A Subset of the URM Language; FA and NFA

Lecture #22; Nov. 30

The FA and NFA of Notes #9 and #10 provide finite descriptions of regular languages, since an FA/NFA M is finite (a graph, say) and a regular language is an $L(M)$ for some M.

The next section proposes *another type of finite description* of regular languages.

0.1. Regular Expressions

Regular expressions are familiar to users of the UNIX operating system.

They are *names* for regular sets as we will see.

- Do they name ALL regular sets, i.e., all sets of the type $L(M)$ where M is a FA (or NFA, equivalently)?

- Do they name any NON regular sets?

We will see that we must answer YES, NO.

Regular Expressions are more than "just names" as they *embody enough information* —as we will see— to be *mechanically transformable* into a NFA (and thus to a FA as well).

0.1.1 Definition. (Regular expressions over Σ) Given the *finite alphabet of atomic symbols* Σ, we form the *extended alphabet*

$$\Sigma \cup \{\emptyset, +, \cdot, *, (,)\} \tag{1}$$

where the symbols $\emptyset, +, \cdot, *, (,)$ (not including the comma separators) are all abstract or *formal** and *do not occur in* Σ. In particular, "$\emptyset$" in this alphabet is just a symbol — do NOT interpret it! (Yet!)

So are "+", "$\cdot$", "$*$" and the brackets. *All these symbols will be* interpreted *shortly*.

The set of *regular expressions over* Σ is *a set of strings over the augmented alphabet above*, given inductively by

Regular expressions are names, formed as strings over the alphabet (1) as follows :

(1) Every member of $\Sigma \cup \{\emptyset\}$ is a regular expression.

Examples for case (1): If $\Sigma = \{0, 1\}$ then $0, 1$, and $\emptyset$, all viewed as *abstract symbols* with no interpretation are each a regular expression.

(2) If α and β are *(names of) regular expressions*, then so is the string $(\alpha + \beta)$

(3) If α and β are *(names of) regular expressions*, then so is the string $(\alpha \cdot \beta)$

(4) If α is a *(name of) regular expression*, then so is the string (α^*)

*Employed to define form or structure.

The letters α, β, γ are used as *metavariables* (*syntactic variables*) in this definition. They will stand for *arbitrary regular expressions* (*we may add primes or subscripts to increase the number of our metavariables*). □

0.1.2 Remark.

(i) We emphasize that regular expressions are built starting from the *objects* contained in $\Sigma \cup \{\emptyset\}$.

We *also emphasize* that we have *NOT* talked about *semantics* yet, that is, we *did NOT say YET* what *sets* these expressions will *name*, nor, what "+, "·" and "*" *mean*.

(ii) We *will often omit the "dot"* in $(\alpha \cdot \beta)$ and write simply $(\alpha\beta)$.

(iii) We assign the *highest priority* to *, the *next lower* to $\cdot$ and the lowest to $+$.

We will let $\alpha \circ \alpha' \circ \alpha'' \circ \alpha'''$ group ("associate") from right to left, *for any* $\circ \in \{+, \cdot, ^*\}$.

Given these priorities, we may omit some brackets, as is usual.

Thus, $\alpha + \beta\gamma^*$ means $\Big(\alpha + \big(\beta(\gamma^*)\big)\Big)$

and $\alpha\beta\gamma$ means $(\alpha(\beta\gamma))$. □

We next define what sets these expressions name (semantics).

0.1.3 Definition. (Regular expression semantics) We define the *semantics* of any regular expression over Σ by recursion on the Definition 0.1.1.

We use the notation $L(\alpha)$ **to indicate *the set* named *by*** α.

(1) $L(\emptyset) = \emptyset$, where the left "$\emptyset$" is the symbol in the augmented alphabet (1) above, while the right "$\emptyset$" is the *name of the empty set in ordinary MATH*.

(2) $L(a) = \{a\}$, for each $a \in \Sigma$

(3) $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$

(4) $L(\alpha \cdot \beta) = L(\alpha)L(\beta)$ —where for two languages (sets of strings!) L and L', LL' —the *concatenation of the SETS* in this order— stands for $\{xy : x \in L \wedge y \in L'\}$.

(5) $L(\alpha^*) = \Big(L(\alpha)\Big)^{*}$[†] —where for any set S —finite or not— S^* denotes *the set of all strings*

$x_1x_2 \ldots x_n$, for $n \geq 0$, and where all (strings) $x_i \in S$

where $n = 0$ means that $x_1x_2 \ldots x_n = \lambda$.

