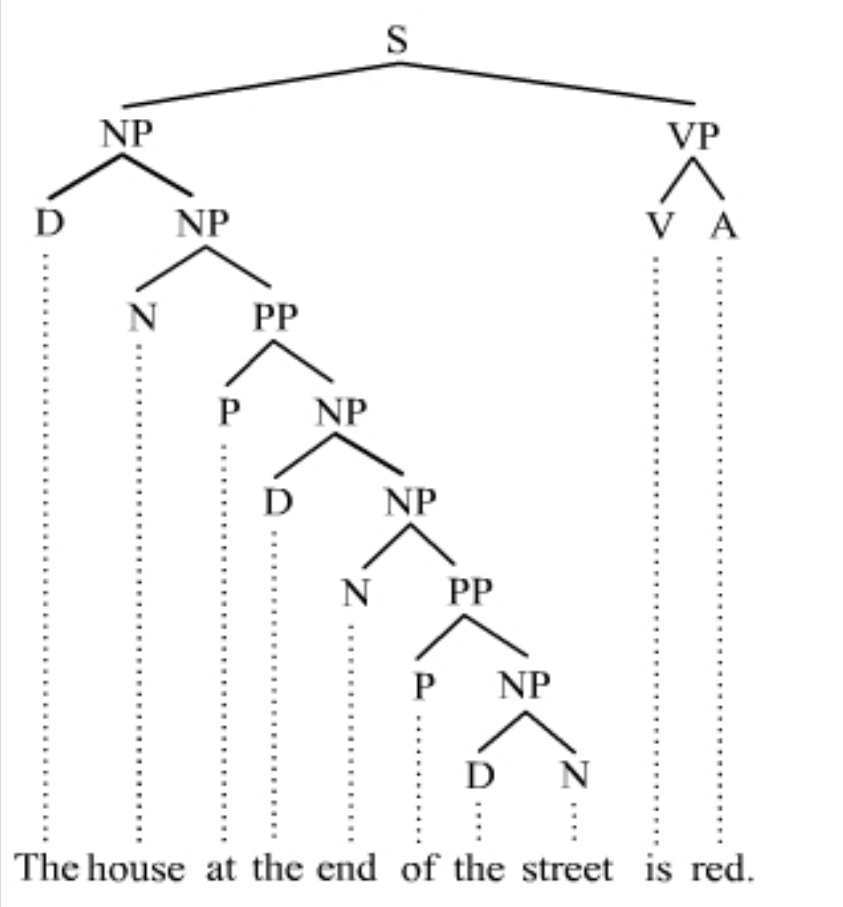
Cesar.Aguilar72@gmail.com

Integrando sintaxis y probabilidades (1)

En esta sesión vamos a tratar de resumir lo que hemos visto sobre la relación que establece el análisis sintáctico con métodos probabilísticos. Así, empecemos por recordar qué es una frase:

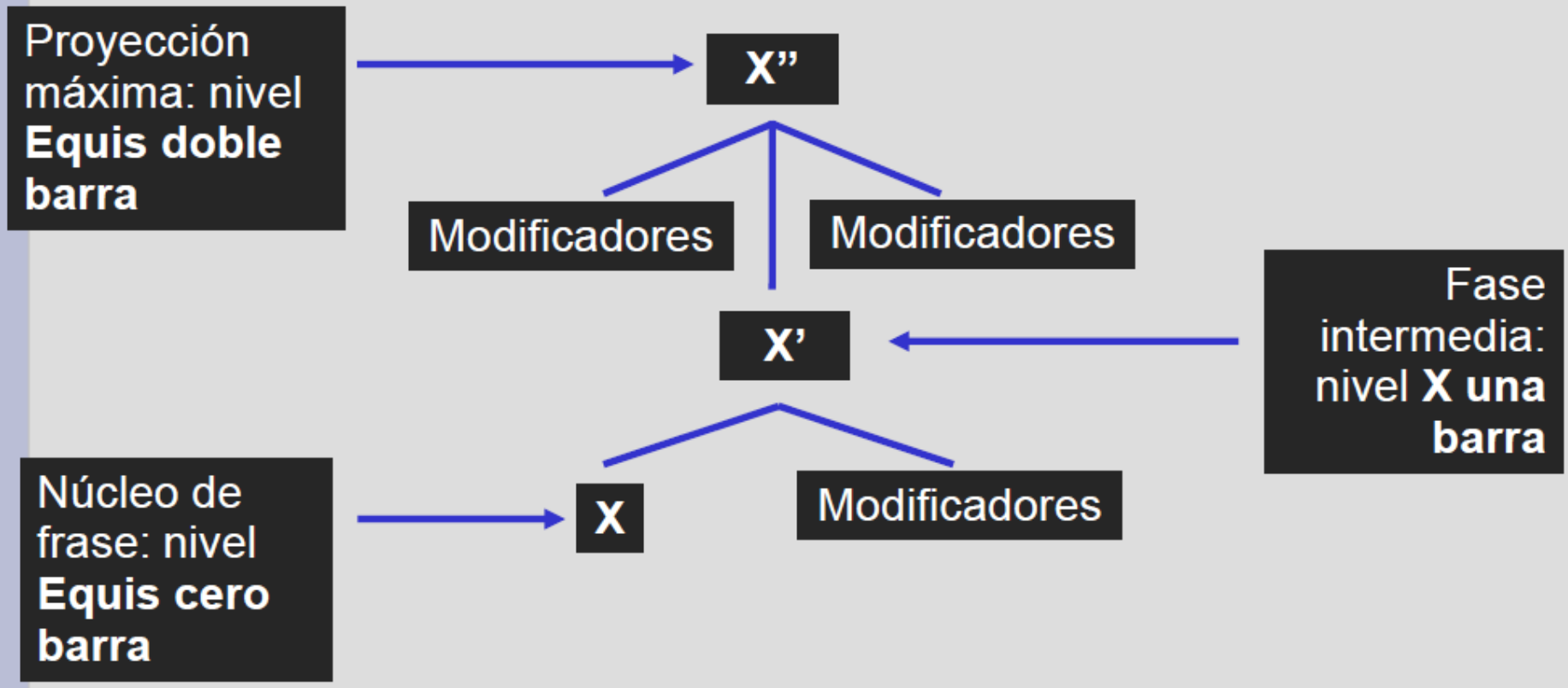
Conjunto de palabras que conforman secuencias específicas configuradas en torno a una serie de reglas.

Dicha configuración puede ser esquematizada a través de un árbol sintáctico, p.e.:



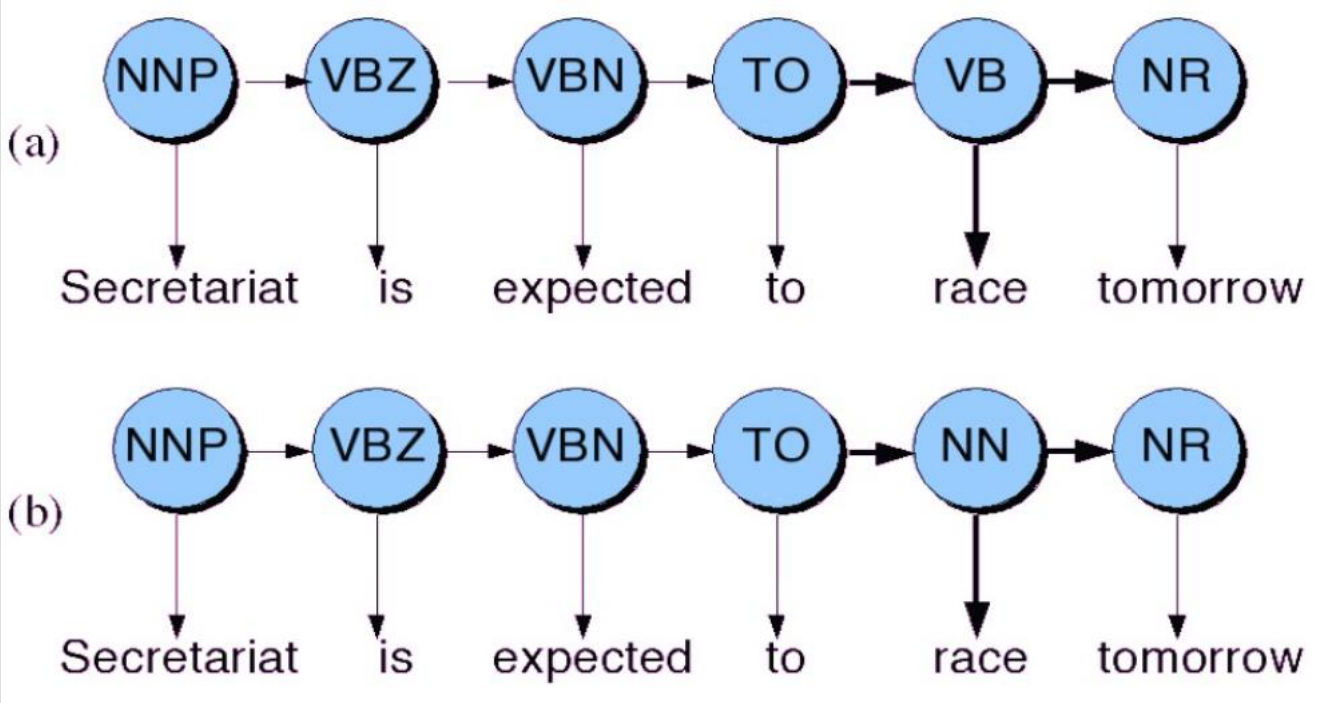
Integrando sintaxis y probabilidades (2)

