Formulas 2

Wednesday, September 22, 2021 3:28 PM

CFL (Context free Languaje):

- permiten generar frases de una forma recursiva y se componen de reglas a las que se les llama CFG (context free grammar)

Colección de reglas de substitución o producciones de la forma:

Variable → Cadena de vars. y terminales

ej: $G_1 = \{ A \to 0A1,\ A \to B,\ B \to \# \}$

Una derivación de $000\#111 \in G_1$ es:

$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

→ todas las frases que se pueden generar a partir de una CFG forman el lenguaje de dicha gramatica.

- todos los lenguajes descritos por una GFL se llaman CFG.

Def: (formal) Una gramática indep. del contexto es una 4tupla:

$$(V, \Sigma, R, S)$$

- V → Conjunto finito de
- Σ → Conjunto de
- R → [illegible]
- S → variable inicial

Conjunto finito de vars.

Conjunto de terminales

Conjunto finito de reglas

Def: (Derivación)

V deriva U si $U = V$ o si existe una secuencia $U_1, U_2, \dots, U_K$, $K \geq 0$ t.q. $U_1 \Rightarrow U_2 \Rightarrow \dots \Rightarrow U_K \Rightarrow V$

que se escribe $U \overset{*}{\Rightarrow} V$.

Def: (lenguaje de una gramática)

$$\{ \underbrace{w \in \Sigma^*}_{\text{palabras que pertenecen al conjunto de potencia de los terminales de la gramatica}}, \underbrace{S \overset{*}{\Rightarrow} w}_{\text{tales que se pueden derivar desde la variable inicial.}} \}$$

ej:

Sea $G = (V, \Sigma, R, \langle \texttt{expr} \rangle)$, donde $V = \{\langle \texttt{expr} \rangle, \langle \texttt{term} \rangle, \langle \texttt{factor} \rangle\}$ y $\Sigma = \{\texttt{a}, +, \times, (,)\}$. Las reglas de G son

$$\begin{aligned} \langle \texttt{expr} \rangle &\to \langle \texttt{expr} \rangle + \langle \texttt{term} \rangle | \langle \texttt{term} \rangle \\ \langle \texttt{term} \rangle &\to \langle \texttt{term} \rangle \times \langle \texttt{factor} \rangle | \langle \texttt{factor} \rangle \\ \langle \texttt{factor} \rangle &\to (\langle \texttt{expr} \rangle) | \texttt{a} \end{aligned}$$

Esta gramática describe un pedazo de la aritmética.

Def: (gramática Regular)

Una gramática es regular sii sus reglas son de la forma:

$$A \to aB$$
$$A \to a$$
$$A \to \varepsilon$$

Teorema

Las gramáticas regulares y los DFA son equivalentes

pd: note que por ello si un CFL (context free language) es regular se le puede hacer un DFA y sacar de allí una CFG

Jerarquía de Chomsky

El lingüista Noam Chomsky propuso en 1956 la siguiente jerarquía de lenguajes.

Tipo 0: Lenguajes recursivamente enumerables (o con estructura de frase):

$$x \to y, \qquad \text{con } x, y \in (V \cup \Sigma)^*.$$

Tipo 1: Lenguajes sensibles al contexto:

$$xAy \to xuy, \qquad \text{con } x, u, y \in (V \cup \Sigma)^*.$$

Tipo 2: Lenguajes independientes del contexto:

$$A \to x, \qquad \text{con } x \in (V \cup \Sigma)^*.$$

Tipo 3: Lenguajes regulares:

$$A \to \mathrm{a}B \mid \mathrm{a} \mid \varepsilon, \qquad \text{con } \mathrm{a} \in \Sigma,\ A, B \in V.$$

Def: (ambigüedad)

- Una palabra es derivada de forma ambigua si hay más de una derivación que la produce.
- Si una gramática genera una palabra de forma ambigua diremos que la gramática misma es ambigua.

Def: (forma Normal de Chomsky)

- Si todas las reglas son de la forma:

$A \to BC$] Variable en par de variables

$A \to a$] Variable en terminal

Se permite: $S \to \varepsilon$

No se permite $A \to S \mid SC \mid SB$] Variable en variable inicial

Teorema

Cualquier CFL se puede generar a través de una CFG en forma normal de Chomsky.

Construcciones

1. Si mi CFL también es un lenguaje regular se puede construir un DFA que reconoce el lenguaje y traducirlo en una CFG así:

y traducirlo en

→ Cada estado del autómata corresponde a una variable.
q_i (estado) es R_i (variable).

→ Si hay transición $\delta(q_i, a) = q_j$, se crea la regla $R_i \to aR_j$

→ Si q_i es un estado final se añade la regla $R_i \to \varepsilon$.

