Dependencias, parsing

GFLN

InCo

2015

Métodos de parsing

Nos concentraremos en representaciones en dependencias de tipo árbol (proyectivo o no).
El objetivo del parsing es, dada una oración, construir uno o varias representaciones en dependencias.
Podemos distinguir entre 3 tipos de métodos:

- Basado en gramática

Métodos de parsing

Nos concentraremos en representaciones en dependencias de tipo árbol (proyectivo o no).
El objetivo del parsing es, dada una oración, construir uno o varias representaciones en dependencias.
Podemos distinguir entre 3 tipos de métodos:

- Basado en gramática
- Basado en restricciones

Métodos de parsing

Nos concentraremos en representaciones en dependencias de tipo árbol (proyectivo o no).
El objetivo del parsing es, dada una oración, construir uno o varias representaciones en dependencias.
Podemos distinguir entre 3 tipos de métodos:

- Basado en gramática
- Basado en restricciones
- Basado en datos

Parsing basado en datos (data-driven)

- Es el enfoque más difundido actualmente.
- No hay gramática ni restricciones explícitas.
- Aprendizaje a partir de un corpus anotado con árboles (treebank).
- Para el inglés se usó el Penn, transformado a dependencias.
- Veremos 2 variantes:
  - Basado en la historia de parsing, con decisiones locales. (Nivre, MALTPARSER, 2004)
  - Inferencia global sobre árboles posibles.(Mc Donald, 2007)

Aprendizaje a partir de historia

- Algoritmo goloso (greedy) que toma decisiones locales óptimas
- Guiado por un clasificador entrenado con secuencias de derivaciones (gold standard)
- Ventajas:
  - fácil de implementar
  - determínistico, por lo tanto eficiente (tiempo)
  - curva de aprendizaje empinada (requiere relativamente pocos datos de entrenamiento)
  - se aprende la historia de las transiciones de un parser determinista

Aprendizaje a partir de historia (2)

Principales sistemas desarrollados

- 2002 - Japonés - Kudo, Matsumoto (sin etiquetas)
- 2003 - Inglés - Yamada, Matsumoto (sin etiquetas)
- 2004 - Sueco, Inglés, Nivre et al. con etiquetas
- 2005 en adelante - Maltparser: generador de parsers, resultados reportados para más de 10 idiomas

Parsing basado en transiciones(2)

Podemos reconocer los siguientes módulos:

- Parser determinista de dependencias basado en transiciones
- Generación de historia de parsing a partir de un árbol de dependencias
- Formulación parámetrica de un oráculo en función de atributos
- Modelo de aprendizaje de los parámetros del oráculo

Definiciones previas

Definición

Grafo de dependencias para una oración.

Dado un conjunto R de etiquetas, un grafo de dependencias para una oración $x = (w_1, w_2, ..., w_n)$ es un grafo dirigido etiquetado $\mathrm{G} = (\mathrm{V}, \mathrm{E}, \mathrm{L})$ tal que:

1. $\mathrm{V} = (\mathrm{R}oot, w_1, ..., w_n)$
2. $\mathrm{E} \subseteq \mathrm{V} \times \mathrm{V}$
3. $\mathrm{L} : \mathrm{E} \to \mathrm{R}$
4. El nodo Root es una raíz (ninguna arista incide en Root)
5. G es un árbol proyectivo

Definiciones previas, notación

- Las aristas son pares ordenados (i,j), usamos también la notación $i \rightarrow j$.
  En la arista $i \rightarrow j$, j es el dependiente, el nodo i es el head

Definiciones previas, notación

- Las aristas son pares ordenados (i,j), usamos también la notación $i \rightarrow j$.
  En la arista $i \rightarrow j$, j es el dependiente, el nodo i es el head
- Llamamos V^+ a los nodos correspondientes a una oración $x = (w_1, w_2, ..., w_n)$.
  Se cumple $\mathrm{V}^+ = \mathrm{V} - \mathrm{R}oot$.

Definiciones previas

Definición

Configuración.

