Harvard CS 121 and CSCI E-121

Lecture 5:
Regular Expressions

Harry Lewis

September 17, 2013

- **Reading:** Sipser, §1.3.

Is the Subset Construction Optimal?

The subset construction shows that any n-state NFA can be implemented as a 2^n-state DFA.

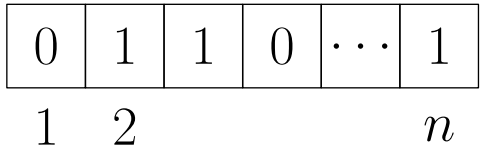
NFA States	DFA States
4	16
10	1024
100	2^{100}
1000	$2^{1000} \gg$ the number of particles in the universe

How to simulate this construction on the fly on an ordinary digital computer?

NFA states

$1, \ldots, n$

DFA state bit vector



Is this construction the best we can do?

Could there be a construction that always produces an n^2 state DFA for example?

Theorem: For every $n \geq 1$, there is a language L_n such that

1. There is an $(n+1)$-state NFA recognizing L_n.
2. There is no DFA recognizing L_n with fewer than 2^n states.

Conclusion: For finite automata, nondeterminism provides an *exponential savings* over determinism (in the worst case).

(The bound can be tightened.)

Proving that exponential blowup is sometimes unavoidable

(Could there be a construction that always produces an n^2 state DFA for example?)

Consider (for some fixed n=17, say)

$$L_n = \{w \in \{a,b\}^* : \text{the } n\text{th symbol from the right end of } w \text{ is an } a\}$$

- There is an $(n+1)$-state NFA that accepts L_n.
- There is no DFA that accepts L_n and has $< 2^n$ states

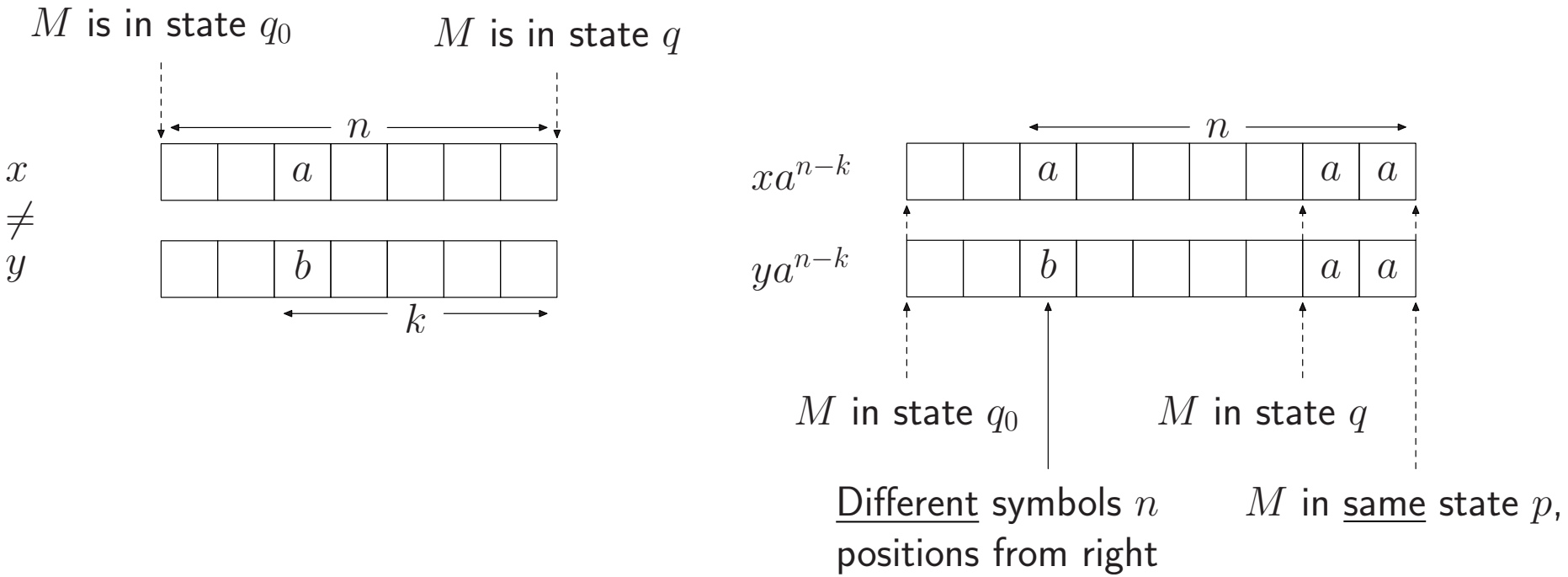
A "Fooling Argument"

