Exp reg que defina um inteiro

$$Int = \begin{cases} 5 \\ +3 \\ -24 \end{cases}$$

$digito = \emptyset \mid 1 \mid \ldots \mid 9$

$V = \{ '+', '-', \emptyset \ldots 9 \}$

$('+' \mid '-' \mid \varepsilon) . digito^+$

Exp reg que defina um real

$$Real = \begin{cases} 5,0 \\ 133.5 \\ -43,58 E +5 \\ 5865 E 43 \end{cases}$$

$('+' \mid '-' \mid \varepsilon) . digito^+ . ('.' . digito^+ \mid \varepsilon) . (\varepsilon \mid ('E' . ('+' \mid '-') . digito^+))$

Nuno Miguel Tavares Costa e Silva - 50204

28/02/08

-> Construção de 1 expressão regular // strings de 0's e 1's q̄ ñ contém

-> Construção de 1 programa em C p/ a expressão regular gerada.
(Recursivo descendente)

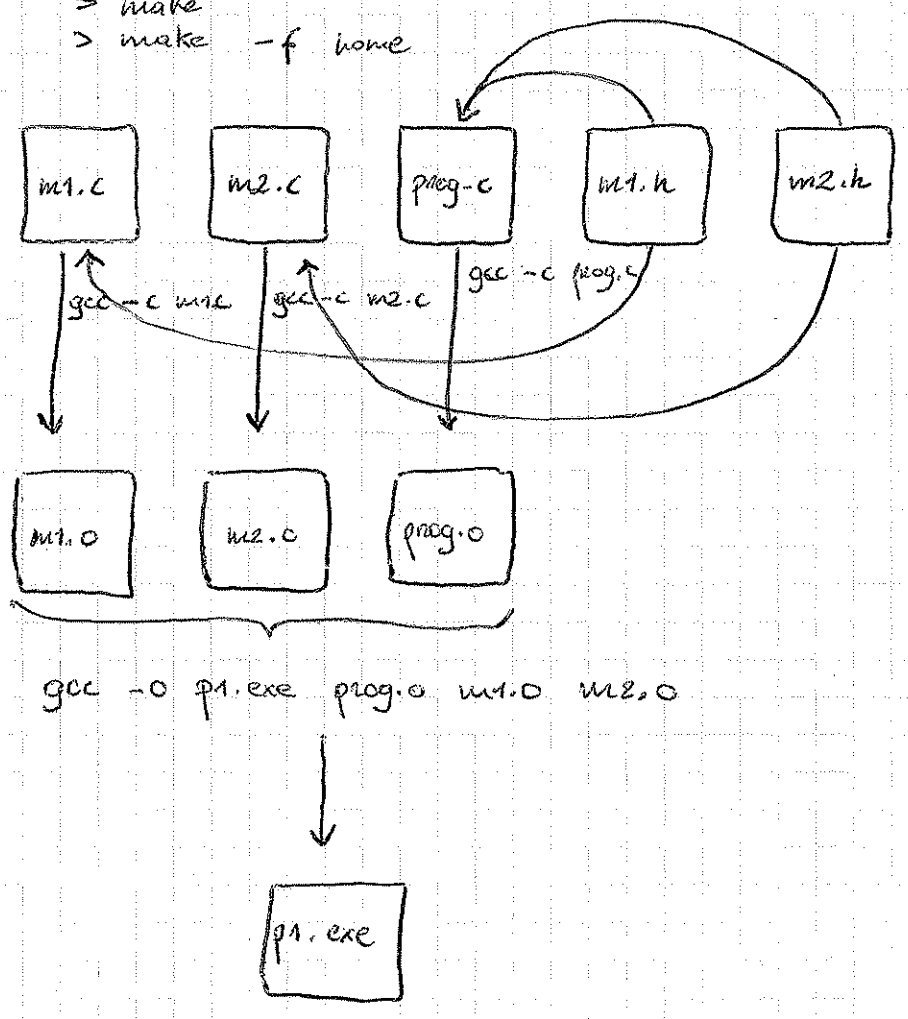
-> Construção de 1 programa em C q̄ verifique a leitura de inteiros

-> Construção de 1 autómato // a expressão regular gerada acima.