Dado un conjunto $\mathrm{R} = r_0, r_1, ..., r_m$ de etiquetas y una oración $x = (w_1, w_2, ..., w_n)$ una **configuración de parsing** para x es una cuádrupla $c = (\rho, \tau, h, d)$, donde:

1. ρ es un stack de nodos de V
2. τ es una cola ordenada de nodos de V^+
3. $h : \mathrm{V}^+ \rightarrow \mathrm{V}$ es una función que indica el padre de cada nodo (único, por hipótesis)
4. $d : \mathrm{V}^+ \rightarrow \mathrm{R}$ es una función que indica la etiqueta de la única arista que incide en cada nodo. Si $h(i) = \mathrm{R}oot$, $d(i) = root$

Definiciones previas

Definición

Grafo definido por una configuración

La configuración $c = (\rho, \tau, h, d)$ define el grafo de dependencias $G_c = (V, E_c, L_c)$ donde:

1. $E_c = \{(i, j) | h(j) = i, j = w_1, w_2, ..., w_n\}$
2. $L_c = \{((i, j), r) | d(j) = r, j = w_1, w_2, ..., w_n\}$

Definiciones previas

Definición

Grafo definido por una configuración

La configuración $c = (\rho, \tau, h, d)$ define el grafo de dependencias $\mathrm{G}_c = (\mathrm{V}, \mathrm{E}_c, \mathrm{L}_c)$ donde:

1. $\mathrm{E}_c = \{(i, j) | h(j) = i, j = w_1, w_2, ..., w_n\}$
2. $\mathrm{L}_c = \{((i, j), r) | d(j) = r, j = w_1, w_2, ..., w_n\}$

Toda configuración por la que pasa el algoritmo define un grafo de dependencias válido.

Definiciones previas

Definición

Grafo definido por una configuración

La configuración $c = (\rho, \tau, h, d)$ define el grafo de dependencias $G_c = (V, E_c, L_c)$ donde:

1. $E_c = \{(i, j) | h(j) = i, j = w_1, w_2, ..., w_n\}$
2. $L_c = \{((i, j), r) | d(j) = r, j = w_1, w_2, ..., w_n\}$

Toda configuración por la que pasa el algoritmo define un grafo de dependencias válido.

Se parte de una configuración inicial donde todos los nodos $token$ dependen del nodo raíz, con etiqueta $root$.

Definiciones previas

Definición

Configuración inicial

La configuración $c = (\mathrm{R}oot, (1, 2, ..., n), h_0, d_0)$ es una configuración inicial sii:

1. $h_0(i) = \mathrm{R}oot$, para todo $i \in \mathrm{V}^+$
2. $d_0(i) = root$, para todo $i \in \mathrm{V}^+$

Definiciones previas

Definición

Configuración inicial

La configuración $c = (\mathrm{R}oot, (1,2,...,n), h_0, d_0)$ es una configuración inicial sii:

1. $h_0(i) = \mathrm{R}oot$, para todo $i \in \mathrm{V}^+$
2. $d_0(i) = root$, para todo $i \in \mathrm{V}^+$

Inicializamos todos los nodos dependiendo de la raíz, con la correspondiente etiqueta. Al final del algoritmo, los nodos para los cuales no se encontró $head$ quedan dependiendo de la raíz y se asegura la buena formación del árbol de dependencias.

Definiciones previas

Definición

Configuración inicial

La configuración $c = (\mathrm{R}oot, (1,2,...,n), h_0, d_0)$ es una configuración inicial sii:

1. $h_0(i) = \mathrm{R}oot$, para todo $i \in \mathrm{V}^+$
2. $d_0(i) = root$, para todo $i \in \mathrm{V}^+$

Inicializamos todos los nodos dependiendo de la raíz, con la correspondiente etiqueta. Al final del algoritmo, los nodos para los cuales no se encontró $head$ quedan dependiendo de la raíz y se asegura la buena formación del árbol de dependencias.

El algoritmo opera por transiciones hasta llegar a una configuración final, en esta se debe haber vaciado la cola de entrada.

Definiciones previas

Definición

Configuración final

Definiciones previas

Definición

Configuración final

Cualquier configuración $c = (\rho, \epsilon, h, d)$ es una configuración final.

