CMPS-104A
ch 3.1
Dragon 3:109

Lexical Analysis

we will use flex

task : read chars
make lexemes
produce tokens

$n = $ yylex()
yytext →

Separate from Parser

- simplify design : ex: parser can ignore comments
- buffering
- portability : input separation
  ex: ASCII, ISO Latin 1, Unicode
  UTF8
  UTF16

Tokens

~~Token = pair {token name, lexeme~~}

Token = pair { $\underset{V_T}{\text{tokenname}}$, attribute }

Pattern = description of form (regex)

Lexeme = seq of chars

ex:

Token	Lexeme	
ID	foo	bar
STRING	"foo"	"\n"
IF	if	
LE	<=	
'<'	<	

attributes

each token has a lexeme
plus source coordinates

file
line
offset

$G = \langle V_T, V_N, P, S \rangle$

36 "foo"

CMPS-104A
ch 3.2
Dragon 3:114

Lex Errors

- skip char ← we will do.
- ins/repl/swap

note -3 is two tokens not 1

x=3-4; x=3+-4;

binop unop

lexer can't tell.

Buffering

- need to look ahead (usu. 1 char, more?)

ex Fortran DO 5 I = 1.25
DO 5 I = 1, 25

C: < <= <<
\- -> --
foo<=

x+++y
x+ ++y

buffer if (den< num) foo = f(bar);

lex begin (yytext)
lex end.

yylex
scan () {
replace null plug.
yytext lexbegin = old lexend
advance lexend
~~*lexend = '\0'~~
t = *lexend
*lexend = 0
return lexbegin

}

CMPS-104A
ch3.3
Dragon 3:115

<u>two ptrs</u>

lexbegin → 1st char of lexeme
lexend → 1st char *after* lexeme

end of buffer: shift back.

Specifications

Strings & Languages.

alphabet: finite set of symbols.

EBCDIC

ASCII: 0..127
flex → ISOLatin1: 0..255 *
Unicode: 0.. 0x10FFFF

never use 7-bit charset. ↑ = 1,114,111

UTF8 } warning
UTF16 }

string: over an αβ is a seq of sym ∈ that αβ

empty string: ε ~~(ϵ vs ∈)~~ (ϵ vs ∈)

language: set of strings over an αβ
may be $\emptyset$

$t = \text{prefix}(s) \rightleftarrows \exists u : tu = s$

$t = \text{suffix}(s) \quad \exists u : ut = s$

$t = \text{substring}(s) \quad \exists u, v : utv = s$

proper means ≠

CMPS-104A
ch 3.4
Dragon 3:119

Operation on Lang

~~union, concat, closure~~

Union $L \cup M = \{ s \mid s \in L \vee s \in M \}$

concat $LM = \{ st \mid s \in L \wedge t \in M \}$

~~closure~~

Kleene closure $L^* = \bigcup_{i=0}^{\infty} L^i$

Positive closure $L^+ = \bigcup_{i=1}^{\infty} = L^i$

Regular Exprs

~~is~~ a C ident $= \lambda (\lambda \mid \delta)^*$

if $\lambda \in a \ldots z \, A \ldots Z$ _

$\delta \in 0 \ldots 9$

ⓔⓧ in Flex: `[a-zA-Z_]([a-zA-Z_]|[0-9])*`

or: `[a-zA-Z_][a-zA-Z_0-9]*`

~~Def~~

each regex ~~r~~ denotes a lang $L(r)$

~~given an alphabet Σ~~

~~Basis 1. ε is a regex, $L(\varepsilon) = \{\varepsilon\}$~~ ↑ empty string

~~2. if $a \in \Sigma$ then a is a~~ regex

```
LETTER      [A-Za-z_]
DIGIT       [0-9]
IDENT       {LETTER}({LETTER}|{DIGIT})*
```

CMPS-104A
ch 3.5
Dragon 3:121

each regex r denotes a lang $L(r)$ defined recursively

given an $\alpha\beta$ Σ

Basis 1. ε is a r.e.; $L(\varepsilon) = \{\varepsilon\}$

2. if $a \in \Sigma$ then a is a ~~re~~ RE and $L(a) = \{a\}$

ε

$\in$

Induction assume r, s are RE denoting $L(r)$, $L(s)$

1.	$(r) \mid (s)$ is a RE denoting	$L(r) \cup L(s)$
2.	$(r)(s)$	$L(r)L(s)$
3.	$(r)^*$	$(L(r))^*$
4.	(r)	$L(r)$

Precedence

(a) * highest left assoc

(b) concat medium left assoc

(c) | lowest left assoc.

Laws of RE

commutative

$r|s = s|r$

x

associative

$r|(s|t) = (r|s)|t$

$r(st) = (rs)t$

identity

$\varepsilon r = r \varepsilon = r$

distributive

$r(s|t) = rs|rt$

$(s|t)r = sr|tr$

closure

$r^* = (r|\varepsilon)^*$

$r^{**} = r^*$

CMPS-104A
ch 3.6
Dragon 3:123

Definitions (flex)

```
LETTER      [A-Za-z_]
DIGIT       [0-9]
IDENT       {LETTER}({LETTER}|{DIGIT})*

INTEGER     {DIGIT}+                  fails with octal
FRACTION    {DIGIT}+\.?|{DIGIT}*\.{DIGIT}+
EXPONENT    [Ee][+-]?{DIGIT}+
FNUMBER     {FRACTION}({EXPONENT})?
```

Extensions to Regex

$(r)+$ denotes $(L(r))^+$

?, +, * same prec, assoc.

