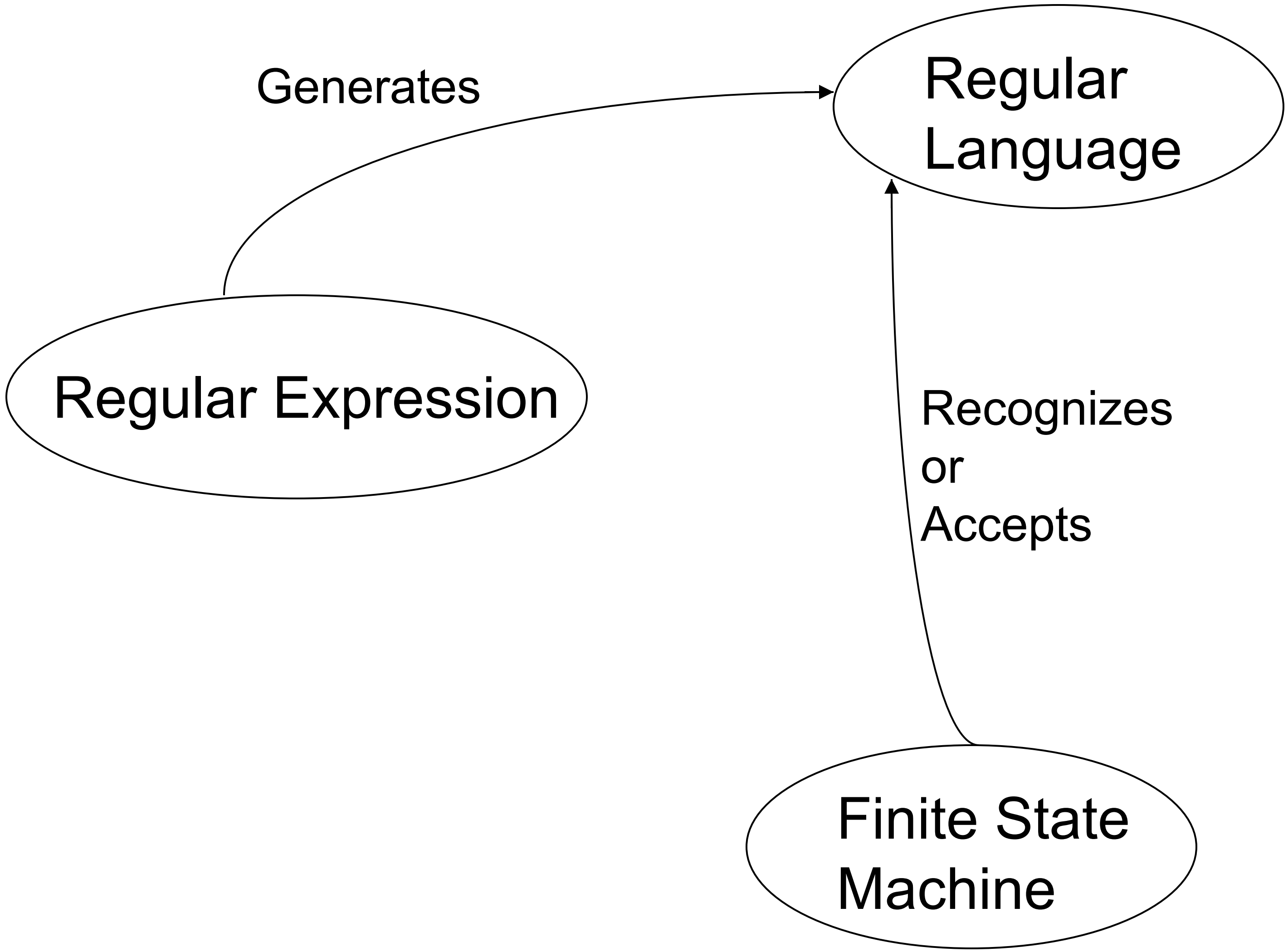
Regular Expressions

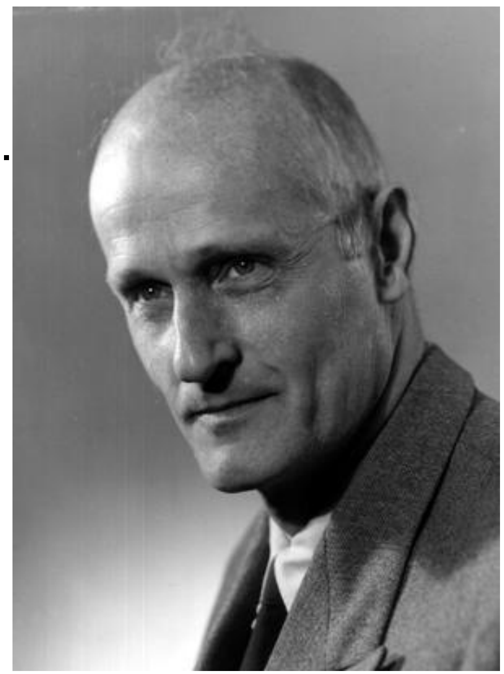
Chapter 6

Regular Languages



Stephen Cole Kleene

- 1909 – 1994, mathematical logician
- One of many distinguished students (e.g., Alan Turing) of Alonzo Church (lambda calculus) at Princeton.
- Best known as a founder of the branch of mathematical logic known as recursion theory.
- Also invented regular expressions.
- Kleene pronounced his last name *KLAY-nee*. *`kli:ni* and *`kli:n* are common mispronunciations.
    - His son, Ken Kleene, wrote: "As far as I am aware this pronunciation is incorrect in all known languages. I believe that this novel pronunciation was invented by my father. "



- Kleeneness is next to Godelness
    - Cleanliness is next to Godliness

Regular Expressions

Regular expression Σ contains two kinds of symbols:

- special symbols, $\varnothing$, ε, *, +, $\cup$, (,) …
- symbols that regular expressions will match against

The regular expressions over an alphabet Σ are all and only the strings that can be obtained as follows:

1. $\varnothing$ is a regular expression.
2. ε is a regular expression.
3. Every element of Σ is a regular expression.
4. If α , β are regular expressions, then so is $\alpha\beta$.
5. If α , β are regular expressions, then so is $\alpha\cup\beta$.
6. If α is a regular expression, then so is α^*.
7. α is a regular expression, then so is α^+.
8. If α is a regular expression, then so is (α).