Definiciones previas

Definición

Configuración final

Cualquier configuración $c = (\rho, \epsilon, h, d)$ es una configuración final.

Solo exigimos que se agote la cola de entrada. De este modo, habremos realizado una pasada completa, de izquierda a derecha, sobre la tira de entrada, y se habrá formado el árbol de dependencias.

Para especificar el algoritmo, solo resta definir cuáles son las transiciones posibles. Las transiciones se definen como mapeos entre configuraciones; definimos a continuación notación para configuraciones:

- C – conjunto de configuraciones posibles (dada una oración x).
- C^0 – conjunto de configuraciones iniciales
- C^ϵ – conjunto de configuraciones finales
- C^n – conjunto de configuraciones no finales

Para especificar el algoritmo, solo resta definir cuáles son las transiciones posibles. Las transiciones se definen como mapeos entre configuraciones; definimos a continuación notación para configuraciones:

- C – conjunto de configuraciones posibles (dada una oración x).
- C^0 – conjunto de configuraciones iniciales
- C^ϵ – conjunto de configuraciones finales
- C^n – conjunto de configuraciones no finales

Notar que C^0 es siempre $((\mathrm{R}oot),(1,2,...,n),h_0,d_0)$.

Para especificar el algoritmo, solo resta definir cuáles son las transiciones posibles. Las transiciones se definen como mapeos entre configuraciones; definimos a continuación notación para configuraciones:

- C – conjunto de configuraciones posibles (dada una oración x).
- C^0 – conjunto de configuraciones iniciales
- C^ϵ – conjunto de configuraciones finales
- C^n – conjunto de configuraciones no finales

Notar que C^0 es siempre $((\mathrm{R}oot),(1,2,...,n),h_0,d_0)$.
Solo se puede aplicar una transición a una configuración $\in \mathrm{C}^n$.

Algoritmo

Definición

Transición

Una transición es una función parcial $\mathrm{C}^n \rightarrow \mathrm{C}$, de alguno de los siguientes tipos:

1. STACK-HIJO(r):
   $(\rho|i, j|\tau, h, d) \rightarrow (\rho, j|\tau, h[j \longmapsto i], d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$
2. STACK-PADRE(r):
   $(\rho|i, j|\tau, h, d) \rightarrow (\rho|i|j, \tau, h[i \longmapsto j], d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$
3. REDUCE:
   $(\rho|i, \tau, h, d) \rightarrow (\rho, \tau, h, d)$ si $h(i) \neq \mathrm{R}oot$
4. SHIFT:
   $(\rho, i|\tau, h, d) \rightarrow (\rho|i, \tau, h, d)$

Algoritmo, ejemplo, 1

Root Con inflación creciente , Economía busca acuerdo de precios .

- Configuración inicial, Stack : naranja, Cola : verde

Algoritmo, ejemplo, 2

Root Con inflación creciente , Economía busca acuerdo de precios .

- SHIFT, $(\rho, i|\tau, h, d) \rightarrow (\rho|i, \tau, h, d)$, i=Con

Algoritmo, ejemplo, 3

pobj

Root Con inflación creciente , Economía busca acuerdo de precios .

- STACK-PADRE(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$,
  i=Con, j=inflación

Algoritmo, ejemplo, 4

- STACK-PADRE(r):
  $(\rho|i, j|\tau, h, d) \rightarrow (\rho|i|j, \tau, h[i \longmapsto j], d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$,
  i=inflación, j=creciente

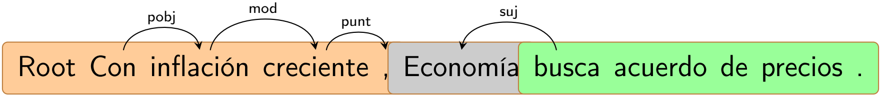
Algoritmo, ejemplo, 5

- STACK-PADRE(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$,
  i=creciente, j=,

Algoritmo, ejemplo, 6

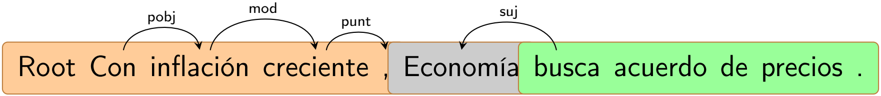
- SHIFT:
  $(\rho, i|\tau, h, d) \rightarrow (\rho|i, \tau, h, d)$, i=Economía