$(r)?$ denotes $L(r) \cup \{\varepsilon\}$

[abc...] denotes a|b|c|...

CMPS-104A
ch 3.8
Dragon 3:128

3.4 Recognition of Tokens

patterns → code.

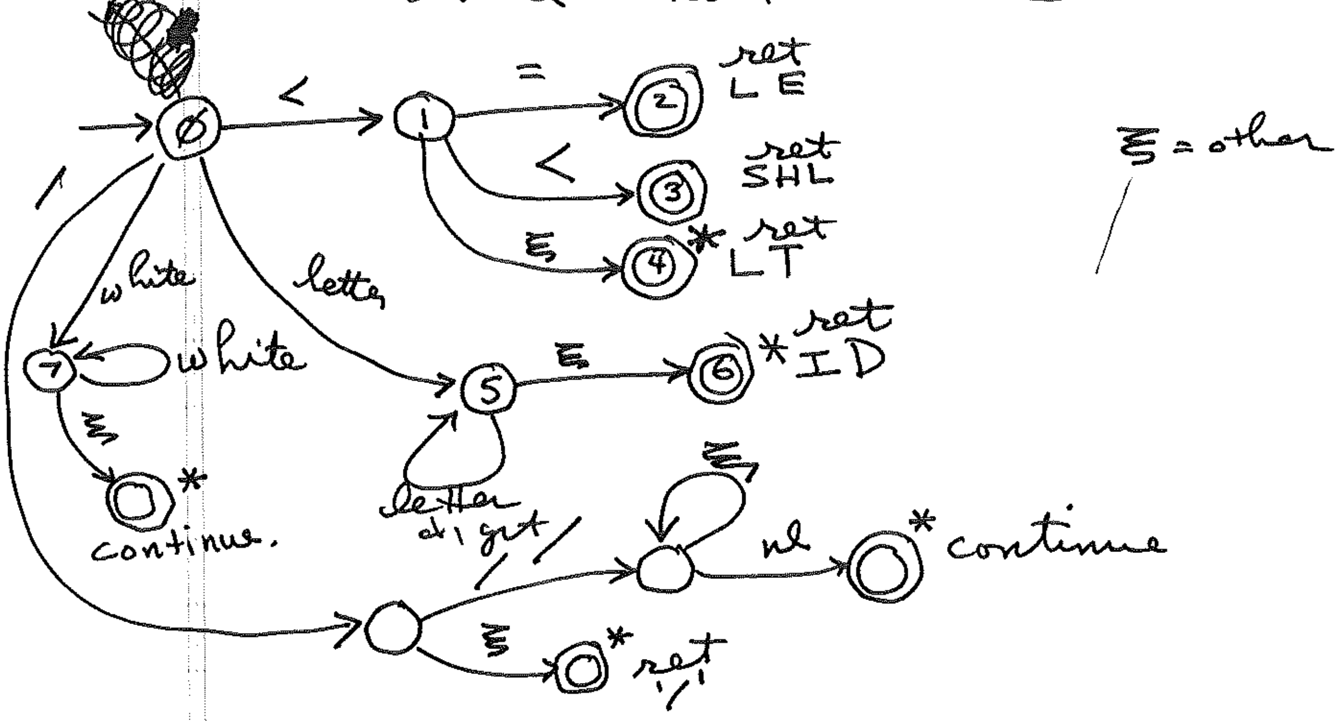
Transition diagrams

states = nodes } directed cyclic graph
label edges }

~~→○ 1. init state~~
◎ ~~2. final states~~

assume deterministic

1. some states are final ◎
2. some final are retracting ◎*
3. uniq start state →○

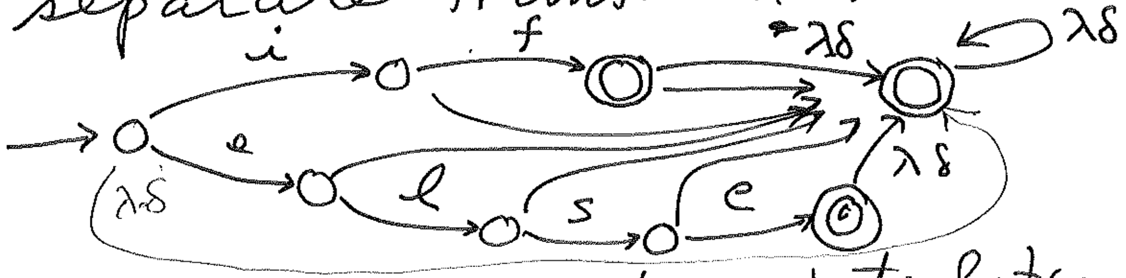


CMPS-104A
ch 3.9
Dragon 3:132

Recog Reg. words

- scan as ~~IDEnt~~ IDENT

1. install res wds in string table. before scan begins
   - extra field in token node
2. separate transition ∀ KW.



cost: many extra states & trans.

3. flex pats.