Regular Expression Examples

If $\Sigma = \{a, b\}$, the following are regular expressions:

$\varnothing$

ε

a

$(a \cup b)^*$

$abba \cup \varepsilon$

Regular Expressions Define Languages

- Regular expressions are useful because each RE has a meaning
- If the meaning of an RE α is the language *A*, then we say that α defines or describes *A*.

Define *L*, a **semantic interpretation function** for regular expressions:

1. $L(\varnothing) = \varnothing$. //the language that contains no strings
2. $L(\varepsilon) = \{\varepsilon\}$. //the language that contains just the empty string
3. $L(c) = \{c\}$, where $c \in \Sigma$.
4. $L(\alpha\beta) = L(\alpha)\ L(\beta)$.
5. $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$.
6. $L(\alpha^*) = (L(\alpha))^*$.
7. $L(\alpha^+) = L(\alpha\alpha^*) = L(\alpha)\ (L(\alpha))^*$. If $L(\alpha)$ is equal to $\varnothing$, then $L(\alpha^+)$ is also equal to $\varnothing$. Otherwise $L(\alpha^+)$ is the language that is formed by concatenating together one or more strings drawn from $L(\alpha)$.
8. $L((\alpha)) = L(\alpha)$.

The Role of the Rules

- Rules 1, 3, 4, 5, and 6 give the language its power to define sets.
- Rule 8 has as its only role grouping other operators.
- Rules 2 and 7 appear to add functionality to the regular expression language, but they don't.

2. ε is a regular expression.

7. α is a regular expression, then so is α^+.

Analyzing a Regular Expression

The compositional semantic interpretation function lets us map between regular expressions and the languages they define.

$$L((\mathtt{a} \cup \mathtt{b})^*\mathtt{b}) = L((\mathtt{a} \cup \mathtt{b})^*)\ L(\mathtt{b})$$

$$= (L((\mathtt{a} \cup \mathtt{b})))^*\ L(\mathtt{b})$$

$$= (L(\mathtt{a}) \cup L(\mathtt{b}))^*\ L(\mathtt{b})$$

$$= (\{\mathtt{a}\} \cup \{\mathtt{b}\})^*\ \{\mathtt{b}\}$$

$$= \{\mathtt{a}, \mathtt{b}\}^*\ \{\mathtt{b}\}.$$

Examples

$L(\ a^*b^*\ ) =$

$L(\ (a \cup b)^*\ ) =$

$L(\ (a \cup b)^*a^*b^*\ ) =$

$L(\ (a \cup b)^*abba(a \cup b)^*\ ) =$

$L(\ (a \cup b)\,(a \cup b)a(a \cup b)^*\ ) =$

Going the Other Way

Given a language, find a regular expression

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*: |w| \text{ is even}\}$

$((\mathtt{a} \cup \mathtt{b})\ (\mathtt{a} \cup \mathtt{b}))^*$

$(\mathtt{aa} \cup \mathtt{ab} \cup \mathtt{ba} \cup \mathtt{bb})^*$

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*: w \text{ contains an odd number of } \mathtt{a}\text{'s}\}$

$\mathtt{b}^*\ (\mathtt{ab}^*\mathtt{ab}^*)^*\ \mathtt{a}\ \mathtt{b}^*$

$\mathtt{b}^*\ \mathtt{a}\ \mathtt{b}^*\ (\mathtt{ab}^*\mathtt{ab}^*)^*$

Common Idioms

$(\alpha \cup \varepsilon)$

- Optional α, matching α or the empty string

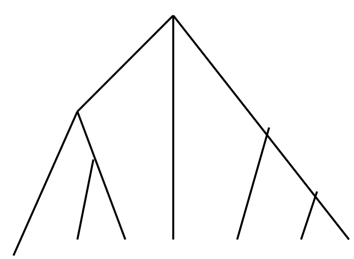
$(\mathtt{a} \cup \mathtt{b})^*$

- Set of all strings composed of the characters a and b

- The regular expression a* is simply a string. It is different from the language L(a*) ={w: w is composed of zero or more a’s}.
- However, when no confusion, we do not write the semantic interpretation function explicitly. We will say things like, “The language a* is infinite”

Operator Precedence in Regular Expressions

	Regular Expressions	Arithmetic Expressions
Highest	Kleene star	exponentiation
	concatenation	multiplication
Lowest	union	addition



$a\ b^* \cup c\ d^*$

$x\ y^2 + i\ j^2$

The Details Matter

$a^* \cup b^* \neq (a \cup b)^*$

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

Kleene's Theorem

Finite state machines and regular expressions define the same class of languages. To prove this, we must show:

***Theorem**:* Any language that can be defined with a regular expression can be accepted by some FSM and so is regular.

Theorem: Every regular language (i.e., every language that can be accepted by some DFSM) can be defined with a regular expression.

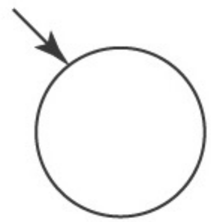
- Sometimes FSM is easy, sometimes RE is easy.

For Every Regular Expression α, There is a Corresponding FSM *M* s.t. $L(\alpha) = L(M)$

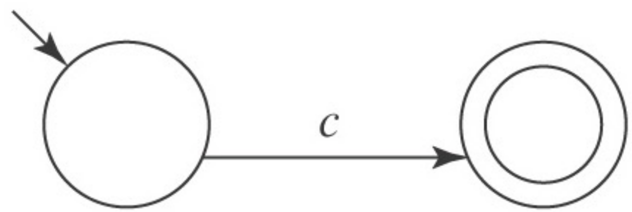
We'll show this by construction.