Algoritmo, ejemplo, 7



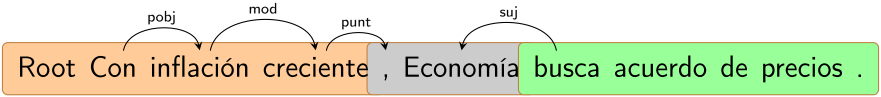
- STACK-HIJO(r):
  $(\rho|i, j|\tau, h, d) \rightarrow (\rho, j|\tau, h[j \longmapsto i], d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$,
  i=Economía, j=busca

Algoritmo, ejemplo, 7



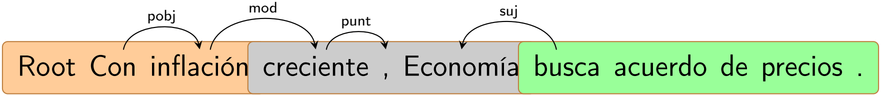
- STACK-HIJO(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho,j|\tau,h[j \longmapsto i],d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$,
  i=Economía, j=busca
- Las palabras eliminadas se ven en fondo gris

Algoritmo, ejemplo, 8



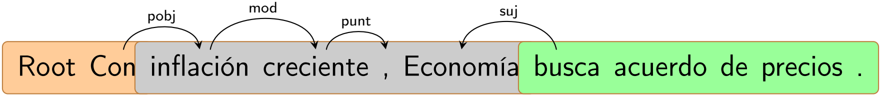
- REDUCE:
  $(\rho|i, \tau, h, d) \rightarrow (\rho, \tau, h, d)$ si $h(i) \neq \mathrm{R}oot$ i=,

Algoritmo, ejemplo, 9



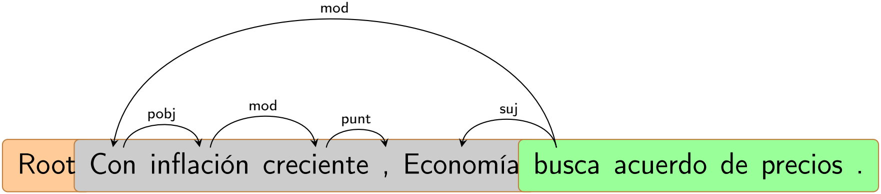
- REDUCE:
  $(\rho|i,\tau,h,d) \rightarrow (\rho,\tau,h,d)$ si $h(i) \neq \mathrm{R}oot$ i=creciente

Algoritmo, ejemplo, 10



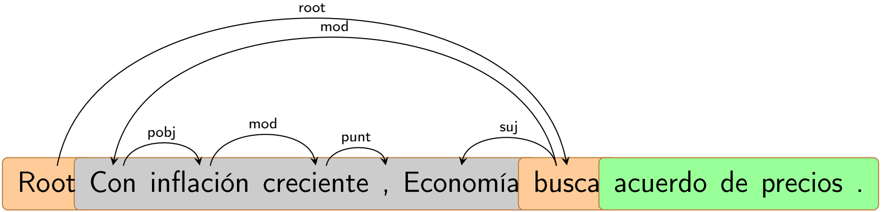
- REDUCE:
  $(\rho|i, \tau, h, d) \rightarrow (\rho, \tau, h, d)$ si $h(i) \neq Root$, i=inflación

Algoritmo, ejemplo, 11



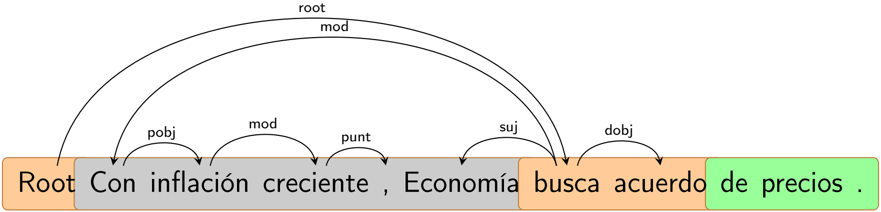
- STACK-HIJO(r):
  $(\rho|i, j|\tau, h, d) \rightarrow (\rho, j|\tau, h[j \longmapsto i], d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$,
  i=con, j=busca