Coding

```
begin = forward;
state = 0;
for(;;) {
    ~~state = trans[*forward++][state];~~
    nstate = trans(state, *forward++);
    if (nstate == ERROR) fail();
    else if (final (nstate)) {
        ~~if (select~~
        return token(nstate)
    }else{
        state = nstate

    }

}
```

CMPS-104A
ch3.10
Dragon 3:131

actions

- trans to non final → continue
- trans to final space/comment
  begin = forward
- trans to final token
  retract if needed
  return token, lexeme
- end of buffer - refill
- graph repr = 2D array
  but SPARSE
- use char classes.
  instead of n states & 256 cols
  (Unicode ~~1000000~~?)
- init state very branchy
- others simpler
- or code directly.

Java idents

kitty
mačka
кошка
پیشی
猫

$¥£€

'A' ≠ 'A' ≠ 'A'

---

scan tokens example

jflex
cup

0x10FFFF
1114111

Regular expressions with a HUGE number of states

Article: 1642 of ucsc.class.cmps104a
Newsgroups: ucsc.class.cmps104a
Subject: Regular expressions with a HUGE number of states.
Distribution: ucsc
Organization: UC Santa Cruz CIS/CE
From: mackey@cse.ucsc.edu (W. Mackey)
NNTP-Posting-Host: asphalt3.cse.ucsc.edu
Message-ID: <47181434$1@darkstar>
Date: 18 Oct 2007 19:19:32 -0800
X-Trace: darkstar 1192760372 128.114.58.48 (18 Oct 2007 19:19:32 -0800)
Lines: 45
Path: darkstar!mackey
Xref: darkstar ucsc.class.cmps104a:1642

CS154 - Introduction to Automata and Complexity Theory
Stanford University
Jeffrey Ullman (Dragonbook author)
http://InfoLab.Stanford.EDU/~ullman/ialc/win00/win00.html

Here are some test files for your regular-expression processor (Project Part 1).

1. test1.txt: a RE whose DFA has about 1000 states.

```
(0|1)*1(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)
```

2. test2.txt: a RE whose DFA has about 32,000 states.

```
(0|1)*1(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)
       (0|1)(0|1)(0|1)(0|1)(0|1)
```

3. test3.txt: a RE whose DFA has about 1,000,000 states.

```
(0|1)*1(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)
       (0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)
```

4. test4.txt: a RE whose DFA has about 32,000,000 states.

```
(0|1)*1(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)
       (0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)(0|1)
       (0|1)(0|1)(0|1)(0|1)(0|1)
```

[01]{25, }

5. test5.txt: a RE with a big NFA but a very small DFA.

```
[a-z]?[a-z]?[a-z]?[a-z]?[a-z]?[a-z]?[a-z]?[a-z]?[a-z]?[a-z]?
```

6. test6.txt: an even more extreme version of test5.txt.

```
[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*
[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*[0-9]*
```

CMPS-104A
3.12
Dragon 3:140

3.5 Lex

```
int yylex () {
    for (;;) {
        n=scan pattern
        switch(n) {
            case: {act} break
            case:  {act} brk
        }
}
```

return tokencode.

yylex() fn

FILE *yyin —

yytext ⟶ flex buffer.

yylval = semantic info.

yyleng = #ch. scanned.

token codes 0 = EOF

1..255 = 'x'

256.. named tokens.

ambiguity

1. always longest char match
2. else use earlier patt.

3 evil thing

goto

global

pointer.

$a[i] \equiv *(a+i)$

$\equiv *(i+a)$

$\equiv i[a]$

CMPS-104A
3.13
Dragon 3:147

3.6 Finite Automata

NFA: nodes & directed edges
DFA: at most one edge
leaving a state for given symbol.

$NFA = \{S, \Sigma, t, s_0, F\}$

S = finite set of states
Σ = finite input alphabet, $\varepsilon \notin \Sigma$
t: trans function.
$$t : S \times (\Sigma \cup \{\varepsilon\}) \longrightarrow S$$
s_0: start state
F: set of final states, $F \subseteq S$

$\in$

ε

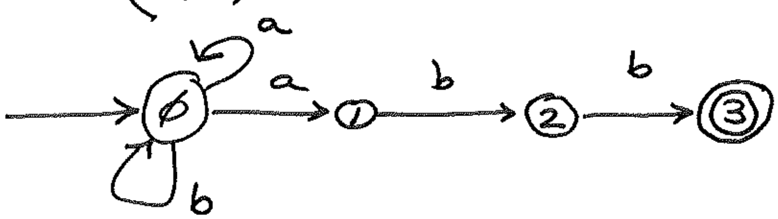
transition diagram

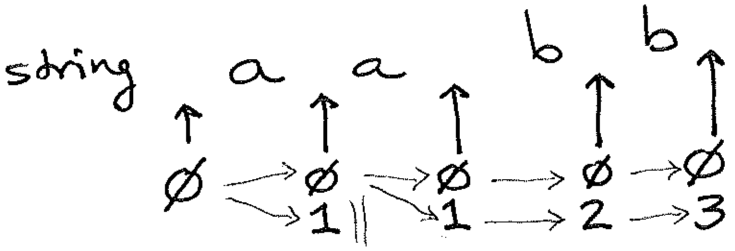
$(s) \xrightarrow{a \in \Sigma} (t)$ eats a when move