Thus, in particular, we have always $\lambda \in S^*$.

□

[†]The $*$ in S^* is called the Kleene closure. So S^* is the Kleene closure of S.

0.1.4 Example. Let $\Sigma = \{0, 1\}$. Then $L\Big((0+1)^*\Big) = \Sigma^*$. Indeed, this is because $L\Big(0+1\Big) = L(0) \cup L(1) = \{0\} \cup \{1\} = \{0, 1\} = \Sigma$. □

0.1.5 Example. We note that $L(\emptyset^*) = \Big(L(\emptyset)\Big)^* = \emptyset^* = \{\lambda\}$.

Why so?

Because Σ^* is λ along with the set of all strings formed using symbols from Σ.

$\emptyset$ has no symbols to form strings with. So all we got is λ.

See last "red" comment in Def. 0.1.3.

Because of the above, we add "λ" as a *DEFINED NAME* —not in the original alphabet— for the set $\{\lambda\}$. □

Of course, two regular expressions α and β over the same alphabet Σ are equal, written $\alpha = \beta$, iff they are so *as strings*.

We also have another, *semantic*, concept of regular expression "equality":

0.1.6 Definition. (Regular expression equivalence) We say that two regular expressions α and β over the same alphabet Σ are *equivalent*, written $\alpha \sim \beta$, iff they *name the same set/language*, that is, iff $L(\alpha) = L(\beta)$.

□

0.1.7 Example. Let $\Sigma = \{0, 1\}$. Then $(0+1)^* \sim \big(0^*1^*\big)^*$.
Indeed, $L\big((0+1)^*\big) = \Sigma^*$, by 0.1.4.

So, if anything, we do have

$$L\Big((0+1)^*\Big) \supseteq L\Big((0^*1^*)^*\Big)$$

Now —for $L\Big((0+1)^*\Big) \subseteq L\Big((0^*1^*)^*\Big)$— the set

$$L\Big((\underbrace{0^*1^*}_{A})^*\Big)$$

is A^* where

$$A = L(0^*1^*) = \{0^n1^m : n \geq 0 \wedge m \geq 0\}$$

because

$$L(0^*) = L(0)^* = \{0\}^* = \{0^n : n \geq 0\}$$

and similarly for

$$L(1^*) = L(1)^* = \{1\}^* = \{1^m : m \geq 0\}$$

It should be clear that *any string of 0s and 1s can be built using as building blocks* 0^n1^m *judiciously choosing* n *and* m *values*.

E.g., $01^{10}0^{11}$ can be thought of as

$$0^11^0\,0^01^{10}\,0^{11}1^0$$

More generally, to show that an arbitrary string over Σ,

$$\ldots 0^k \ldots 1^r \ldots \tag{1}$$

is in A^* view (1) as

$$\ldots 0^k1^0 \ldots 0^01^r \ldots$$

But then the statement between the signs simply says that $\Sigma^* \subseteq L\Big((0^*1^*)^*\Big)$. Done. □

By the above example, $\alpha \sim \beta$ *does NOT imply* $\alpha = \beta$.

0.2. From a Regular Expression to NFA and Back

There is a *mechanical procedure* (*algorithm*), which from a given regular expression α *constructs* a NFA M so that $L(\alpha) = L(M)$, and conversely:

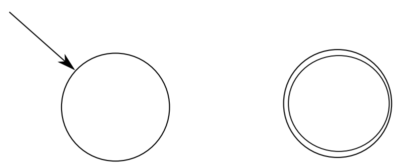
Given a NFA M constructs a regular expression α so that $L(\alpha) = L(M)$.

We split the procedure into two directions. *First, we go from regular expression to a NFA*.

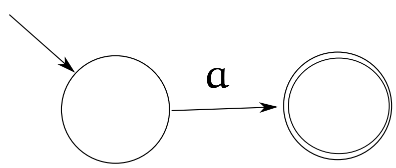
0.2.1 Theorem. (Kleene) *For any regular expression α over an alphabet Σ we can construct a NFA M with input alphabet Σ so that $L(\alpha) = L(M)$.*

Proof. Induction over the closure of Definition 0.1.1 — that is, on the formation of a regular expression α according to the said definition. For the basis we consider the cases