Algoritmo, ejemplo, 12



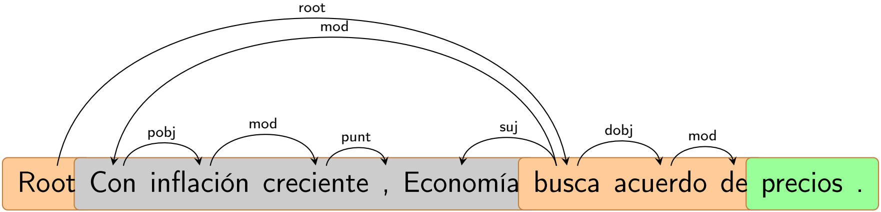
- STACK-PADRE(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$,
  i=Root, j=busca

Algoritmo, ejemplo, 13



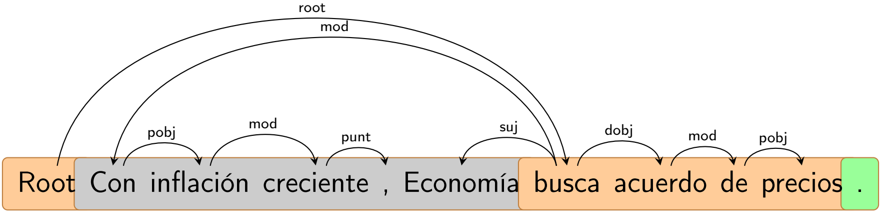
- STACK-PADRE(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$,
  i=busca, j=acuerdo

Algoritmo, ejemplo, 14



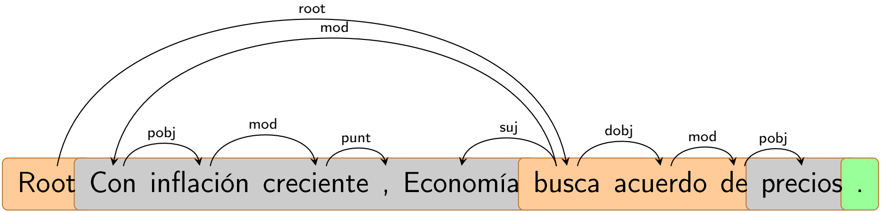
- STACK-PADRE(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$,
  i=acuerdo, j=de

Algoritmo, ejemplo, 15



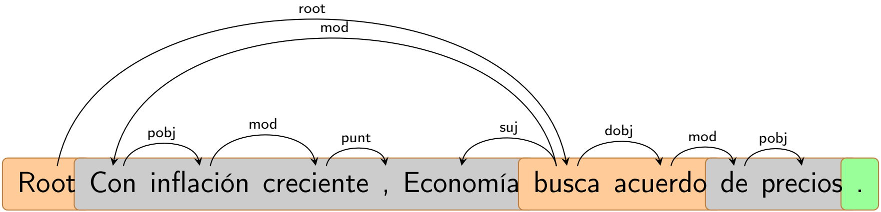
- STACK-PADRE(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j)=\mathrm{R}oot$,
  i=de, j=precios

Algoritmo, ejemplo, 16



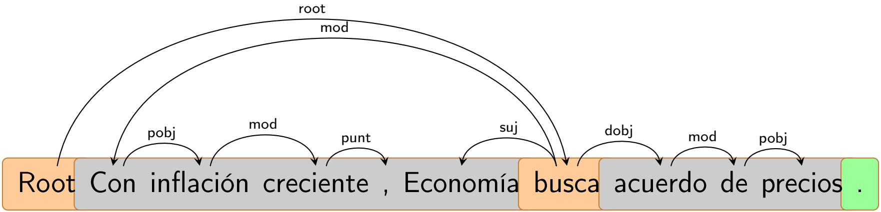
- REDUCE:
  $(\rho|i, \tau, h, d) \rightarrow (\rho, \tau, h, d)$ si $h(i) \neq Root$, i=precios