$(s) \xrightarrow{\varepsilon} (t)$ move w/o ~~[illegible]~~ eat a char.

ex: $(a|b)^* abb$

not DFA





✓ ($3 \in F$)

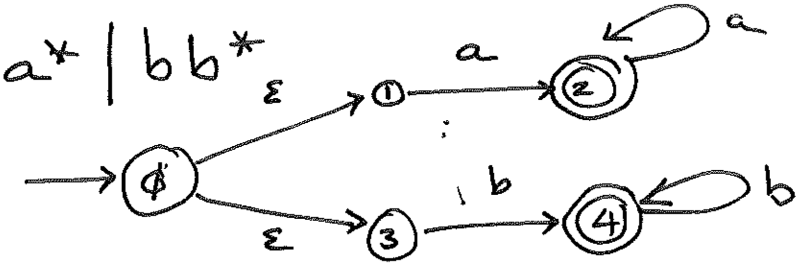
CMPS-104A
3.14
Dragon 3:148

simulate NFA

```
$C \leftarrow \{S_0\}$
loop {
    $a \leftarrow$ getc();
    if($a$ == EOF) break;
    $C \leftarrow \{t' \mid t \in C \wedge \exists t' = move(t, a)\}$
}
return $\exists t \mid t \in C \wedge t \in F$
```

$t \xrightarrow{a} t'$

ex: $aa^* \mid bb^*$

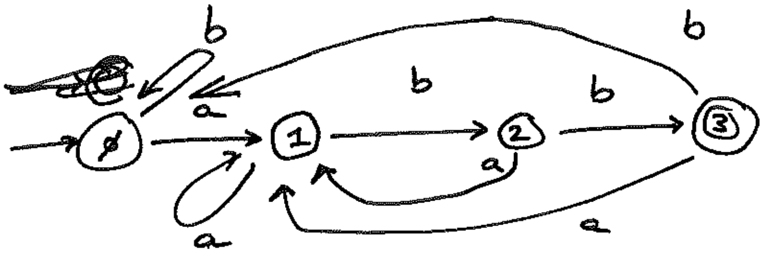


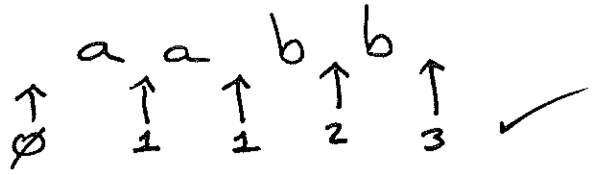
DFA

a DFA is an NFA except:

- no ε moves
- each outgoing edge ~~is unique~~ has unique label

ex: $(a|b)^* abb$





CMPS-104A
3.15
Dragon 3 :151

simulate DFA

```
s = s0
loop {
    a = getc()
    if (a == EOF) break
    s = move (s, a)
}
return s ∈ F
```

NOTE: we don't use dead states <u>in actual</u> scanner.

```
s = s0
loop {
    a = getc()
    t = move(s,a)
    if( ∄t ) break.
    s = t
}
ungetc(a);
if s ∈ F OK else error .
```

note: we scan until stuck
then exit
don't scan to EOF.

theory DFA stops @ EOF
scanner stops @ end of token

CMPS - 104A
3.16
Dragon 3:152

~~ex: strict DFA~~

simulate NFA:

```
S = $\varepsilon$closure ($s_0$)
loop {
    a = getc() ;
    if ( c == EOF) break;
    S = $\varepsilon$closure (move (S,a))
}
if (S ∩ F == $\phi$) error else OK
```

3.7 Regex → Automata

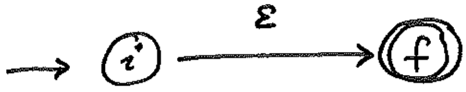
Thompson's construction
RE → NFA

input: RE r over Σ
output: NFA accepting L(r)

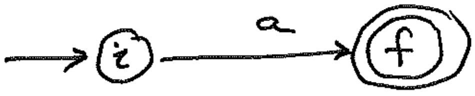
~~Induction~~

Basis

(1) ε



(2) $a \in \Sigma$

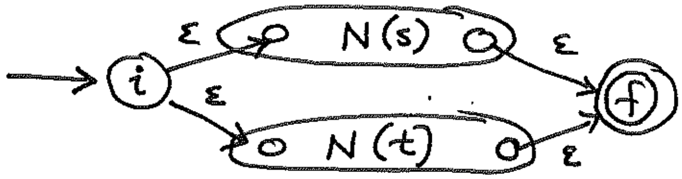


for simple symbols.

