

1. Introducción

La clase anterior revisamos:

- La formalización de Stabler (2011) *external* e *internal merge*.
- Centramos la atención en la conformación de los ítems léxicos y en la relevancia que tiene el ordenamiento de los rasgos para esta propuesta en particular.
- Presentamos la noción de *movimiento de remanente* y la propuesta de emplear este movimiento de manera generalizada para obtener los órdenes de palabras apropiados en ciertas lenguas.
- Consideramos diversas alternativas en la distribución de los rasgos para implementar gramáticas con licenciamiento de caso.

En esta clase, proponemos

- revisar dos implementaciones en Python de gramáticas minimalistas (sin movimiento de núcleo);
- presentar la formalización del movimiento de núcleo (incluido *Affix Hopping*) y las operaciones de adjunción;
- introducir el símbolo $\langle o / o \rangle$ para la adjunción de núcleos en el *baretree* y los operadores:
 - \Rightarrow / \Leftarrow (movimiento nuclear)
 - $\Rightarrow\Rightarrow / \Leftarrow\Leftarrow$ (affix hopping)
 - \gg / \ll (adjunción)
- probar las implementaciones para el *parser* sintáctico `mghapx` y el *parser* de secuenciación lineal `lhapx` de Stabler.

2. Repaso: MG

Recordemos los “ingredientes” de las gramáticas minimalistas que vimos la última clase.

Vocabulario	$\{ \text{casa, juan, comió, } \epsilon, \dots \}$
Tipos de items	$\{ ::, : \}$
Rasgos sintácticos	C, T, V, N, \dots $=C, =T, =V, =N, \dots$ $+wh, +focus, +caso, \dots$ $-wh, -focus, -caso, \dots$
Árboles	$\{ >, < \}$
Operaciones	$\text{em}(t_1[=x], t_2[x]) = \begin{cases} < & \text{if } t_1 =1 \\ \begin{array}{c} \wedge \\ t_1 \quad t_2 \end{array} & \\ > & \text{otherwise} \\ \begin{array}{c} \wedge \\ t_2 \quad t_1 \end{array} & \end{cases}$ $\text{im}(t_1[+x]) = \begin{array}{c} > \\ \wedge \\ t_2^M \quad t_1\{t_2[-x]^M \rightarrow \epsilon\} \end{array} \quad \text{si SMC}$

3. MG en Python

A continuación revisaremos dos implementaciones de las Minimalist Grammars en Python, a saber: las implementaciones de Stabler y la de Warstadt.

3.1. Implementaciones de Stabler

Además de las implementaciones de Prolog que vimos las clases anteriores, Stabler construyó implementaciones para las MGs en

otros lenguajes de programación: F# (F Sharp), Haskell, OCaml y Python. Nosotros nos limitaremos a revisar la implementación para Python.

3.1.1. Notación y codificación

Como punto de partida, es importante tener en cuenta que la notación empleada en las gramáticas de Python difiere de la que vimos en Prolog. En cierto sentido, esta notación resulta menos transparente con respecto a la notación que venimos empleando para representar los ítems léxicos. Consideremos las notaciones empleadas hasta el momento:

- | | | |
|-----|---|------------|
| (1) | a. <code>el :: =N,D,-case</code> | Formalismo |
| | b. <code>[e1] :: [= 'N' , 'D' , -case]</code> | Prolog |

En la implementación de Python de Stabler los items léxicos se codifican de la siguiente manera:

- (2) `([fon],[sin])`

FON corresponde al ítem de vocabulario y SIN al conjunto de rasgos sintácticos; esto es, rasgos de selección, rasgos categoriales y rasgos de licenciamiento. Cada uno de estos rasgos se especifican indicando el tipo de rasgo y su valor, del siguiente modo:

(3)	Tipo de rasgo	Codificación	observación
	Categorial	<code>('cat', 'X')</code>	
	Selección	<code>('sel', 'X')</code>	
	Licenciamiento	<code>('pos', 'x')</code>	SONDA
		<code>('neg', 'x')</code>	META

Así, el ítem léxico de (1) se codifica en esta implementación como en (4):

- (4) `([e1],[('sel', 'N'),('cat', 'D'),('neg', 'case')])`

La gramática se codica, entonces, con un símbolo `g` seguido de símbolo `=`, seguido del inventario encorchetado de ítems léxicos separados por comas:

(5) Ejemplo gramática

```
g = [[ [ casa ], [ ( 'cat ', 'N' ) ] ],  
      [ [ la ], [ ( 'sel ', 'N' ), ( 'cat ', 'D' ), ( 'neg ', 'case' ) ] ],  
      [ [ pinta ], [ ( 'sel ', 'D' ), ( 'pos ', 'case' ), ( 'cat ', 'V' ) ]  
      ]
```

 Veamos las gramáticas `mg0.py` y `mg0sp.py` para la implementación de Python desarrollada por Stabler.

Instrucciones:

1. Abrir una terminal.
2. Para iniciar el *parser* es necesario introducir una orden que tiene la siguiente secuencia de elementos
`python3 parser.py gramática raíz probabilidad`
3. Si utilizamos:
 - `parser: mgtdbp-dev.py`
 - `gramática: mg0.py`
 - `raíz: C`
 - `probabilidad: 0.001`

entonces la orden tendrá la forma:

```
python3 mgtdbp-dev.py mg0 C 0.001
```

Ejercicio 1. Cree una gramática `mgex1.py` que permita derivar la oración:

(6) Iván pinta la casa

Como punto de partida puede emplear los ítems de (5) y agregar los que sean necesario para derivar esta oración.

3.2. Implementación de Warstadt para C&S2016

En la clase 05 vimos que Collins y Stabler (2016) definen formalmente la GU como:

- (7) La **Gramática universal** es una 6-tupla
⟨PHON, SYN, SEM, Select, Merge, Transfer⟩

De los seis elementos que conforman la tupla, nos hemos concentrado fundamentalmente en cuatro, estos son, la operación *merge* y la 3-tupla de rasgos ⟨PHON, SYN, SEM⟩ que conforman los ítems léxicos. Al ver el funcionamiento de las gramáticas de Stabler la operación *Select* aparece, en cierto sentido, implícita en la derivación. La implementación de Warstadt, que revisaremos a continuación, nos permite ver de manera más clara cómo interactúan *Select* y *Merge*.

Observación

A diferencia de los desarrollos de Stabler, la implementación de Warstadt no está completa y, en consecuencia, tiene algunas limitaciones. No obstante, resulta valiosa para ver diversos modos de implimentar computacionalmente una propopuesta teórica.

3.3. Codificación

La implementación de Warstadt difiere de las implementaciones de Stabler, entre otras cuestiones, en el hecho de que el léxico está construido como un archivo independiente en lenguaje XML.

El lexicón está formado por un inventario de ítems léxicos, que están codificados de la siguiente manera:

- (8)

```
<root>
  <word>
    <phon> ... </phon>
    <syn>
      <cat> ... </cat>
      <sel> ... </sel>
      <sel> ... </sel>
    </syn>
    <sem> ... </sem>
  </word>
```

```
<word>
    ...
</word>
</root>
```

Los valores de cada etiqueta se detallan a continuación:

- (9)
- | | |
|--|---------------------------------|
| <code><root></root></code> | requerido por XML |
| <code><word></word></code> | ítem léxico |
| <code><phon></phon></code> | rasgos fonológicos |
| <code><sem></sem></code> | rasgos semánticos |
| <code><syn></syn></code> | rasgos sintácticos |
| <code><cat></cat></code> | rasgos sintácticos categoriales |
| <code><sel></sel></code> | rasgos sintácticos de selección |

 Veamos la implementación de python desarrollada por Alex Warstadt para *A Formalization of Minimalist Syntax* (Collins y Stabler, 2016).

