Teoría de la Computación

Clase 13: Parsing

Mauro Artigiani

6 septiembre 2021

Universidad del Rosario, Bogotá

Scanners y Parsers

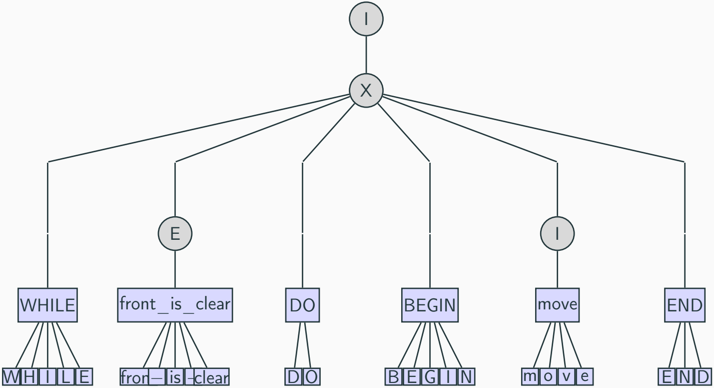
Parsing

Sea G una `CFG` y sea $w \in L(G)$. El problema de parsing para w es encontrar un árbol de parsing (o de análisis) de w en G.

Parsing

Sea G una CFG y sea $w \in L(G)$. El problema de parsing para w es encontrar un árbol de parsing (o de análisis) de w en G.

WHILE `front_is_clear` **DO** **BEGIN** `move` **END**

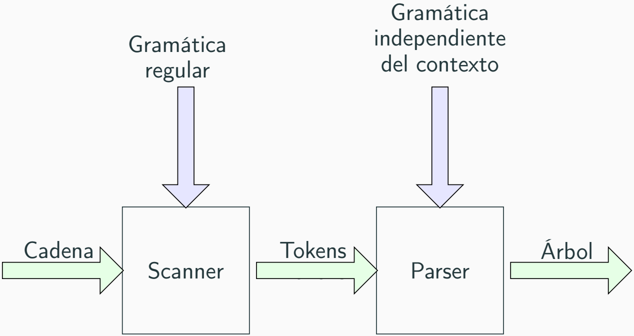


Scanners y Parsing

Un scanner divide semánticamente, la cadena en palabras o *tokens* y las envía al parser que encuentra un árbol de parsing para cada token.

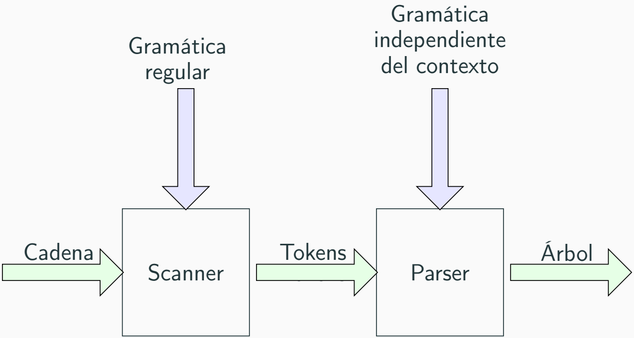
Scanners y Parsing

Un scanner divide semánticamente, la cadena en palabras o *tokens* y las envía al parser que encuentra un árbol de parsing para cada token.



Scanners y Parsing

Un scanner divide semánticamente, la cadena en palabras o *tokens* y las envía al parser que encuentra un árbol de parsing para cada token.



Usualmente, los scanners utilizan expresiones regulares (es decir: lenguajes regulares). Los parsers utilizan a menudo `CFGs`.

Métodos de parsing

Métodos de parsing

Dada una gramática independiente del contexto G y una palabra $w \in L(G)$, hay dos métodos para encontrar un árbol de parsing para w:

- El método *top-down* (arriba-abajo), que consiste en arrancar con la variable inicial S de G y buscar una derivación $S \overset{*}{\Rightarrow} w$.

Métodos de parsing

Dada una gramática independiente del contexto G y una palabra $w \in L(G)$, hay dos métodos para encontrar un árbol de parsing para w:

- El método *top-down* (arriba-abajo), que consiste en arrancar con la variable inicial S de G y buscar una derivación $S \stackrel{*}{\Rightarrow} w$.
- El método *bottom-up* (abajo-arriba), que consiste en empezar con la palabra w y buscar *escalar* hacía la variable inicial S.

Métodos de parsing

Dada una gramática independiente del contexto G y una palabra $w \in L(G)$, hay dos métodos para encontrar un árbol de parsing para w:

- El método *top-down* (arriba-abajo), que consiste en arrancar con la variable inicial S de G y buscar una derivación $S \overset{*}{\Rightarrow} w$.
- El método *bottom-up* (abajo-arriba), que consiste en empezar con la palabra w y buscar *escalar* hacía la variable inicial S.

El primer método se puede hacer incluso de *fuerza bruta*.

Métodos de parsing

Dada una gramática independiente del contexto G y una palabra $w \in L(G)$, hay dos métodos para encontrar un árbol de parsing para w:

- El método *top-down* (arriba-abajo), que consiste en arrancar con la variable inicial S de G y buscar una derivación $S \stackrel{*}{\Rightarrow} w$.
- El método *bottom-up* (abajo-arriba), que consiste en empezar con la palabra w y buscar *escalar* hacía la variable inicial S.

El primer método se puede hacer incluso de *fuerza bruta*.

Para el segundo, utilizaremos los *autómatas deterministas*, por lo menos para una clase especial de gramáticas.