- Suppose a DFA M has $< 2^n$ states, and $L(M) = L_n$
- There are 2^n strings of length n.
- By the pigeonhole principle, two such strings $x \neq y$ must drive M to the same state q.
- Suppose x and y differ at the k^{th} position from

  the right end (one has a, the other has b)

  ($k = 1, 2, \ldots,$ or n)
- M must treat xa^{n-k} and ya^{n-k} identically (accept both or reject both). These strings differ at position n from the right end.
- So $L(M) \neq L_n$, contradiction. QED.

Illustration of the fooling argument



- x and y are different strings

  (so there is a position k where one has a and the other has b)

- But both strings drive M from s to the same state q

What the argument proves

- This shows that the subset construction is within a factor of 2 of being optimal
- In fact it is optimal, i.e., as good as we can do in the *worst case*.
- Still, in many cases, the "generate-states-as-needed" method yields a DFA with $\ll 2^n$ states

  (e.g. if the NFA was deterministic to begin with!)

Regular Expressions

- Let $\Sigma = \{a, b\}$. The **regular expressions** over Σ are certain expressions formed using the symbols $\{a, b, (,), \varepsilon, \emptyset, \cup, \circ, {}^*\}$
- We use **red** for the strings under discussion (the **object language**) and **black** for the ordinary notation we are using for doing mathematics (the **metalanguage**).
- Construction Rules (= inductive/recursive definition):

1. a, b, ε, $\emptyset$ are regular expressions (of size 1)
2. If R_1 and R_2 are REs (of size s_1 and s_2), then $(R_1 \circ R_2)$, $(R_1 \cup R_2)$, and (R_1^*) are REs (of sizes $s_1 + s_2 + 3$, $s_1 + s_2 + 3$, and $s_1 + 3$, respectively).

- Examples:

$(a \circ b)$ $\qquad$ $((((a \circ (b^*)) \circ c) \cup ((b^*) \circ a))^*)$ $\qquad$ $(\emptyset^*)$

What REs Do

- Regular expressions (which are strings) represent languages (which are sets of strings), via the function L:

$$
\begin{array}{lrcl}
(1) & L(a) & = & \{a\} \\
(2) & L(b) & = & \{b\} \\
(3) & L(\varepsilon) & = & \{\varepsilon\} \\
(3) & L(\emptyset) & = & \emptyset \\
(4) & L((R_1 \circ R_2)) & = & L(R_1) \circ L(R_2) \\
(5) & L((R_1 \cup R_2)) & = & L(R_1) \cup L(R_2) \\
(6) & L((R_1^*)) & = & L(R_1)^*
\end{array}
$$

- Example:

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

- $L(\cdot)$ is called the **semantics** of the expression.

Syntactic Shorthand

- Omit many parentheses, because union and concatenation of languages are associative. For example,

  for any languages L_1, L_2, L_3:

  $$(L_1 L_2) L_3 = L_1 (L_2 L_3)$$

  and therefore for any regular expressions R_1, R_2, R_3,

  $$L\big((R_1 \circ (R_2 \circ R_3))\big) = L\big((R_1 \circ (R_2 \circ R_3))\big)$$

- Omit $\circ$ symbol

- Drop the distinction between red and black, between object language and metalanguage.

Semantic equivalence

The following are equivalent:

$((ab)c)$ $(a(bc))$ abc

or strictly speaking

$((a \circ b) \circ c)$ $(a \circ (b \circ c))$

- **Equivalent** means:

  "same semantics—same $L(\cdot)$-value—maybe different syntax"

More syntactic sugar

- By convention, $*$ takes precedence over $\circ$, which takes precedence over $\cup$.

  So $a \cup bc^*$ is equivalent to $(a \cup (b \circ (c^*)))$.

