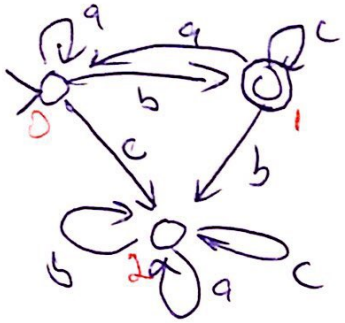
Finite State Automata (FSA)



◎	accept (yes) state
○	state
→ ○	start state
○ —a→ ○	transition under reading a

FSA

What language does this machine accept? $\Sigma = \{a, b, c\}$ alphabet

~~$(b c)^*$~~ $b\,c^*$? No

$b(a \vee \lambda)$ ---- ?

$(a \vee bc^*a)^* b c^*$ is proposed line by Prof.

a|a|ba|bcca|bcccc

An FSA is a five tuple (S, s_0, Σ, f, F)

where S is a finite set (of states), non empty

$s_0 \in S$ the start state

Σ is a finite alphabet

$f: S \times \Sigma \rightarrow S$ transition function

↑ current state ↑ character read next ↑ next state

$f(p, a)$ is the state reached from state p when reading a.

$F \subseteq S$ accepting states ◎

An FSA configuration is a pair in $S \times \Sigma^*$, i.e.

a pair of a state p and a remaining input string w

(p, w) is a configuration when $p \in S$ and $w \in \Sigma^*$

When computing on input string w, the starting configuration is (s_0, w)

For any $a \in \Sigma$, string $w \in \Sigma^*$, and state $p \in S$,

the configuration (p, aw) transitions to $(f(p,a), w)$

(q, w) when $q = f(p,a)$ written $(p, aw) \vdash_M (f(p,a), w)$

one step of FSA computation

for FSA M.

Our example FSA accepts aabababccabcccc :

$(0, aabababccabcccc) \vdash (0, ababccabcccc) \vdash (0, babccabcccc) \vdash$

$(1, abccabcccc) \vdash (0, bccabcccc) \vdash (1, ccabcccc) \vdash$

$(1, cabcccc) \vdash (1, abcccc) \vdash (0, bcccc) \vdash (1, cccc) \vdash \cdots$

$\cdots (1, \lambda)$

yes aababccabcccc is in the language accepted.

For FSA M the language accepted is

$$L(M) = \{ w \mid (s_0, w) \vdash_M \cdots \vdash_M (p, \lambda) \text{ for some } p \in F \}$$

$$= \{ w \mid (s_0, w) \vdash_M^* (p, \lambda) \text{ for some } p \in F \}$$

R^* is the reflexive transitive closure of R, for binary relation R

$L(\alpha)$ gives the set of strings matching reg. exp α ⇓

A regular expression is either:

$\emptyset$	empty language	$L(\emptyset) = \emptyset$
λ	language of just the empty string	$L(\lambda) = \{\lambda\}$
a	for any $a \in \Sigma$	$\{a\}$ $L(a) = \{a\}$
$\alpha\beta$	for α & β regular expressions "concatenation"	$L(\alpha\beta) = \{\omega x \mid \omega \in L(\alpha), x \in L(\beta)\}$
$\alpha \vee \beta$	"union"	$L(\alpha \vee \beta) = L(\alpha) \cup L(\beta)$
α^*	"iteration"	$L(\alpha^*) = L(\lambda) \cup L(\alpha) \cup L(\alpha\alpha) \cup \cdots$ $= \bigcup_{i \in \mathbb{N}} L(\alpha^i)$ where $\alpha^0 = \lambda$ and $\alpha^{i+1} = \alpha\alpha^i$

$a(b^*)$
$(ab)^*$

Kleene's Thm: $\forall L \subseteq \Sigma^*$ $\exists$ FSA M $L(M) = L$ iff $\exists$ reg. exp α $L(\alpha) = L$

Focus first on Finding M given α s.t. ~~$L = L(\alpha)$~~

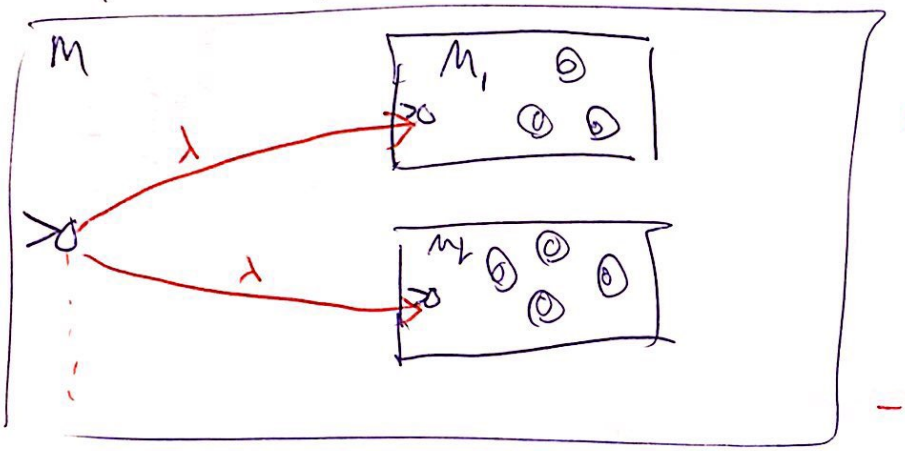
~~s. that M~~ $L(M) = L$ s.t. $L(M) = L(\alpha)$

idea: handle each method of building up a regex:

e.g.

given M_1 s.t. $L(M_1) = L(\alpha_1)$ and M_2 s.t. $L(M_2) = L(\alpha_2)$

find M s.t. $L(M) = L(\alpha_1 \vee \alpha_2)$



Accept the input if any choice available works to reach an accept state

- specifies an exponential search
- NONDETERMINISM