Retomando la clase anterior ...

Ejemplo:

$S \to SS$
$S \to (S)$
$S \to \varepsilon$

es un CFG que genera expresiones balanceadas

reglas

$R \subseteq V \setminus \Sigma \times V^*$ (NO TERMINALES)

$G = (V, \Sigma, R, S)$ (V: $\Sigma \cup \{S\}$; Σ: $\{(,)\}$; S: símbolo inicial)

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

(Σ: símbolos terminales.)

$\overset{*}{\Rightarrow}$: es posible llegar mediante finitas aplicaciones de las reglas

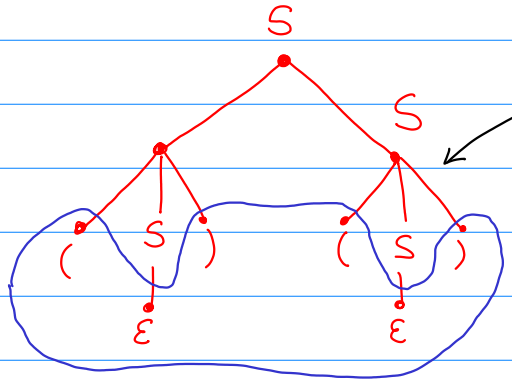
Ejemplo: Hay muchas derivaciones distintas del mismo w,

D_1: $S \overset{L}{\Rightarrow} SS \overset{L}{\Rightarrow} (S)S \overset{L}{\Rightarrow} ()S \overset{L}{\Rightarrow} ()(S) \overset{L}{\Rightarrow} ()()$

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 serán 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

$x \overset{L}{\Rightarrow} y \iff x = \alpha A \beta$, con $\alpha \in \Sigma^*$ ← TERMINALES
$y = \alpha \gamma \beta$
$A \to \gamma \in R$
$\beta \in V^*$

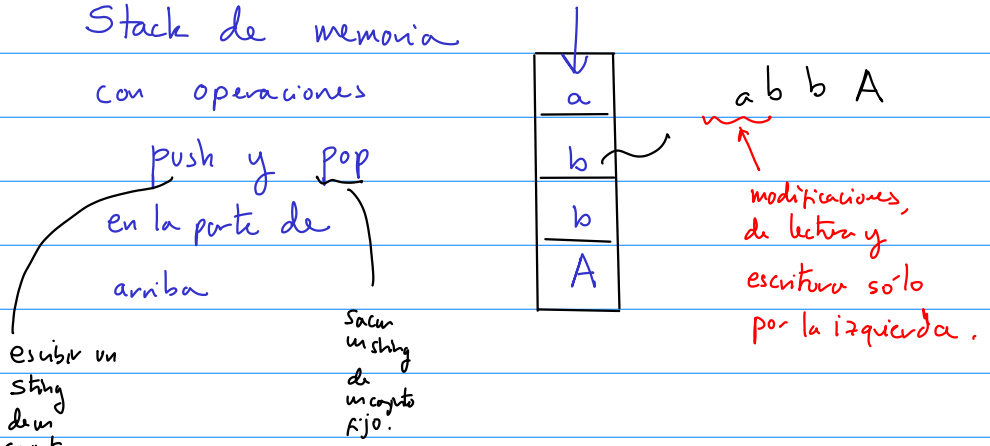
Consecuencia $*$ arriba

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 $(S \to 0S1 \;,\; S \to \varepsilon)$

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



Más formalmente tenemos

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

(Γ: nuevo!)

K — Conjunto finito de estados

Σ — Conjunto finito de símbolos

Γ — Conjunto finito de símbolos del Stack

Δ — relación de transición; subconjunto finito de

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

s — Estado inicial $s \in K$

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

La interpretación de elementos en Δ es así:

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

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

$$\text{stack\_antes} = \gamma\delta \rightsquigarrow \text{stack\_despues} = \beta\delta$$

Caso especial $\gamma = \varepsilon$ "push β"

Caso especial $\beta = \varepsilon$ "pop γ"

Para representar el estado actual de un cálculo necesitamos,

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

- p: estado de M
- x: símbolos que quedan en el input $\in \Sigma^*$
- α: Info guardada en el stack $\in \Gamma^*$

Def:

$$(p, x, \alpha) \vdash_M (q, y, \zeta) \iff$$

("da en un paso")

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

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

Nuestros autómatas son NO DETERMINISTAS y podría haber muchas reglas $\in \Delta$ que aplican en cada instante.

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

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

$\exists\, r \in F$ (de aceptación) tal que

$$(s, w, \varepsilon) \vdash_M^* (r, \varepsilon, \varepsilon)$$

(ε: stack)

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

a b b b a b → en stack → b a b b b a

↑ queda al revés aut.

$K = \{s, f\}$

$[(s, a, \varepsilon), (s, a)]$
$[(s, b, \varepsilon), (s, b)]$
} escribe la palabra antes de ver c

$[(s, c, \varepsilon), (f, \varepsilon)]$ ← cambia al estado "compara"

$[(f, a, a), (f, \varepsilon)]$
$[(f, b, b), (f, \varepsilon)]$
} compara letra por letra y va borrando

Ejercicio: Construya un autómata 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 autómata intentará 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.

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

Δ:

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

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

$[(q, \varepsilon, A), (q, w)]$ ← Si $A \to w \in R$

Ejemplo: Considere $G =$

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

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

Qué hace en $abbcbba$?

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

Dem: Mostremos que $S \overset{L}{\Rightarrow}{}^* w\alpha$ $\left(\iff (q, w, S) \vdash_M^* (q, \varepsilon, \alpha)\right.$

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

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