- $\alpha = \emptyset$; the NFA below works

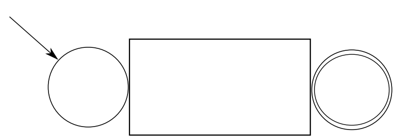


- $\alpha = a$, where $a \in \Sigma$; the NFA below works



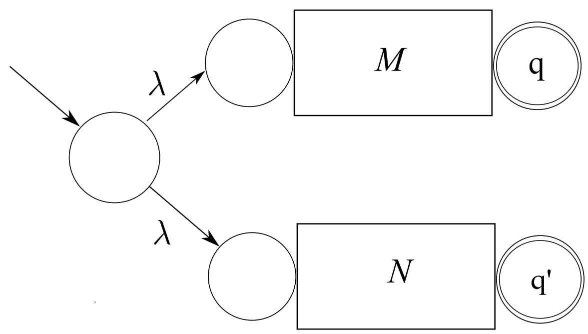
Both of the above NFA have EXACTLY ONE accepting state. Our construction maintains this property throughout.

That is, **all the NFA we construct in this proof will have that form**, namely

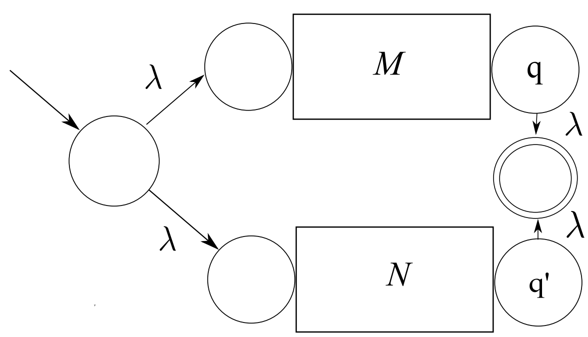


Assume now (*the I.H. on regular expressions!*) that we have built NFA for α and β —M and N— so that $L(\alpha) = L(M)$ and $L(\beta) = L(N)$. *Moreover, these M and N have the form above.* For the induction step we have three cases:

- To build a NFA for $\alpha + \beta$, that is, one that accepts the language $L(M) \cup L(N)$. The NFA below works since the accepting paths are precisely those from M and those from N.



However, to maintain the single accepting state form, we modify it as the NFA below.

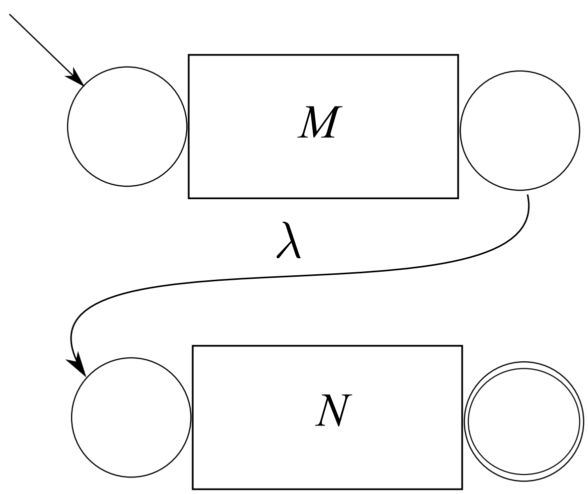


- To build a NFA for $\alpha\beta$, that is, one that accepts the language $L(M)L(N)$.

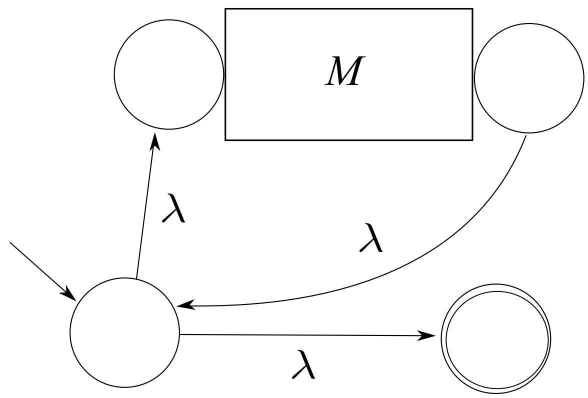
  The NFA below works —since *the accepting paths are precisely those formed by concatenating an* accepting path of M *(labeled by some $x \in L(M)$)* with *an λ-move and then with* an accepting path of N *(labeled by some $y \in L(N)$)*;

  in that left to right order.

  The λ that connects M and N will not affect the path name: $x\lambda y = xy$.



- To build a NFA for α^*, that is, one that accepts the language $L(M)^*$. The NFA below, that we call P, works. That is, $L(P) = L(M)^*$.