First, primitive regular expressions, then regular expressions that exploit the operations of union, concatenation, and Kleene star.

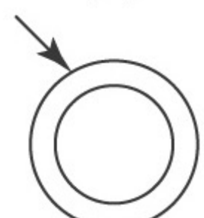
$\varnothing$:



A single element of Σ:

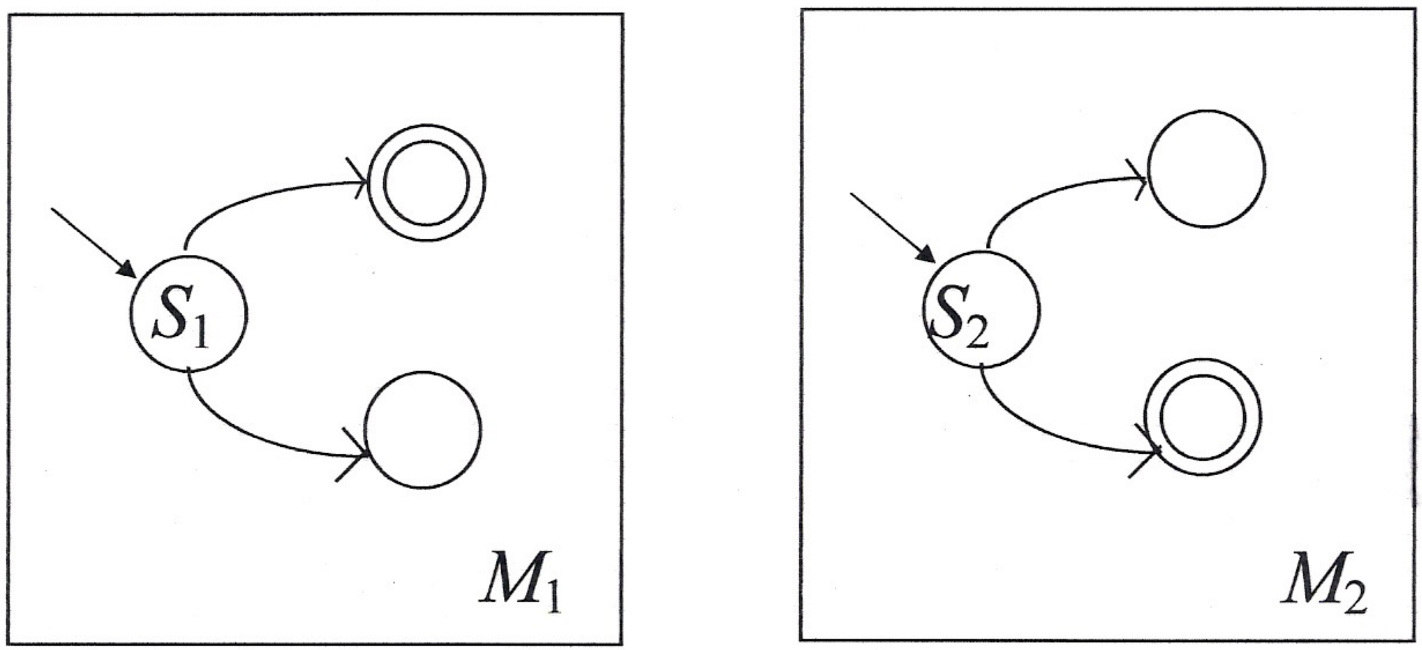


ε ($\varnothing^*$):



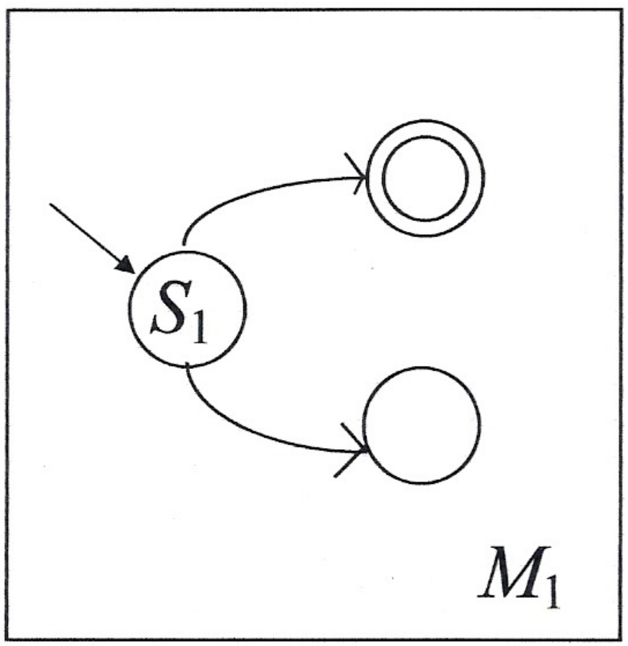
Union

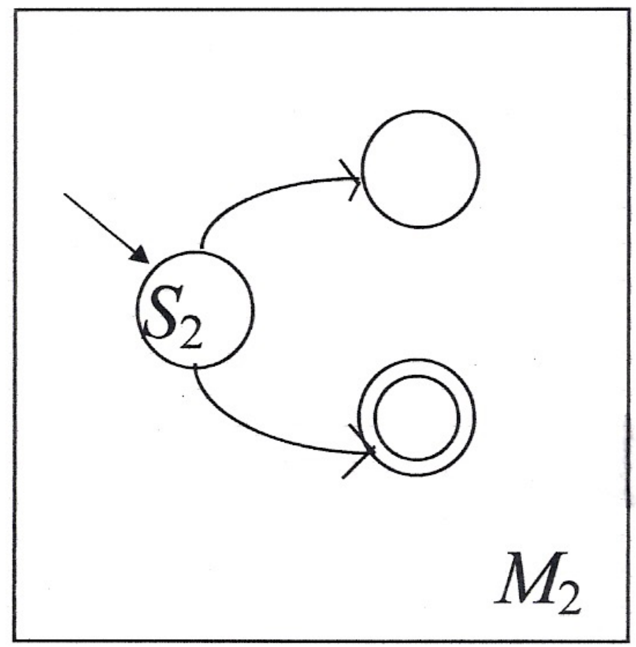
If α is the regular expression $\beta \cup \gamma$ and if both $L(\beta)$ and $L(\gamma)$ are regular:



Concatenation

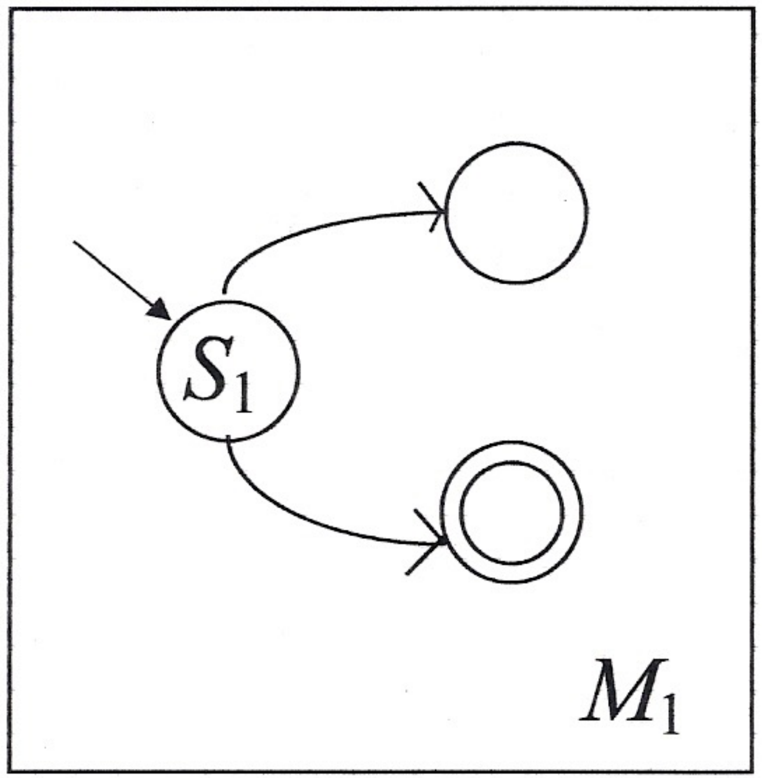
If α is the regular expression $\beta\gamma$ and if both $L(\beta)$ and $L(\gamma)$ are regular:





Kleene Star

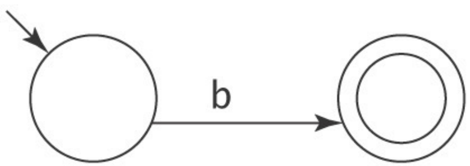
If α is the regular expression β^* and if $L(\beta)$ is regular:



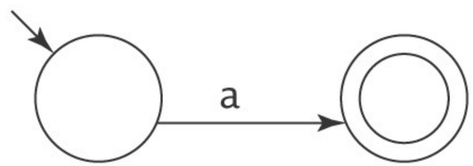
From RE to FSM: An Example

(b ∪ ab)*

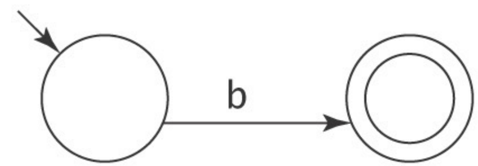
An FSM for b



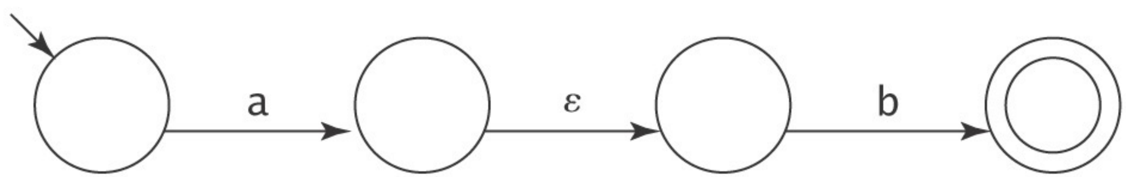
An FSM for a



An FSM for b



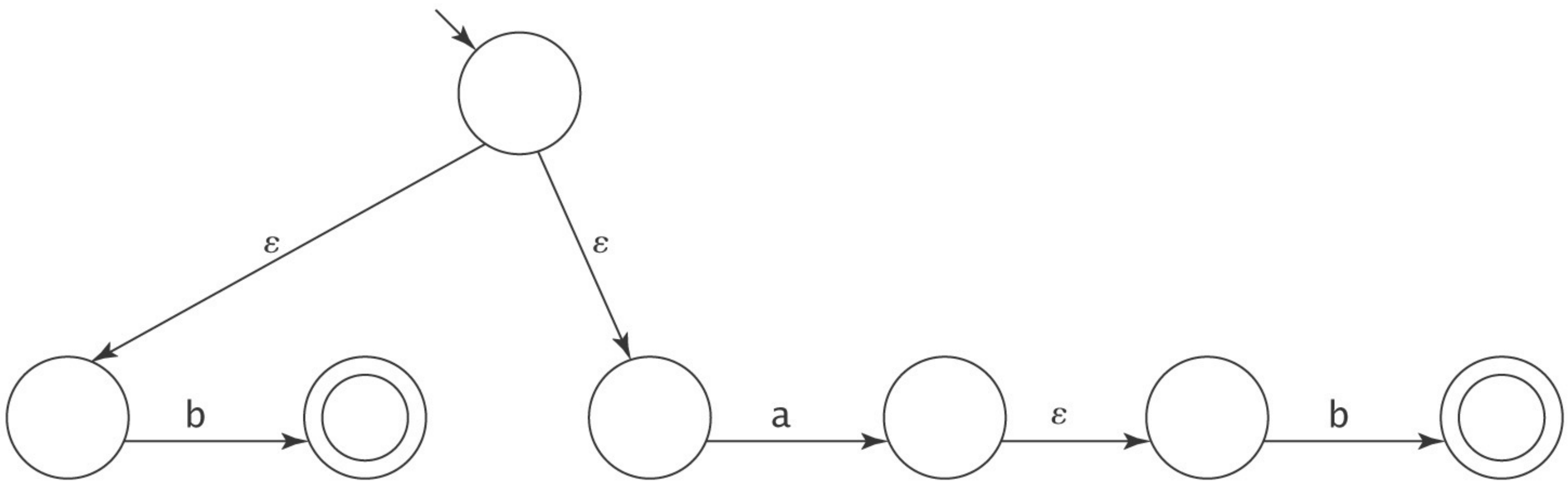
An FSM for ab:



An Example

(b $\cup$ ab)*

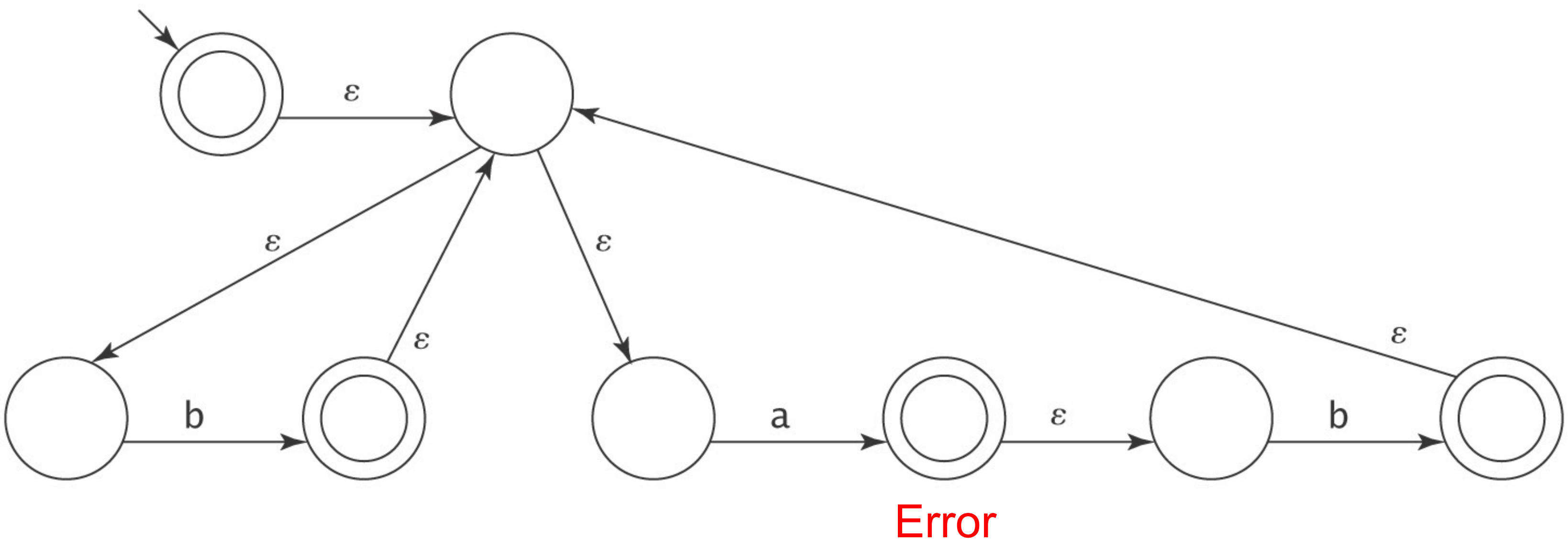
An FSM for (b $\cup$ ab):



An Example

(b ∪ ab)*

An FSM for (b ∪ ab)*:



The Algorithm *regextofsm*

regextofsm(α: regular expression) =

Beginning with the primitive subexpressions of α and working outwards until an FSM for all of α has been built do:

Construct an FSM as described above.

For Every FSM There is a Corresponding Regular Expression

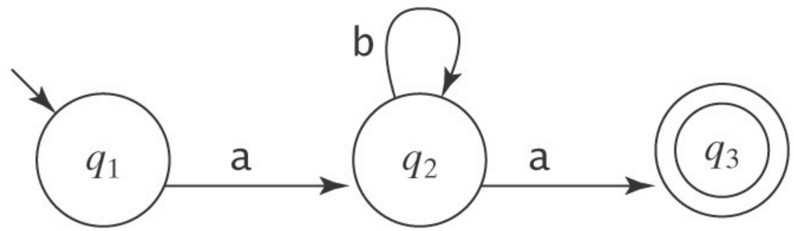
We' ll show this by construction.

The key idea is that we' ll allow arbitrary regular expressions to label the transitions of an FSM.

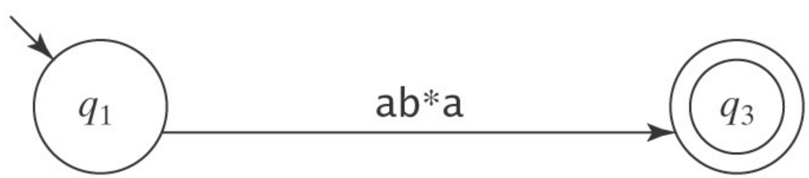
Read if interested …

A Simple Example

Let *M* be:



Suppose we rip out state 2:



The Algorithm *fsmtoregexheuristic*

fsmtoregexheuristic(*M*: FSM) =

1. Remove unreachable states from *M*.
2. If *M* has no accepting states then return ∅.
3. If the start state of *M* is part of a loop, create a new start state *s* and connect *s* to *M*' s start state via an ε-transition.
4. If there is more than one accepting state of *M* or there are any transitions out of any of them, create a new accepting state and connect each of *M*' s accepting states to it via an ε-transition. The old accepting states no longer accept.
5. If *M* has only one state then return ε.
6. Until only the start state and the accepting state remain do:
   6.1 Select *rip* (not *s* or an accepting state).
   6.2 Remove *rip* from *M*.
   6.3 *Modify the transitions among the remaining states so *M* accepts the same strings.
7. Return the regular expression that labels the one remaining transition from the start state to the accepting state.