- Σ is shorthand for $a \cup b$ (or the analogous RE for whatever alphabet is in use).

Examples of Regular Languages

Strings ending in $a = \Sigma^* a$

Strings containing the substring $abaab =$?

Strings of even length $= (aa \cup ab \cup ba \cup bb)^*$

Strings with even # of a's $= (b \cup ab^*a)^*$
$= b^*(ab^*ab^*)^*$

Strings with $\leq$ two a's $=$?

Strings of form $x_1 x_2 \cdots x_k$, $k \geq 0$, each $x_i \in \{aab, aaba, aaa\}$ = ?

Decimal numerals, no leading zeroes
$= 0 \cup ((1 \cup \ldots \cup 9)(0 \cup \ldots \cup 9)^*)$

All strings with an even # of a's and an even # of b's
$= (b \cup ab^*a)^* \cap (a \cup ba^*b)^*$ but this isn't a regular expression

Equivalence of REs and FAs

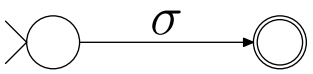
Recall: we call a language **regular** if there is a finite automaton that recognizes it.

Theorem: For every regular expression R, $L(R)$ is regular.

Proof:

Induct on the construction of regular expressions ("structural induction").

Base Case: R is a, b, ε, or $\emptyset$



accepts $\{\sigma\}$

accepts $\emptyset$

accepts $\{\varepsilon\}$

Equivalence of REs and FAs, continued

Inductive Step: If R_1 and R_2 are REs and $L(R_1)$ and $L(R_2)$ are regular (inductive hyp.), then so are:

$$\begin{aligned} L((R_1 \circ R_2)) &= L(R_1) \circ L(R_2) \\ L((R_1 \cup R_2)) &= L(R_1) \cup L(R_2) \\ L((R_1^*)) &= L(R_1)^* \end{aligned}$$

(By the closure properties of the regular languages).

Proof is constructive (actually produces the equivalent finite automaton, not just proves its existence).

Example Conversion of a RE to a FA

$$(a \cup \varepsilon)(aa \cup bb)^*$$

Converting Finite Automata to Regular Expressions

Theorem: For every regular language L, there is a regular expression R such that $L(R) = L$.

Proof:

Define generalized NFAs (GNFAs) (of interest only for this proof)

- Transitions labelled by regular expressions (rather than symbols).
- One start state q_{start} and only one accept state q_{accept}.
- Exactly one transition from q_i to q_j for every two states $q_i \neq q_{\text{accept}}$ and $q_j \neq q_{\text{start}}$ (including self-loops).

NFAs to GNFAs

Lemma: For every NFA N, there is an equivalent GNFA G.

- Add new start state, new accept state. Transitions?

- If multiple transitions between two states, combine. How?

- If no transition between two states, add one. With what label?

GNFAs to REs

Lemma: For every GNFA G, there is an equivalent RE R.

- By induction on the number of states k of G.
- Base case: $k = 2$. Set R to be the label of the transition from q_{start} to q_{accept}.
- Inductive Hypothesis: Suppose every GNFA G of k or fewer states has an equivalent RE (where $k \geq 2$).
- Induction Step: Given a $(k+1)$-state GNFA G, we will construct an equivalent k-state GNFA G'.

  Rip: Remove a state q_r (other than q_{start}, q_{accept}).

  Repair: Augment labels on all transitions $q_i \to q_j$ to also include strings that could have followed the transitions $q_i \to q_r \to q_j$.

Ripping and repairing GNFAs: details

Given a $(k+1)$-state GNFA G ($k \geq 2$), we construct an equivalent k-state GNFA G' as follows.

For any two (not necessarily distinct) states q_i, q_j, let R_{ij} be the regular expression labeling the transition $q_i \longrightarrow q_j$.

Rip: Remove a state q_r (other than q_{start}, q_{accept}).

Repair: For every two states q_i, q_j such that $q_i \notin \{q_{\text{accept}}, q_r\}$, $q_j \notin \{q_{\text{start}}, q_r\}$ simultaneously

put $R_{ij} \cup R_{i,r}R^*_{r,r}R_{r,j}$ on transition $q_i \longrightarrow q_j$.

Argue that $L(G') = L(G)$, which generated by a regular expression by IH.