TP
04-03-2008

Makefile

- Directoria
  makefile
  Makefile
- Linha de comandos
  > make
  > make -f nome




```
m1.o : m1.c m1.h
	gcc -c m1.c
```

(TAB → it)

```
m2.o : m2.c m2.h
	gcc -c m2.c

prog.o : prog.c m1.h m2.h
	gcc -c prog.c

p1.exe : prog.o m1.o m2.o
	gcc -o p1.exe prog.o m1.o m2.o
```

→ Regra do objectivo final em 1º lugar

Ficha 2
Exercício nº1

a)
```
%%
[Ee][Uu]:    ;
[Ee][Ll][Ee]:    ;
```

b)
```
%%
[Ee][Uu]:    printf("JCR:");
[Ee][Ll][Ee]:    .*\n    ;
```

fich1.l

> flex fich1.l
↳ lex.yy.c

> gcc -o fich1 lex.yy.c -lfl

> make

makefile
```
fich1: lex.yy.c
    gcc -o fich1 lex.yy.c -lfl

lex.yy.c: fich1.l
    flex fich1.l
```

Contar tags num XML

contags.l
```
%{
int contador = 0;
%}

%%
\<    contador++;

.    ;
\n    ;
%%

int main()
{
  yylex();
}

int yywrap()
{
  printf("\n\n TAGS: %d \n\n", contador);
}
```

Escola de Engenharia
Universidade do Minho

Campus de Gualtar
4710-057 Braga

Departamento de
Informática

Gilberto Martins Felgueiras nº 50201

06.03.08

Autómato finito Não Determinista - AFND.

ER → AFND → AFD → GR.

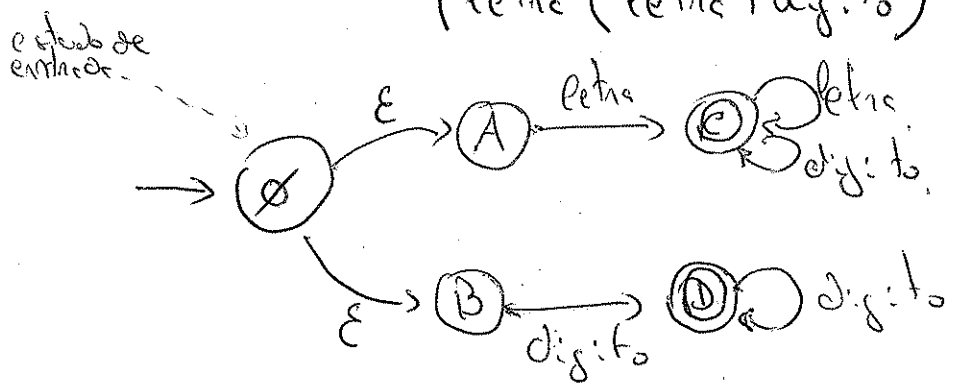
AFND $J = (Q, \Sigma, \delta(Q \times \Sigma), q_0 \in Q, F)$

- Q: Estados
- Σ: Símbolos do linguagem $\vee \in \Sigma$
- $\delta(Q \times \Sigma)$: Função transição
- $q_0 \in Q$: Estado de entrada
- F: estado Final

Exemplo:

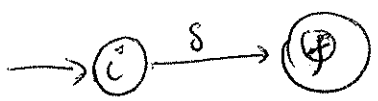
$V = \{letra, digito\}$

$er = (letra\ (letra \mid digito)^*) \mid digito^+$



ER → AFND

1. Para cada s em ER.

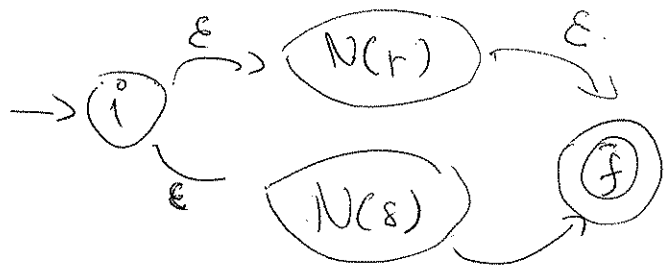


2. AFND (r) = AFND r.

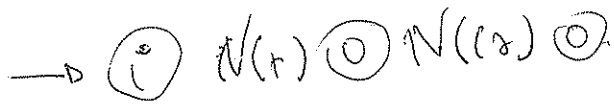
3. Dado 2 AFND para ER res: N(s) e N(s)