¿Cómo se configura un árbol sintáctico? Podemos explicar esto usando el siguiente esquema:



Integrando sintaxis y probabilidades (3)

Cuando decimos que las palabras son los ladrillos que construyen frases, ¿podríamos pensar que tales combinaciones pueden ser regulares, de modo que resultan patrones *predecibles*? Se supone que sí, y para eso nos sirven los *n*-gramas, ¿recuerdan?:



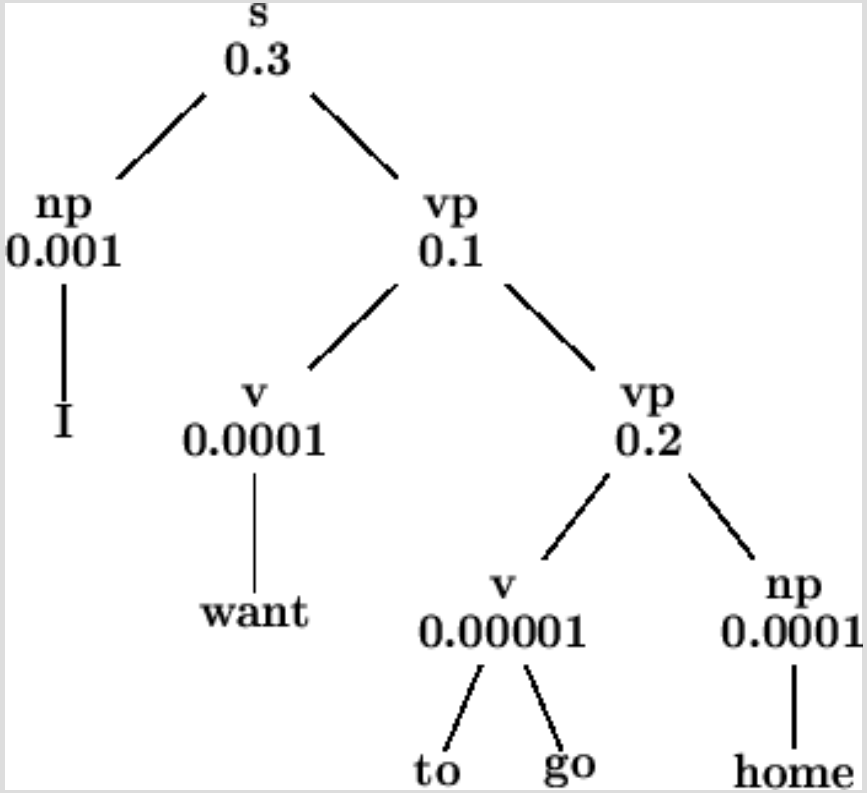
En este ejemplo, ¿por qué *to race* era mejor candidato a verbo que a nombre?

¿Cómo funciona? (1)

Si lo anterior ocurre entre palabras, ¿también ocurre entre estructuras sintácticas, que al final son secuencias de palabras? Podríamos suponer que sí.

Digamos entonces que la diferencia ahora es establecer probabilidades que nos permitan determinar qué estructura de frase es más probable de construir, si tengo un conjunto determinado de palabras, además de una serie de reglas de combinación específicas.

Esto es justo lo que quieren calcular las gramáticas probabilísticas.