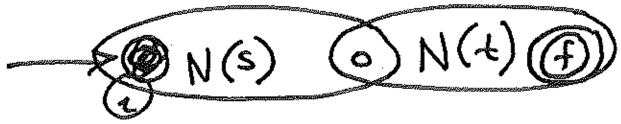
Induction

let N(s), N(t) be the NFA for reg s, t

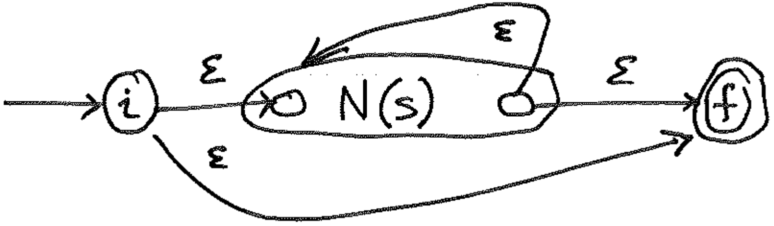
(a) r = s|t. then N(r) is



(b) r = st. then N(r) is



(c) r = s* then N(r) is



(d) r = (s) then L(r) = L(s)

∴ N(r) = N(s)

each step consists of at most 2 states

∴ size (N(r)) is O(n) for size(r)

[states] [symbols]

size: O(|r|)

construction speed: O(|r|)

scanning speed: O(|r| * |s|)

string s

CMPS-104A
3.18
Dragon 3:157

Efficiency of NFA simulation

(1) two sets: old set, newset
(2) temp bool set already on if in new set.
(3) array move[s,a] elt is set of states
(4) perform εclosure on each state [could precompute]

---

for each char:
    for each state

Conv NFA → DFA

alg: <u>subset construction</u>

each state of DFA is a set of states of NFA
worst case: $O(2^n)$
where NFA has n states
usually not so bad.

input: NFA N
output: DFA D accepts same lang

εclosure(s) = set of NFA states reachable from s via ε trans

εclosure(T) = set of NFA states reachable from NFA state $s \in T$

$$= \bigcup_{s \in T} \varepsilon\text{closure}(s)$$

CMPS-104A
3.19

Dragon 3:153

move(T,a) set of NFA states
$\exists$ trans from $a \in \Sigma$
from $s \in T$

subset construction

```
Dstates = $\varepsilon$closure($S_0$)
while($\exists$ unmarked T $\in$ Dstates)
        mark T
        $\forall$ a $\in \Sigma$ {
            U = $\varepsilon$closure(move(T,a))
            if U $\notin$ Dstates {
                add U to Dstates
                unmarked
            }
            Dtran[T,a] = U
        }
    }
```

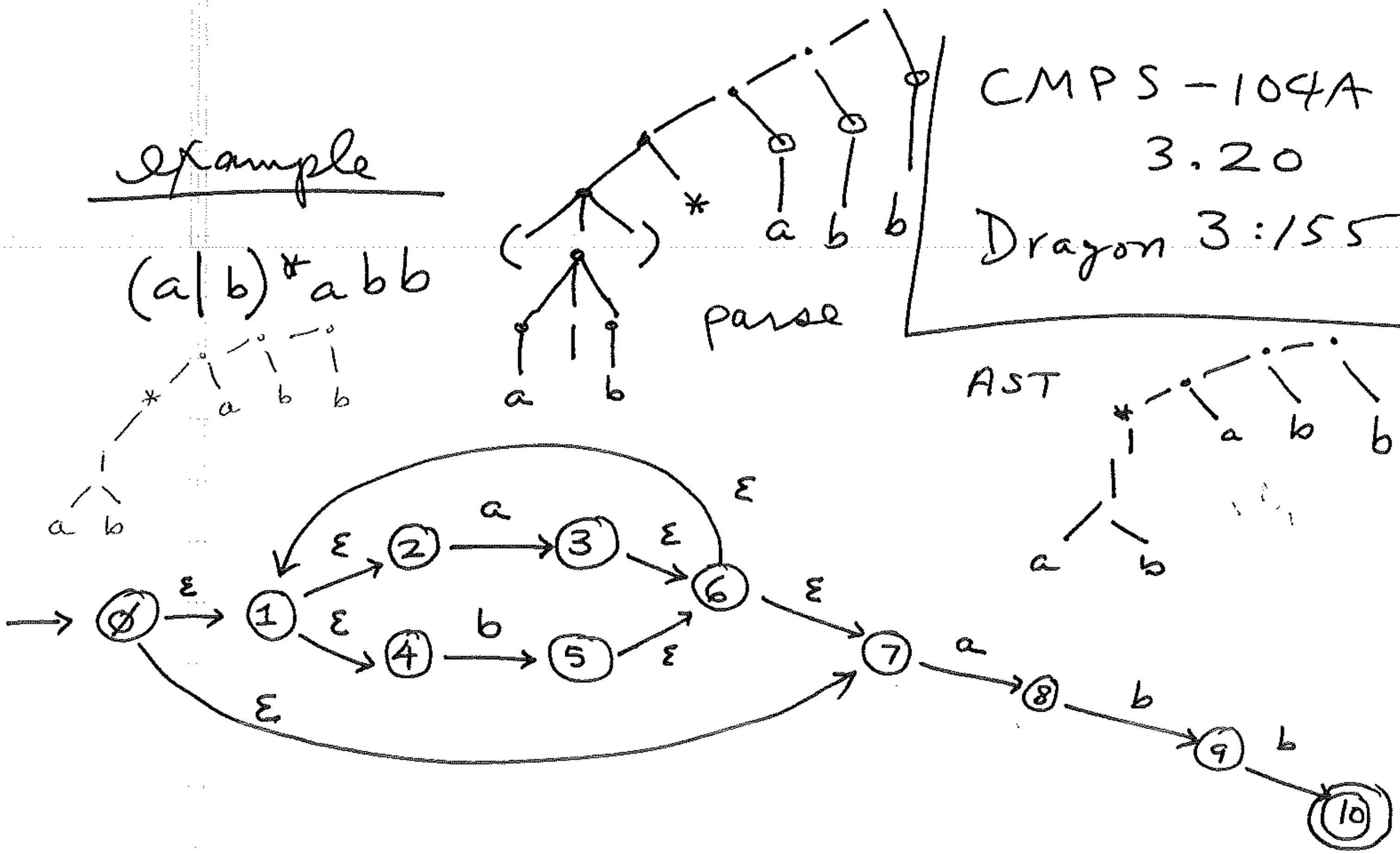
Eclosure(T)

```
push all states of T on stk
$\varepsilon$closure(T) = T
while (stk not empty) {
    t = pop stk
    ~~for~~ $\forall$ u : edge t $\xrightarrow{\varepsilon}$ u {
        if (u $\notin$ $\varepsilon$closure(T) :
            add u to $\varepsilon$closure(T)
            push u
    }
  }
}
```