Método *top-down*

Veamos un ejemplo del método *top-down* para la palabra

$$\Rightarrow q\neg p$$

en la gramática G.

$$
\begin{aligned}
G = \{S &\to NF \mid DF \mid p \mid q \\
F &\to NF \mid DF \mid p \mid q \\
D &\to CF \\
N &\to \neg \\
C &\to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\}
\end{aligned}
$$

$\boxed{\Rightarrow}$ $\boxed{q}$ $\boxed{\neg}$ $\boxed{p}$

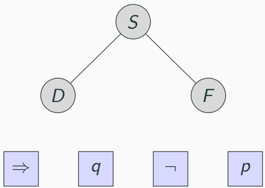
Método *top-down*

Veamos un ejemplo del método *top-down* para la palabra

$$\Rightarrow q\neg p$$

en la gramática G.

$$
\begin{aligned}
G = \{S &\rightarrow NF \mid DF \mid p \mid q \\
F &\rightarrow NF \mid DF \mid p \mid q \\
D &\rightarrow CF \\
N &\rightarrow \neg \\
C &\rightarrow \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\}
\end{aligned}
$$



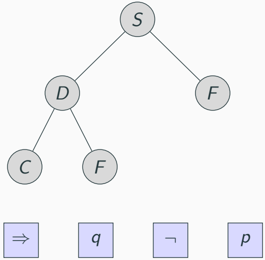
Método *top-down*

Veamos un ejemplo del método *top-down* para la palabra

$$\Rightarrow q\neg p$$

en la gramática G.

$$
\begin{aligned}
G = \{S &\to NF \mid DF \mid p \mid q\\
F &\to NF \mid DF \mid p \mid q\\
D &\to CF\\
N &\to \neg\\
C &\to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\}
\end{aligned}
$$



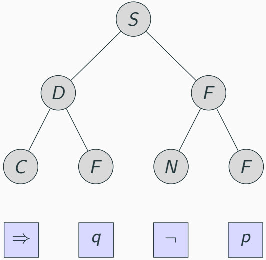
Método *top-down*

Veamos un ejemplo del método *top-down* para la palabra

$$\Rightarrow q\neg p$$

en la gramática G.

$$G = \{S \to NF \mid DF \mid p \mid q$$
$$F \to NF \mid DF \mid p \mid q$$
$$D \to CF$$
$$N \to \neg$$
$$C \to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\}$$



Top-down por fuerza bruta

Un computador no sabe adivinar cual es la elección correcta para cada derivación. Entonces tiene que tratar todos los casos, por fuerza bruta.

Top-down por fuerza bruta

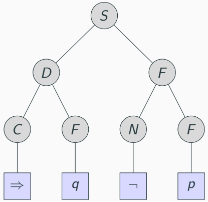
Un computador no sabe adivinar cual es la elección correcta para cada derivación. Entonces tiene que tratar todos los casos, por fuerza bruta.

Supongamos que G sea en forma normal de Chomsky. En este caso, cualquier árbol de parsing será un árbol binario. Veamos en este caso cuántos casos se necesitan.

Top-down por fuerza bruta

Un computador no sabe adivinar cual es la elección correcta para cada derivación. Entonces tiene que tratar todos los casos, por fuerza bruta.

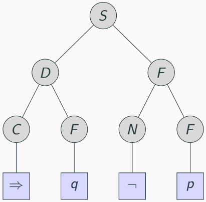
Supongamos que G sea en forma normal de Chomsky. En este caso, cualquier árbol de parsing será un árbol binario. Veamos en este caso cuántos casos se necesitan.



Top-down por fuerza bruta

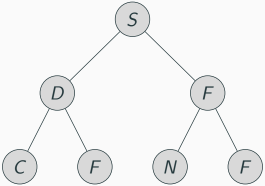
Un computador no sabe adivinar cual es la elección correcta para cada derivación. Entonces tiene que tratar todos los casos, por fuerza bruta.

Supongamos que G sea en forma normal de Chomsky. En este caso, cualquier árbol de parsing será un árbol binario. Veamos en este caso cuántos casos se necesitan.

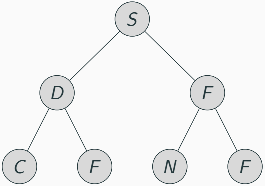


Para una cadena de longitud n se requieren n aplicaciones de reglas de tipo $A \to a$.

Top-down por fuerza bruta

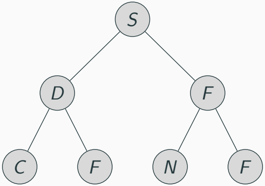


Top-down por fuerza bruta



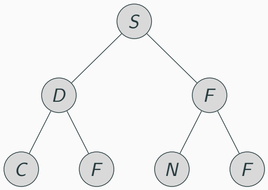
El número de reglas de tipo $A \to BC$ es igual a la mitad de aristas del árbol.

Top-down por fuerza bruta



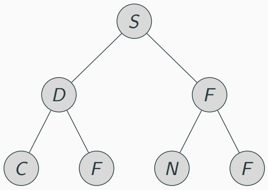
El número de reglas de tipo $A \rightarrow BC$ es igual a la mitad de aristas del árbol. En un árbol binario con n hojas tenemos $2(n-1)$ aristas.

Top-down por fuerza bruta



El número de reglas de tipo $A \to BC$ es igual a la mitad de aristas del árbol. En un árbol binario con n hojas tenemos $2(n-1)$ aristas. Entonces, necesitamos aplicar $n-1$ reglas de tipo $A \to BC$.