□

Lecture #23, Dec. 7

0.2.2 Theorem. (Kleene) *For any FA or NFA M with input alphabet Σ we can construct a regular expression α over Σ so that $L(\alpha) = L(M)$.*

Proof. Given a FA M (if a NFA is given, *then we convert it to a FA first*).

We will construct an α with the required properties. The idea is to express $L(M)$ in terms of simple to describe (indeed, regular themselves) sets of strings over Σ *by repeatedly using* the *operations* $\cdot$, $\cup$ *and Kleene star*, a finite number of times.

These regular sets —NAMEABLE by RegEXs— are called by Kleene "R^k_{ij}", where $k \le n$ and where the state set of the FA is

$$q_1, q_2, \ldots, q_n \text{ —the same "}n\text{" as above}$$

It turns out that "$\bigcup_j R^n_{1j}$" is the set of all FA-acceptable strings, the union taken *over all accepting q_j*.

So let $Q = \{q_1, q_2, \ldots, q_n\}$ be the set of states of M, where q_1 is the start state.† We will refer to the set of M's accepting states as F.

We next define several *sets* of strings (over Σ) —denoted by R^k_{ij}, for $k = 0, 1, \ldots, n$ and each i and j ranging from 1 to n.

$$R^k_{ij} = \{x \in \Sigma^* : x \text{ labels a path from } q_i \text{ to } q_j \text{ and every } q_m \text{ in this path, other than the endpoints } q_i \text{ and } q_j\text{, satisfies } m \leq k\} \tag{1}$$

A superscript of n removes the *restriction* on the path

$$q_i \overset{x}{\frown} q_j \tag{2}$$

since *every state q_m satisfies $m \leq n$*.

Thus R^n_{ij} contains ALL strings that name FA-paths from q_i to q_j —no restriction on where these paths pass through.

†We start numbering states from 1 rather than 0 for technical convenience; see the blue sentence at the top of next page.

We first note that for $k = 0$ we get very small finite sets.

Indeed, since state numbering starts at 1, the condition $m \leq 0$ is false and therefore in R^0_{ij} we have the cases:

- if we have $i \neq j$, then the condition (2) on p.17 can hold precisely when $x = a \in \Sigma$ for some a —since there can be no nodes in the interior of x.

  That is, we have precisely the case:

  $$\textcircled{q_i} \xrightarrow{a} \textcircled{q_j} \tag{†}$$

- The case $i = j$ also allows λ in the set, since we have ONE state:

  $$\textcircled{q_i = q_j} \tag{‡}$$

  In words, "*I can go from q_i to q_j DETERMINISTICALLY without consuming ANY input*".

To summarize, for all i and j we have

$$R^0_{ij} = \begin{cases} \{a \in \Sigma : \text{ Case } (\dagger)\} & \text{if } i \neq j \\ \{\lambda\} \cup \{a \in \Sigma : \text{ Case } (\dagger)\} & \text{if } i = j \end{cases} \tag{3}$$

Since every *finite* set of strings *can be named by a regular expression* (Exercise!),

there are RegEx: α^0_{ij} such that $L(\alpha^0_{ij}) = R^0_{ij}$, for all i, j (4)

For example, say $A = \{3, 5, 8, \lambda\}$. This is a finite set. It is NOT an alphabet (contains λ).

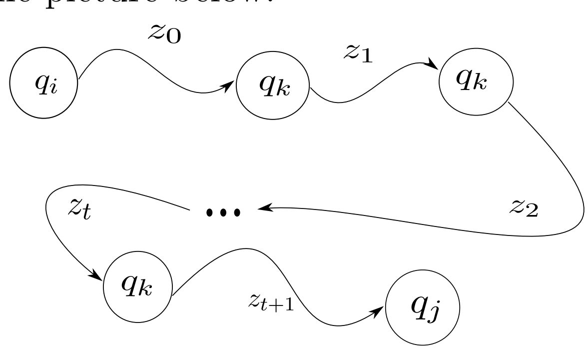
Then the RegEX $3+5+8+\lambda = 3+5+8+\emptyset^*$ NAMES A.

Why? Because $A = \{3\} \cup \{5\} \cup \{8\} \cup \{\lambda\}$.

Next note that the R_{ij}^k can be COMPUTED **recursively using k as the recursion variable** and i, j as parameters, and taking (3) as the basis of the recursion.