www.di.uminho.pt

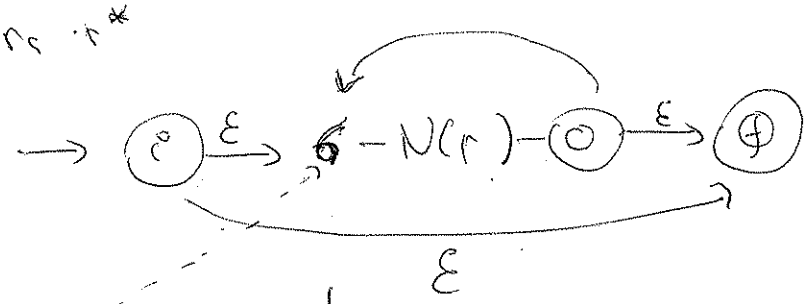
- Para r|s



- Para r.s



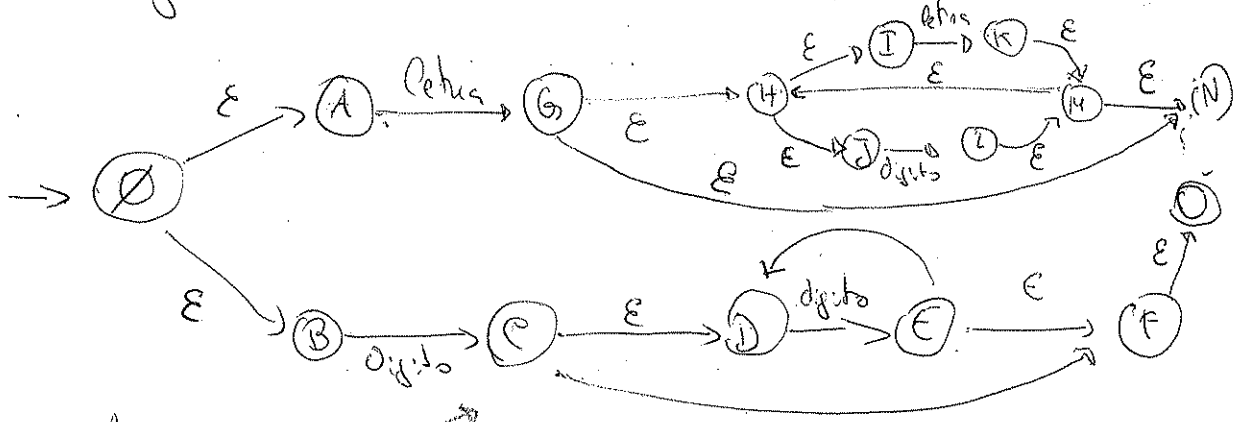
- Para r*



estado inicial de r.

aplicar regras ao automato anterior

AFND Pelas Regras.



(letra.(letra|dígito)*) | dígito+ / dígito.dígito*

Escola de Engenharia
Universidade do Minho

Campus de Gualtar
4710-057 Braga

Departamento de
Informática

		letra	dígito
> {∅, A, B}	(1)	{G, H, N, I, J, O}*	{C, D, E, O}*
{G, H, N, I, J, O}	(2)	{K, M, N, O, H, I, J}4 *	{L, M, N, O, H, I, J}5 *
{C, D, F, O}	(3)	—	{E, F, O, D}*
{K, M, N, O, H, I, J}	(4)	{K, M, N, O, H, I, J}4	{L, M, N, O, H, I, J}5
{K, M, N, O, H, I, J}	(5)	{K, M, N, O, H, I, J}4	{L, M, N, O, H, I, J}5
{E, F, O, D}	(6)	—	{E, F, O, D}6

