

Retomando la clase anterior ...

Ejemplo:

$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow (S) \\ S &\rightarrow \epsilon \end{aligned}$$

es un CFG que genera expresiones balanceadas,

$\Sigma \cup \{S\}$

reglas

$$R \subseteq V \setminus \Sigma \times V^*$$

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

símbolo inicial

NO TERMINALES

$\{ (, ) \}$

$$L(G) = \{ w \in \Sigma^* : S \Rightarrow^* w \}$$

símbolos terminales.

es posible

llegar mediante finitas aplicaciones de las reglas

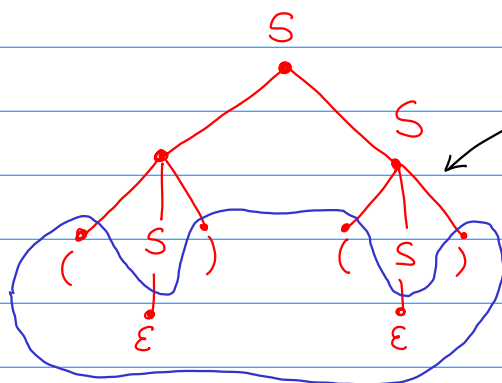
Ejemplo: Hay muchas derivaciones distintas del mismo  $w$ ,

$$D_1 : S \xRightarrow{L} SS \xRightarrow{L} (S)S \xRightarrow{L} (())S \xRightarrow{L} (())(S) \xRightarrow{L} (())()$$

$$D_2 : S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow (S)() \Rightarrow (())()$$

$$D_3 : S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$$

\* no van a ser modificados a futuro por reglas



Árbol de análisis  
"PARSE TREE"

Obs: Entre las derivaciones de una fórmula con un mismo parse tree hay una "Left-most", es decir una en la que toda transición satisface

$$x \xRightarrow{L} y \Leftrightarrow \begin{aligned} x &= \alpha A \beta, \text{ con } \alpha \in \Sigma^* \leftarrow \text{TERMINALES} \\ y &= \alpha \gamma \beta \quad A \rightarrow \gamma \in R \\ &\quad \beta \in V^* \end{aligned}$$

consecuencia \* arriba

Queremos definir un nuevo tipo de máquina capaz de saber si un string  $w$  viene o no de aplicar las reglas de un CFG.

Como vimos es fácil crear  $\{0^n 1^n : n \in \mathbb{N}\} = L$  mediante CFG  $(S \rightarrow 0S1, S \rightarrow \epsilon)$

lo que sugiere que necesitamos "memoria" para poder recordar el número de ceros y poder compararlo con el de 1's para poder reconocer  $L$ .

Un PUSHDOWN AUTOMATA es una máquina no determinista con finitos estados y un

Stack de memoria

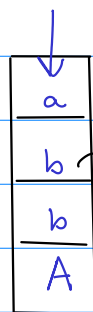
con operaciones

push y pop  
en la parte de arriba

escribir un string de un conjunto fijo

sacar un string de un conjunto fijo.

Más formalmente tenemos



$a b b A$

modificaciones de lectura y escritura sólo por la izquierda.

Def: Un PDA es una 6-tupla  $(K, \Sigma, \Gamma, \Delta, s, F)$

nuevo!

$K$  — Conjunto finito de estados

$\Sigma$  — Conjunto finito de símbolos

$\Gamma$  — Conjunto finito de símbolos del Stack

$\Delta$  — relación de transición; subconjunto finito de

$\Delta \subseteq \underbrace{K \times (\Sigma \cup \{\epsilon\})}_{\text{input}} \times \underbrace{\Gamma^* \times K \times \Gamma^*}_{\text{nueva situación}}$

$s$  — Estado inicial  $s \in K$

$F$  — Conjunto de estados de aceptación  $F \subseteq K$

La interpretación de elementos en  $\Delta$  es así:

$$((q, a, \gamma), (r, \beta)) \in \Delta$$

Estando en estado  $q$ , lee símbolo  $a$  y los primeros  $|\gamma|$  símb.  
del stack son  $\gamma$  entonces paso al estado  $r$  y reemplazo  
 $\gamma$  por  $\beta$  en el stack,

$$\text{stack-antes} = \gamma\delta \rightsquigarrow \text{stack-después} = \beta\delta$$