Algoritmo, ejemplo, 17



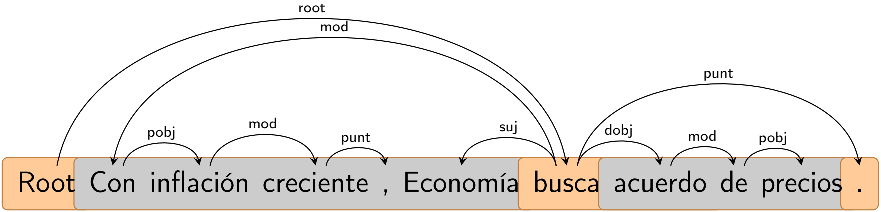
- REDUCE:
  $(\rho|i, \tau, h, d) \rightarrow (\rho, \tau, h, d)$ si $h(i) \neq Root$, i=de

Algoritmo, ejemplo, 18



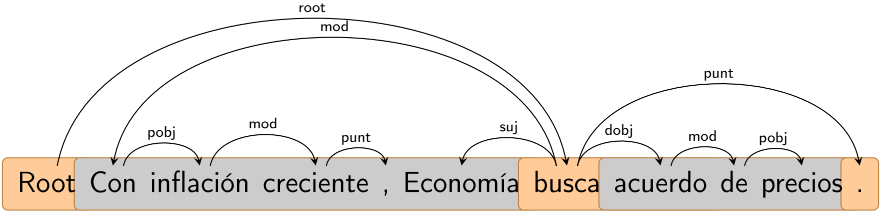
- REDUCE:
  $(\rho|i, \tau, h, d) \rightarrow (\rho, \tau, h, d)$ si $h(i) \neq \mathrm{R}oot$, i=acuerdo

Algoritmo, ejemplo, 19



- STACK-PADRE(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$,
  i=busca, j=.

Algoritmo, ejemplo, 19



- STACK-PADRE(r):
  $(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$,
  i=busca, j=.
- FIN , la cola de entrada está vacía !!

Algoritmo, observaciones

STACK-HIJO(r):

$(\rho|i, j|\tau, h, d) \rightarrow (\rho, j|\tau, h[j \longmapsto i], d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$

Algoritmo, observaciones

STACK-HIJO(r):

$(\rho|i,j|\tau,h,d) \to (\rho,j|\tau,h[j \longmapsto i],d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$

- STACK-HIJO genera un hijo izquierdo (i) del nodo j.

Algoritmo, observaciones

STACK-HIJO(r):
$(\rho|i, j|\tau, h, d) \rightarrow (\rho, j|\tau, h[j \longmapsto i], d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$

- STACK-HIJO genera un hijo izquierdo (i) del nodo j.
- Se debe cumplir que a i no se le haya asignado antes un head (el algoritmo no asigna head más de una vez por cada nodo).

Algoritmo, observaciones

STACK-HIJO(r):
$(\rho|i, j|\tau, h, d) \rightarrow (\rho, j|\tau, h[j \longmapsto i], d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$

- STACK-HIJO genera un hijo izquierdo (i) del nodo j.
- Se debe cumplir que a i no se le haya asignado antes un head (el algoritmo no asigna head más de una vez por cada nodo).
- Notar que i no puede tener descendientes entre los nodos que falta procesar, ya que en ese caso se violaría la proyectividad. i se saca del stack

Algoritmo, observaciones

STACK-HIJO(r):
$(\rho|i,j|\tau,h,d) \to (\rho,j|\tau,h[j \longmapsto i],d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$

- STACK-HIJO genera un hijo izquierdo (i) del nodo j.
- Se debe cumplir que a i no se le haya asignado antes un head (el algoritmo no asigna head más de una vez por cada nodo).
- Notar que i no puede tener descendientes entre los nodos que falta procesar, ya que en ese caso se violaría la proyectividad. i se saca del stack
- j (el padre) podría tener más hijos izquierdos en el stack. Sigue entonces en la cola.