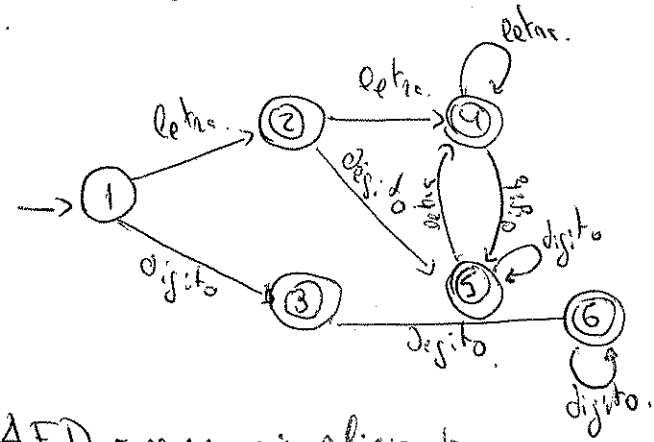
~~letra~~	letra	dígito
> 1	2	3
2	4	5
3	—	6
4	4	5
5	4	5
6	—	6

Estes três estados são finais

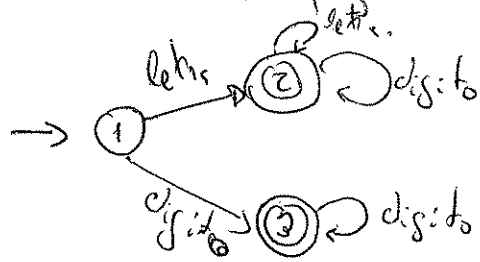
↓

Podem ser fundidos.
Porque são do mesmo classe de equivalência.

AFD



AFD - mais simplificado.



www.di.uminho.pt

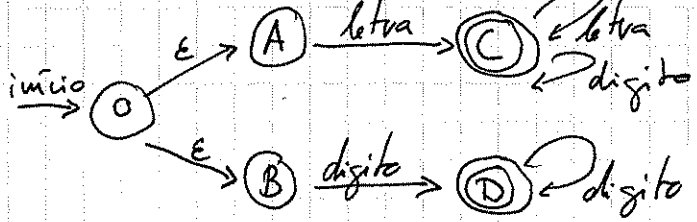
2008-03-06

Autómatos

AFND

$D = (Q, I, \delta, q_0, F)$ — Q: estados; I: símbolos c/ ϵ; $\delta \to (Q \times I) \times Q$; $q_0 \in Q$; $F \in PQ - \phi$

Exemplo: $V = \{$ letra, digito $\}$ $\quad er =$ letra (letra | digito)* | digito+



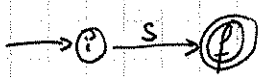
	0	A	B	C	D
ε	A,B				
letra		C		C	
digito			D	C	D

Erro de Reconhecimento:

- o autómato stá em q_i, ainda nõ processou o sufixo na entrada e nõ existe $q_j \in Q$ tal que $\{ < q_i, a_j > \mapsto q_j \} \in \delta$
- o último estado atingido nõ é membro de F.

Geraçõ de um AFND a partir de uma ER:

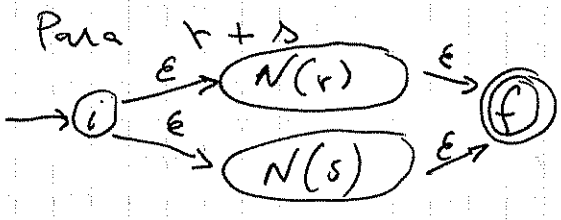
1. Para cada símbolo s



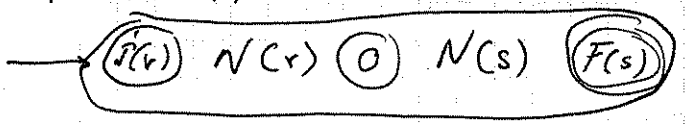
2. AFND (r) = AFND r

3. Dados 2 AFND para ER r e s: N(r) e N(s)

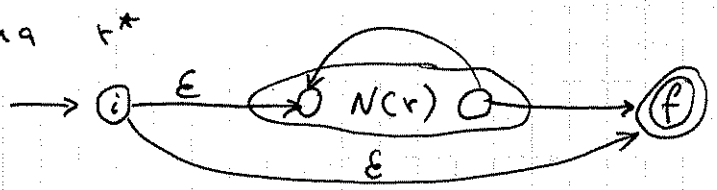
≡ Para r + s



≡ Para r, s



≡ Para r*



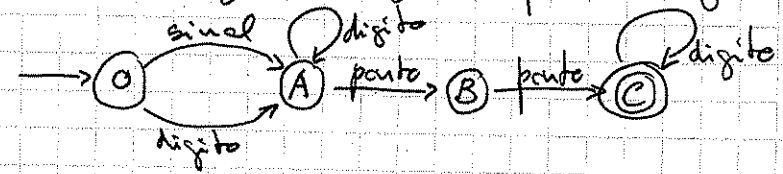
AFD

$\varepsilon \notin V$

δ é uma função

Exemplo: $V = \{$ dígito, sinal, ponto $\}$

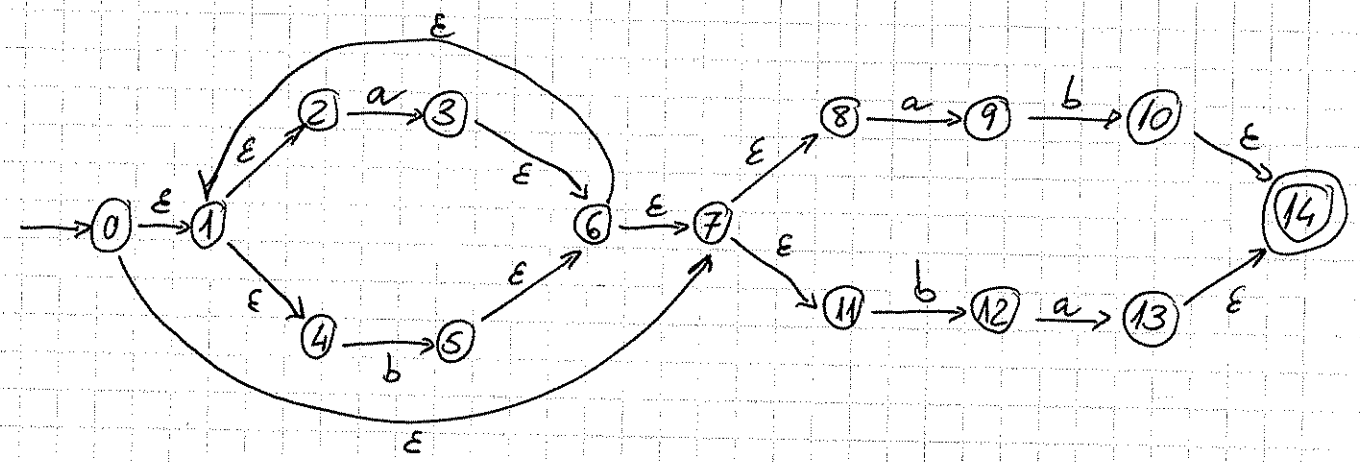
(sinal + dígito) (dígito*) ponto dígito (dígito)*