Regular Expressions in Perl

Syntax	*Name*	*Description*
abc	Concatenation	Matches *a*, then *b*, then *c*, where *a*, *b*, and *c* are any regexs
a | *b* | *c*	Union (Or)	Matches *a* or *b* or *c*, where *a*, *b*, and *c* are any regexs
*a**	Kleene star	Matches 0 or more *a*' s, where *a* is any regex
a+	At least one	Matches 1 or more *a*' s, where *a* is any regex
a?		Matches 0 or 1 *a*' s, where *a* is any regex
a{*n*, *m*}	Replication	Matches at least *n* but no more than *m* *a*' s, where *a* is any regex
*a**?	Parsimonious	Turns off greedy matching so the shortest match is selected
a+?	"	"
.	Wild card	Matches any character except newline
^	Left anchor	Anchors the match to the beginning of a line or string
$	Right anchor	Anchors the match to the end of a line or string
`[a-z]`		Assuming a collating sequence, matches any single character in range
`[^a-z]`		Assuming a collating sequence, matches any single character not in range
`\d`	Digit	Matches any single digit, i.e., string in [`0-9`]
`\D`	Nondigit	Matches any single nondigit character, i.e., [`^0-9`]
`\w`	Alphanumeric	Matches any single "word" character, i.e., [`a-zA-Z0-9`]
`\W`	Nonalphanumeric	Matches any character in [`^a-zA-Z0-9`]
`\s`	White space	Matches any character in [space, tab, newline, etc.]

Regular Expressions in Perl

Syntax	*Name*	*Description*
\S	Nonwhite space	Matches any character not matched by \s
\n	Newline	Matches newline
\r	Return	Matches return
\t	Tab	Matches tab
\f	Formfeed	Matches formfeed
\b	Backspace	Matches backspace inside []
\b	Word boundary	Matches a word boundary outside []
\B	Nonword boundary	Matches a non-word boundary
\0	Null	Matches a null character
nnn	Octal	Matches an ASCII character with octal value *nnn*
\x*nn*	Hexadecimal	Matches an ASCII character with hexadecimal value *nn*
\c*X*	Control	Matches an ASCII control character
char	Quote	Matches *char*; used to quote symbols such as . and \
(*a*)	Store	Matches *a*, where *a* is any regex, and stores the matched string in the next variable
\1	Variable	Matches whatever the first parenthesized expression matched
\2		Matches whatever the second parenthesized expression matched
…		For all remaining variables

Testing. many other online tools

Using Regular Expressions in the Real World

Matching numbers:

-?([0-9]+(\.[0-9]*)?|\.[0-9]+)

Matching ip addresses:

[0-9]{1,3}(\.[0-9]{1,3}){3}

Trawl for email addresses:

```
\b[A-Za-z0-9_%-]+@[A-Za-z0-9_%-]+ (\.[A-Za-
  z]+){1,4}\b
```

From Friedl, J., Mastering Regular Expressions, O’Reilly,1997.

IE: information extraction, unstructured data management

A Biology Example – BLAST

Given a protein or DNA sequence, find others that are likely to be evolutionarily close to it.

ESGHDTTTYYNKNRYPAGWNNHHDQMFFWV

Build a DFSM that can examine thousands of other sequences and find those that match any of the selected patterns.

Simplifying Regular Expressions

Regex’s describe sets:

- Union is commutative: $\alpha \cup \beta = \beta \cup \alpha$.
- Union is associative: $(\alpha \cup \beta) \cup \gamma = \alpha \cup (\beta \cup \gamma)$.
- $\varnothing$ is the identity for union: $\alpha \cup \varnothing = \varnothing \cup \alpha = \alpha$.
- Union is idempotent: $\alpha \cup \alpha = \alpha$.

Concatenation:

- Concatenation is associative: $(\alpha\beta)\gamma = \alpha(\beta\gamma)$.
- ε is the identity for concatenation: $\alpha\,\varepsilon = \varepsilon\,\alpha = \alpha$.
- $\varnothing$ is a zero for concatenation: $\alpha\,\varnothing = \varnothing\,\alpha = \varnothing$.

Concatenation distributes over union:

- $(\alpha \cup \beta)\,\gamma = (\alpha\,\gamma) \cup (\beta\,\gamma)$.
- $\gamma\,(\alpha \cup \beta) = (\gamma\,\alpha) \cup (\gamma\,\beta)$.

