</body>

23/10/19

COMPILER DESIGN (JC Martin Books)

* Set of all strings $\Sigma = \{a, b\}$ with length 2.
  $L = (a+b).(a+b)$ [language RE]

* Set of all even length strings:
  $L = \{\varepsilon, aa, ab, ba, bb, aaaa, abab, \ldots \infty\}$
  $RE = ((a+b).(a+b))^*$

* Set of all strings with exactly two a's
  $L = \{aa, baa, aabb, aaba, \ldots\}$
  $RE = b^* a b^* a b^*$

* Set of all strings with even nos of a's
  $L = \{b, bb, bbb, aab\}$
  $RE = (b^* a b^* a b^*)^* + b^*$

* Set of all string that begins and ends with 2 diff symbols
  $L = \{ab, abb, ba, baaba \ldots\}$
  ~~RE = a*b* + b*a*~~ $RE = a(a+b)^* b + b(a+b)^* a$

* [* + → OR
  . → AND]

⊛ Grammers : Set of rules.

$< V, T, P, S >$

V → Set of non terminals { A, B } (variables)

T → Set of terminals { a, b, } = Σ

P → Set of Production Rules { Head → Body (combination of symbols (V, T))

S → Starting Symbol.

eg: $V = \{ S \}$

$T = \{ +, -, a \}$

$P = \{ S \to SS+ \mid SS- \mid a \}$

$S = S$.

$S \Rightarrow \underline{S}S-$

$\Rightarrow \underline{S}S+S-$

$\Rightarrow a\underline{S}+S-$

$\Rightarrow aa+\underline{S}-$

$\Rightarrow aa+a-$

When there is no V then string is generated

⊛ Grammer & Language : If we can generate all strings with a particular grammer, then that grammer is said to generate all strings of that particular language.

⊛ $L = (a+b).(a+b)$

Grammer : $V = \{ S, A \}$. $T = \{ a, b \}$

$P = \{ S \to AA, \quad A \to a \mid b \}$

$S = S$.

Unacademy
Date
Page

① $L = (a+b)^*$

grammer: $P = \{ S \to aS \mid bS \mid \epsilon \}$

② $L = a(a+b)^* b$

$P = \{ S \to aAb \mid \epsilon,$
$A \to a,$
$B \to b \}$

$P = \{ S \to aAb,\ A \to aA \mid bA \mid \epsilon \}$

③ $L = a^n b^n \mid n > 1$

$P = \{ S \to AB \mid$
$A \to a \mid$
$B \to b \mid \}$

$P = \{ S \to aSb \mid ab \}$

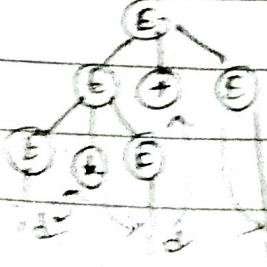
④ Set of all Palindromes over $\Sigma = \{a, b\}$.

$P = \{ S \to aSa \mid bSb \mid a \mid b \mid \epsilon \}$.

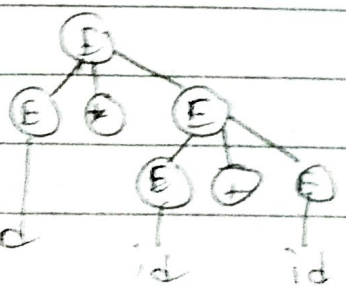
⑤ $E \to E+E \mid E*E \mid id$. → grammer

id * id + id (String)

$E \Rightarrow E*E$
$\Rightarrow E*E+E$
$\Rightarrow id*id+id$



$E \Rightarrow E*E$
$\Rightarrow E*E+E$
$\Rightarrow id*id+id$



* Ambigious grammer → for same string if we can come up with different tree it is known as ambigious grammer.

30/10/19

COMPILER DESIGN

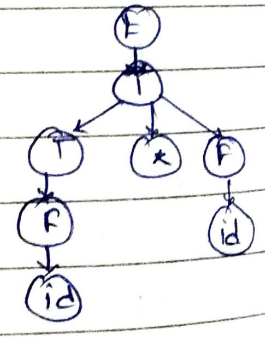
* Top Down Parsing - We try generating some string from the grammer.
* Bottom up parsing - We move from the string to starting symbol.