¿Cómo funciona? (2)

Para entender mejor esta clase de análisis, analicemos lo siguiente: supongamos que toda gramática libre de contexto (o **CFG**) de acuerdo con la concepción planteada por Chomsky (denominada Forma normal de Chomsky, **CNF**). Si es esto, deben respetar dos secuencias de producción básicas:

Expansión de un No-terminal a dos no-terminales:

$$A \rightarrow B\ C$$

Expansión de un No-terminal a un terminal:

$$A \rightarrow a$$

¿Cómo funciona? (3)

De acuerdo con esta regla, podemos proponer la siguiente gramática:

Alice called Bob from Cardiff

Context-Free Grammar:

$S \rightarrow NP\ VP$	$NP \rightarrow$ Alice
$NP \rightarrow NP\ PP$	$NP \rightarrow$ Bob
$VP \rightarrow V\ NP$	$NP \rightarrow$ Cardiff
$VP \rightarrow VP\ PP$	$V \rightarrow$ called
$PP \rightarrow P\ NP$	$P \rightarrow$ from

¿Cómo funciona? (4)

Si pensamos en generar un árbol sintáctico para describir la composición de esta oración, ¿por dónde empezamos? Veamos:

				Cardiff
			from	
		Bob		
	called			
Alice				

Aquí tenemos la secuencia que siguen los componentes de nuestra oración. Hipotéticamente, es fácil reconocer una NP por cada nombre que tenemos en ella.

¿Cómo funciona? (5)

Identifiquemos entonces nombres:

				Cardiff
			from	
		Bob		
NP	called			
Alice				

Base case

Y así reconocemos al resto de los componentes de nuestra oración:

				NP
			P	**Cardiff**
		NP	from	
	V	Bob		
NP	called			
Alice				

Base case

¿Cómo funciona? (6)

Ahora, ¿con qué componentes se puede relacionar una frase nominal?:

				NP
			P	Cardiff
		NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

Para identificar esta relación, consideremos la presencia de patrones recursivos, previamente establecidos en nuestra gramática.

¿Cómo funciona? (7)

Así, podemos decir que una frase verbal admite como complemento una frase nominal, y a su vez tal secuencia se puede relacionar con una frase nominal previa:

				NP
			P	Cardiff
	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

¿Cómo funciona? (8)

Por otro lado, cuando identificamos una preposición, podemos tener un buen indicio para reconocer una frase prepositiva (PP):

			PP	NP
		X	P	Cardiff
	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

¿Cómo funciona? (9)

Otra secuencia posible de reconocer es: NP + VP + NP:

			PP	NP
		X	P	Cardiff
S	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

¿Cómo funciona? (10)

Una similar a la anterior es: NP + PP + NP:

		NP	PP	NP
	X	X	P	Cardiff
S	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

¿Cómo funciona? (11)

Como comentamos en algún momento, a partir de segmentos simples podemos construir otros más complejos, p.e.:

	VP_1	NP	PP	NP
X	X	X	P	Cardiff
S	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

¿Cómo funciona? (12)

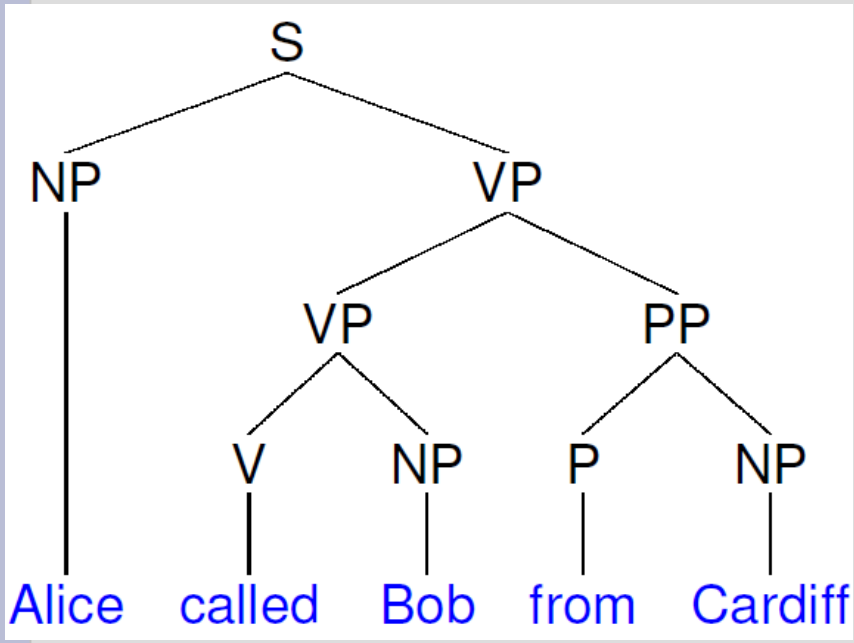
Con todo, no siempre es fácil determinar cuál es el patrón canónico a seguir para la constitución de una oración, o para hacer el recorrido de su estructura arbórea. Un camino alterno es:

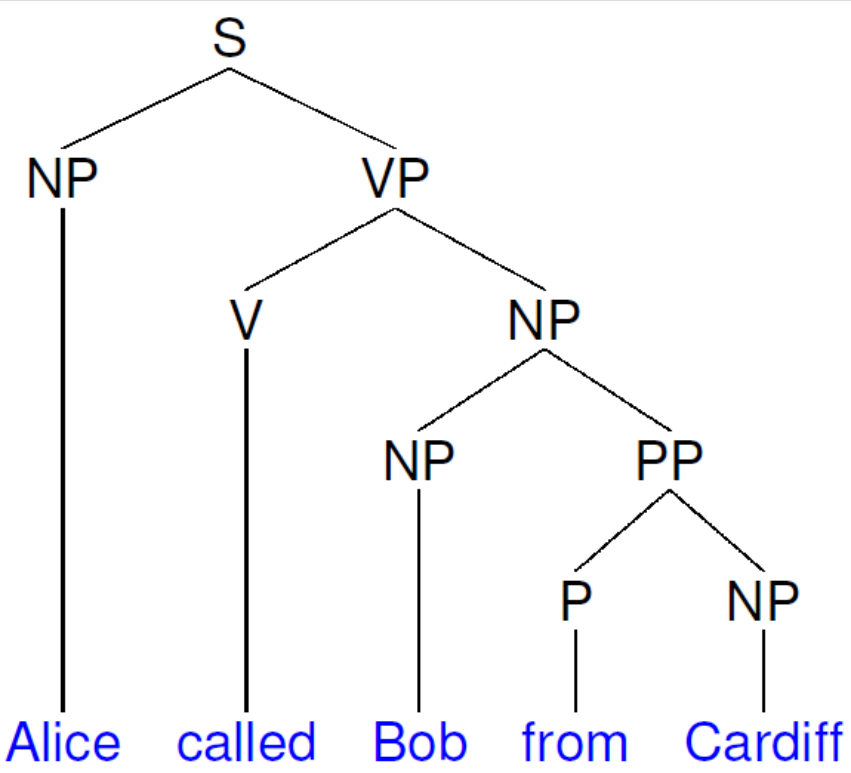
	VP_2	NP	PP	NP
X	X	X	P	Cardiff
S	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

¿Cómo funciona? (13)

Con todo, no siempre es fácil determinar cuál es el patrón canónico a seguir para la constitución de una oración, o para hacer el recorrido de su estructura arbórea. Un camino alterno es:





¿Cómo funciona? (14)

Ya tenemos identificadas las estructuras arbóreas que subyacen en nuestra oración, e incluso sospechamos cuál es el árbol canónico esperable. Ahora, para completar el análisis, determinemos cuáles son las probabilidades posibles que tenemos para generar uno de estos dos árboles. Podemos hacerlo considerando:

- A probabilistic context-free grammar augments each rule in a CFG with a conditional probability p
  $A \rightarrow \alpha \quad (p)$
- This probability is the probability that given non-terminal A it will be expanded to the sequence α; written as $P(A \rightarrow \alpha | A)$ or $P(A \rightarrow \alpha)$
- Probability of a parse tree is the product of the rule probabilities used to construct the parse

¿Cómo funciona? (15)

Desarrollemos algunas reglas, p.e.:

- Consider the rule:

$$\mathrm{S} \rightarrow \mathrm{NP\ VP}$$

  Then the probability of S is the product of the rule probability and the probability of each subtree:

$$P(S) = P(\mathrm{S} \rightarrow \mathrm{NP\ VP}) \cdot P(\mathrm{NP}) \cdot P(\mathrm{VP})$$

- We are doing bottom-up parsing... so we already know the subtree probabilities $P(\mathrm{NP})$ and $P(\mathrm{VP})$

¿Cómo funciona? (16)

Desarrollemos algunas reglas, p.e.:

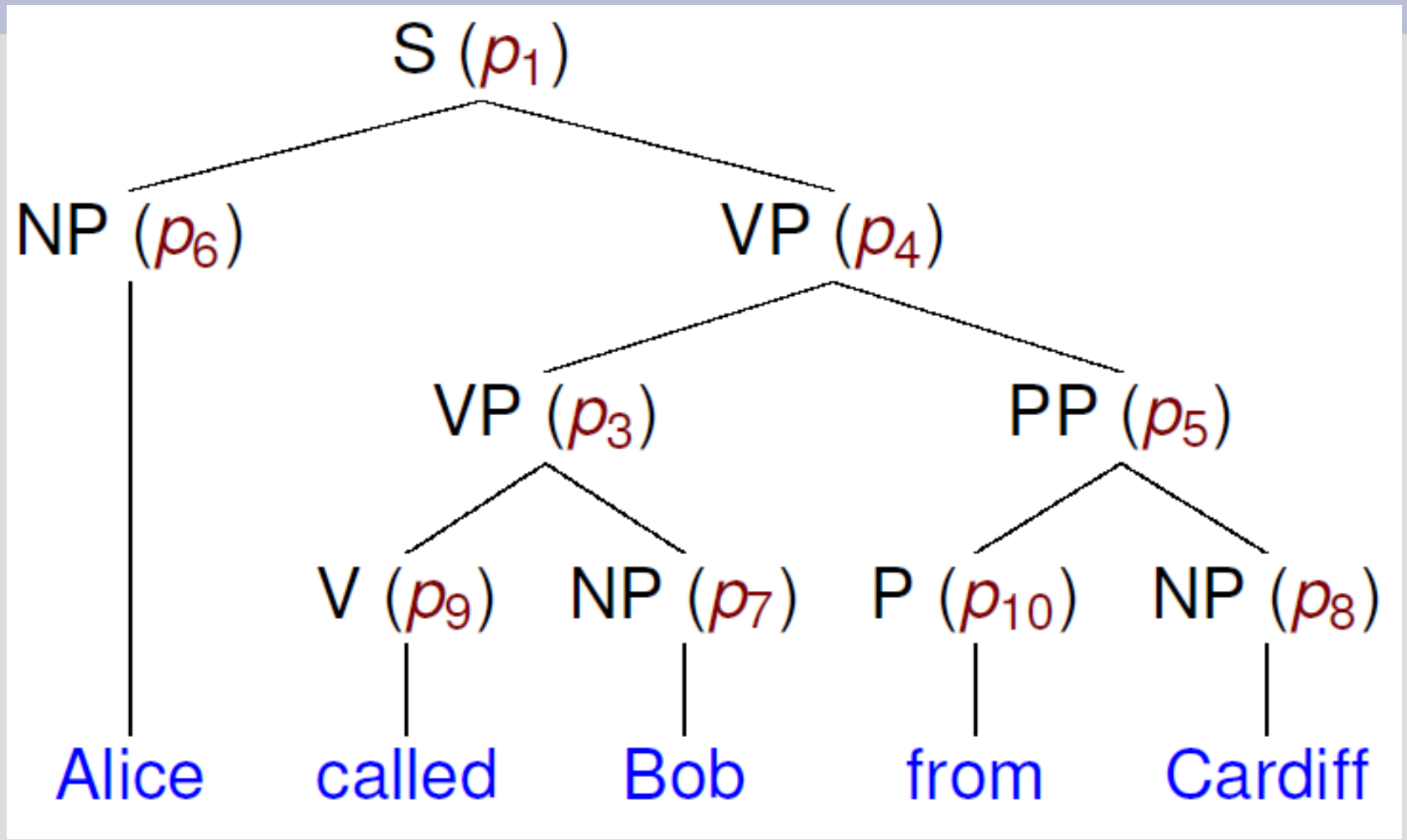
Alice called Bob from Cardiff

Context-Free Grammar:

$S \rightarrow NP\ VP \quad (p_1)$	$NP \rightarrow \text{Alice} \quad (p_6)$
$NP \rightarrow NP\ PP \quad (p_2)$	$NP \rightarrow \text{Bob} \quad (p_7)$
$VP \rightarrow V\ NP \quad (p_3)$	$NP \rightarrow \text{Cardiff} \quad (p_8)$
$VP \rightarrow VP\ PP \quad (p_4)$	$V \rightarrow \text{called} \quad (p_9)$
$PP \rightarrow P\ NP \quad (p_5)$	$P \rightarrow \text{from} \quad (p_{10})$