Algoritmo, observaciones

STACK-HIJO(r):
$(\rho|i, j|\tau, h, d) \rightarrow (\rho, j|\tau, h[j \longmapsto i], d[i \longmapsto r])$, si $h(i) = \mathrm{R}oot$

- STACK-HIJO genera un hijo izquierdo (i) del nodo j.
- Se debe cumplir que a i no se le haya asignado antes un head (el algoritmo no asigna head más de una vez por cada nodo).
- Notar que i no puede tener descendientes entre los nodos que falta procesar, ya que en ese caso se violaría la proyectividad. i se saca del stack
- j (el padre) podría tener más hijos izquierdos en el stack. Sigue entonces en la cola.

Ejemplos: *Juan corre. / Juan últimamente ha corrido mucho.*

Algoritmo, observaciones

STACK-PADRE(r):

$(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$

Algoritmo, observaciones

STACK-PADRE(r):
$(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$

- STACK-PADRE genera un hijo derecho (j) del nodo i en el tope del stack.

Algoritmo, observaciones

STACK-PADRE(r):

$(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$

- STACK-PADRE genera un hijo derecho (j) del nodo i en el tope del stack.
- Ambos nodos deben quedar en el stack, ya que tanto i como j pueden tener más hijos derechos.

Algoritmo, observaciones

STACK-PADRE(r):
$(\rho|i,j|\tau,h,d) \rightarrow (\rho|i|j,\tau,h[i \longmapsto j],d[j \longmapsto r])$, si $h(j) = \mathrm{R}oot$

- STACK-PADRE genera un hijo derecho (j) del nodo i en el tope del stack.
- Ambos nodos deben quedar en el stack, ya que tanto i como j pueden tener más hijos derechos.

Ejemplos: (*comen huesos / comen huesos lentamente / comen huesos de vaca*)

Algoritmo, observaciones

REDUCE:
$(\rho|i, \tau, h, d) \rightarrow (\rho, \tau, h, d)$ si $h(i) \neq \mathrm{R}oot$

Algoritmo, observaciones

REDUCE:
$(\rho|i,\tau,h,d) \rightarrow (\rho,\tau,h,d)$ si $h(i) \neq Root$

- REDUCE - saca el elemento del tope del stack.

Algoritmo, observaciones

REDUCE:

$(\rho|i, \tau, h, d) \rightarrow (\rho, \tau, h, d)$ si $h(i) \neq \mathrm{R}oot$

- REDUCE - saca el elemento del tope del stack.
- Es necesaria para sacar nodos que fueron puestos por STACK-PADRE y completaron sus hijos derechos.

Algoritmo, observaciones

REDUCE:
$(\rho|i,\tau,h,d) \rightarrow (\rho,\tau,h,d)$ si $h(i) \neq \mathrm{R}oot$

- REDUCE - saca el elemento del tope del stack.
- Es necesaria para sacar nodos que fueron puestos por STACK-PADRE y completaron sus hijos derechos.
- Hay que sacarlos porque el nodo anterior en el stack puede tener más hijos derechos

Algoritmo, observaciones

REDUCE:
$(\rho|i,\tau,h,d) \rightarrow (\rho,\tau,h,d)$ si $h(i) \neq \mathrm{R}oot$

- REDUCE - saca el elemento del tope del stack.
- Es necesaria para sacar nodos que fueron puestos por STACK-PADRE y completaron sus hijos derechos.
- Hay que sacarlos porque el nodo anterior en el stack puede tener más hijos derechos

Ejemplos: (*comen huesos lentamente / un banco viejo de madera*)

Algoritmo, observaciones

SHIFT:
$(\rho, i|\tau, h, d) \rightarrow (\rho|i, \tau, h, d)$

Algoritmo, observaciones

SHIFT:
$(\rho, i|\tau, h, d) \rightarrow (\rho|i, \tau, h, d)$

- SHIFT- pone un nodo en el tope del stack.

Algoritmo, observaciones

SHIFT:
$(\rho, i|\tau, h, d) \rightarrow (\rho|i, \tau, h, d)$

- SHIFT- pone un nodo en el tope del stack.
- es necesaria para procesar nodos cuya head está a la derecha.

