Introduction to the Theory of Computation

XU Ming

School of Software Engineering, East China Normal University

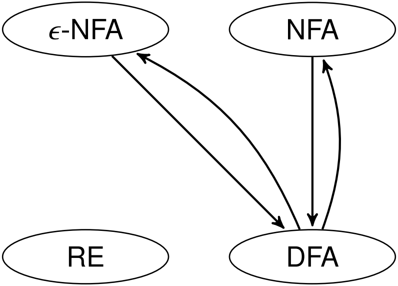
March 27, 2024

OUTLINE

- Finite Automata and Regular Expressions
- Regular Expressions in UNIX

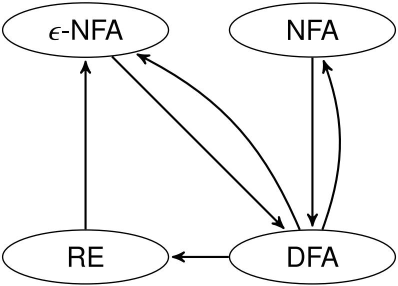
Finite Automata and Regular Expressions

We have already shown that DFA, NFA and ϵ-NFA all are equivalent. That is, they accept the same class of languages.



To show FA is equivalent RE we need to establish that

1. For every DFA A we can find a regular expression R such that $L(R) = L(A)$.
2. For every RE R there is an ϵ-NFA A such that $L(A) = L(R)$.



From DFA to RE

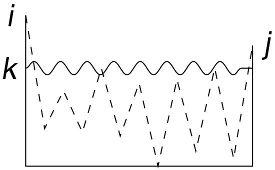
Theorem 3.1

For every DFA $A = (Q, \Sigma, \delta, q_0, F)$ *there is a regular expression* R *such that* $L(R) = L(A)$.

Let the states of A be $\{1, 2, \ldots, n\}$ for some integer n, with 1 being the start state. Let $R_{ij}^{(k)}$ be the regular expression describing the set of labels of all paths in A from state i to state j going through intermediate states $\{1, 2, \ldots, k\}$ only.

The Table Filling Technique

The figure suggests the requirement on the paths represented by $R_{ij}^{(k)}$.



$R_{ij}^{(k)}$ will be defined inductively. Eventually, $\sum_{j \in F} L(R_{1j}^{(n)}) = L(A)$.

Basis step: $k = 0$, i.e. no intermediate states.

- Case 1: $i \neq j$

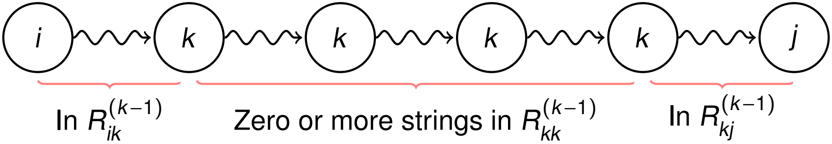
$$R_{ij}^{(0)} = \sum_{a \in \Sigma \text{ s.t. } \delta(i,a)=j} \mathbf{a}$$

- Case 2: $i = j$

$$R_{ij}^{(0)} = \epsilon + \sum_{a \in \Sigma \text{ s.t. } \delta(i,a)=j} \mathbf{a}$$

Inductive step:

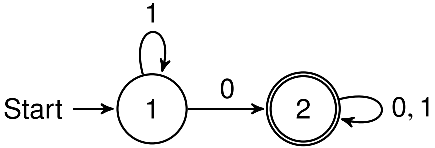
$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$$



□

Example

Let's find regular expression R for DFA A where



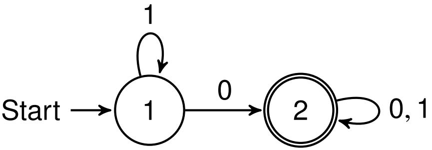
Clearly,

$$L(A) = \{x0y \mid x \in \{1\}^* \text{ and } y \in \{0,1\}^*\}.$$

Now, we use Theorem 3.1 to construct regular expression R.