→ la variable inicial R_0 es q_0.

2 Como pasar una CFG a una CFG en forma normal de Chomsky:

1. Se añade una variable inicial S_0 con la regla $S_0 \to S$.

2. Se remueven todas las reglas $A \to \varepsilon$ y añadimos una regla para cada regla que se removió. (modificamos todas las reglas afectadas por este cambio y las ponemos de una forma que sea equiv. a la anterior. para no afectar la gramática). ej: $\cancel{A \to \varepsilon} \mid CD$
$B \to uAx \rightsquigarrow B \to ux \mid uCDx$
Repetir hasta completar ∀ las reglas excepto $A = S_0$

3. eliminamos todas las reglas unitarias, de la forma $A \to B$, para no cambiar el lenguaje cada vez que había un $B \to u$, donde u es una cadena de variables y terminales, se

es una cadena de variables y [illegible]
agrega la regla $A \to U$.

4. Por último, convertimos las reglas que quedan en forma estándar. Remplazamos cada regla de la forma $A \to u_1 u_2 \cdots u_k$, donde cada u_i es una variable o un terminal, con las reglas:

$$A \to u_1 A_1, \quad A_1 \to u_2 A_2, \ldots, A_{k-2} \to u_{k-1} u_k.$$

Los A_i son variables nuevas. También remplazamos todos los terminales u_i con nuevas variables U_i y añadimos las reglas $U_i \to u_i$. □

$A \to U_i A_1$

Def: (parsing) el problema de parsing para una palabra w es encontrar un árbol de parsing de w en la gramática G.

Método de parsing top-down

- Se puede hacer a fuerza bruta
- empieza con una variable inicial $S \in G$ y busca una derivación $S \overset{*}{\Rightarrow} w$.

⚠ m^{2n-1} ⇒ combinaciones.
m: # reglas
$2n-1$: # reglas a aplicar

Método de parsing bottom-up

- empezar con la palabra w y escalar hacia la variable inicial S.

Def: (CFG determinista) Una CFG es determinista si cada cadena en el procesamiento bottom-up de la palabra w, tiene solo una manija

Una manija para $w = xhy$ es una regla de la forma $A \to h$. que reduce w a xAy.

Determinista:

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

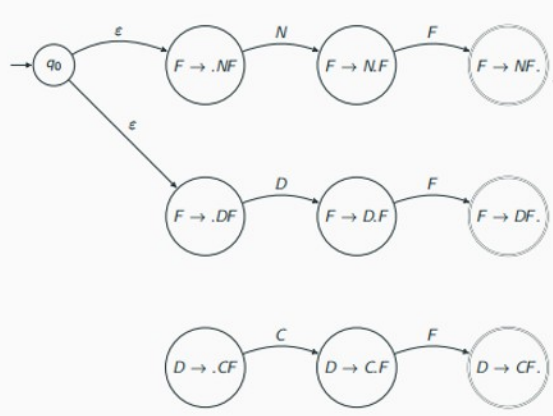
No determinista:

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

Para poder ubicarnos mejor empezamos a usar las reglas punteadas ej:

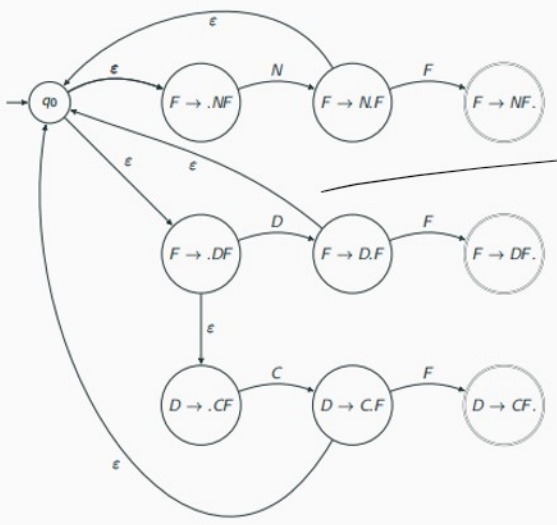
$F \to .NF$ (leo desde el '.') , $F \to N.F$ (leo desde el '.') $F \to NF.$ (terminé de leer.)

Autómata K (para encontrar manijas)

① 

→ estados de aceptación

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

② 

puede entenderse como:
"si me equivoco y no puedo ir al siguiente me regreso a ver cuál de antes me sirve".

o como:
"abro un proceso para ver que pasa si me voy por este camino".