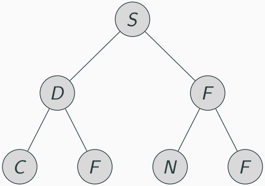
Top-down por fuerza bruta



El número de reglas de tipo $A \to BC$ es igual a la mitad de aristas del árbol. En un árbol binario con n hojas tenemos $2(n-1)$ aristas. Entonces, necesitamos aplicar $n-1$ reglas de tipo $A \to BC$.

En total tenemos que aplicar $n + (n-1) = 2n - 1$ reglas.

Top-down por fuerza bruta



El número de reglas de tipo $A \to BC$ es igual a la mitad de aristas del árbol. En un árbol binario con n hojas tenemos $2(n-1)$ aristas. Entonces, necesitamos aplicar $n-1$ reglas de tipo $A \to BC$.

En total tenemos que aplicar $n + (n-1) = 2n - 1$ reglas. En conclusión, si la gramática tiene m reglas, hay

$$m^{2n-1}$$

posibles combinaciones.

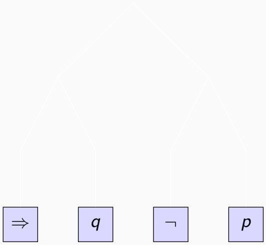
Método *bottom-up*

Veamos ahora un ejemplo del método *bottom-up* para la misma palabra

$$\Rightarrow q\neg p$$

en la gramática G.

$$
\begin{aligned}
G = \{S &\to NF \mid DF \mid p \mid q \\
F &\to NF \mid DF \mid p \mid q \\
D &\to CF \\
N &\to \neg \\
C &\to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\}
\end{aligned}
$$



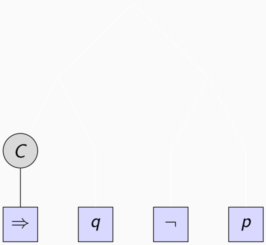
Método *bottom-up*

Veamos ahora un ejemplo del método *bottom-up* para la misma palabra

$$\Rightarrow q \neg p$$

en la gramática G.

$$\begin{aligned} G = \{ S &\to NF \mid DF \mid p \mid q \\ F &\to NF \mid DF \mid p \mid q \\ D &\to CF \\ N &\to \neg \\ C &\to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow \} \end{aligned}$$



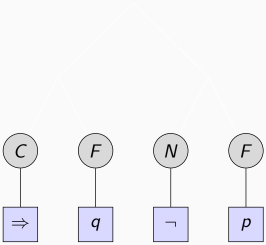
Método *bottom-up*

Veamos ahora un ejemplo del método *bottom-up* para la misma palabra

$$\Rightarrow q \neg p$$

en la gramática G.

$$
\begin{aligned}
G = \{S &\rightarrow NF \mid DF \mid p \mid q \\
F &\rightarrow NF \mid DF \mid p \mid q \\
D &\rightarrow CF \\
N &\rightarrow \neg \\
C &\rightarrow \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\}
\end{aligned}
$$



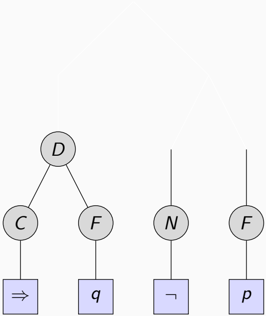
Método *bottom-up*

Veamos ahora un ejemplo del método *bottom-up* para la misma palabra

$$\Rightarrow q \neg p$$

en la gramática G.

$$\begin{aligned} G = \{ S &\to NF \mid DF \mid p \mid q \\ F &\to NF \mid DF \mid p \mid q \\ D &\to CF \\ N &\to \neg \\ C &\to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow \} \end{aligned}$$



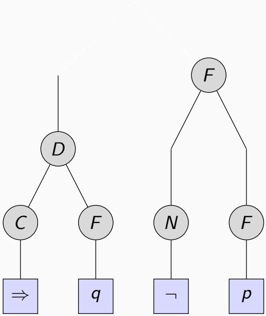
Método *bottom-up*

Veamos ahora un ejemplo del método *bottom-up* para la misma palabra

$$\Rightarrow q\neg p$$

en la gramática G.

$$\begin{aligned} G = \{S &\to NF \mid DF \mid p \mid q \\ F &\to NF \mid DF \mid p \mid q \\ D &\to CF \\ N &\to \neg \\ C &\to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\} \end{aligned}$$



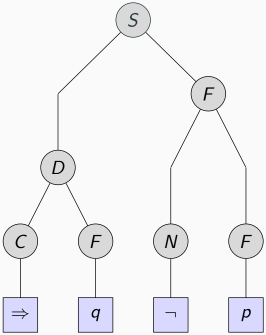
Método *bottom-up*

Veamos ahora un ejemplo del método *bottom-up* para la misma palabra

$$\Rightarrow q\neg p$$

en la gramática G.

$$
\begin{aligned}
G = \{S &\to NF \mid DF \mid p \mid q\\
F &\to NF \mid DF \mid p \mid q\\
D &\to CF\\
N &\to \neg\\
C &\to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\}
\end{aligned}
$$



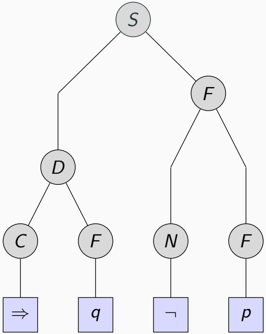
Método *bottom-up*

Veamos ahora un ejemplo del método *bottom-up* para la misma palabra

$$\Rightarrow q\neg p$$

en la gramática G.

$$\begin{aligned} G = \{S &\to NF \mid DF \mid p \mid q \\ F &\to NF \mid DF \mid p \mid q \\ D &\to CF \\ N &\to \neg \\ C &\to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\} \end{aligned}$$



Veamos ahora como realizar de manera algorítmica este método.