The basis expression in the construction is

$R_{11}^{(0)}$	$\epsilon + \mathbf{1}$
$R_{12}^{(0)}$	$\mathbf{0}$
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\epsilon + \mathbf{0} + \mathbf{1}$



Computing expression $R_{ij}^{(1)}$

$R_{11}^{(0)}$	$\epsilon + \mathbf{1}$
$R_{12}^{(0)}$	$\mathbf{0}$
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\epsilon + \mathbf{0} + \mathbf{1}$

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)}(R_{11}^{(0)})^{*}R_{1j}^{(0)}$$

	By directive substitution	Simplified
$R_{11}^{(1)}$	$\epsilon + \mathbf{1} + (\epsilon + \mathbf{1})(\epsilon + \mathbf{1})^*(\epsilon + \mathbf{1})$	$\mathbf{1}^*$
$R_{12}^{(1)}$	$\mathbf{0} + (\epsilon + \mathbf{1})(\epsilon + \mathbf{1})^*\mathbf{0}$	$\mathbf{1}^*\mathbf{0}$
$R_{21}^{(1)}$	$\emptyset + \emptyset(\epsilon + \mathbf{1})^*(\epsilon + \mathbf{1})$	$\emptyset$
$R_{22}^{(1)}$	$\epsilon + \mathbf{0} + \mathbf{1} + \emptyset(\epsilon + \mathbf{1})^*\mathbf{0}$	$\epsilon + \mathbf{0} + \mathbf{1}$

For building the expression in induction part, we will need the following simplification rules.

- $(R^*)^* = R^*$ (Idempotence)
- $(\epsilon + R)^* = R^*$
- $R + RS^* = RS^*$ and $R + S^*R = S^*R$
- $\emptyset R = R\emptyset = \emptyset$ (Annihilation)
- $\emptyset + R = R + \emptyset = R$ (Identity)

Computing expression $R_{ij}^{(2)}$

$R_{11}^{(1)}$	$\mathbf{1}^*$
$R_{12}^{(1)}$	$\mathbf{1}^*\mathbf{0}$
$R_{21}^{(1)}$	$\emptyset$
$R_{22}^{(1)}$	$\epsilon + \mathbf{0} + \mathbf{1}$

$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)}(R_{22}^{(1)})^* R_{2j}^{(1)}$$

	By directive substitution	Simplified
$R_{11}^{(2)}$	$\mathbf{1}^* + \mathbf{1}^*\mathbf{0}(\epsilon + \mathbf{0} + \mathbf{1})^*\emptyset$	$\mathbf{1}^*$
$R_{12}^{(2)}$	$\mathbf{1}^*\mathbf{0} + \mathbf{1}^*\mathbf{0}(\epsilon + \mathbf{0} + \mathbf{1})^*(\epsilon + \mathbf{0} + \mathbf{1})$	$\mathbf{1}^*\mathbf{0}(\mathbf{0} + \mathbf{1})^*$
$R_{21}^{(2)}$	$\emptyset + (\epsilon + \mathbf{0} + \mathbf{1})(\epsilon + \mathbf{0} + \mathbf{1})^*\emptyset$	$\emptyset$
$R_{22}^{(2)}$	$\epsilon + \mathbf{0} + \mathbf{1} + (\epsilon + \mathbf{0} + \mathbf{1})(\epsilon + \mathbf{0} + \mathbf{1})^*(\epsilon + \mathbf{0} + \mathbf{1})$	$(\mathbf{0} + \mathbf{1})^*$

The final regular expression for A is $R = R_{12}^{(2)} = \mathbf{1}^*\mathbf{0}(\mathbf{0} + \mathbf{1})^*$.

Complexity

The method of last section for converting a DFA to a regular expression always works. But such construction is too expensive!

- There are n^3 expressions $R_{ij}^{(k)}$.
- Each inductive step grows the expression 4-fold, $R_{ij}^{(n)}$ has size 4^n.
- For any $i, j \in \{1, 2, \ldots, n\}$, $R_{ij}^{(k)}$ uses $R_{kk}^{(k-1)}$. So we have to write n^2 times $R_{kk}^{(k-1)}$.