Algoritmo, observaciones

SHIFT:
$(\rho, i|\tau, h, d) \rightarrow (\rho|i, \tau, h, d)$

- SHIFT- pone un nodo en el tope del stack.
- es necesaria para procesar nodos cuya head está a la derecha.
- Siempre se puede aplicar

Algoritmo, observaciones

SHIFT:
$(\rho, i|\tau, h, d) \rightarrow (\rho|i, \tau, h, d)$

- SHIFT- pone un nodo en el tope del stack.
- es necesaria para procesar nodos cuya head está a la derecha.
- Siempre se puede aplicar

Ejemplo: Juan corre.

Algoritmo, seudocódigo

Parse($x = (w_1, ..., w_n)$)

  $c \leftarrow ((\mathrm{R}oot), (1, ..., n), h_0, d_0)$

  while $c = (\rho, \tau, h, d)$ no es final

    $c \leftarrow [\mathrm{ORACULO}(c, x)](c)$

  $\mathrm{G} \leftarrow (\mathrm{V}_x, \mathrm{E}_c, \mathrm{L}_c)$

return G

Algoritmo, seudocódigo

Parse($x = (w_1, ..., w_n)$)

$c \leftarrow ((\mathrm{R}oot), (1, ..., n), h_0, d_0)$

while $c = (\rho, \tau, h, d)$ no es final

$c \leftarrow [\mathrm{ORACULO}(c, x)](c)$

$\mathrm{G} \leftarrow (\mathrm{V}_x, \mathrm{E}_c, \mathrm{L}_c)$

return G

ORACULO(c,x) realiza una de las 4 transiciones vistas, generando una nueva configuración.

Correctitud y orden del algoritmo (ver referencias)

Correctitud y orden del algoritmo (ver referencias)

Se prueban los siguientes teoremas:

1. Dada una oración de largo n el algoritmo termina en a lo sumo $2n-1$ transiciones.

Correctitud y orden del algoritmo (ver referencias)

Se prueban los siguientes teoremas:

1. Dada una oración de largo n el algoritmo termina en a lo sumo $2n-1$ transiciones.
2. Toda secuencia que termine de transiciones $\mathrm{C}_{0,m}$ para una oración x define un grafo de dependencias $\mathrm{G} = (\mathrm{V}_x, \mathrm{E}_m, \mathrm{L}_m)$ que cumple las siguientes 5 condiciones:

Correctitud y orden del algoritmo (ver referencias)

Se prueban los siguientes teoremas:

1. Dada una oración de largo n el algoritmo termina en a lo sumo $2n-1$ transiciones.
2. Toda secuencia que termine de transiciones $\mathrm{C}_{0,m}$ para una oración x define un grafo de dependencias $\mathrm{G} = (\mathrm{V}_x, \mathrm{E}_m, \mathrm{L}_m)$ que cumple las siguientes 5 condiciones:
   - 2.1 RAÍZ única
   - 2.2 CONEXO
   - 2.3 SINGLE-HEAD
   - 2.4 ACÍCLICO
   - 2.5 PROYECTIVO

Correctitud y orden del algoritmo (ver referencias)

Se prueban los siguientes teoremas:

1. Dada una oración de largo n el algoritmo termina en a lo sumo $2n-1$ transiciones.
2. Toda secuencia que termine de transiciones $\mathrm{C}_{0,m}$ para una oración x define un grafo de dependencias $\mathrm{G} = (\mathrm{V}_x, \mathrm{E}_m, \mathrm{L}_m)$ que cumple las siguientes 5 condiciones:
   - 2.1 RAÍZ única
   - 2.2 CONEXO
   - 2.3 SINGLE-HEAD
   - 2.4 ACÍCLICO
   - 2.5 PROYECTIVO
3. Para todo grafo proyectivo de dependencias existe una secuencia de transiciones que conduce a una configuración final.

Referencias

- Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S. and Marsi, E. (2007) MaltParser: A language-independent system for data-driven dependency parsing. Natural Language Engineering, 13(2), 95-135
- Nivre, J. (2008) Algorithms for Deterministic Incremental Dependency Parsing. Computational Linguistics 34(4), 513-553
- Ballesteros, M. and Nivre, J. (2013) Going to the Roots of Dependency Parsing. Computational Linguistics 39(1): 5-13