Autómatas de Knuth

Manijas

Para escalar se necesitan buenos agarres en la pared. Nuestro agarres serán las manijas.

Manijas

Para escalar se necesitan buenos agarres en la pared. Nuestro agarres serán las manijas.

Una manija para una cadena $w = xhy$ es una regla de la forma $A \to h$.

Manijas

Para escalar se necesitan buenos agarres en la pared. Nuestro agarres serán las manijas.

Una manija para una cadena $w = xhy$ es una regla de la forma $A \to h$. La regla $A \to h$ permite reducir la expresión xhy a la expresión xAy. Las manijas se denotan así:

$$xhy \rightarrowtail xAy.$$

Manijas

Para escalar se necesitan buenos agarres en la pared. Nuestro agarres serán las manijas.

Una manija para una cadena $w = xhy$ es una regla de la forma $A \to h$. La regla $A \to h$ permite reducir la expresión xhy a la expresión xAy. Las manijas se denotan así:

$$xhy \rightarrowtail xAy.$$

Veamos un ejemplo, para la gramática

$$G = \{S \to NF \mid DF \mid p \mid q, F \to NF \mid DF \mid p \mid q, D \to CF, N \to \neg, \\ C \to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\},$$

y la palabra $\Rightarrow q\neg p$.

Manijas

Para escalar se necesitan buenos agarres en la pared. Nuestro agarres serán las manijas.

Una manija para una cadena $w = xhy$ es una regla de la forma $A \to h$. La regla $A \to h$ permite reducir la expresión xhy a la expresión xAy. Las manijas se denotan así:

$$xhy \rightarrowtail xAy.$$

Veamos un ejemplo, para la gramática

$$G = \{S \to NF \mid DF \mid p \mid q, F \to NF \mid DF \mid p \mid q, D \to CF, N \to \neg,$$
$$C \to \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow\},$$

y la palabra $\Rightarrow q\neg p$.

$$\underline{\Rightarrow} q\neg p \rightarrowtail C\underline{q}\neg p \rightarrowtail CF\underline{\neg} p \rightarrowtail CFN\underline{p} \rightarrowtail \underline{CF}NF \rightarrowtail$$
$$\rightarrowtail D\underline{NF} \rightarrowtail \underline{DF} \rightarrowtail S$$

Búsqueda de manijas

Describiremos como buscar manijas para una clase más sencilla de gramáticas independientes del contexto:

Búsqueda de manijas

Describiremos como buscar manijas para una clase más sencilla de gramáticas independientes del contexto:

Definición

Decimos que una `CFG` G es determinista si cada cadena en el procesamiento *bottom-up* de una palabra w en $L(G)$ tiene sólo una manija.

Búsqueda de manijas

Describiremos como buscar manijas para una clase más sencilla de gramáticas independientes del contexto:

Definición

Decimos que una `CFG` G es determinista si cada cadena en el procesamiento *bottom-up* de una palabra w en $L(G)$ tiene sólo una manija.

Determinista:

$$G = \{F \to NF \mid DF$$
$$D \to CF\}$$

No determinista:

$$G' = \{S \to NF \mid DF$$
$$\{F \to NF \mid DF$$
$$D \to CF\}$$

Búsqueda de manijas

Describiremos como buscar manijas para una clase más sencilla de gramáticas independientes del contexto:

Definición

Decimos que una `CFG` G es determinista si cada cadena en el procesamiento *bottom-up* de una palabra w en $L(G)$ tiene sólo una manija.

Determinista:

$$G = \{F \to NF \mid DF \\ D \to CF\}$$

No determinista:

$$G' = \{S \to NF \mid DF \\ \{F \to NF \mid DF \\ D \to CF\}$$

Ahora construiremos un autómata no determinista mediante el cual buscar cada manija en el procesamiento *bottom-up* de w.

El autómata K para encontrar manijas

Para describir en qué lugar estamos haciendo la comparación de nuestra palabras introducimos las reglas punteadas.

El autómata K para encontrar manijas

Para describir en qué lugar estamos haciendo la comparación de nuestra palabras introducimos las reglas punteadas. Una regla punteada es una regla más un punto. Por ejemplo, de la regla $F \to NF$ salen las siguientes reglas punteadas:

$$F \to .NF \qquad F \to N.F \qquad F \to NF.$$

El autómata K para encontrar manijas

Para describir en qué lugar estamos haciendo la comparación de nuestra palabras introducimos las reglas punteadas. Una regla punteada es una regla más un punto. Por ejemplo, de la regla $F \to NF$ salen las siguientes reglas punteadas:

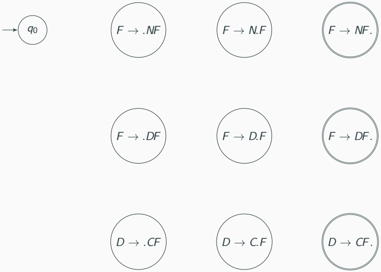
$$F \to .NF \qquad F \to N.F \qquad F \to NF.$$

El `NFA` K que estamos construyendo tendrá un estado inicial q_0 y todos los estados correspondientes a reglas punteadas.