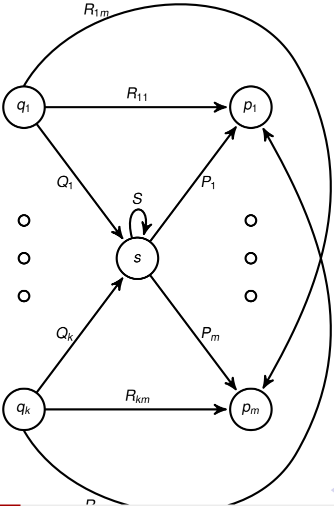
Overall, the complexity is given by

$$\underbrace{n^2 \cdot 1}_{\text{for constructing } R_{ij}^{(k)} \text{ when } k=0} + \underbrace{n^2 \cdot 4}_{\text{for } k=1} + \cdots + \underbrace{n^2 \cdot 4^n}_{\text{for } k=n} \in \Theta(n^2 \cdot 4^n).$$

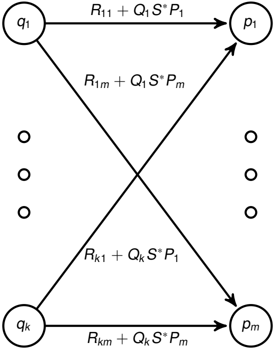
We need a more efficient approach.

The State Elimination Technique

Let's label the edges with regular expression instead symbols.



Now, let's eliminate state s.

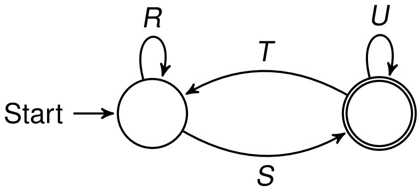


Such approach is called the state elimination technique.

The strategy for constructing regular expression from FA is as follows:

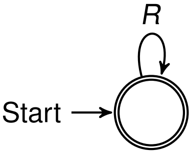
For each accepting state q, eliminate from the original automaton all states except q_0 and q. For each $q \in F$, we will be left with an A_q

- If A_q looks like



then the corresponding regular expression is $E_q = (R + SU^*T)^*SU^*$.

- If A_q looks like



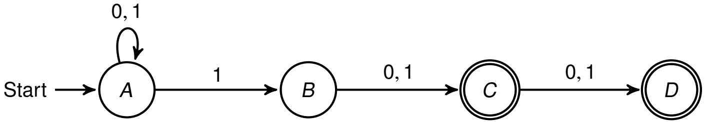
then the corresponding regular expression is $E_q = R^*$.

The final regular expression is

$$\sum_{q \in F} E_q$$

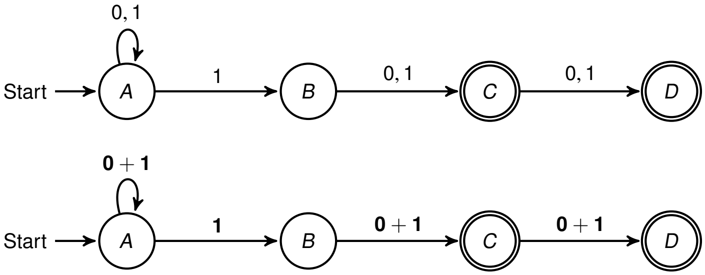
Example

Convert NFA N to regular expression by the state elimination technique.

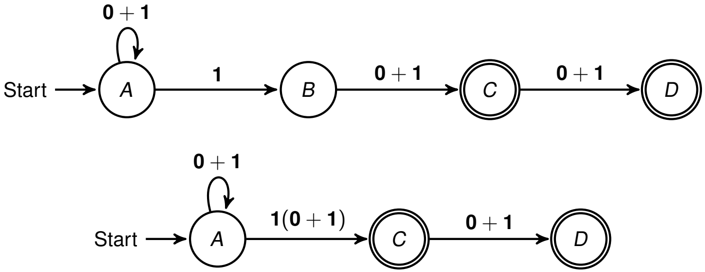


$$L(N) = \{w \mid w = x1b \text{ or } w = x1bc, x \in \{0, 1\}^*, b, c \in \{0, 1\}\}$$