CMPS-104A
3.20
Dragon 3:155

example

(a|b)*abb



ex a b a b b

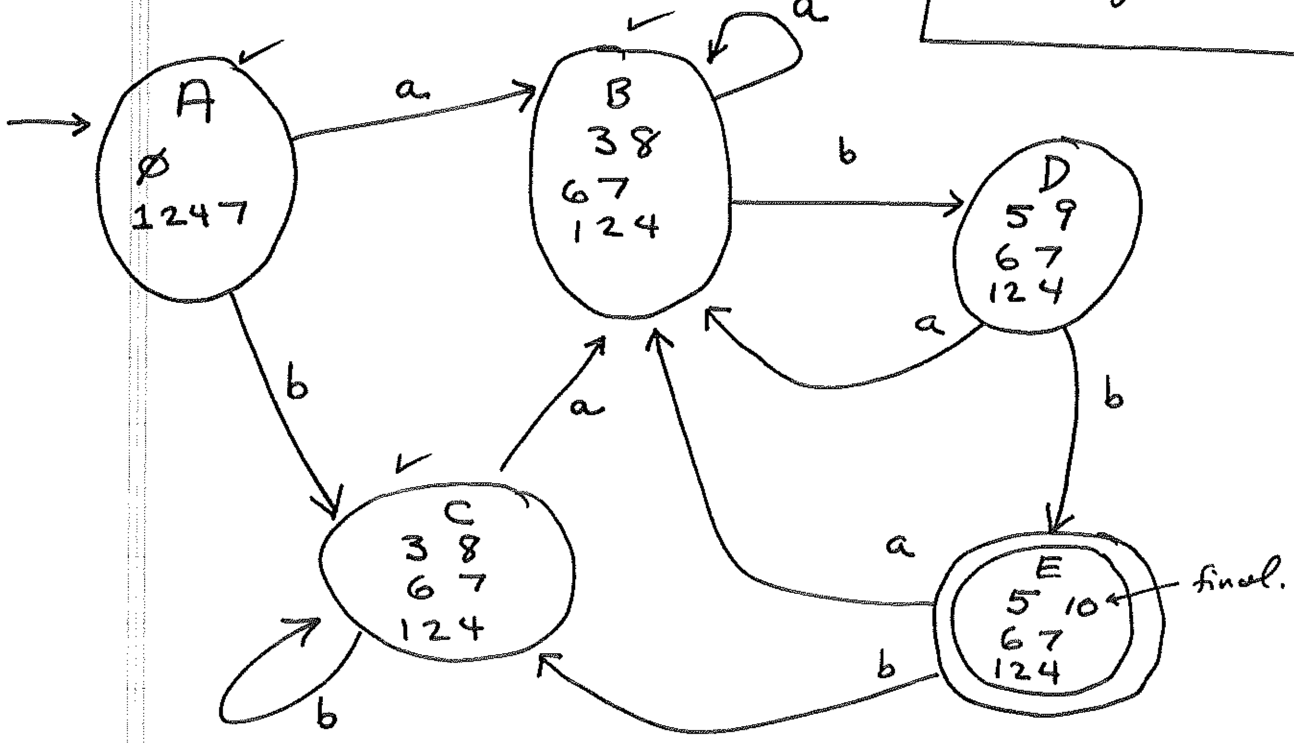
↑	↑	↑	↑	↑	↑
0	3	5	3	[illegible]	5
1	8	9	6	5	10
2	6	6	8	9	6
4	1	1	1	6	7
7	2	2	2	7	1
	4	4	4	1	2
	7	7	7	2	4
				4	

← 10 is final.

S	εclosure(s)
0	0 1 2 4 7
1	1 2 4
2	2
3	3 1 2 4 6 7
4	4
5	5 1 2 4 6 7
6	6 1 2 4 7
7	7
8	8
9	9
10	10

NFA → DFA

CMPS-104A
p. 3.21
Dragon 3:155



a b a b b

A B D B D E.