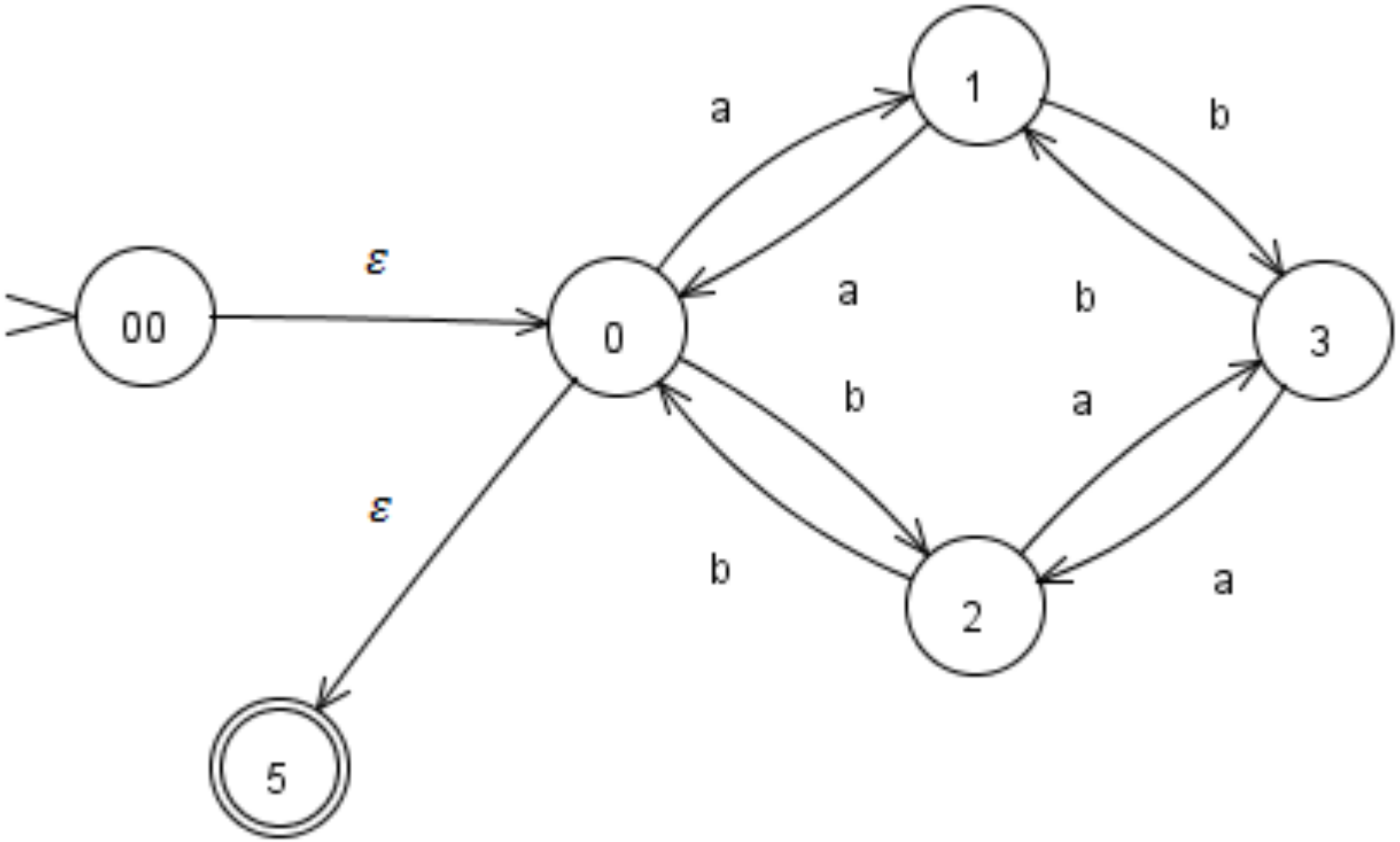
Example: The even as and even bs language