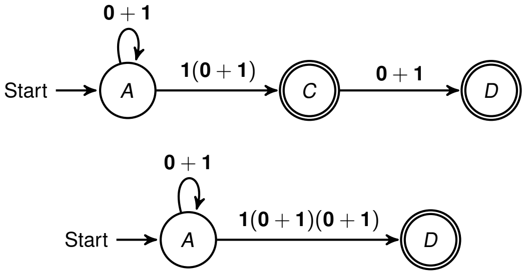
We turn this into an automaton with regular expression labels.



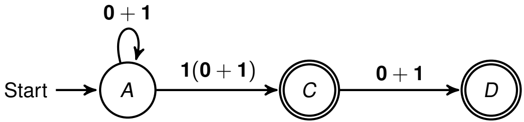
Let's elimінate state B.

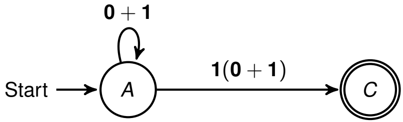


Then we eliminate state C and obtain N_D with regular expression $(\mathbf{0}+\mathbf{1})^*1(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})$.



Also, we can eliminate state D and obtain N_C with regular expression $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$.





The final expression is $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1}) + (\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})$.

Complexity

The complexity is boiled down to

$$\underbrace{(n-1)^2 \cdot 1}_{\text{for eliminating 1st state}} + \underbrace{(n-2)^2 \cdot 4}_{\text{for eliminating 2nd state}} + \cdots + \underbrace{1^2 \cdot 4^{n-1}}_{\text{for eliminating } (n-1)\text{st state}}$$

$$= \sum_{k=1}^{n} (n-k)^2 \cdot 4^{k-1}$$

$$= \sum_{k=1}^{n} [n^2 - (2n+1)k + k(k+1)] \cdot 4^{k-1}$$

$$= n^2 \cdot \left[\sum_{k=1}^{n} 4^{k-1}\right] - (2n+1) \cdot \left[\sum_{k=1}^{n} x^k\right]'_{x=4} + \left[\sum_{k=1}^{n} x^{k+1}\right]''_{x=4}$$

$$= n^2 \cdot \frac{4^n - 1}{3} - (2n+1) \cdot \left[\frac{x^{n+1} - x}{x-1}\right]'_{x=4} + \left[\frac{x^{n+2} - x^2}{x-1}\right]''_{x=4}$$

$$\in \Theta(4^n).$$

Complexity

The complexity is boiled down to

$$\underbrace{(n-1)^2\cdot 1}_{\text{for eliminating 1st state}} + \underbrace{(n-2)^2\cdot 4}_{\text{for eliminating 2nd state}} + \cdots + \underbrace{1^2\cdot 4^{n-1}}_{\text{for eliminating }(n-1)\text{st state}}$$

$$= \sum_{k=1}^{n}(n-k)^2\cdot 4^{k-1}$$

$$= \sum_{k=1}^{n}[n^2-(2n+1)k+k(k+1)]\cdot 4^{k-1}$$

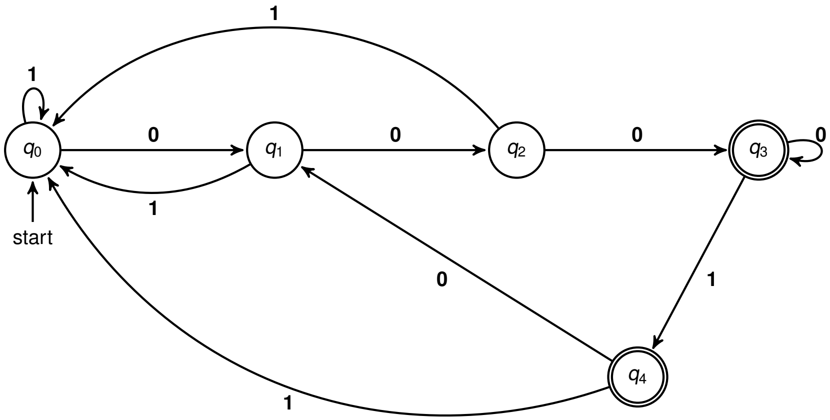
$$= n^2\cdot\left[\sum_{k=1}^{n}4^{k-1}\right]-(2n+1)\cdot\left[\sum_{k=1}^{n}x^k\right]'_{x=4}+\left[\sum_{k=1}^{n}x^{k+1}\right]''_{x=4}$$