Ejercicio 2. Cree el lexicón que permita derivar las siguientes oraciones del español:

- (10)
- El perro ladra.
 - El perro muerde el hueso.
 - El perro creció.

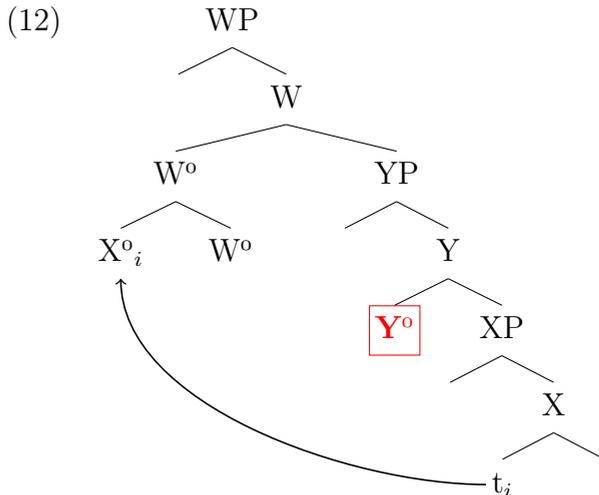
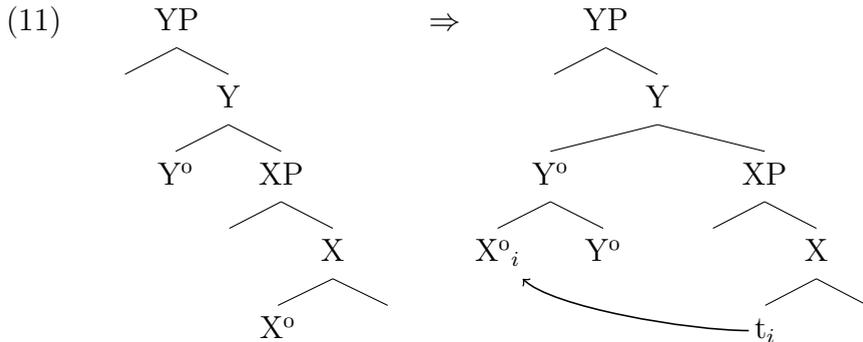
3.4. Síntesis

Hasta el momento revisamos dos tipos de implementaciones computacionales en Python para las gramáticas minimalista. Ninguna de estas gramáticas, ni las gramáticas que estudiamos las clases anteriores involucraban movimiento de núcleos o adjunción. En lo que sigue, centraremos la atención en estas dos operaciones.

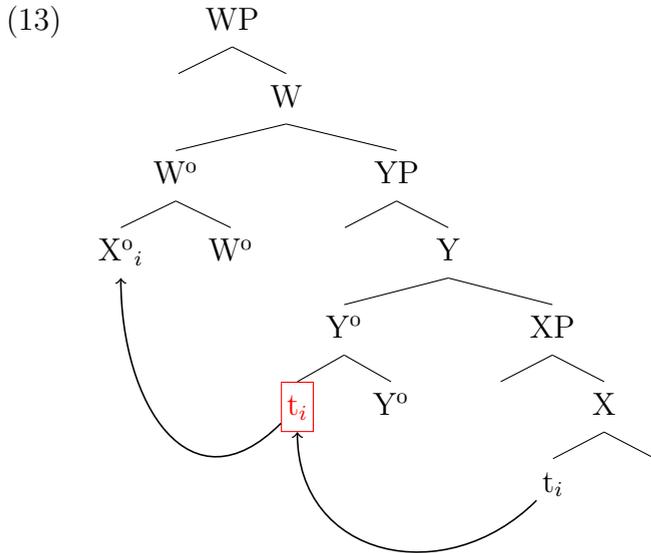
4. Movimiento de Núcleos

Hasta el momento nos hemos limitado al ensamble de núcleos con frases (por **em**) y de frases con frases (por **em** o **im**). En lo que sigue nos ocuparemos de la operación de movimiento que involucra núcleos. No discutiremos la naturaleza de este movimiento, pero quien tenga interés en la cuestión puede consultar una serie de trabajos clásicos como Travis (1984); Baker (1985, 1988); Chomsky (2000, 2001) y otros trabajos más recientes como Matushansky (2006); Harizanov y Gribanova (2019); Arregi y Pietraszko (2021).

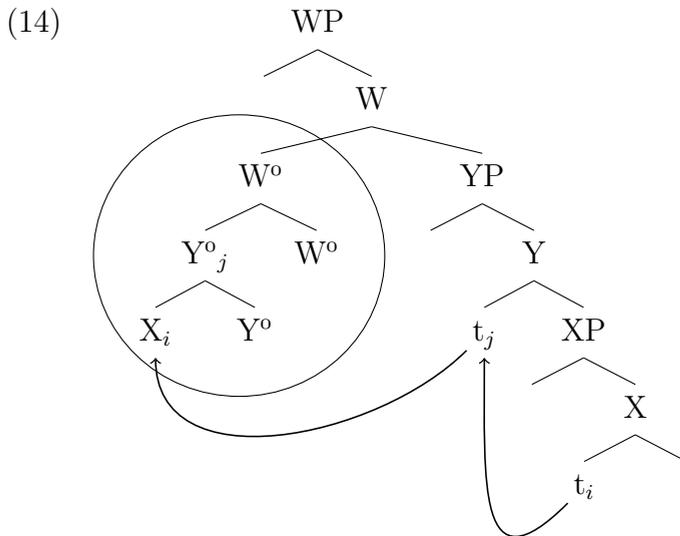
Este tipo de movimiento consiste, básicamente, en el desplazamiento de un núcleo hacia otro núcleo mediante una operación adjunción, lo que crea un núcleo complejo. Esta operación es estrictamente local. Esto significa que el movimiento solo es posible hacia el núcleo inmediatamente superior (o inferior, si se considera *affix hopping* como una operación sintáctica).



El movimiento de núcleo tampoco admite excorporación, por lo que una derivación como la de (14) queda descartada:

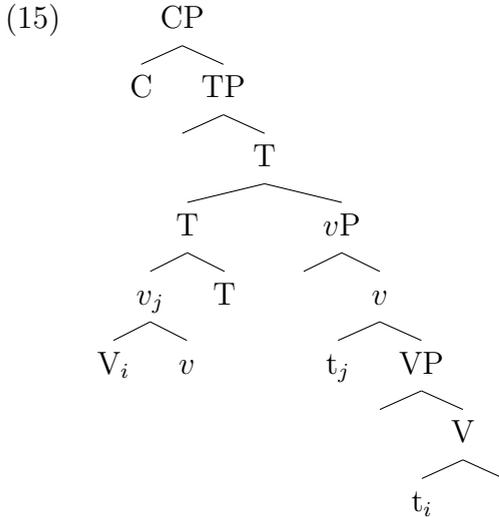


Si en una derivación dada, el núcleo X debiera llegar al núcleo W, solo podría hacerlo arrastrando todos los núcleos entre X y W.