Caso especial  $\gamma = \epsilon$  "push  $\beta$ "

Caso especial  $\beta = \epsilon$  "pop  $\gamma$ "

Para representar el estado actual de un cálculo necesitamos

$$(p, x, \alpha)$$

estado de  $M$  (pointing to  $p$ )  
símbolos que quedan en el input  $\in \Sigma^*$  (pointing to  $x$ )  
Info guardada en el stack  $\in \Gamma^*$  (pointing to  $\alpha$ )

Def:

$$(p, x, \alpha) \xrightarrow{M} (q, y, \gamma) \quad (\Leftrightarrow \text{"da en un paso"})$$

$$\exists ((p, a, \gamma), (q, \beta)) \in \Delta \text{ tal que}$$

$$x = ay, \quad \alpha = \gamma\delta, \quad \gamma = \beta\delta$$

Nuestros automatas son NO DETERMINISTAS  
y podría haber muchas reglas  $\in \Delta$   
que apliquen en cada inst. de  $x$ .

$$\vdash_M^* := \text{Reflexive, transitive closure de } \vdash$$

Def:  $M$  acepta  $w \in \Sigma^*$  si

$$\exists r \in F \text{ tal que}$$

$$(s, w, \epsilon) \xrightarrow{*} (r, \epsilon, \epsilon)$$

(de aceptación) (pointing to  $r$ )  
stack (pointing to  $\epsilon$  in the triple)

Ejercicio: Construya un PDA que capture expresiones balanceadas en paréntesis

Ejercicio:  $L = \{wcw^R : w \in \{a,b\}^*\} \subseteq \{a,b,c\}^*$

"Acumulo  $w$  en el stack y luego la leo al revés"

$\underline{a} \ \underline{b} \ \underline{b} \ \underline{a} \ b \xrightarrow{\text{en stack}} \underline{b \ a \ b \ b \ b \ a}$   
↑  
quedo al revés aut.

$K = \{s, f\}$

$[(s, a, \epsilon), (s, a)]$

$[(s, b, \epsilon), (s, b)]$

$[(s, c, \epsilon), (f, \epsilon)]$

$[(f, a, a), (f, \epsilon)]$

$[(f, b, b), (f, \epsilon)]$

escribe la palabra antes de ver  $c$

cambio al estado "compro"

compro letra por letra y va

balanceado

Ejercicio: Construya un automata que capture el lenguaje  $L$  de palabras con la misma cantidad de  $a$ 's y  $b$ 's.  
(El stack debe recordar de cual hay más)

Teorema: Para todo CFG  $G$  existe un PDA  $M$  con  $L(G) = L(M)$

Dem: El automata intenta verificar que la palabra dada  $w$  pueda obtenerse mediante un "left-most derivation" del símbolo inicial usando el NO-determinismo para adivinar cuál regla de sustitución utilizar.

El uso de leftmost derivations es para hacerlas compatibles con la estructura de stack pues la máquina debe ir haciendo sustituciones y comparando.

$M = \{ \{p, q\}, \Sigma, V, \Delta, p, \{q\} \}$

$\Delta$ :

$[(p, \epsilon, \epsilon), (q, S)]$  ← inicio deducción poniendo  $S$  en el stack

$[(q, a, a), (q, \varepsilon)] \leftarrow$  Si  $a$  es terminal  
la borro.  
 $\forall a \in \Sigma$

$[(q, \varepsilon, A), (q, w)] \leftarrow$  Si  $A \rightarrow w \in R$

Ejemplo: Consider  $G =$

$R = \{ S \rightarrow aSa, S \rightarrow bSb, S \rightarrow c \}$

con  $V = \{S, a, b, c\}$  en  $L(G) = \{w c w^R : w \in \{a, b\}^*\}$

¿Qué hace en  $abbc bba$ ?

si  $\alpha \in (V - \Sigma)^* \cup \{\varepsilon\}$

Dem: Mostremos que  $S \xRightarrow{*} wd \iff (q, w, S) \vdash_M^* (q, \varepsilon, \alpha)$

" $\Rightarrow$ " Inducción en la longitud de una "LEFTMOST DERIVATION"  
para  $wd$

" $\Leftarrow$ " number of uses of transitions coming from the rules  $R$