~~Minimizing a DFA~~

CMPS-104A
p. 3.22
Dragon 3:180

Minimizing a DFA ($D \to D'$)

1. Start with partition Π
two groups, F and S-F
S = set of all states
F = set of final states
S-F = " " non final states

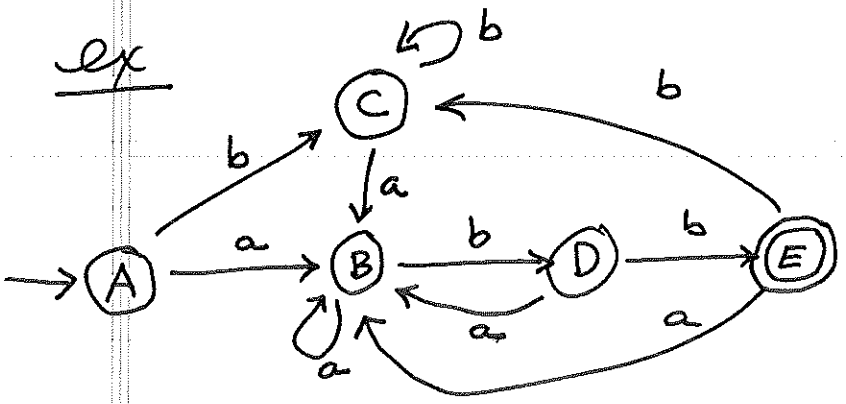
2. $\Pi' = \Pi$

```
$\forall (G \in \Pi)$ {
    partition G into subgroups where
        states s, t ∈ same subgroup
        iff $\forall a \in \Sigma$ $s,t \xrightarrow{a}$ same subgroup of $\Pi$
    replace G in $\Pi'$ by new subgroups
}
```

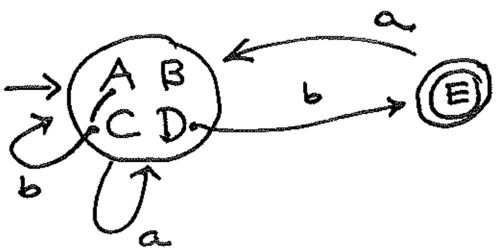
3. if ($\Pi' = \Pi$) let Πfinal = Π & goto (4)
else $\Pi = \Pi'$ and repeat (2)

(4) Choose each state of each group ∈ Πfinal
as representative of minimum D'
(a) start s of D' contains start s of D
(b) final states of D' are any that have
final of D
(c) Let s be represent a~~tive~~ of one G ∈ Πfinal
trans $s \xrightarrow{a} t$

ex



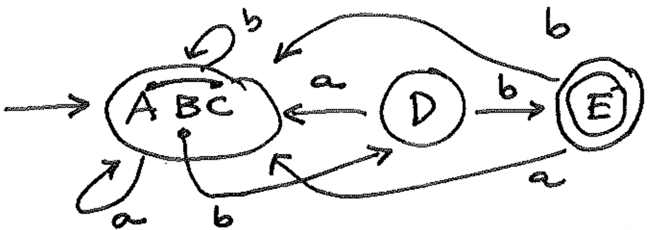
pass 1



a: agrees.

b: ABC | D

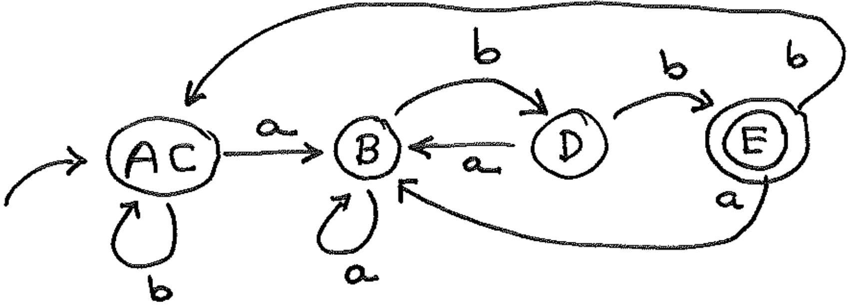
pass 2



a : agree

b : AC | B

pass 3



Efficiency

regex r
string ~~B~~ x

lex - prefer DFA

grep - prefer NFA?

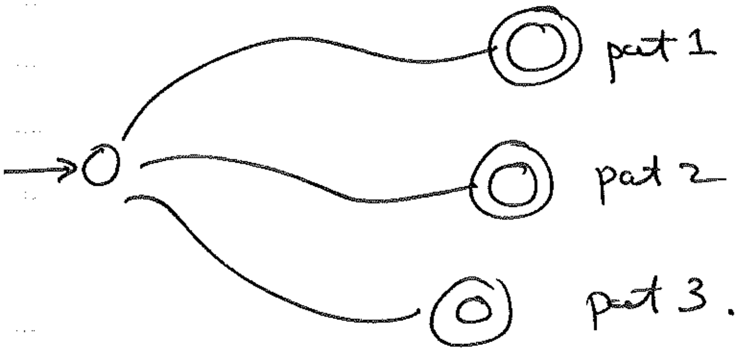
~~ato~~ αυτομaτον	space	constr time	parsing
NFA	$O(\lvert r\rvert)$	$O(\lvert r\rvert)$	$O(\lvert r\rvert \times \lvert x\rvert)$
DFA typical		$O(\lvert r\rvert^3)$	$O(\lvert x\rvert)$
DFA worst	$O(2^{\lvert r\rvert})$	$O(\lvert r\rvert^2 2^{\lvert r\rvert})$	$O(\lvert x\rvert)$

CMPS-104a
p3.24
Dragon 3:

~~example~~
~~a(b|c)* | bb~~

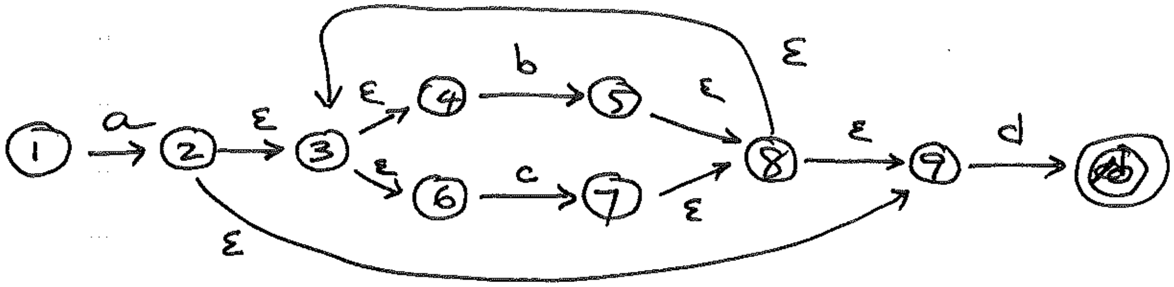
Flex

need sp final state ∀ rule



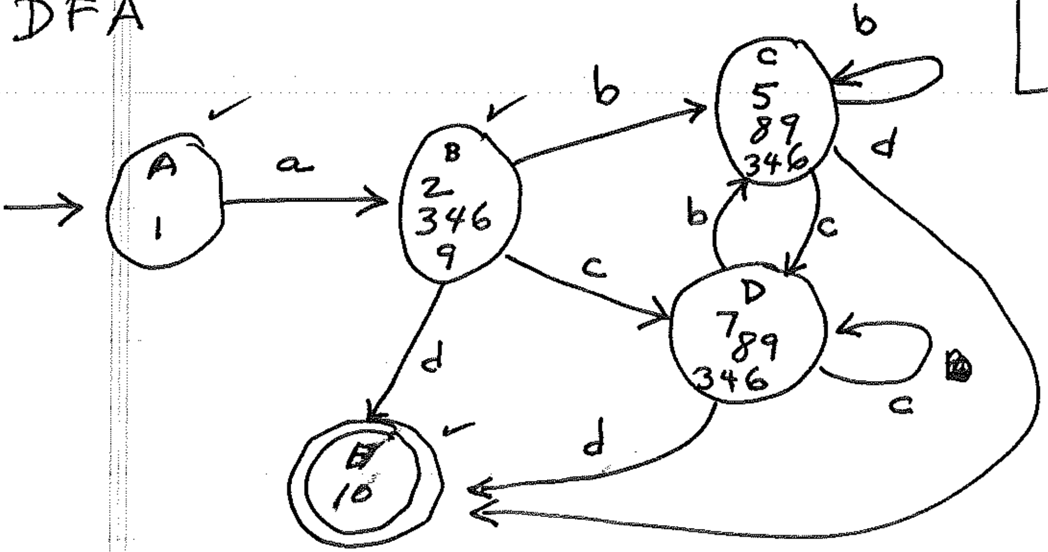
---

another ex a(b|c)*d

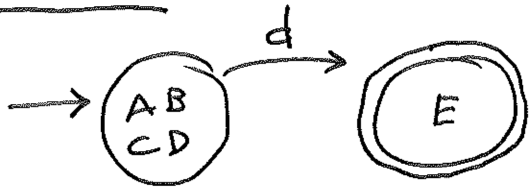


S	Eclosure(s)
1	1
2	2 3 9 4 6
3	3 4 6
4	4
5	5 8 9 3 4 6
6	6
7	7 8 9 3 4 6
8	8 9 3 4 6
9	9
10	10

DFA

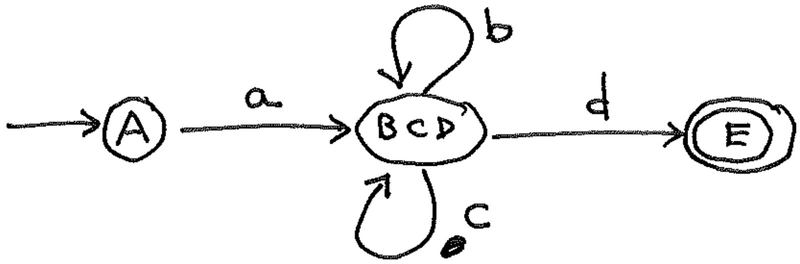


Minimize



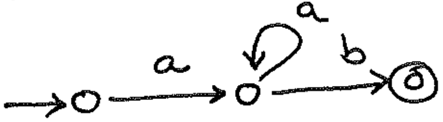
state ABCD

d: A | B CD
a: A | BCD
b: A | BCD
c: A | BCD



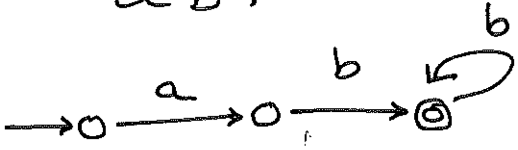
---

a+b

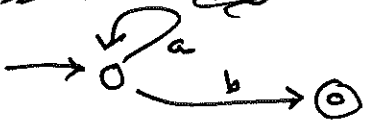


DFA

ab+



a*b



ab*