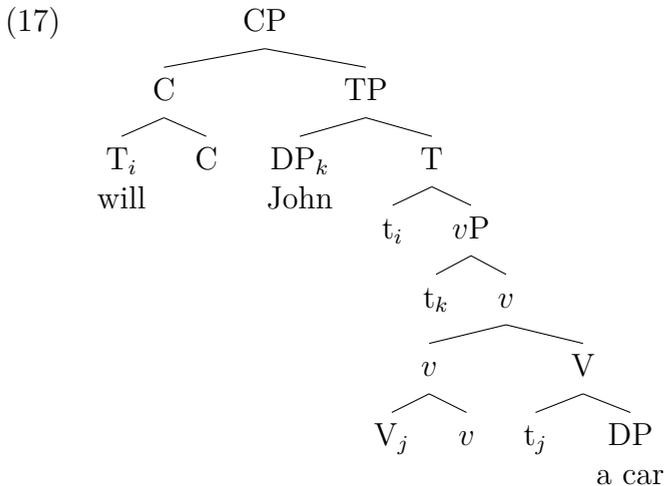
Se ha propuesto el movimiento de núcleos en una amplia variedad de dominios empíricos. Por ejemplo, en las lenguas románicas,

en general, el núcleo V se mueve a T. Así, si consideramos el esqueleto de la cláusula que vimos la clase anterior, el movimiento de núcleo daría como resultado la estructura de (15)



El movimiento de núcleo también ha sido empleado para explicar la inversión *auxiliar-sujeto* en las preguntas del inglés.

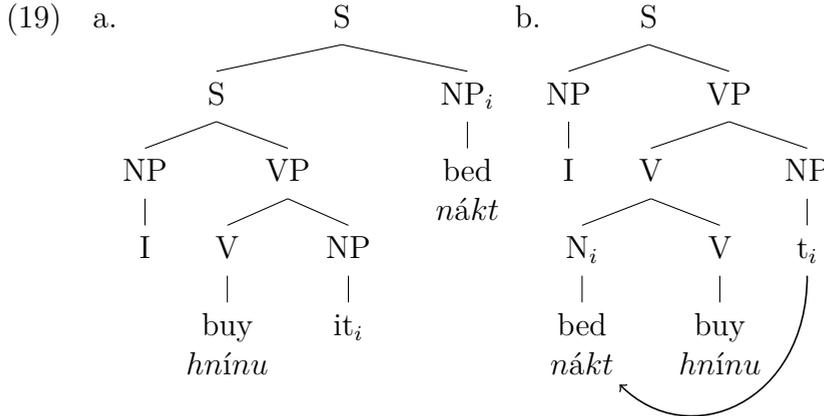
- (16) a. John will buy a car.
 b. Will John buy a car?



Finalmente, un dominio en el que ha sido preponderante es en el fenómeno de incorporación, que se puede apreciar en los siguientes ejemplos del mohicano.

- (18) a. Wa'-k-hnínu-' ne ka-**nákt**-a'.
 FACT-1sS-buy-PUNC NE NsS-bed-NSF
 'I bought the/a bed.'
- b. Wa'-ke-**nákt**-a-hnínu-'.
 FACT-1sS-bed-Ø-buy-PUNC
 'I bought the/a bed.'

(mohicano Baker, 1996, 279)



(Adaptado de Baker, 1996, 282)

4.1. Minimalist Grammar with Head-movement

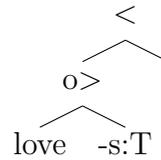
Stabler observa que existen ciertas dificultades en incorporar el movimiento de núcleos a una Minimalist Grammar. En efecto, señala que solo tiene que mover el núcleo, sin argumentos ni adjuntos y, además, el núcleo movido no es un especificador, sino un núcleo seleccionador que se incorpora a un núcleo seleccionado. La definición formal del movimiento de núcleos requiere del agregador de un nuevo operador \Rightarrow o \Leftarrow , cuyo efecto es hacer que un núcleo deba adjuntarse a la izquierda o a la derecha respectivamente. Este operador indica cuál es el núcleo al cual debe adjuntarse el otro núcleo. Veamos la definición.

$$(20) \text{ mHM}(t_{1[\Rightarrow f]}, t_{2[f]}) =$$

si t_1 tiene exactamente 1 nodo.

El resultado de la combinación de dos objetos sintácticos mediante ensamble de núcleos se marca con un $\langle o \text{ o } o \rangle$, según si el núcleo se encuentra a la izquierda o a la derecha respectivamente:

(21) $-s::=>V \text{ T} + \text{love}::V \rightarrow$

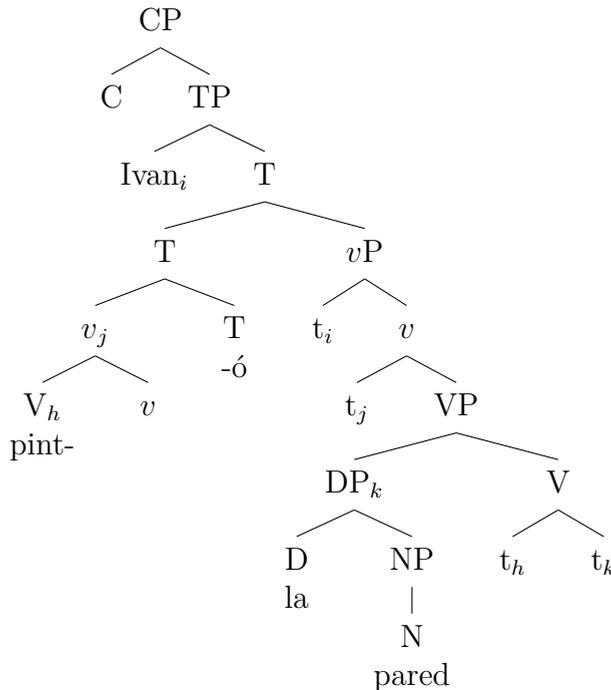


extraído de [Stabler \(2003\)](#)

Vale notar que ahora la combinación de rasgos de los ítems léxicos resulta ligeramente más compleja, en tanto deben incluir los rasgos que desencadenan el movimiento de núcleos. Revisemos los ítems léxicos necesarios para derivar (22a), cuyo árbol en la representación habitual es el de (22b):

(22) a. Ivan pintó la pared

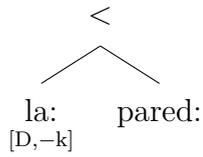
b.



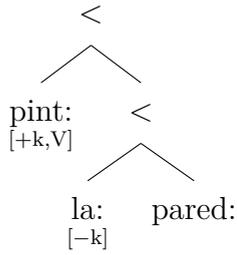
(23)	1 Ivan :: D, -k	5 $\epsilon :: =>V, =D, v$
	2 la :: =N, D, -k	6 $-\acute{o} :: =>v, +k, T$
	3 pared :: N	7 $\epsilon :: =T, C$
	4 pint :: =D, +k, V	

Veamos la derivación paso a paso.