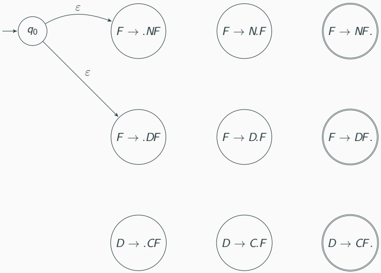
$$\boxed{F \to .NF} \qquad \boxed{F \to N.F} \qquad \boxed{\boxed{F \to NF.}}$$

Las reglas en donde un punto esté al final del lado derecho serán nuestro estados de aceptación.

El autómata K para encontrar manijas:

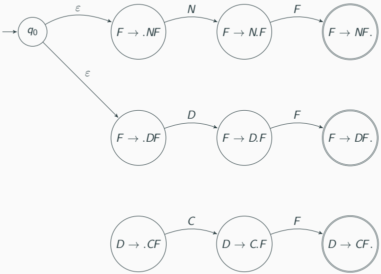


El autómata K para encontrar manijas:



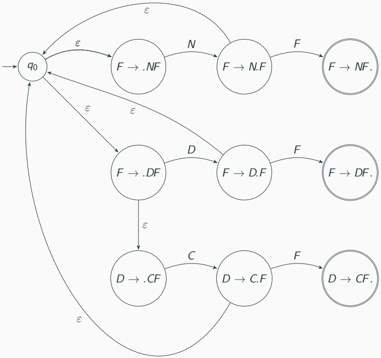
Ponemos transiciones ε de q_0 a las reglas con el símbolo inicial.

El autómata K para encontrar manijas:



Definimos *shift-moves* (reglas de desplazamiento) entre las reglas punteadas correspondientes: $\boxed{B \to U.AV} \xrightarrow{A} \boxed{B \to UA.V}$

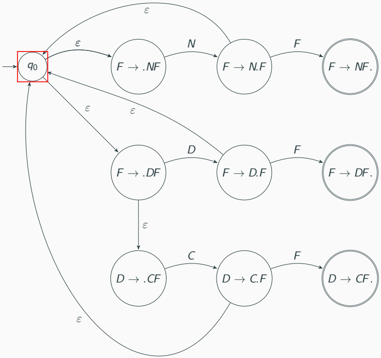
El autómata K para encontrar manijas:



Definimos *ε-moves* desde $.C$ hasta una regla que comienza con C:

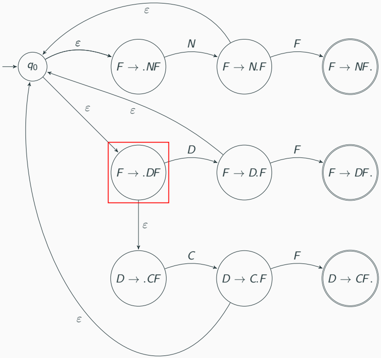
$$\boxed{B \to U.CV} \xrightarrow{\varepsilon} \boxed{C \to .R}$$

El autómata K para encontrar manijas:



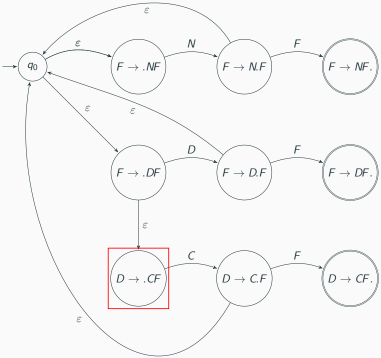
Ejemplo: $CFNF$

El autómata K para encontrar manijas:



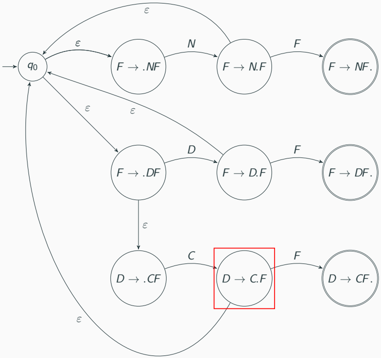
Ejemplo: $CFNF$

El autómata K para encontrar manijas:



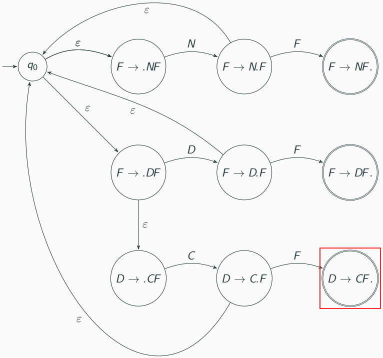
Ejemplo: $CFNF$

El autómata K para encontrar manijas:



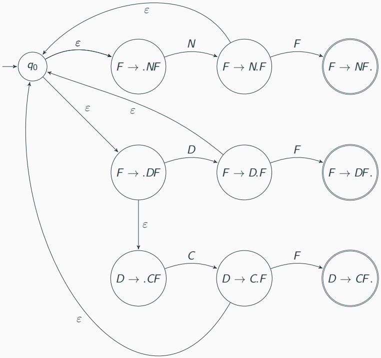
Ejemplo: $CFNF$

El autómata K para encontrar manijas:



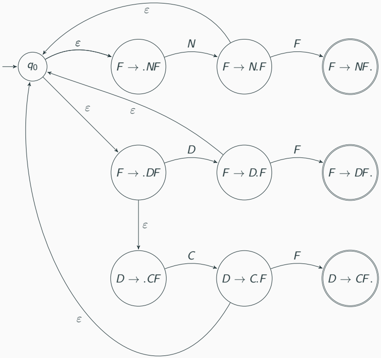
Ejemplo: $\underline{CF}NF$ ¡Se encontró la manija!

