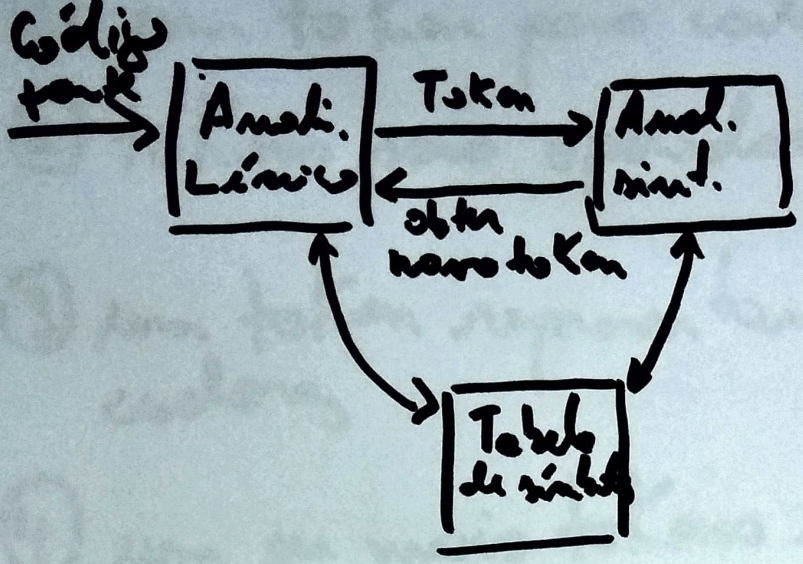
# Ferramentas p/ construção de compiladores

* **Geradores de analisadores léxicos**
  Geram automaticamente analisadores léxicos, dada a especificação dos tokens (através de exp. reg. ou aut. finito). Ex.: Lex, ANTLR, PLY
* **Geradores de analisadores sintáticos**:
  geram automaticamente analisadores sintáticos, dada uma gramática livre-de-contexto. Ex.: Yacc, ANTLR, PLY
* **Dispositivos de tradução dirigida pela sintaxe**: Percorrer uma árvore de derivação
* **Geradores automáticos de código**: traduzem (código Int. => código Alvo)

# O analisador léxico:






const pi = 3.1416;

const é lexema para token "const"
pi " " "identificador"
= " " "AT"
3.1416 " " "NU"
; " " "PONTO VÍRGULA"

Os tokens são tratados como símbolos terminais da gramática para a linguagem-fonte.

Importante: geralmente temos:

① um token para cada palavra reservada

② tokens para operadores (individual ou agrupados)

③ um token representando todos os identificadores

④ um ou mais tokens representando constantes como números e strings.

⑤ tokens para cada símbolo de pontuação

Especificação de tokens: especificamos através de expressões regulares.

**Alfabeto**: é qualquer conjunto finito de símbolos. Ex. $\{0,1\}$ alfabeto binário

$\{a, b, \ldots, z, A, b, \ldots, Z\}$

$\{0, 1, \ldots, 9\}$

**Cadeia sobre um alfabeto**: é uma sequência finita de símbolos deste alfabeto.

Ex.: 0110111 cadeia sobre $\{0,1\}$.

**Linguagem**: qualquer conjunto de cadeia sobre algum alfabeto fixo.

Ex.: $\{$Construção, de, compiladores$\}$

$\emptyset$, $\{\varepsilon\}$

↳ notação para cadeia vazia.

**Operadores sobre linguagens**:

**União**: $L \cup M$ : $L \cup M = \{s \mid s$ está em $L$ ou em $M\}$

**Concatenação** $L_1 \cdot L_2$: $L_1 L_2 = \{st \mid s \text{ está em } L_1 \text{ e } t \text{ está em } L_2\}$

**Fecho de Kleene** $L$: $L^* = \bigcup_{i=0}^{\infty} L^i$ (gera as cadeias concatenações de $L$)

$L^i = L L^{i-1}$; $L^2 = LL$; $L^0 = \{\varepsilon\}$.

Cada expressão regular $r$ define uma linguagem $L(r)$.

Regras que definem expressões regulares sobre um alfabeto $\Sigma$:

① $\varepsilon$ é uma expressão regular e $\{\varepsilon\}$ denota a linguagem $L(\varepsilon)$.