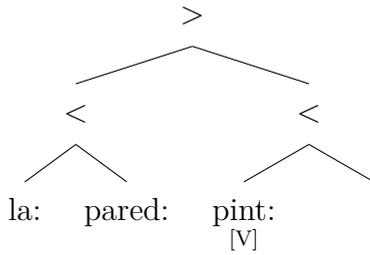
(24) $em(2,3) = \boxed{8}$



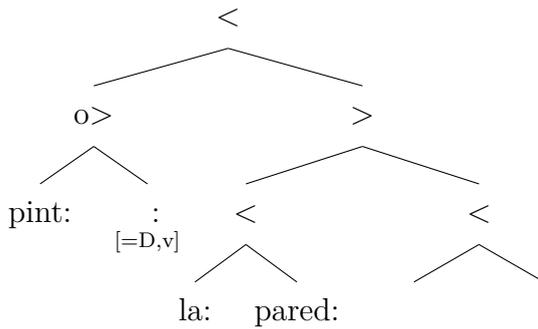
(25) $em(4,8) = \boxed{9}$



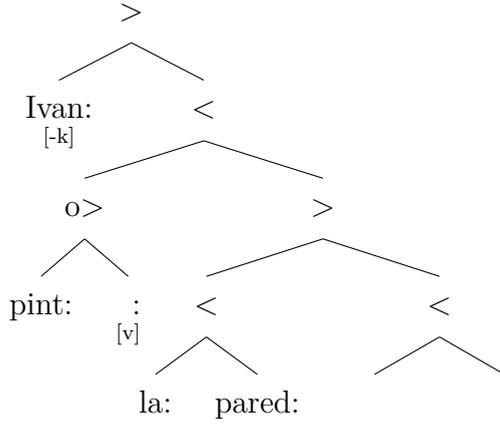
(26) $im(9) = \boxed{10}$



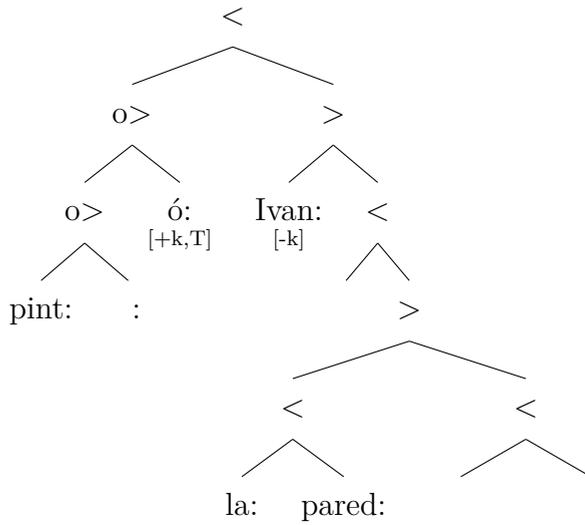
(27) $em(10,5) = \boxed{11} + mHM(11) = \boxed{12}$



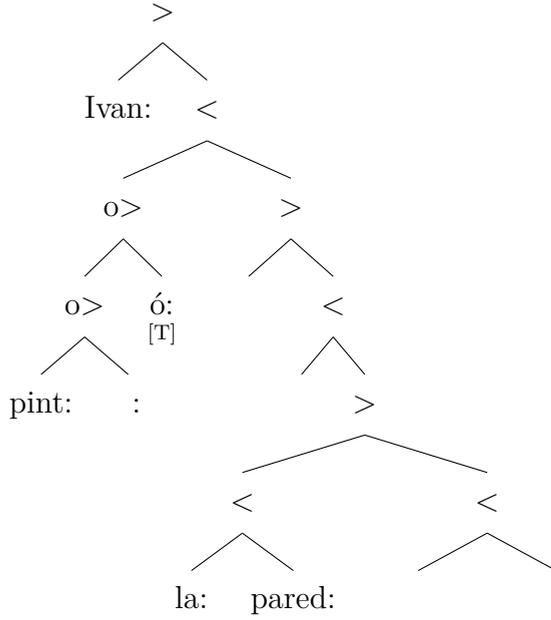
(28) $em(12,1) = \boxed{13}$



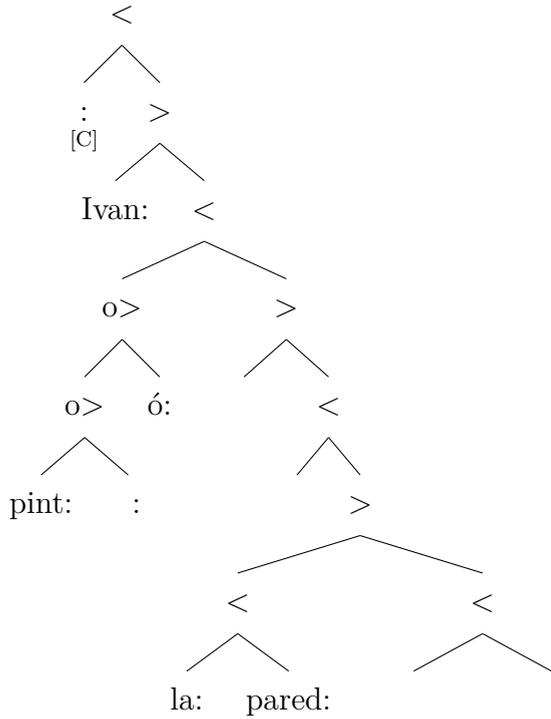
(29) $\text{em}(13,6) = \boxed{14} + \text{mHM}(14) = \boxed{15}$



(30) $\text{im}(15) = \boxed{16}$



(31) $em(16,7) = \boxed{17}$



Ejercicio 3. Agregue al léxico de (54) los ítems léxicos necesarios que permitan generar la siguientes oraciones:

- (32) a. ¿Qué pintó Iván?
b. ¿Quién pintó la pared?

Note que en (32a) hay una inversión en el orden entre el verbo y el sujeto.

 Veamos la gramática `spanish.pl` para ver cómo funciona una Minimalist Grammar con Head-Movement en español.

4.2. Affix Hopping

Affix Hopping es el sobrenombre que recibió una de las reglas transformacionales formuladas por Chomsky (1957), que originalmente había sido bautizada con el nombre de *Auxiliary Transformation*. Su importancia reside en la originalidad y precisión con la que se describe la morfología verbal del inglés y la distribución de los afijos.

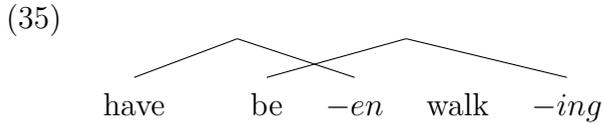
La regla de *Affix Hopping* es una de las primeras reglas que involucran el desplazamiento de núcleos. Las observaciones asociadas a esta regla han sido importantes para comprender algunas diferencias paramétricas en relación con ciertos movimientos. Una diferencia central entre la operación de Head-movement que solemos considerar, con respecto a *Affix Hopping*, es que esta última parece inducir que los núcleos desciendan, es decir, que un núcleo se adjunte a otro núcleo inmediatamente inferior en la estructura de la oración.

A continuación, veremos brevemente cómo funciona *Affix Hopping* en su formulación original y posteriormente veremos la implementación de Stabler.