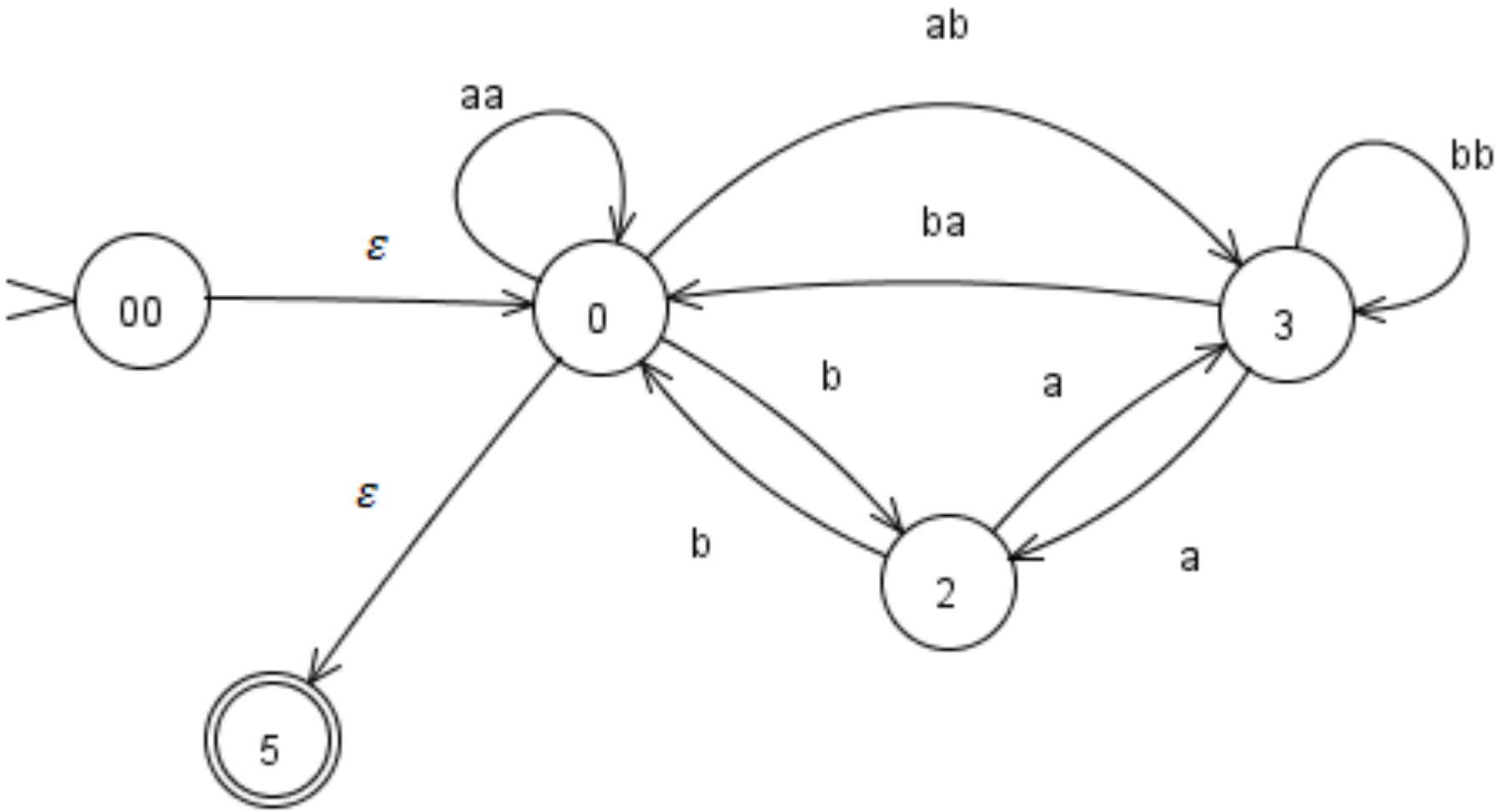
All strings with an even # of a's and an even # of b's