② Se $\underline{a}$ é um símbolo de $\Sigma$, então $\underline{a}$ é uma expressão regular e $L(a) = \{a\}$

③ Se $r$ e $s$ são exp. regulares cujas linguagens são $L(r)$ e $L(s)$, então

$r|s$ é uma exp. regular que denota a linguagem $L(r) \cup L(s)$;

$rs$ é " " $L(r)L(s)$;

$r^*$ " " $L(r)^*$.

Precedência dos operadores: $a|b^*c$

1º *, 2º concatenação, 3º união.

Definições regulares: é uma seq. de definições da seguinte forma:

$d_1 \to r_1$
$d_2 \to r_2$
$\vdots$
$d_n \to r_n$

onde cada $d_i$ é um nome distinto e cada $r_i$ é uma expressão regular sobre $\Sigma \cup \{d_1, \ldots, d_{n-1}\}$.

Ex1. Identificadores em Pascal

letra → A | B | ... | Z | a | b | ... | z

dígito → 0 | 1 | ... | 9

id → letra (letra | dígito)*

Ex.2: "Números" em Pascal

dígitos → dígito dígito*

fração_opcional → . dígitos | ε

expoente_opcional → (E (+ | - | ε) dígitos | ε

num → dígitos fraçãoopcional expoente opcio

CMD → if EXP then CMD |
if EXP then CMD else CMD |
ε

EXP → TERMO relop TERMO |
TERMO

TERMO → id | num

Definição regulares para os terminais

if → if

then → then

else → else

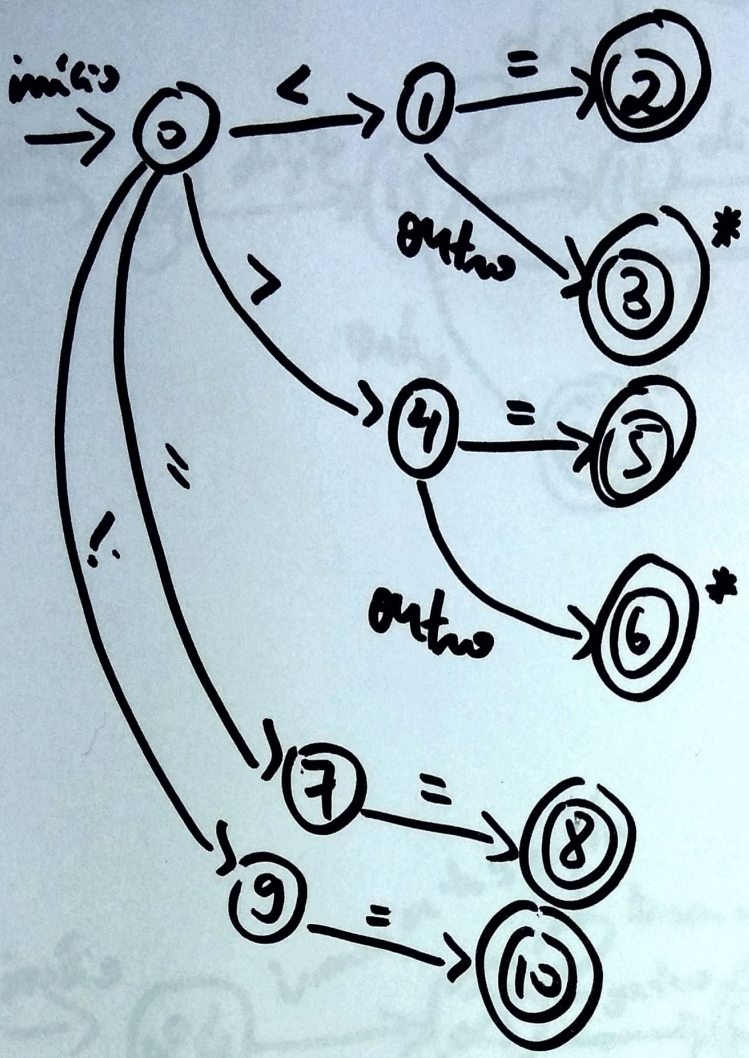
relop → < | <= | == | != | > | >=

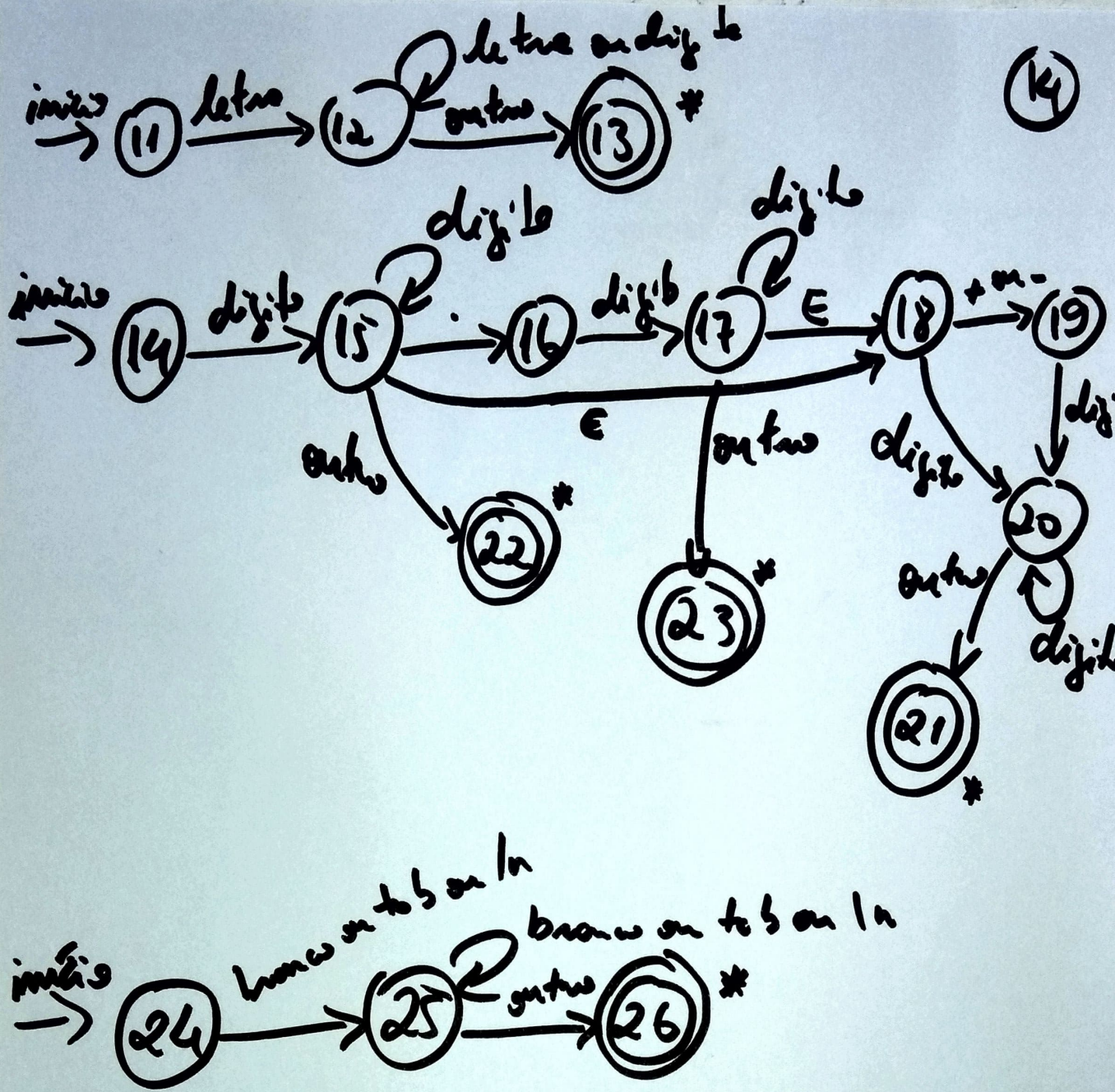
id → letra ( letra | dígito )*

num → dígito⁺ (. dígito⁺)? (E (+|-)? dígito⁺)?

NS → (branco | tabulação | novalinha)⁺

# Diagrama de transição:




14
inicio
11
letra
12
letra o dígito
otro
13
*
inicio
14
dígito
15
dígito
.
16
dígito
17
dígito
ε
18
+ o -
19
ε
otro
22
*
otro
23
*
dígito
20
otro
21
*
dígito
inicio
24
blanco o tab o ln
25
blanco o tab o ln
otro
26
*