4.2.1. Morfología verbal del inglés

- (33) a. Peter *walked*.
b. Peter *was walking*
c. Peter *had been walking*.
d. Peter *could have been walking*

- (34) a. V+past
 b. be+past - V+ing
 c. have+past - be+en - V+ing
 d. can+past - have - be+en - V+ing



- (36) Reglas del componente base
 ...
 8. Verb \rightarrow Aux + V
 9. V \rightarrow hit, take, walk, read, ...
 10. Aux \rightarrow C(M) + (have + en) (be + ing)
 11. M \rightarrow will, can, make, shall, must
 ...

Reglas Transformacionales

- (37) Number transformation
 DE: X-C-Y
 CE: C \rightarrow $\left\{ \begin{array}{l} -s \text{ en el contexto de } NP_{sg} \text{ ---} \\ \emptyset \text{ en otros contextos} \\ past \text{ en cualquier conexto} \end{array} \right\}$

- (38) Auxiliary transformation (Affix Hopping)
 DE: X - Af - v - Y
 CE: X₁ - X₂ - X₃ - X₄ \rightarrow X₁ - X₃ - X₂ # - X₄
 (donde Af = cualquier C o -en o -ing y v = M o V o have o be)

Para ver el funcionamiento de la regla de *Affix Hopping* consideremos las siguientes oraciones:

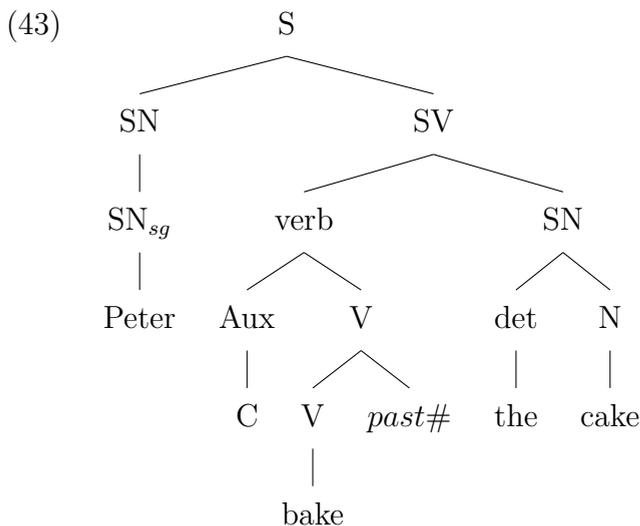
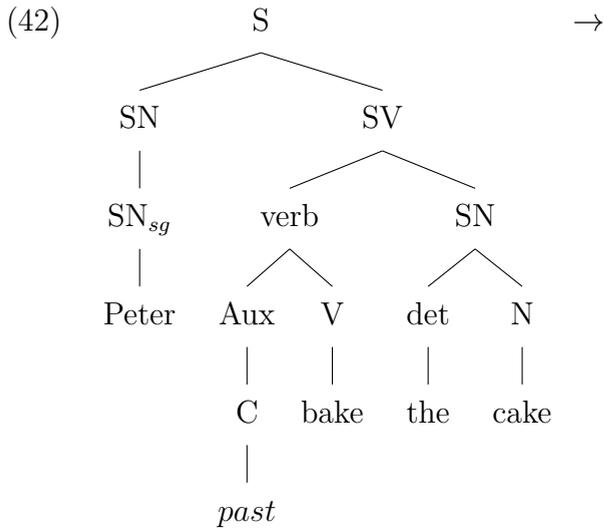
- (39) a. Peter baked the cake.
 b. Peter was baking the cake.
 c. Peter had been baking the cake.

Comencemos por el primer ejemplo. Nótese que resulta relevante la segmentación morfológica de *baked*.

- (40) Peter bak-ed the cake.
 Peter bake-PST the cake

El verbo y el morfema de pasado aparecen juntos como resultado de la regla, pero el contexto estructural que induce su aplicación requiere que se encuentren en otras posiciones.

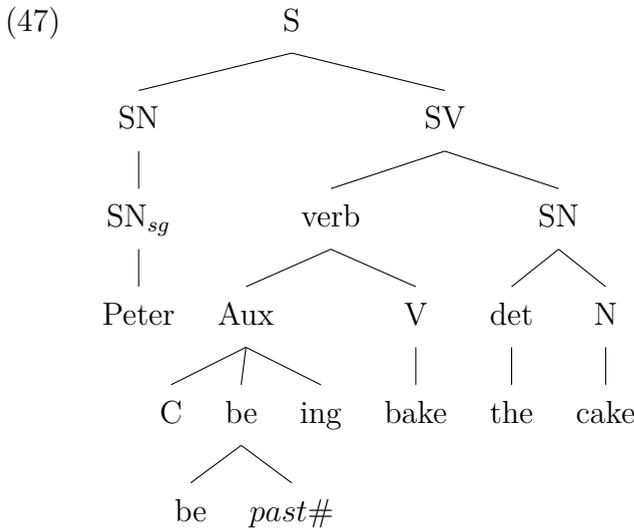
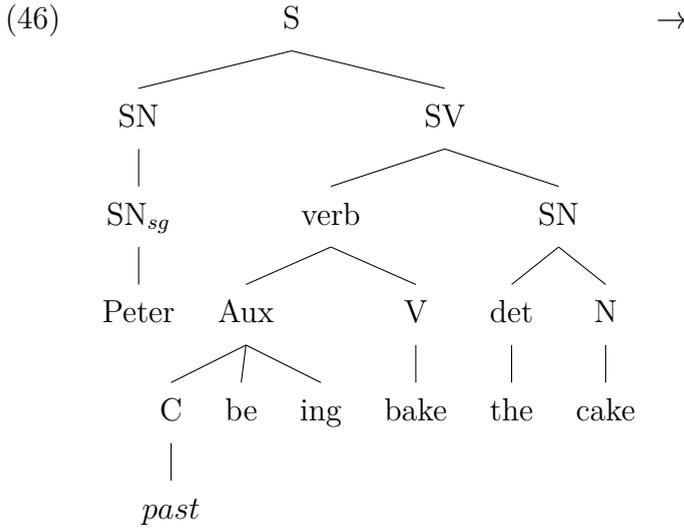
- (41) $\underbrace{\text{Peter}}_X$ $\underbrace{\text{past}}_{Af}$ $\underbrace{\text{bake}}_v$ $\underbrace{\text{the cake}}_Y$