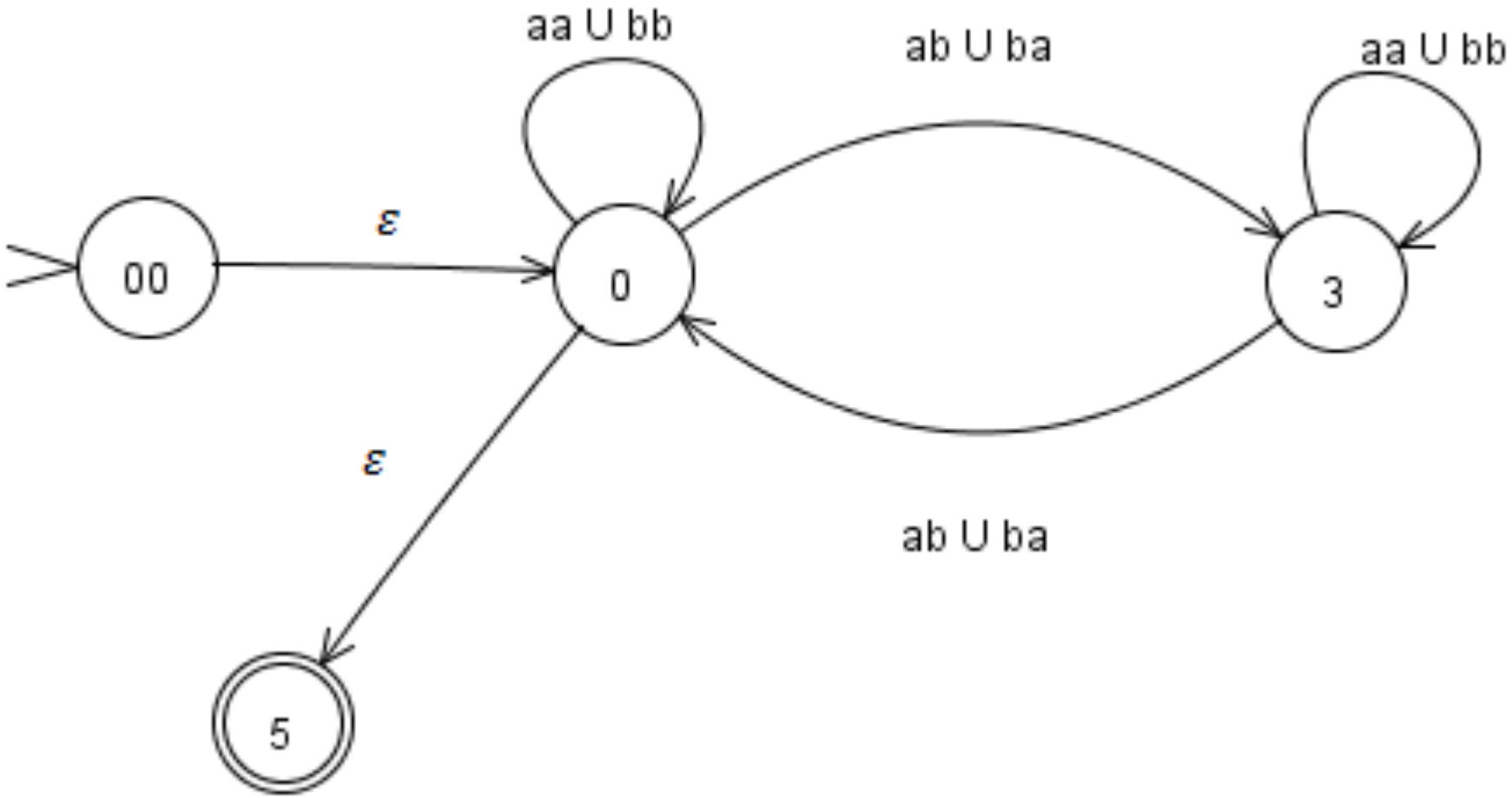
$= (b \cup ab^*a)^* \cap (a \cup ba^*b)^*$