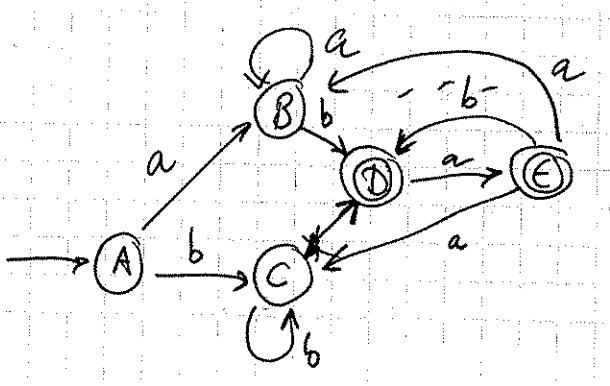
Conversão de um AFND → AFD

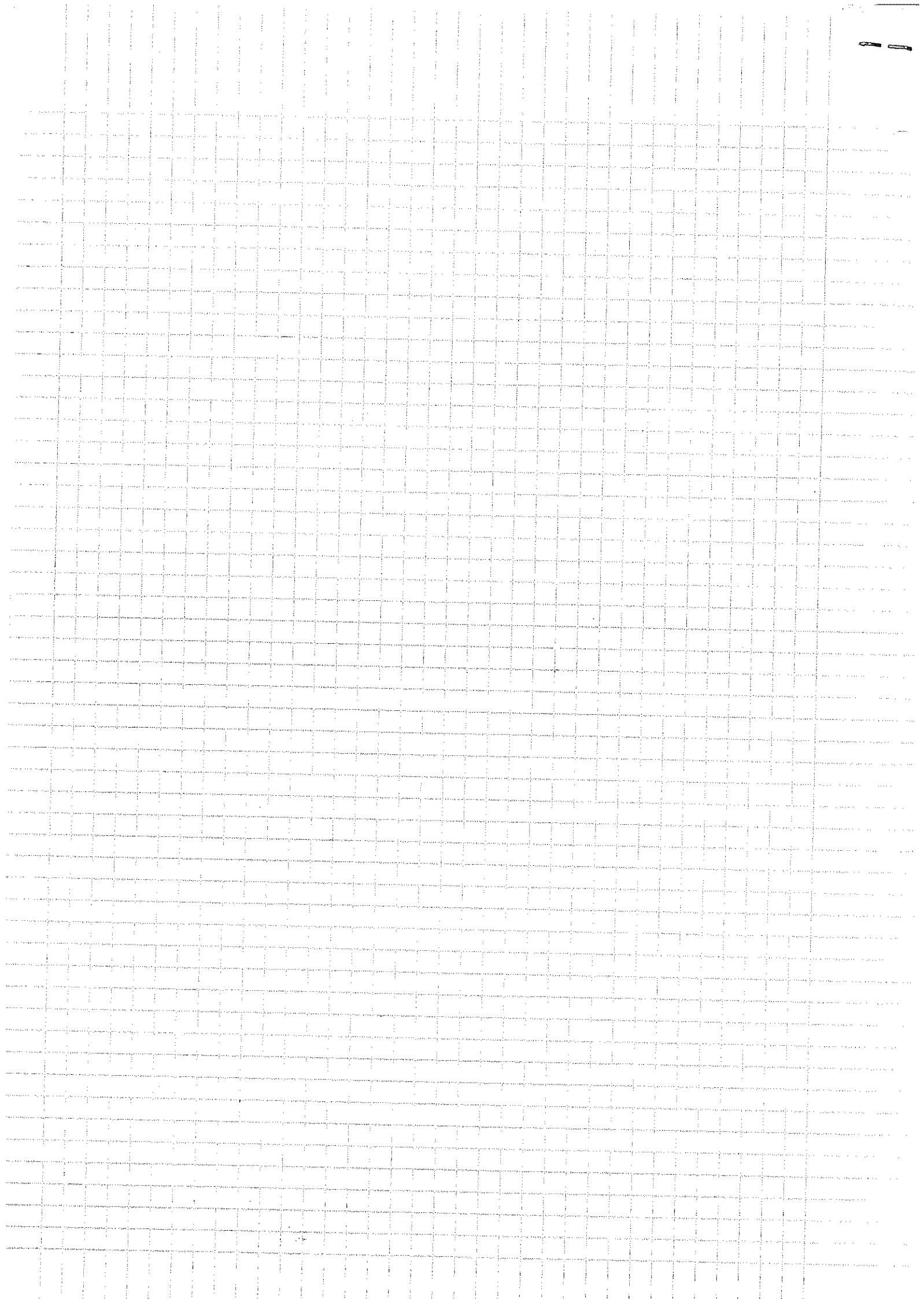
- fecho-ε

Exemplo: $(a+b)^*\ (ab + ba)$



	A $\{0,1,2,4,7,8,11\}$	B	C
a	$\{3,6,1,2,4,7,8,11\} = B$	$B + \{9\} = D$	
b	$\{5,6,1,2,4,7,8,11\} = C$	$C + \{12\} = E$	





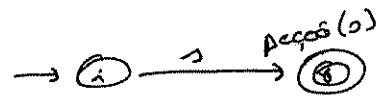
Nome: Rui Peixoto nº 51166 10-3-08

- Autómatos com condição de contexto
- Autómatos reactivos.