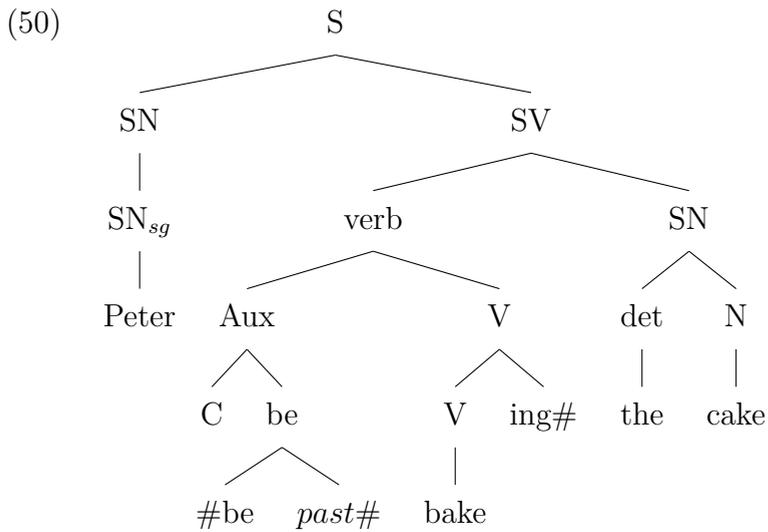
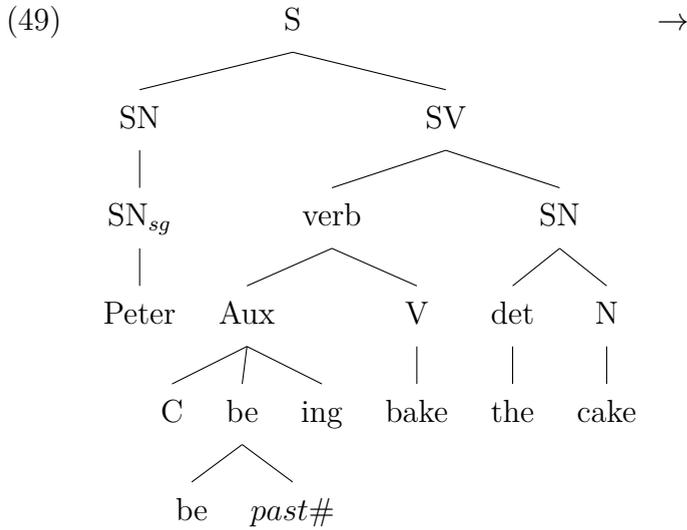
Veamos, ahora, el segundo ejemplo. ¿Cuántas veces debería aplicarse la regla en este caso?

(44) Peter was baking the cake

(45) $\underbrace{\text{Peter}}_X$ $\underbrace{\text{past}}_{Af}$ $\underbrace{\text{be}}_v$ $\underbrace{\text{ing bake the cake}}_Y$



(48) $\underbrace{\text{Peter \#be past\#}}_X \quad \underbrace{\text{ing}}_{Af} \quad \underbrace{\text{bake}}_v \quad \underbrace{\text{the cake}}_Y$



Ejercicio 4. Indique cómo se aplicarían *Affix hopping* para dar cuenta de la siguiente oración:

(51) Peter had been baking the cake

4.2.2. Affix hopping: definición formal

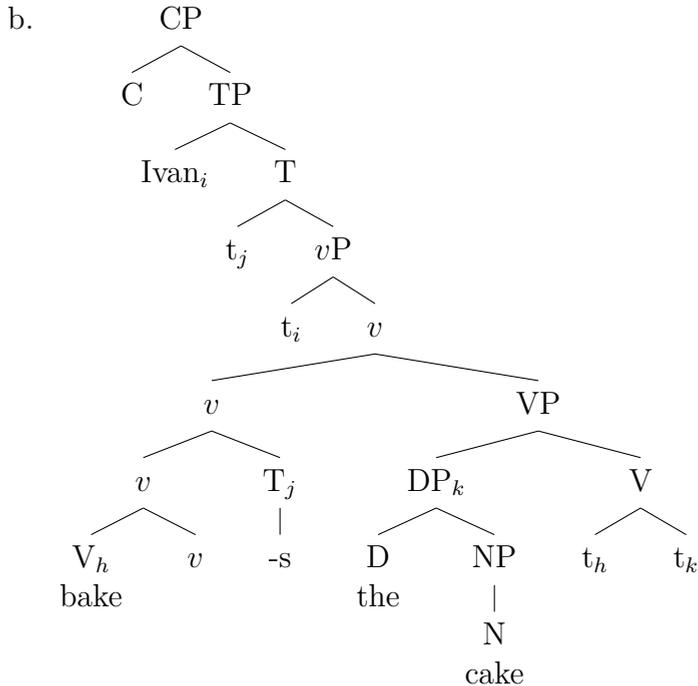
Tal como vimos, la regla de *Affix Hopping*, que resulta tan importante para describir la morfología verbal del inglés, constituye una especie de operación de descenso, también conocida como *lowering*. Formalmente, esta operación se puede formular como en (52):

$$(52) \quad \text{mAH}(t_1[f==>], t_2[f]) = \begin{array}{c} < \\ / \quad \backslash \\ t_1\{hd_1 \rightarrow \varepsilon\} \quad t_2\{hd_2 \rightarrow hd_2hd_1\} \end{array}$$

si t_1 tiene exactamente 1 nodo.

Con esta regla en mente, veamos cómo estarían formados los ítems léxicos para derivar (53a) con esta operación. En (53b) se puede apreciar la estructura en la representación de X-barra:

(53) a. Peter bakes the cake.

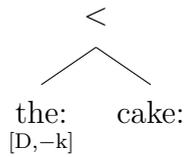


- (54)

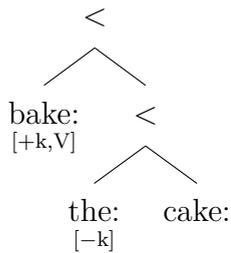
1	Peter :: D, -k	5	$\epsilon :: =>V, =D, v$
2	the :: =N, D, -k	6	$-s :: v==>, +k, T$
3	cake :: N	7	$\epsilon :: =T, C$
4	bake :: =D, +k, V		

Hagamos la derivación paso por paso:

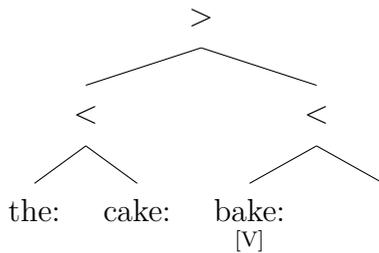
- (55) $em(2,3)=\boxed{8}$



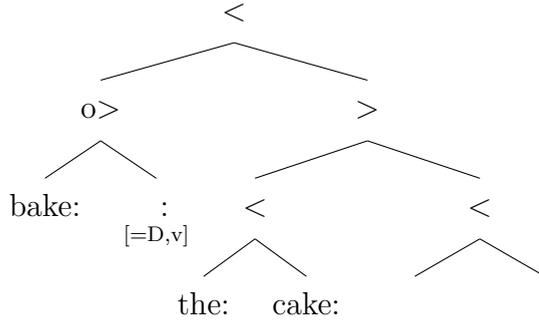
- (56) $em(4,8)=\boxed{9}$



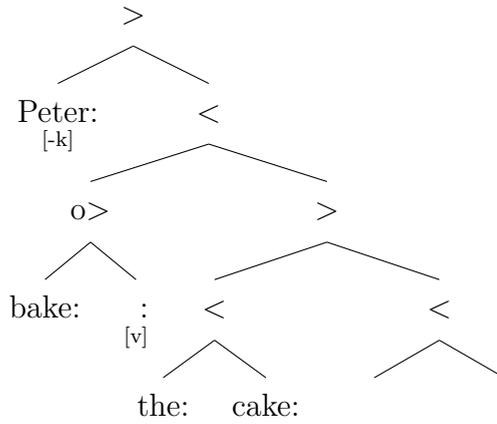
- (57) $im(9)=\boxed{10}$



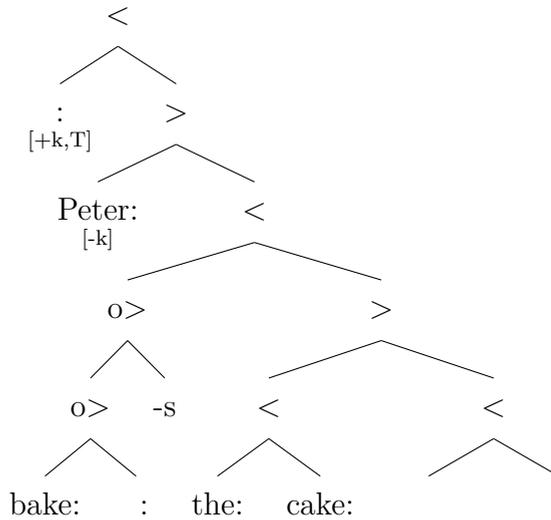
- (58) $em(10,5)=\boxed{11} + mHM(11)=\boxed{12}$