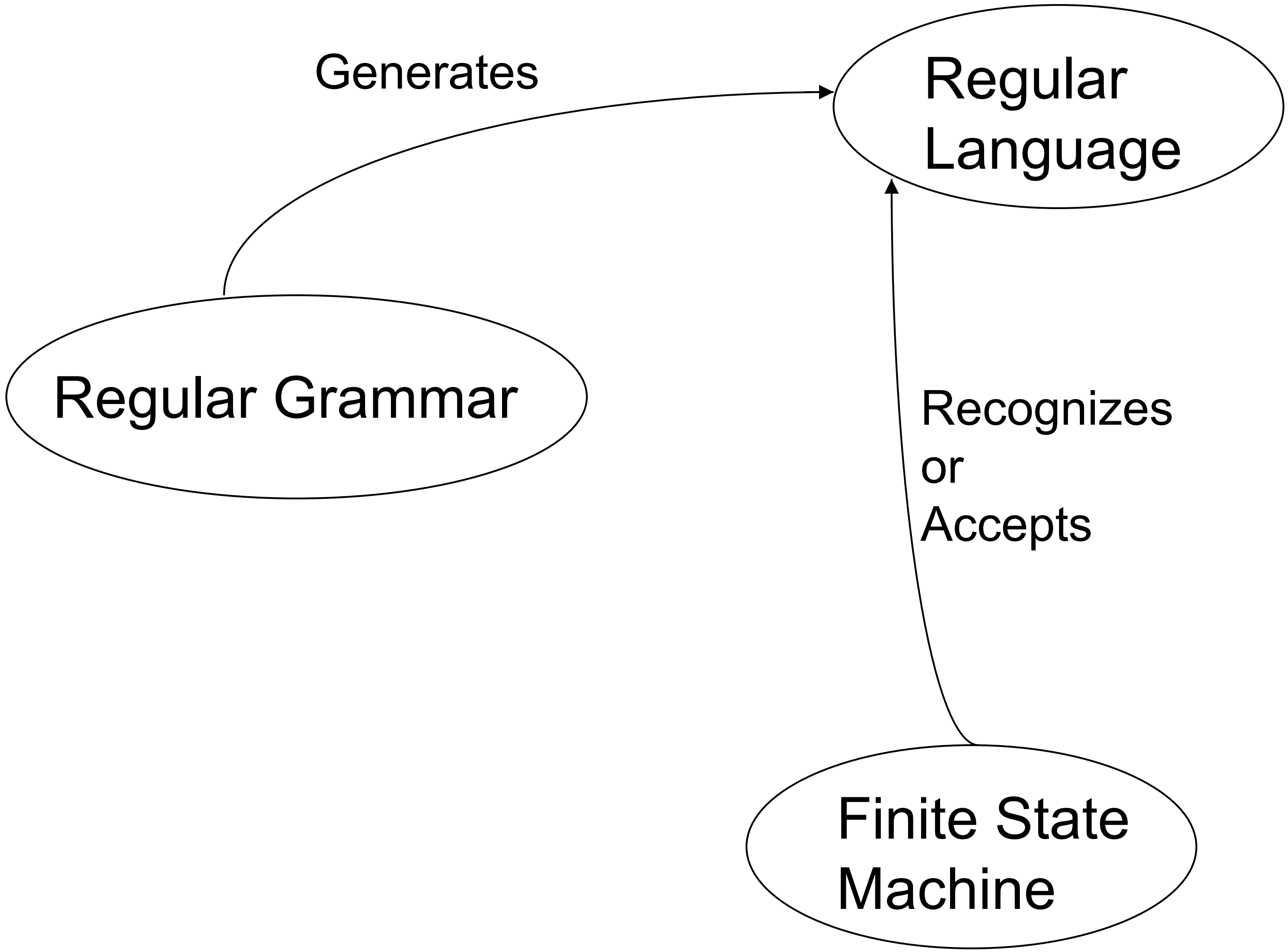
Kleene star:

- $\varnothing^* = \varepsilon$.
- $\varepsilon^* = \varepsilon$.
- $(\alpha^*)^* = \alpha^*$.
- $\alpha^*\alpha^* = \alpha^*$.
- $\alpha \cup \beta)^* = (\alpha^*\beta^*)^*$.

Regular Grammars

Chapter 7

Regular Languages



Regular Grammars

A regular grammar G is a quadruple (V, Σ, R, S), where:

- V (rule alphabet) contains nonterminals and terminals
  - terminals: symbols that can appear in strings generated by G
  - nonterminals: symbols that are used in the grammar but do not appear in strings of the language
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite set of rules of the form:

$$X \rightarrow Y$$

- S (the start symbol) is a nonterminal

Regular Grammars

In a regular grammar, all rules in R must:

- have a left hand side that is a single nonterminal
- have a right hand side that is:
  ε, or a single terminal, or a single terminal followed by a single nonterminal.

Legal: $S \rightarrow \mathtt{a}$, $S \rightarrow \varepsilon$, and $T \rightarrow \mathtt{a}S$

Not legal: $S \rightarrow \mathtt{a}S\mathtt{a}$ and $\mathtt{a}S\mathtt{a} \rightarrow T$

- Regular grammars must always produce strings one character at a time, moving left to right.

Regular Grammars

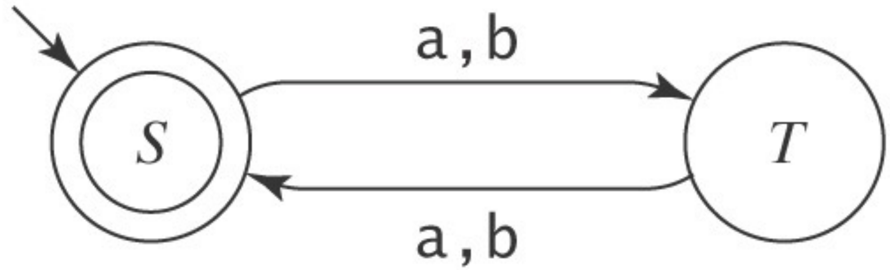
- The one we study is actually ***right regular grammar***.
  - Also called right linear grammar
  - Generates regular languages, recognized by FSM
  - Note FSM reads the input string *w* left to right

- ***Left regular grammar*** (left linear grammar)
  - $S \rightarrow \texttt{a}$, $S \rightarrow \varepsilon$, and $T \rightarrow S\texttt{a}$
  - Does it generate regular languages?

Regular Grammar Example

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : |w| \text{ is even}\}$ $((\mathtt{aa}) \cup (\mathtt{ab}) \cup (\mathtt{ba}) \cup (\mathtt{bb}))^*$

M:



G:

$S \rightarrow \varepsilon$
$S \rightarrow \mathtt{a}T$
$S \rightarrow \mathtt{b}T$
$T \rightarrow \mathtt{a}S$
$T \rightarrow \mathtt{b}S$

- By convention, the start symbol of any grammar G will be the symbol on the left-hand side of the first rule
- **Notice the clear correspondence between *M* and *G***
  - Given one, easy to derive the other

Regular Languages and Regular Grammars

Theorem: The class of languages that can be defined with regular grammars is exactly the regular languages.