$$= n^2\cdot\frac{4^n-1}{3}-(2n+1)\cdot\left[\frac{x^{n+1}-x}{x-1}\right]'_{x=4}+\left[\frac{x^{n+2}-x^2}{x-1}\right]''_{x=4}$$

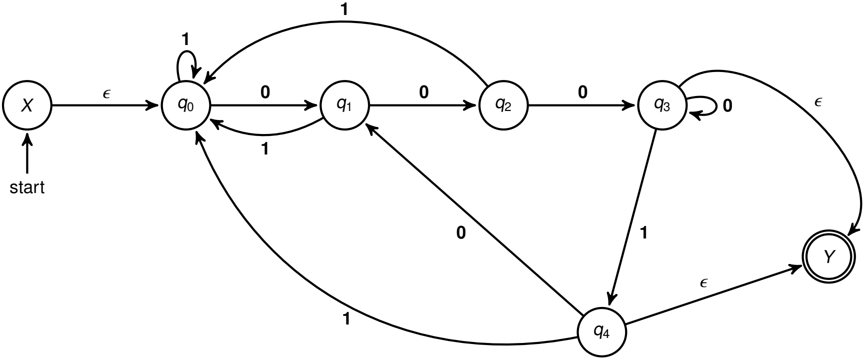
$$\in\Theta(4^n).$$

Example

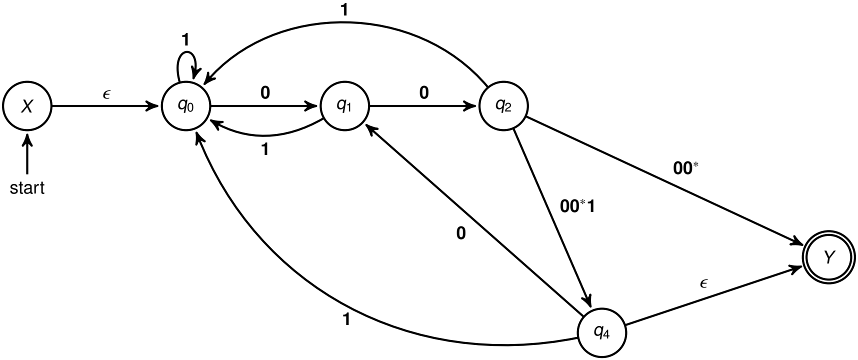
Construct regular expression from DFA A.



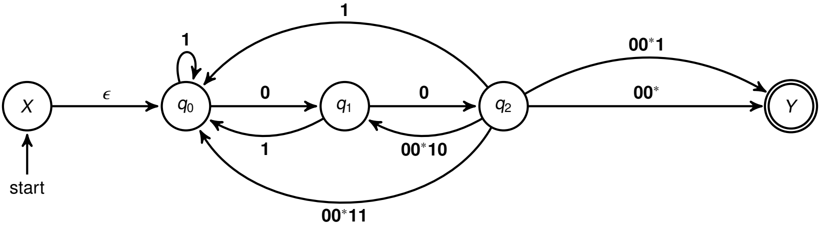
Label X and Y states.



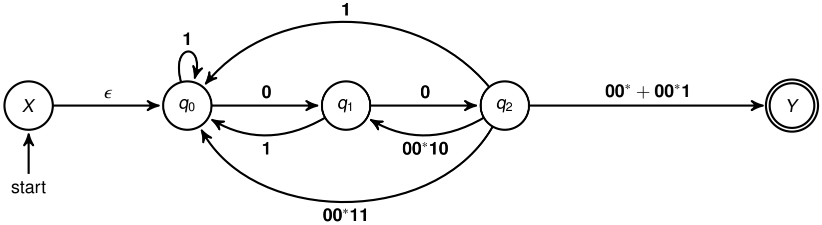
Eliminate state q_3.