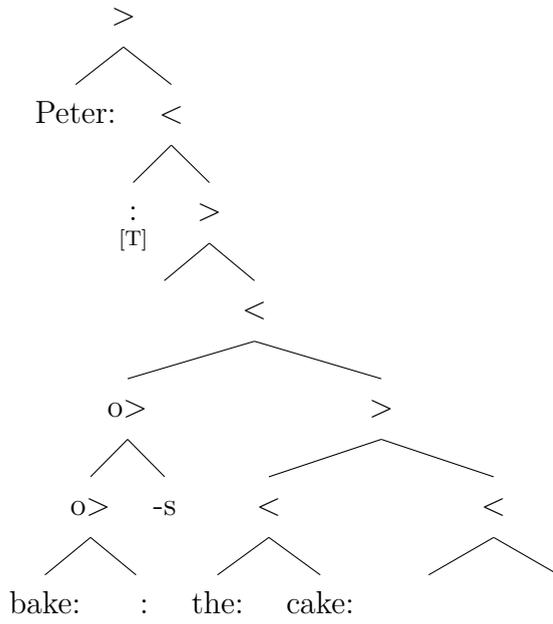
(59) $em(12,1) = \boxed{13}$



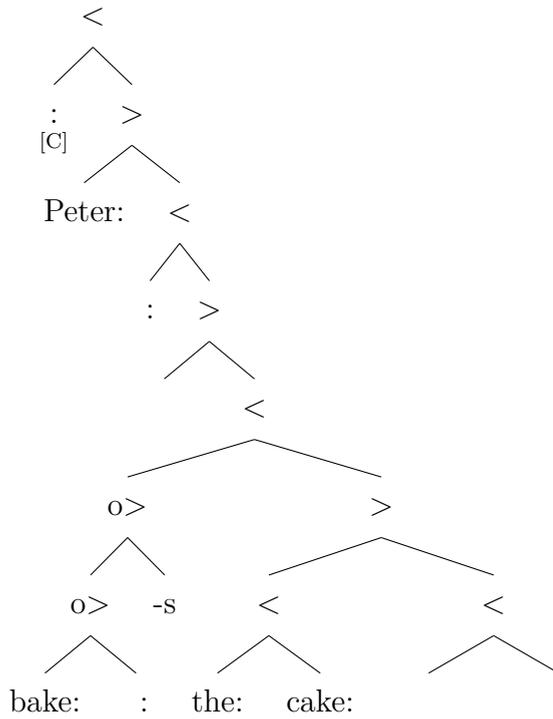
(60) $em(13,6) = \boxed{14} + mAH(14) = \boxed{15}$



(61) $\text{im}(15) = \boxed{16}$



(62) $\text{em}(16,7) = \boxed{17}$



 Revisemos, entonces, las gramáticas `gh6.pl` y `larsonian1.pl`.

Ejercicio 5. Observe con detenimiento las gramáticas `gh6.pl` y `larsonian1.pl`, luego determine qué ítems léxicos inducen Affix Hopping y cuáles inducen head-movement.

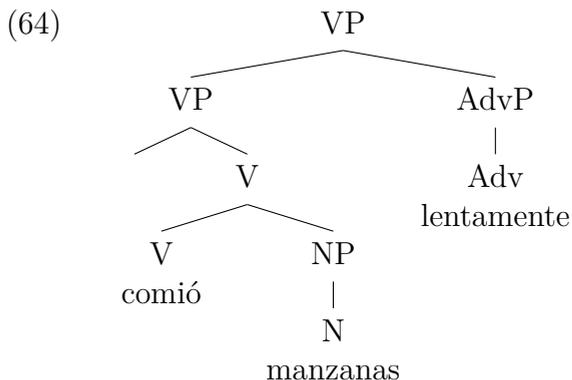
5. Adjunción

Las estructuras de adjunción constituyen un dolor de cabeza para la teoría de la estructura de frase. De hecho, en el marco de la gramática generativa estas estructuras no han recibido una solución totalmente adecuadas. Existen varios problemas asociados con la adjunción. A continuación mencionamos los principales:

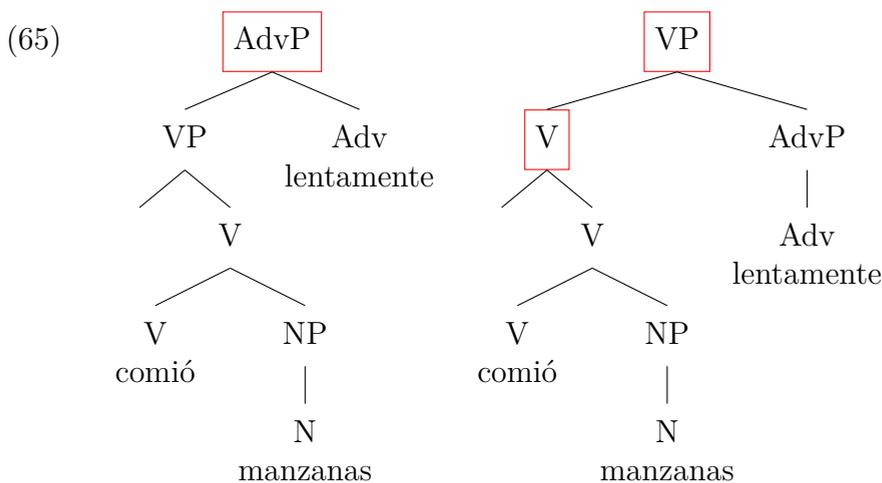
- No están seleccionados.
- Admiten iteración.
- Preservan la categoría sintáctica del constituyente al que se adjuntan.
- No entran en relaciones de concordancia.
- Tienen restricciones de caso diferentes de los argumentos.
- No resulta claro si cotejan rasgos. Si lo hacen, no queda claro qué rasgo cotejan.
- Los adjuntos no se circunscriben a una única categoría, es decir, hay adjuntos prácticamente de cualquier categoría (AP, AdvP, CP, DP, PP, ...).
- No es completamente claro de qué manera se relacionan sintácticamente con los constituyentes a los que modifican.

Consideremos la cuestión con mayor detalle. En la oración (63), *lentamente* es un adjunto del VP:

(63) Comió manzanas lentamente.



Uno de los efectos centrales de la adjunción es que no cambia la categoría de la frase a la que se adjunta, ni modifica el nivel de proyección.



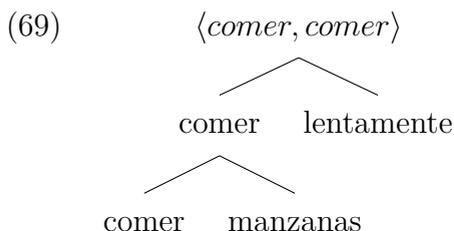
En términos de la teoría de *bare phrase structure*, la adjunción tiene que dar como resultado una proyección máxima, no mínima:

(66) {comer, {comer, manzanas}}

(67) {?, {{comer, {comer, manzanas}}, lentamente} }

La propuesta de Chomsky es representar la etiqueta de la adjunción como un par ordenado $\langle comer, comer \rangle$.