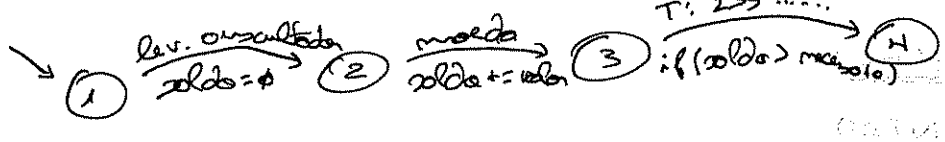
Trabalho:

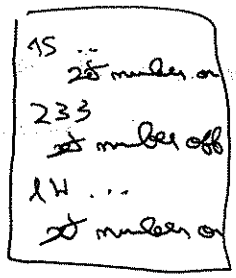
wiki → HTML

Autómatos reactivos:



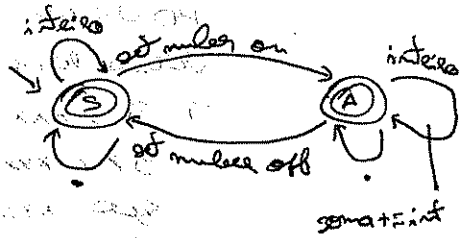
Ex: Cabine Telefónica





A soma foi ... .

set number on
set number off
[0-9]+



Em LEX:

```
%{
int flag = 0;
int soma = 0;
%}
%%
"set number on"     flag = 1;
"set number off"    flag = 0;
[0-9]+              { if (flag) soma += atoi (yytext); }
.                   ;
%%
main() {
yylex();
printf ("%d ", soma);
}
```

Cond. de contexto:

```
%{
int soma = 0;
%}
%start on
%%
    BEGIN 0;
"set ambler on"    BEGIN on;
<on> "set ambler off"    BEGIN 0;
<on> [0-9]+    soma += atoi (yytext);
    ;
.
```

O telefone:

```
LEVANTAR
POUSAR - devolve troco;
MOEDA 10,20,50 - inc. saldo;
T 253 xxx xxx - if (saldo suf)  dec. saldo chamada local;
  9xx xxx xxx -  "    [ " " ]    "    "    "    móvel
  800 xxx xxx - chamada grátis;
  808 xxx xxx - chamada local;
(ABORTAR)
```

Em lex:

```
%{
int saldo = 0;
%}
%start chamada moeda
%%
    BEGIN 0;
LEVANTAR   BEGIN chamada;
(chamada) POUSAR   { printf ("Troco = %d", saldo);
                     saldo = 0;
                     BEGIN 0; }
<chamada> MOEDA    BEGIN moeda;
(moeda) [0-9]+     saldo += atoi (yytext);
<moeda> \,         ;
<moeda> \n         BEGIN chamada;
%% --> [␣\t]+     ;
```

31/03/2008 | 50287

Data limite : 7 de Abril 23:59h

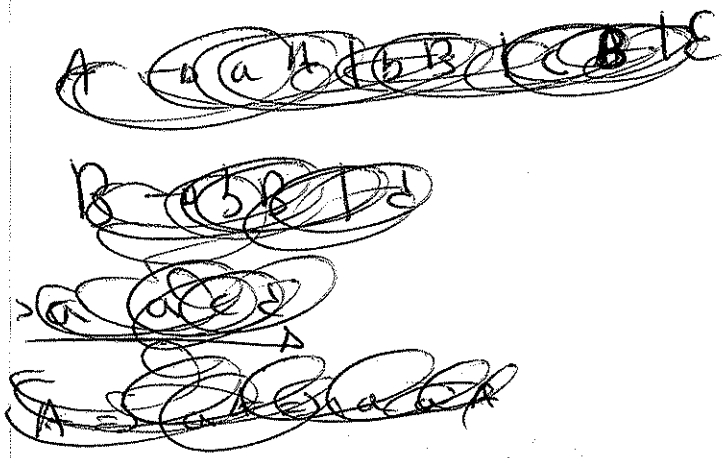
tp1-n1-n2-n3.EIP

$A \to B\,aA \mid bB \mid c$

$B \to bB \mid d$

>a a b b d

$A \Rightarrow aA \Rightarrow aaA \Rightarrow aabB \Rightarrow aabbB \Rightarrow aabbd$