* Top down parsing - eg:

eg: Gr = $P = \{ E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id \}$

String = id * id

$E \rightarrow \underline{T}$
$\rightarrow \underline{T} * F$
$\rightarrow \underline{F} * F$
$\rightarrow id * \underline{F}$
$\rightarrow id * id$.



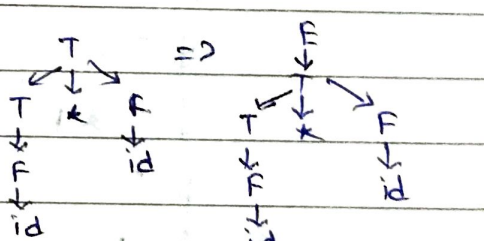
It is called top down parsing as it starts from the very first symbol in grammer and reaches the target string as we move downwards towards the leaf nodes.

Handelpruning?
If not possible to reduce, the substring is discarded, this is called handelpruning.
2 operators
- Shift (going to other symbol)
- Reduce (replace some part of the string using production rule).

① Bottom up parsing - (Reducing an input string to its starting symbol)

id * id	F → id	(LL() parser)
F * id	F → id	(Left most derivation)
F * F	T → F	(LR parser)
T * F	T → T * F |	↳ Right most derivation in Reverse.)
T	E → T	
E		

id * id ⇒ F * id ⇒ F * F ⇒ T * F ⇒ ⇒



Parsing process - reads the string from left to right.

→ part of the string that the parser is focusing on for reduction.

	Handle	
id, * id2	id1	F → id
F * id2	F * id2	T → F
T * id2	id2	F → id.
T * F	T * F	T → T * F
T	T	E → T
E		

LL ↓ LR ↑

id1 + id2	id1	F → id
F + id2	F	T → F
T + id2	T	E → T
E + id2	id2	F → id2
E + F	F	T → F
E + T	E + T	E → E + T

⊛ Stack implementation : String : id * id

marks the end of the stack → $ | marks the end of the string ←$

STACK	INPUT	ACTION
$	id * id $	Shift
$id	* id $	Reduce, F→id
$F	* id $	Reduce, T→F
$T	* id $	Shift
$T*	id $	Shift
$T*id	$	Reduce, F→id
$T*F	$	Reduce, T→T*F
$T	$	Reduce, E→T
$E	$	Stop / Accept

handle part of the string from top of the stack.

challenge to identify the handle

challenge to make the decision to shift when it can't be reduced

14/10/19

IMAGE PROCESSING

⊛ Entropy calculation of an Image:

Formula: $H = -\sum_{k=0}^{L-1} P(r_k) \log_2 P(r_k)$ bits / pixel.

where: $P(r_k)$ - probality of any perticular intensity value.

eg:

$$\begin{bmatrix} 21 & 21 & 21 & 143 \\ 197 & 143 & 67 & 21 \\ 143 & 21 & 67 & 94 \\ 194 & 67 & 94 & 197 \end{bmatrix}_{4\times4}$$

Intensity Values	Count	P(rk)
21	5	5/16 = 0.313
67	3	3/16 = 0.188
94	2	2/16 = 0.125
143	3	3/16 = 0.188
194	1	1/16 = 0.063
197	2	2/16 = 0.125
	16	

$H_1 = 0.313 \log_2 (0.313 = -0.5245$

$H_2 = -0.4533$ $\quad H_4 = -0.4533$

$H_3 = -0.375$ $\quad H_5 = -0.2513$

$H_6 = -0.375$

$H = -(-2.4324) = 2.43$ bits / pixel.

Amount of information gathered from image

$= 6 \times 8 \times 2.43 = 116.64.$

(6 ↓ no. of intensity values; 8 ↓ no. of bits; 2.43 → Entropy value.)

⑧ LZW Encoding Technique:

eg:

Binary image representation ⇒

1 0 1 0 1 1 0 1 1 0 1 0 1 0 1 0

1 0 10 11 01 101 010 1010

1, 0, 10, 11, 01, 101, 010, 1010

1 2 3 4 5 6 7 8 → 2^3

Location	Content	Code	
001	1	0001	Copy the conte
010	0	0000	last bit, as
011	10	0010	code's LSB &
100	11	0011	there is a
101	01	1011	to ire that
110	101	0111	location kee
111	010	1010	the LSB
	1010	1100	

encoded → "0001 0000 0010 0011 1011 0111 1010

image