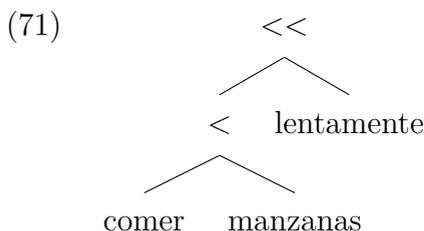
(68) $\{ \langle comer, comer \rangle, \{ \{ comer, \{ comer, manzanas \} \}, lentamente \} \}$



Stabler introduce los operadores \gg y \ll para la adjunción a la izquierda y a la derecha, respectivamente. La formalización de la adjunción puede ser expresada de la siguiente manera:

$$(70) \quad \text{adjoin}(t_1[f], t_2[g]) = \begin{cases} \gg & \\ \begin{array}{c} \swarrow \quad \searrow \\ t_1 \quad t_2\{g\} \end{array} & \text{if } t_1 \in \text{left-adjoiners}(g) \\ \ll & \\ \begin{array}{c} \swarrow \quad \searrow \\ t_2\{g\} \quad t_1 \end{array} & \text{if } t_1 \in \text{right-adjoiners}(g) \end{cases}$$

De modo que la estructura de (71) en un *bare tree* tendría la siguiente forma:



Vale la pena recordar las operaciones *external merge*, *internal merge* o el *movimiento de núcleos* es desencadenado por rasgos especificados en los ítems léxicos. No obstante, dado que los adjuntos nunca son seleccionados, sería problemático incluir el rasgo de adjunción en las ítems léxicos en cuestión. En este sentido, la operación se especifica como una estructura listada. Por ejemplo, para introducir un adjetivo modificador de un nombre alcanza con la siguiente regla:

(72) $N \ll A$



Veamos cómo está implementada la adjunción en las gramáticas `gh6.pl` y `larsonian1.pl` de Stabler.

Ejercicio 6. ¿Cómo podrían formularse las reglas para introducir los adjuntos en las siguientes expresiones?

- (73)
- a. el auto [de Pedro]
 - b. el [famoso] escritor
 - c. Arreglamos el auto ayer.
 - d. Pedro cocino el almuerzo rápidamente.

6. Síntesis

Vocabulario	{casa, juan, comió, ϵ , ... }
Tipos de items	{::, : }
Rasgos sintácticos	C, T, V, N, ... =C, =T, =V. =N, ... +wh, +focus, +caso, ... -wh, -focus, -caso, ...
Árboles	{>, < } {o>, <o} {> >, < < }

$$\begin{array}{l}
\text{em}(t_{1[=x]}, t_{2[x]}) = \begin{cases} < & \text{if } |t_1|=1 \\ \begin{array}{c} \wedge \\ t_1 \quad t_2 \end{array} & \\ > & \text{otherwise} \\ \begin{array}{c} \wedge \\ t_2 \quad t_1 \end{array} & \end{cases} \\
\text{im}(t_{1[+x]}) = \begin{array}{c} > \\ \wedge \\ t_2^M \quad t_1\{t_2[-x]^M \rightarrow \varepsilon\} \end{array} \quad \text{si SMC} \\
\text{mHM}(t_{1[=>f]}, t_{2[f]}) = \begin{cases} < & \text{if } |t_1|=1 \\ \begin{array}{c} \wedge \\ \text{hd}_2 t_1 \quad t_2\{\text{hd}_2 \rightarrow \varepsilon\} \end{array} & \end{cases} \\
\text{mAH}(t_{1[f==>]}, t_{2[f]}) = \begin{cases} < & \text{if } |t_1|=1 \\ \begin{array}{c} \wedge \\ t_1\{\text{hd}_1 \rightarrow \varepsilon\} \quad t_2\{\text{hd}_2 \rightarrow \text{hd}_2 \text{hd}_1\} \end{array} & \end{cases} \\
\text{adjoin}(t_{1[f]}, t_{2[g]}) = \begin{cases} >> & \\ \begin{array}{c} \wedge \\ t_1 \quad t_2\{g\} \end{array} & \text{if } t_1 \in \text{left-adjoiners}(g) \\ << & \\ \begin{array}{c} \wedge \\ t_2\{g\} \quad t_1 \end{array} & \text{if } t_1 \in \text{right-adjoiners}(g) \end{cases}
\end{array}$$

7. Consideraciones finales

En estas tres clases centramos la atención en formalización de las gramáticas minimalistas y algunas implementaciones computacionales en Prolog y Python.

En particular, revisamos los siguientes puntos:

- la formalización de los ítems léxicos y su codificación para ser procesados en implementaciones de Prolog y Python;

- las clases de rasgos que codifican los ítems léxico y el problema de su ordenamiento;
- las operaciones de *external merge* e *internal merge*;
- el movimiento de núcleos y *Affix Hopping*;
- la operación de adjunción.

De manera concomitante, fuimos estudiando algunas implementaciones computacionales de las gramáticas que emplean estas operaciones.

Referencias

- Arregi, K. y Pietraszko, A. (2021). The Ups and Downs of Head Displacement. *Linguistic Inquiry*, 52(2):241–290.
- Baker, M. (1985). The Mirror Principle and Morphosyntactic Explanation. *Linguistic Inquiry*, 16(3):373–415.
- Baker, M. (1988). *Incorporation: A theory of grammatical function changing*. University of Chicago Press, Chicago, Illinois.
- Baker, M. C. (1996). *The Polysynthesis Parameter*. Oxford Studies in Comparative Syntax. Oxford University Press, New York.
- Chomsky, N. (1957). *Syntactic structures*. Mouton, The Hague.
- Chomsky, N. (2000). Minimalist inquiries: The framework. En Martin, R., Michaels, D., y Uriagereka, J., editores, *Step by step: Essays on minimalist syntax in honor of Howard Lasnik*, pp. 89–156. MIT Press.
- Chomsky, N. (2001). Derivation by phase. En Kenstowicz, M., editor, *Ken Hale: A Life in Linguistics*, pp. 1–52. MIT Press, Cambridge, Massachusetts.
- Collins, C. y Stabler, E. (2016). A formalization of minimalist syntax. *Syntax*, 19(1):43–78.

- Harizanov, B. y Gribanova, V. (2019). Whither head movement?
Natural Language & Linguistic Theory, 37(2):461–522.
- Matushansky, O. (2006). Head movement in linguistic theory. *Linguistic Inquiry*, (37):69–109.
- Stabler, E. P. (2003). Comparing 3 perspectives on head movement.
En Mahajan, A., editor, *From Head Movement And Syntactic Theory:UCLA, Postdam Working Papers in Linguistics*, pp. 178–198. UCLA.
- Stabler, E. P. (2011). Computational perspectives on minimalism.
En Boeckx, C., editor, *Oxford handbook of linguistic minimalism*, pp. 617–643. Oxford.
- Travis, L. d. (1984). *Parameters and effects of word order variation*.
Tesis doctoral, Massachusetts Institute of Technology.