El autómata K para encontrar manijas:



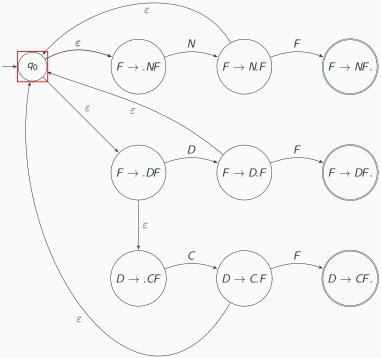
Ejemplo: $\underline{CF}NF \rightarrowtail DNF$

El autómata K para encontrar manijas:



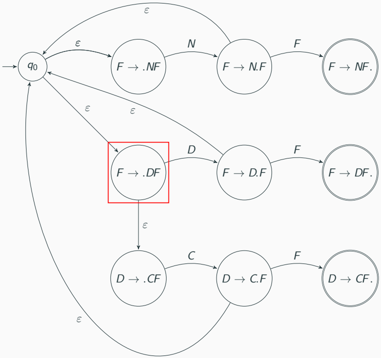
Ejemplo: Ahora se inicializa el autómata para procesar *DNF*

El autómata K para encontrar manijas:



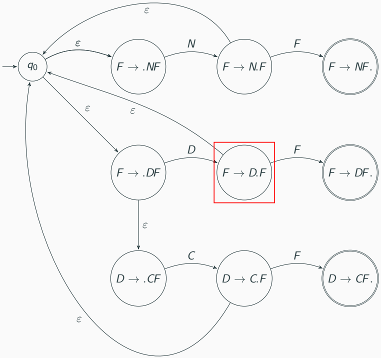
Ejemplo: DNF

El autómata K para encontrar manijas:



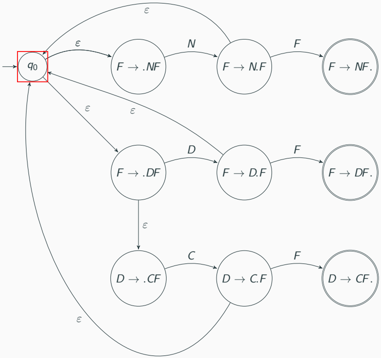
Ejemplo: DNF

El autómata K para encontrar manijas:



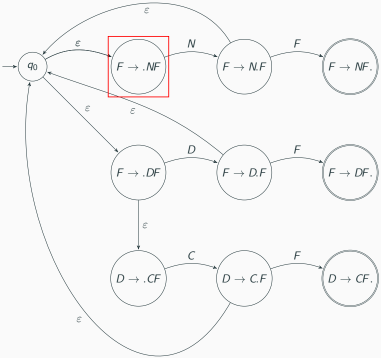
Ejemplo: DNF

El autómata K para encontrar manijas:



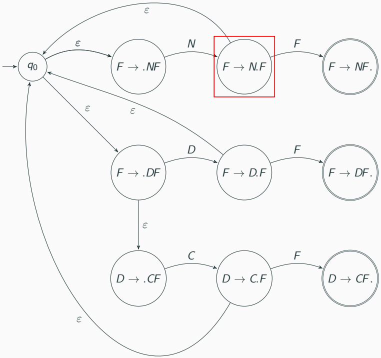
Ejemplo: DNF

El autómata K para encontrar manijas:



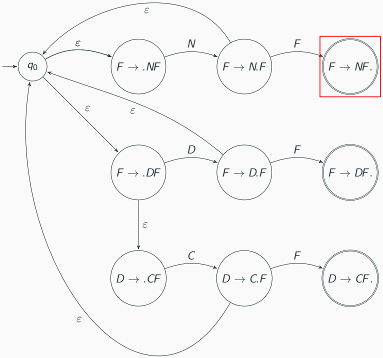
Ejemplo: DNF

El autómata K para encontrar manijas:



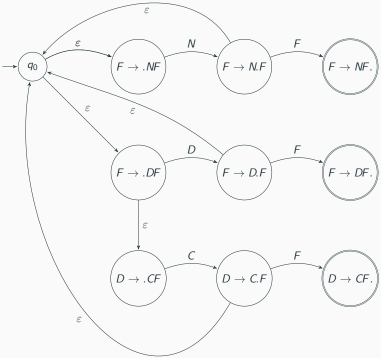
Ejemplo: DNF

El autómata K para encontrar manijas:



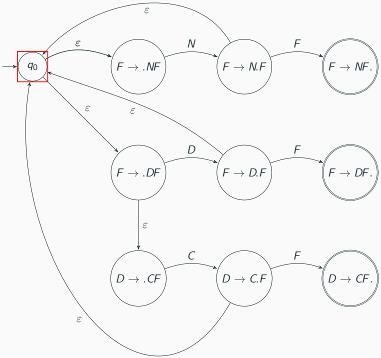
Ejemplo: $D\underline{NF} \rightarrowtail DF$

El autómata K para encontrar manijas:



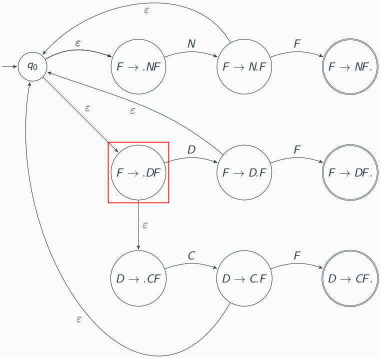
Ejemplo: Ahora se inicializa el autómata para procesar DF

El autómata K para encontrar manijas:



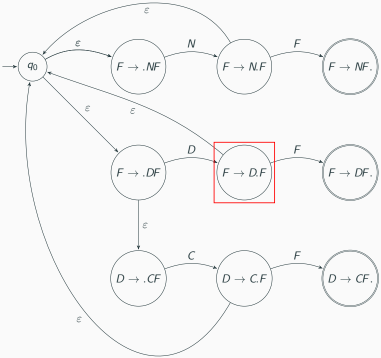
Ejemplo: DF

El autómata K para encontrar manijas:



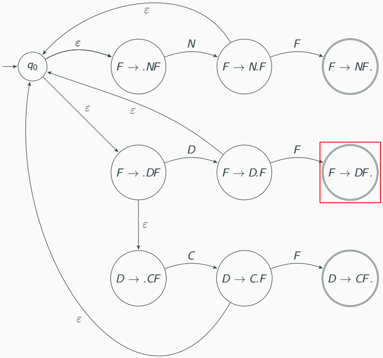
Ejemplo: DF

El autómata K para encontrar manijas:



Ejemplo: DF

El autómata K para encontrar manijas:



Ejemplo: $\underline{DF} \rightarrowtail F$

Autómata DK

DK, La versión determinista de K

Sabemos que cualquier autómata no determinista tiene su equivalente determinista.

DK, La versión determinista de K

Sabemos que cualquier autómata no determinista tiene su equivalente determinista.

Desafortunadamente, el procedimiento estándar para transformar el `NFA` K en DK, su equivalente `DFA`, produciría un autómata enorme. Por ejemplo, en el ejemplo anterior obtendríamos un autómata con $2^{10} = 1024$ estados!

DK, La versión determinista de K

Sabemos que cualquier autómata no determinista tiene su equivalente determinista.

Desafortunadamente, el procedimiento estándar para transformar el `NFA` K en DK, su equivalente `DFA`, produciría un autómata enorme. Por ejemplo, en el ejemplo anterior obtendríamos un autómata con $2^{10} = 1024$ estados!

Vamos a mostrar una manera más fácil (y económica) para construir DK.

DK, La versión determinista de K

Sabemos que cualquier autómata no determinista tiene su equivalente determinista.

Desafortunadamente, el procedimiento estándar para transformar el `NFA` K en DK, su equivalente `DFA`, produciría un autómata enorme. Por ejemplo, en el ejemplo anterior obtendríamos un autómata con $2^{10} = 1024$ estados!

Vamos a mostrar una manera más fácil (y económica) para construir DK.

Simplificamos nuestra gramática G quitandole los terminales y la repetición entre las reglas $S \to \ldots$ y $F \to \ldots$. Obtenemos una gramática determinsta:

$$G = \{F \to NF \mid DF, D \to CF\}.$$

Construcción del autómata DK

Veamos cómo construir el `DFA` DK.

Construcción del autómata DK

Veamos cómo construir el `DFA` DK.

1. Creamos un estado inicial q_0 con todas las reglas punteadas que arrancan con el símbolo inicial y el punto al inicio.

Construcción del autómata DK

Veamos cómo construir el `DFA` DK.

1. Creamos un estado inicial q_0 con todas las reglas punteadas que arrancan con el símbolo inicial y el punto al inicio.
2. Si el estado en consideración tiene una regla de tipo: $A \to B.C$ añadimos al estado todas la reglas punteadas que arrancan con $.C$.

Construcción del autómata *DK*

Veamos cómo construir el `DFA` DK.

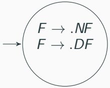
1. Creamos un estado inicial q_0 con todas las reglas punteadas que arrancan con el símbolo inicial y el punto al inicio.
2. Si el estado en consideración tiene una regla de tipo: $A \to B.C$ añadimos al estado todas la reglas punteadas que arrancan con $.C$. Así seguimos este proceso hasta que no se pueda repetir más.

Construcción del autómata DK

Veamos cómo construir el `DFA` DK.

1. Creamos un estado inicial q_0 con todas las reglas punteadas que arrancan con el símbolo inicial y el punto al inicio.
2. Si el estado en consideración tiene una regla de tipo: $A \to B.C$ añadimos al estado todas la reglas punteadas que arrancan con $.C$. Así seguimos este proceso hasta que no se pueda repetir más.

Si tenemos la gramática $G = \{F \to NF \mid DF, D \to CF\}$, empezamos con el estado inicial:

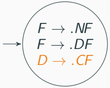


Construcción del autómata DK

Veamos cómo construir el `DFA` DK.

1. Creamos un estado inicial q_0 con todas las reglas punteadas que arrancan con el símbolo inicial y el punto al inicio.
2. Si el estado en consideración tiene una regla de tipo: $A \to B.C$ añadimos al estado todas la reglas punteadas que arrancan con $.C$. Así seguimos este proceso hasta que no se pueda repetir más.