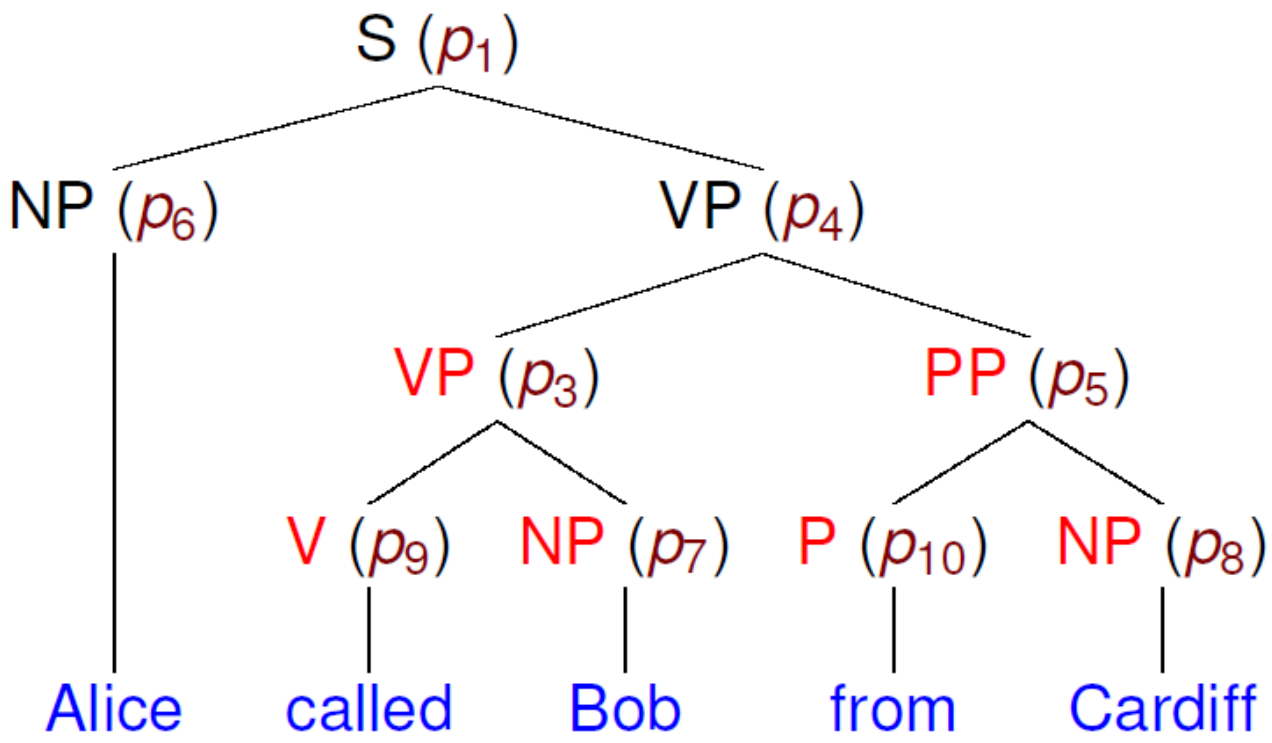
¿Cómo funciona? (17)

Desarrollemos algunas reglas, p.e.:



¿Cómo funciona? (18)

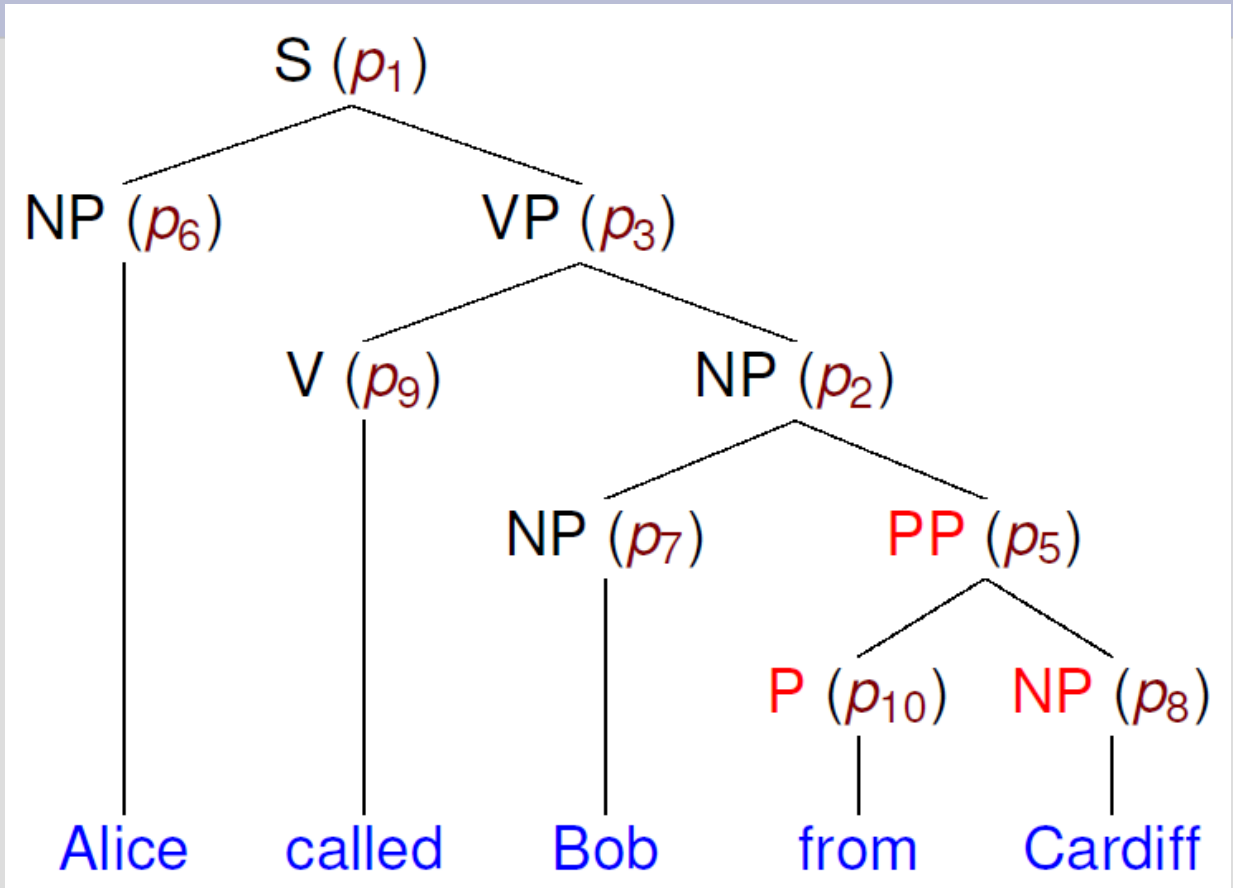
Ahora, ¿cómo establecer esta secuencia de probabilidades? Para el primer árbol tenemos:



$$P(T1, S) = p_1(p_6)(p_4(p_3(p_9 p_7))(p_5(p_{10} p_8)))$$

¿Cómo funciona? (19)

Y para el segundo tenemos:



$$P(T2, s) = p_1(p_6)(p_3(p_9)(p_2(p_7)(p_5(p_{10}p_8))))$$

¿Cómo funciona? (20)

Vamos a elegir la opción más canónica realizando la siguiente operación:

Choose tree 1 if $P(T1, S)/P(T2, S) > 1$.

$$\frac{P(T1,S)}{P(T2,S)} = \frac{p_1(p_6)(p_4(p_3(p_9p_7))(p_5(p_{10}p_9)))}{p_1(p_6)(p_3(p_9)(p_2(p_7)(p_5(p_{10}p_8))))} = \frac{p_4}{p_2}$$

Lo que estamos calculando aquí se puede ver como un problema de decisión, esto es: dada una secuencia de nodos posibles dentro de un árbol sintáctico, podemos preguntarnos *cuál es el camino más corto para recorrer tal árbol*.

Si deducimos esto, podríamos decir entonces que este camino representa el patrón canónico, precisamente porque es *la primera opción que cualquiera podría considerar*.

¿Cómo funciona? (21)

Para desarrollar esto podemos usar un algoritmo llamado **CYK** (Cocke–Younger–Kasami Algorithm), el cual propone deducir tal recorrido usando la siguiente tabla:

If $p_4 p_3 p_7 p_9 p_5 p_8 p_{10} < p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$:

S $p_1 p_6 p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	VP_2 $p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	NP $p_2 p_7 p_5 p_8 p_{10}$	PP $p_5 p_8 p_{10}$	NP p_8
X	X	X	P p_{10}	Cardiff
S $p_1 p_6 p_3 p_7 p_9$	VP $p_3 p_7 p_9$	NP p_7	from	
X	V p_9	Bob		
NP p_6	called			
Alice				

¿Cómo funciona? (22)

El algoritmo CKY, al igual que el de Viterbi, es una estrategia para resolver un problema de decisión, esto es: ¿cómo producir el árbol sintáctico más rápido y predecible de entre las dos opciones posibles?

Para que se den una mejor idea sobre cómo se ha implementado este algoritmo para hacer análisis gramatical, veamos la siguiente demo:

http://lxmls.it.pt/2015/cky.html

Esta demo fue desarrollada por un estudiante llamado Martin Lazarov, quien cursó la licenciatura en lingüística computacional de la Universidad de Tübingen:

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

www.sfs.uni-tuebingen.de/en/courses-of-study/courses-of-study-at-the-sfs/international-studies-in-computational-linguistics.html

Gracias por su atención

Blog del curso:
http://cesaraguilar.weebly.com/introduccioacuten-a-la-linguumliacutestica-computacional.html