To see this, *consider a path labeled x in R_{ij}^k, for $k > 0$. It is possible that* all q_m *(other than q_i and q_j) that occur in the path have $m < k$. Then this x also belongs to R_{ij}^{k-1}.*

If on the other hand we DO have q_k appear in the interior of the path labeled x, one or more times, then we have the picture below.



where the q_k occurrences start immediately after the path named z_0 and are connected by paths named z_i, for $i = 1, \ldots, t$. Thus, $x = z_0 z_1 z_2 \ldots z_t z_{t+1}$. Noting that $z_0 \in R_{ik}^{k-1}$, $z_i \in R_{kk}^{k-1}$ —for $i = 1, \ldots, t$— and $z_{t+1} \in R_{kj}^{k-1}$, we have that $x \in R_{ik}^{k-1} \cdot \left(R_{kk}^{k-1}\right)^* \cdot R_{kj}^{k-1}$. We have established, for all $k \geq 1$ and all i, j, that

$$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1} \cdot \left(R_{kk}^{k-1}\right)^* \cdot R_{kj}^{k-1} \tag{4}$$

Explanation. Noting that

$$\left(R_{kk}^{k-1}\right)^* = \{\lambda\} \cup R_{kk}^{k-1} \cup R_{kk}^{k-1} R_{kk}^{k-1} \cup R_{kk}^{k-1} R_{kk}^{k-1} R_{kk}^{k-1} \cup R_{kk}^{k-1} R_{kk}^{k-1} R_{kk}^{k-1} R_{kk}^{k-1} \cup \ldots$$

the set of paths, from q_i to q_j depicted in the following part of (4):

$$R_{ik}^{k-1} \cdot \left(R_{kk}^{k-1}\right)^* \cdot R_{kj}^{k-1}$$

may contain

one interior q_k case corresponds to λ
two interior q_k case corresponds to R_{kk}^{k-1}
three interior q_k case corresponds to $R_{kk}^{k-1} R_{kk}^{k-1}$
four interior q_k case corresponds to $R_{kk}^{k-1} R_{kk}^{k-1} R_{kk}^{k-1}$
five interior q_k case corresponds to $R_{kk}^{k-1} R_{kk}^{k-1} R_{kk}^{k-1} R_{kk}^{k-1}$
etc.

Now take the I.H. that for $k-1 \geq 0$ (fixed!) and all values of i and j we have regular expressions α_{ij}^{k-1} such that $L(\alpha_{ij}^{k-1}) = R_{ij}^{k-1}$ —that is, α_{ij}^{k-1} NAMES the set R_{ij}^{k-1}.

We see that we can construct —from the α_{ij}^{k-1}— regular expressions α_{ij}^{k} for the R_{ij}^{k}.

Indeed, using the I.H. and (4), *we have the RegEX α_{ij}^{k} GIVEN, for all i, j and the fixed k, by*

$$\alpha_{ij}^{k} = \alpha_{ij}^{k-1} + \alpha_{ik}^{k-1}\left(\alpha_{kk}^{k-1}\right)^* \alpha_{kj}^{k-1} \tag{5}$$

Along with the basis (3) that the R_{ij}^{0} sets *CAN be named being finite*, this induction proves that *all* the R_{ij}^{k} can be named by regular expressions, which we may construct, from the basis up.

Finally, the set $L(M)$ can be so named. Indeed,

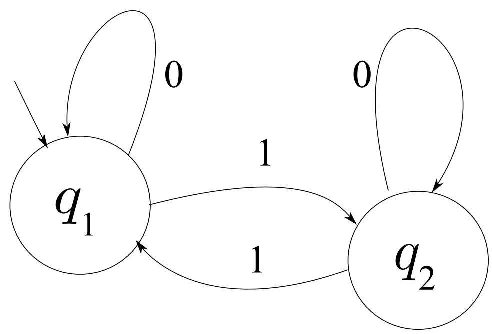
$$L(M) = \bigcup_{q_j \in F} R^n_{1j}$$

Therefore, as a RegEX:

$$\sum_{q_j \in F} \alpha^n_{1j} = \overbrace{\alpha^n_{1j_1} + \alpha^n_{1j_2} + \ldots + \alpha^n_{1j_m}}^{\textit{finitely many terms}}$$

The above is a finite union (F is finite!) of sets named by α^n_{1j} with $q_j \in F$. Thus we may construct its name as the "sum" (using "+", that is) of the names α^n_{1j} with $q_j \in F$. □

0.2.3 Example. Consider the FA below.



We will compute regular expressions for:

- all sets R^0_{ij}
- all sets R^1_{ij}
- all sets R^2_{ij}

Recall the definition of the R^k_{ij}, *here for* $k = 0, 1, 2$ *and* i, j *ranging in* $\{1, 2\}$ (cf. proof of 0.2.2):

$\{x : \textcircled{q_i} \overset{x}{\frown} \textcircled{q_j}$, where no state in this computation, other than possibly the *end-points* q_i and q_j, has index higher than $k\}$

This leads —as we saw— to the recurrence:

$$R^k_{ij} = R^{k-1}_{ij} \cup R^{k-1}_{ik}(R^{k-1}_{kk})^* R^{k-1}_{kj}$$

Below I employ the abbreviated (regular expression) *name* "λ" for $\emptyset^*$.

SET	RegEx
R^0_{11}	$\lambda + 0$
R^0_{12}	1
R^0_{21}	1
R^0_{22}	$\lambda + 0$

Superscript 1 now:

SET	RegEx: By Direct Substitution
$R^1_{11} = R^0_{11} \cup R^0_{11}(R^0_{11})^* R^0_{11}$	$\lambda + 0 + (\lambda + 0)(\lambda + 0)^*(\lambda + 0)$
$R^1_{12} = R^0_{12} \cup R^0_{11}(R^0_{11})^* R^0_{12}$	$1 + (\lambda + 0)(\lambda + 0)^* 1$
$R^1_{21} = R^0_{21} \cup R^0_{21}(R^0_{11})^* R^0_{11}$	$1 + 1(\lambda + 0)^*(\lambda + 0)$
$R^1_{22} = R^0_{22} \cup R^0_{21}(R^0_{11})^* R^0_{12}$	$\lambda + 0 + 1(\lambda + 0)^* 1$

Using the previous table, the reader will have no difficulty to fill in the regular expressions under the heading "RegEx: By Direct Substitution" in the next table.

To make things easier it is best to simplify the regular expressions of the previous table, meaning, finding simpler, equivalent ones. For example, $L\big(\lambda + 0 + (\lambda + 0)(\lambda + 0)^*(\lambda + 0)\big) = \{\lambda, 0\} \cup \{\lambda, 0\}\{\lambda, 0\}^*\{\lambda, 0\} = \{\lambda, 0\} \cup \{\lambda, 0\}\{\lambda, 0, 00, 000, \ldots\}\{\lambda, 0\} = \{0\}^*$, thus

$$\lambda + 0 + (\lambda + 0)(\lambda + 0)^*(\lambda + 0) \sim 0^*$$

Superscript 2:

SET	RegEx: By Direct Substitution
$R^2_{11} = R^1_{11} \cup R^1_{12}(R^1_{22})^* R^1_{21}$	
$R^2_{12} = R^1_{12} \cup R^1_{12}(R^1_{22})^* R^1_{22}$	
$R^2_{21} = R^0_{21} \cup R^0_{22}(R^1_{22})^* R^1_{21}$	
$R^2_{22} = R^1_{22} \cup R^1_{22}(R^1_{22})^* R^1_{22}$	

□

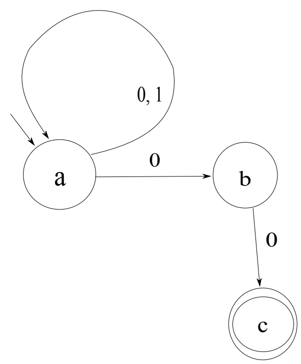
0.3. Another Example

0.3.1 Example. Let us show another NFA to FA conversion.

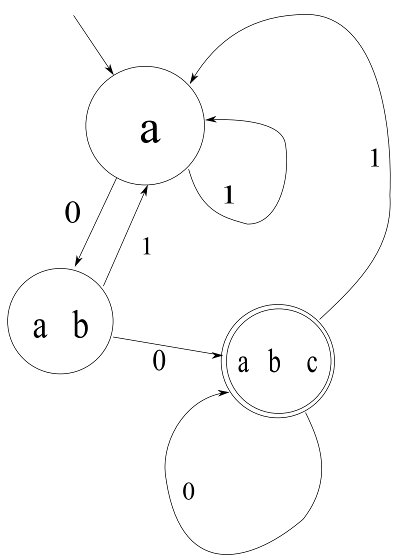
OK, given the following NFA which clearly decides the language over $\Sigma = \{0, 1\}$ given by the RegEx

$$(0 + 1)^*00$$

that is, the language containing ALL strings that end in two 0s.



The DETERMINISTIC FA equivalent to the above is the following:



□