$A \to aAb \mid \epsilon \mid cB$

$B \to d$

>a a a b c d *

$A \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aab$

(

)

+

-

*

/

int

G = (T = {
N = {
P = {

Exp → Exp op Exp | num

op → '+' | '-' | '/' | '*'

2 + 5 * 2

Exemplo : [] , [1] , [1, 2, 3, 4, 5]

lista → '[' conteudo ']'

conteudo → ε | num cont2

cont2 → ',' num cont2 | ε

[1]

lista ⇒ [conteudo] ⇒ [num cont2] ⇒ [num]

Exp → Termo Opad Exp
| Termo

Termo → Fator Opmul Termo
| Fator

Fator → '(' Exp ')' | num

Problema comum, Loop

Exp → Termo Exp2

Exp2 → Opad Exp | ε

Termo → Fator TERMO

Termo → Opmul TERMO

Fator → '(' Exp ')' | num

$A \rightarrow A a \mid b$

$baaa$

$A \rightarrow bA'$

$A' \rightarrow aA' \mid \varepsilon$

$A \rightarrow A\alpha \mid \beta$

$\Downarrow$

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$

Eduardo dos Santos Pereira 51176

3141/2008

INDICE

1 = processador, compilador

2 = CRC, ELI, compilado

Δ = BNF

FIM

processador : 1
compilador : 1, 2
...

ImSesc → INDICE Relations FIM (p1)

Relations → Relation Relations (p2)
| ε (p3)

Relation → id '=' PalList (p4)

PalList → id PalList2 (p5)

PalList2 → ε (p6)
| ';' id PalList2 (p7)

G = (T = {INDICE, FIM, '=', id, ';'},
N = {ImSesc, Relations, Relation, PalList, PalList2},
S = ImSesc
P = ...
)

	INDICE	FIM	'='	';'	id
ImSesc	P1	—	—	—	—
Relations		P3			P2
Relation					P4
PalList					P5
PalList2		P6		P7	P6

CA LA(Relations → ε) = Follow(Relations) = First(FIM) = {FIM}

LA(PalList2 → ε) = Follow(PalList2) = Follow(PalList) = Follow(Relation)
= First(Relation = {id, FIM}

Analisador léxico

```
%%
#define INDICE 400
#define FIM    401
#define PAL    500
#define ERRO   -1
```

terminais sinais 0 - 255
palavras reservadas - 400
variáveis - 500
ERRO - -1

```
%%
INDICE[0\n]*   return(INDICE);
FIM            return(FIM);
\=|\,          return ('='
                       *yytext
                       yytext[0]);

[a-zA-Z0-9]+   return (PAL);
.              return (ERRO);
```

①

```
#include "lex.yy.c"
static int prox_simb;

int main ()
{
   prox_simb = yylex();
   rec_Index();
   printf("\n SUCESSO!\n");
}
```

②

```
int rec_Index()
{
  rec_term (INDICE);
  rec_Relations();
  rec_term (FIM);
}

int rec_Relations ()
{ switch (prox_simb)
  {
   case PAL = rec_Relations()
              rec_Relations();
              break;
   case FIM = break
   default : error (prox_simb);
}}
```

③

```
int rec_term (int simb)
{ if (simb == prox_simb)
       prox_simb = yylex();
  else
     error (prox_simb);
}
```

④

```
int rec_PalList2 ()
{ switch (prox_simb)
   { case PAL;
     case FIM: break;
     case ',' : rec_term(',');
                rec_term (PAL);
                rec_PalList2 ();
                break;
     default = error (prox_simb)
   }
}
```