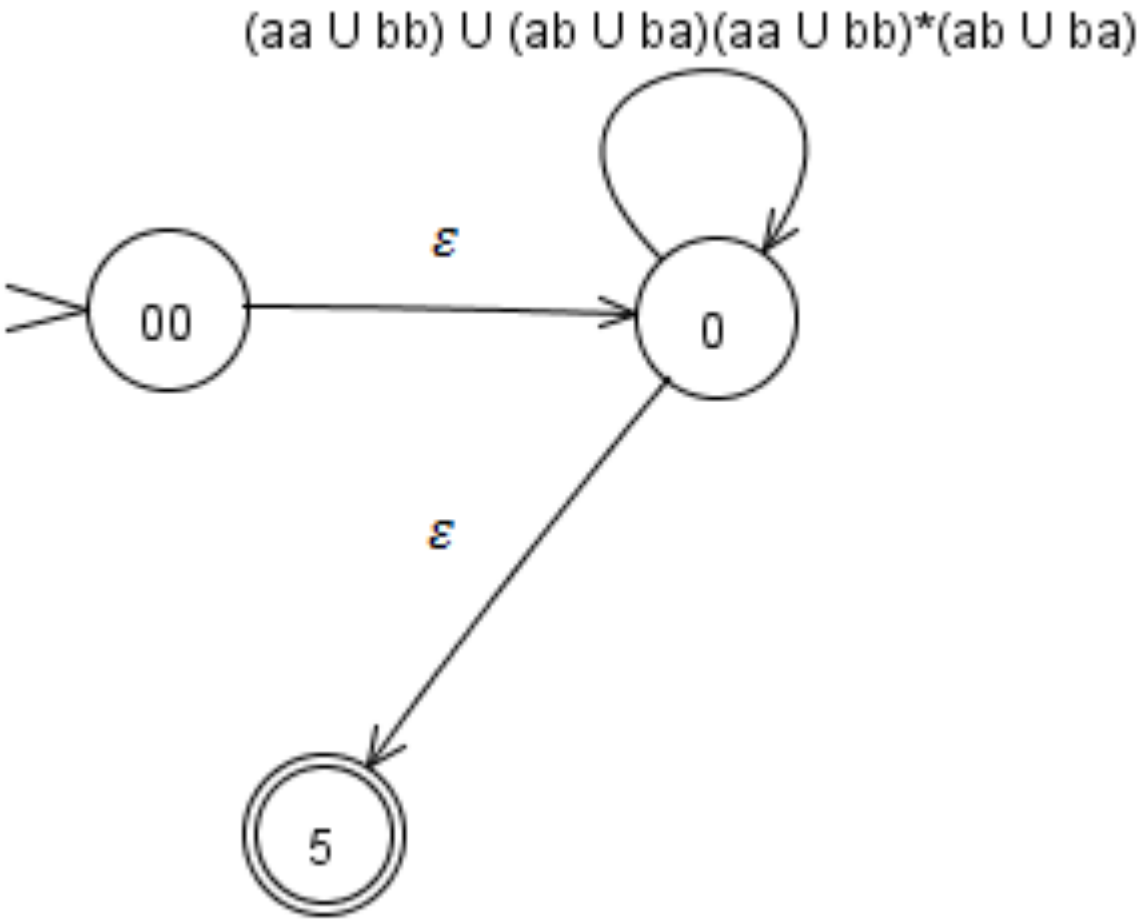
but this isn't a regular expression

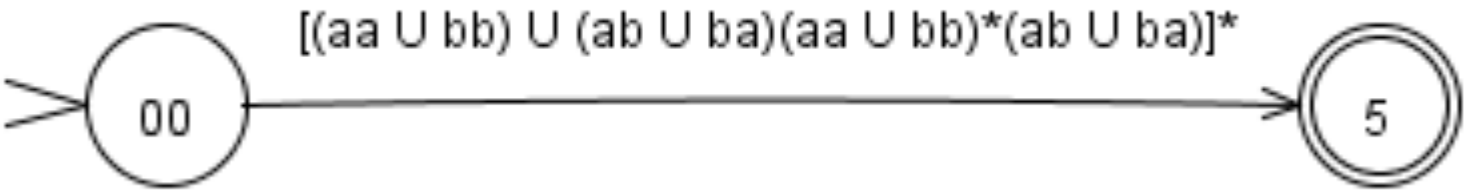
So let's build a DFA and convert it to a regular expression!


00
ε
0
a
1
b
a
b
3
b
a
ε
b
2
a
5


ab
aa
bb
ba
ε
00
0
3
b
a
ε
b
a
2
5


aa U bb
ab U ba
aa U bb
00
ε
0
3
ε
ab U ba
5


(aa U bb) U (ab U ba)(aa U bb)*(ab U ba)
00
ε
0
ε
5


[(aa U bb) U (ab U ba)(aa U bb)*(ab U ba)]*
00
5