Definimos *ε-moves* desde $.C$ hasta una regla que comienza con C: $\boxed{B \to U.CV} \xrightarrow{\varepsilon} \boxed{C \to .R}$

⇒ Revisar ejemplo de ejecución de la clase 13

Autómata DK → versión determinista de un autómata K

(también para

DK
(también para encontrar manijas)

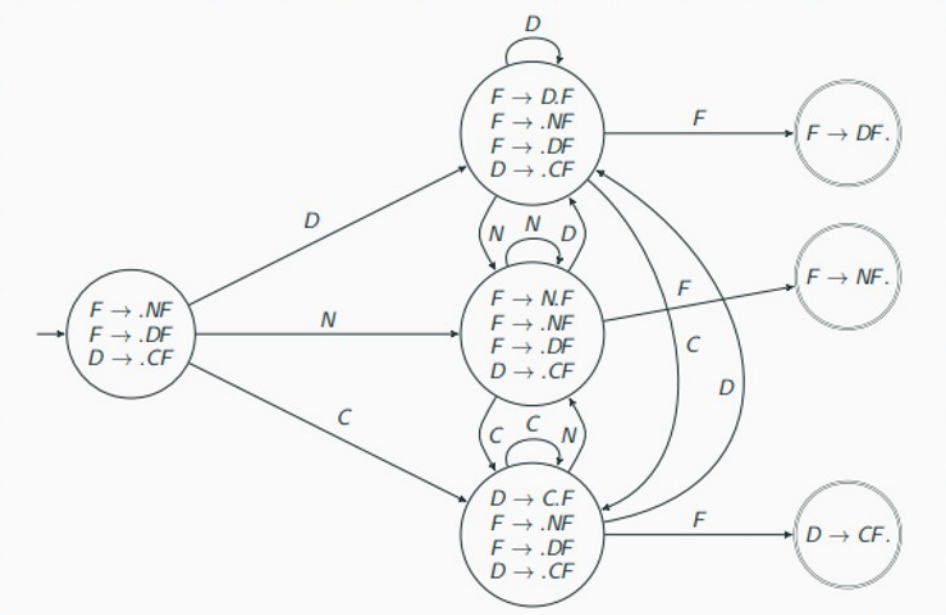
Pasos para hacer un autómata DK:

0 simplificar la gramática quitando los terminales y la repetición entre reglas es:

$$G = \begin{cases} \cancel{S \to NF \mid DF} \\ F \to NF \mid DF \\ D \to CF \end{cases} \equiv G_2 = \begin{cases} F \to NF \mid DF \\ DF \to CF \end{cases}$$

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.
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 $B \to X.$.

ej:



Def: (Autómata de pila)

Definición

Un autómata de pila (no determinista) es una 6-tupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$, donde:

- Q es el conjunto finito de los estados;
- Σ es el alfabeto de input (finito);
- Γ es el alfabeto de la pila (finito);
- $\delta\colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ es la función de transición;
- $q_0 \in Q$ es el estado inicial;

de pila

Definición

Un autómata de pila (no determinista) es una 6-tupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$, donde:

- Q es el conjunto finito de los estados;
- Σ es el alfabeto de input (finito);
- Γ es el alfabeto de la pila (finito);
- $\delta\colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ es la función de transición;
- $q_0 \in Q$ es el estado inicial;
- $F \subseteq Q$ es el conjunto de los estados de aceptación.

Describimos ahora en manera formal el PDA que reconoce el lenguaje $\{0^n 1^n, n \geq 0\}$. Sean $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, \$\}$, $F = \{q_1, q_4\}$ y δ es como sigue (los símbolos que faltan son $\emptyset$):

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \$)\}$					
q_3				$\{(q_3, \$)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

El símbolo \$ es un símbolo especial que utilizamos señalar el comienzo de la pila: cuando lo leemos sabemos que la pila está vacía.

Teorema

Un lenguaje es independiente del contexto si y solo si es reconocido por un autómata de pila.

Para la demostración:

Lema 1

Sea P un PDA. Luego existe un PDA P' *equivalente* a P tales que

1. P' solo tiene un estado de aceptación q_{accept};
2. P' vacía la pila *antes* de aceptar;
3. Cada transición añada un símbolo en la pila (paso de empuje) o borra un símbolo de la pila (paso de extracción), pero no hace los dos al mismo tiempo.

Lema 2

Dado un PDA que satisfaga el Lema 1, se puede construir una CFG G que satisfaga lo siguiente. Si p y q son dos estados de P, G contiene una variable A_{pq} y

$$(p, w, \varepsilon) \overset{*}{\vdash} (q, \varepsilon, \varepsilon) \quad \text{sii} \quad A_{pq} \overset{*}{\Rightarrow} w$$