Si tenemos la gramática $G = \{F \to NF \mid DF, D \to CF\}$, empezamos con el estado inicial:



Ahora aplicamos el punto 2 al estado q_0

Construcción del autómata DK

3. Consideramos el estado q_i. Para cada regla punteada del tipo $B \to U.CA$ en q_i, añadimos una transición con etiqueta C al estado que contiene todas las reglas $B \to UC.A$ de q_i.

Construcción del autómata DK

3. Consideramos el estado q_i. Para cada regla punteada del tipo $B \to U.CA$ en q_i, añadimos una transición con etiqueta C al estado que contiene todas las reglas $B \to UC.A$ de q_i. Si este estado no existe lo creamos.

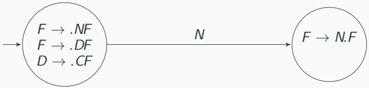
Construcción del autómata DK

3. Consideramos el estado q_i. Para cada regla punteada del tipo $B \to U.CA$ en q_i, añadimos una transición con etiqueta C al estado que contiene todas las reglas $B \to UC.A$ de q_i. Si este estado no existe lo creamos.
4. Los estados finales son los que contienen por lo menos una regla punteada completa del tipo $B \to X.$.

Construcción del autómata DK

3. Consideramos el estado q_i. Para cada regla punteada del tipo $B \to U.CA$ en q_i, añadimos una transición con etiqueta C al estado que contiene todas las reglas $B \to UC.A$ de q_i. Si este estado no existe lo creamos.
4. Los estados finales son los que contienen por lo menos una regla punteada completa del tipo $B \to X.$.

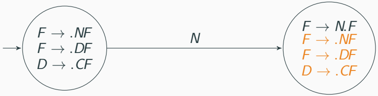
Seguimos el ejemplo anterior de la gramática $G = \{F \to NF \mid DF, D \to CF\}$. Consideramos la regla punteada $F \to .NF$ y obtenemos:



Construcción del autómata DK

3. Consideramos el estado q_i. Para cada regla punteada del tipo $B \to U.CA$ en q_i, añadimos una transición con etiqueta C al estado que contiene todas las reglas $B \to UC.A$ de q_i. Si este estado no existe lo creamos.
4. Los estados finales son los que contienen por lo menos una regla punteada completa del tipo $B \to X.$.

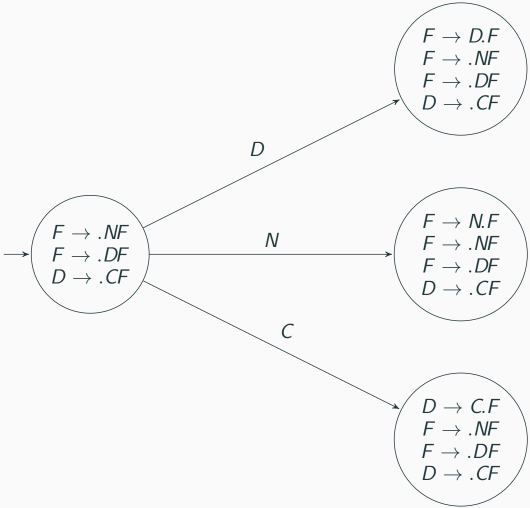
Seguimos el ejemplo anterior de la gramática $G = \{F \to NF \mid DF, D \to CF\}$. Consideramos la regla punteada $F \to .NF$ y obtenemos:



Ahora aplicamos el punto 2 de nuestro algoritmo al estado que acabamos de crear.

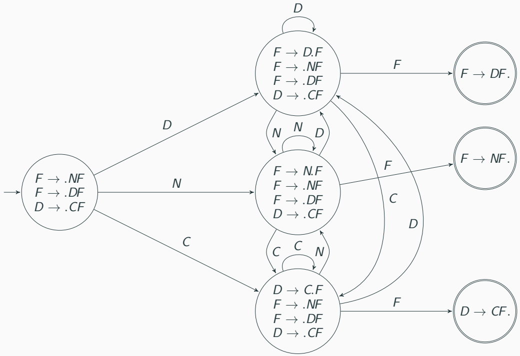
Construcción del autómata *DK*

Iterando los pasos 3 y 2 a la gramática $G = \{F \to NF \mid DF, D \to CF\}$ obtenemos:



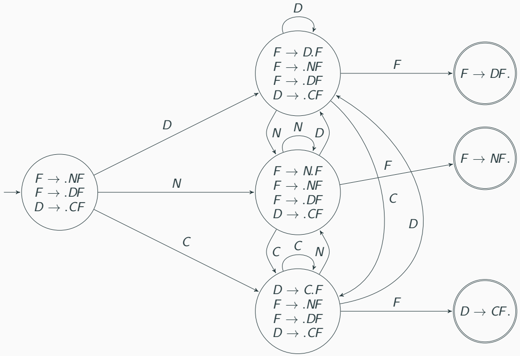
Construcción del autómata *DK*

Al final obtenemos:



Construcción del autómata DK

Al final obtenemos:



Es un buen ejercicio realizar el parsing de $CFNF$ encontrando las manijas mediante este autómata.

Resumen

Resumen

Hoy aprendimos:

- Qué son un scanner y un parser;
- Qué es el parsing *top-down* y el parsing *bottom-up*;
- Cómo calcular la complejidad del método *top-down* para el caso de las gramáticas en forma normal de Chomsky;
- Cómo construir un autómata de Knuth de una gramática determinista;
- Cómo utilizar un autómata de Knuth de una gramática determinista para buscar las manijas en el método *bottom-up*.