Proof: By two constructions.

Regular Languages and Regular Grammars

Regular grammar* $\rightarrow$ *FSM:

grammartofsm($G = (V, \Sigma, R, S)$) =

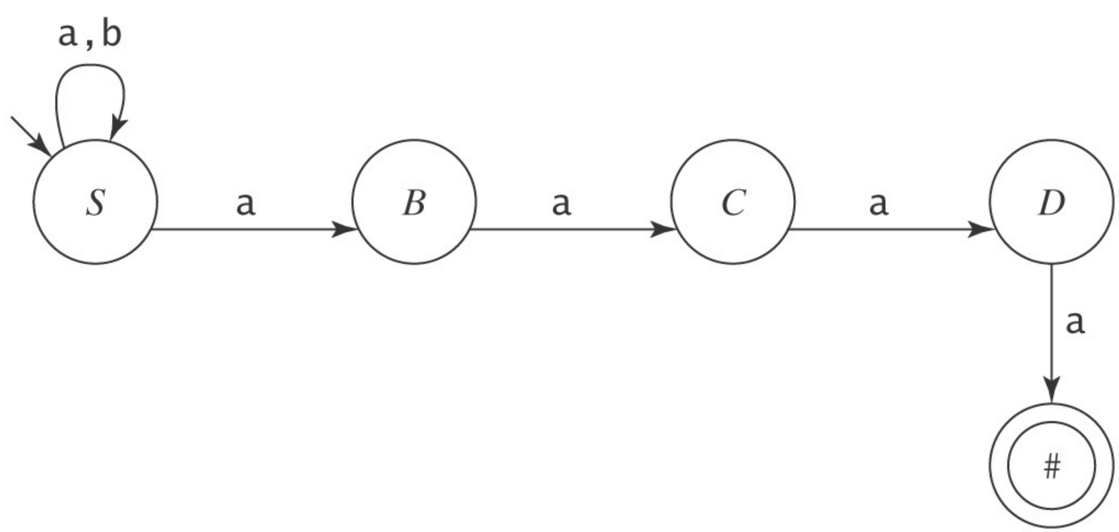
1. Create in M a separate state for each nonterminal in V.
2. Start state is the state corresponding to S .
3. If there are any rules in R of the form $X \rightarrow w$, for some $w \in \Sigma$, create a new state labeled #.
4. For each rule of the form $X \rightarrow w\ Y$, add a transition from X to Y labeled w.
5. For each rule of the form $X \rightarrow w$, add a transition from X to # labeled w.
6. For each rule of the form $X \rightarrow \varepsilon$, mark state X as accepting.
7. Mark state # as accepting.

FSM* $\rightarrow$ *Regular grammar: Similarly.

Strings That End with aaaa

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : w$ ends with the pattern $\mathtt{aaaa}\}$.

$$S \rightarrow \mathtt{a}S$$
$$S \rightarrow \mathtt{b}S$$
$$S \rightarrow \mathtt{a}B$$
$$B \rightarrow \mathtt{a}C$$
$$C \rightarrow \mathtt{a}D$$
$$D \rightarrow \mathtt{a}$$



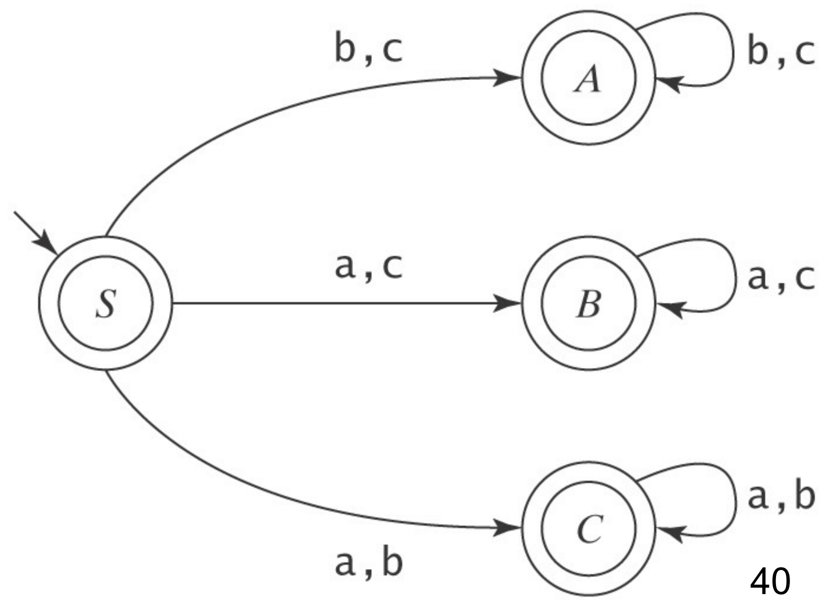
One Character Missing

$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$: there is a symbol in the alphabet not appearing in $w\}$.

$S \rightarrow \varepsilon$
$S \rightarrow \mathtt{a}B$
$S \rightarrow \mathtt{a}C$
$S \rightarrow \mathtt{b}A$
$S \rightarrow \mathtt{b}C$
$S \rightarrow \mathtt{c}A$
$S \rightarrow \mathtt{c}B$

$A \rightarrow \mathtt{b}A$
$A \rightarrow \mathtt{c}A$
$A \rightarrow \varepsilon$
$B \rightarrow \mathtt{a}B$
$B \rightarrow \mathtt{c}B$
$B \rightarrow \varepsilon$

$C \rightarrow \mathtt{a}C$
$C \rightarrow \mathtt{b}C$
$C \rightarrow \varepsilon$



How about ε transitions?