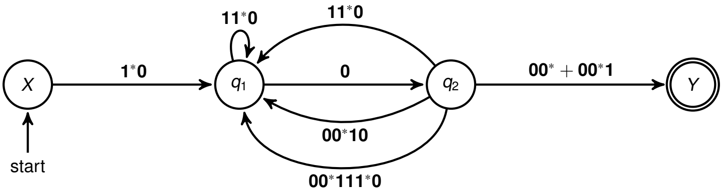
Eliminate state q_4.



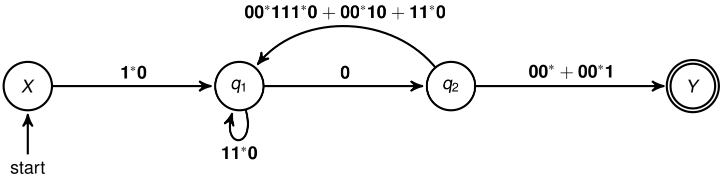
Combine.


1
1
start
X
ϵ
q_0
0
q_1
0
q_2
$\mathbf{00}^* + \mathbf{00}^*\mathbf{1}$
Y
1
$\mathbf{00}^*\mathbf{10}$
$\mathbf{00}^*\mathbf{11}$

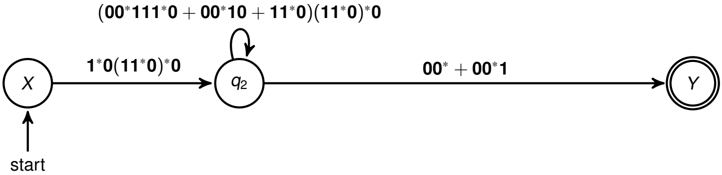
Eliminate state q_0.



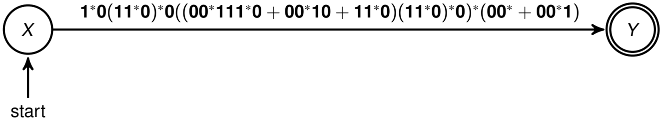
Combine.



Eliminate state q_1.



Eliminate state q_2.



The result is

$$\mathbf{1^*0(11^*0)^*0((00^*111^*0 + 00^*10 + 11^*0)(11^*0)^*0)^*(00^* + 00^*1)}$$

From RE to ϵ-NFA

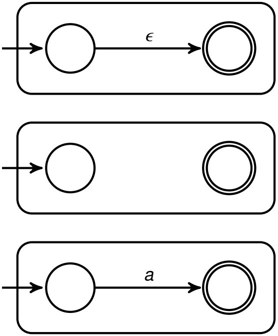
Theorem 3.2

For every regular expression R we can construct an ϵ-NFA A such that $L(R) = L(A)$.

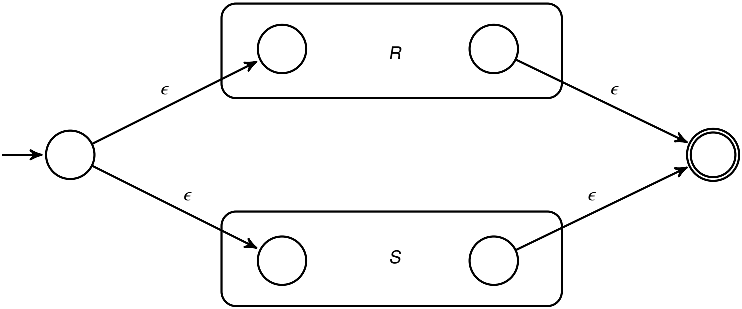
Proof The proof is by structural induction on R. We will construct the ϵ-NFA A with:

1. exactly one accepting state,
2. no arcs into the initial state,
3. no arcs out of the accepting state.

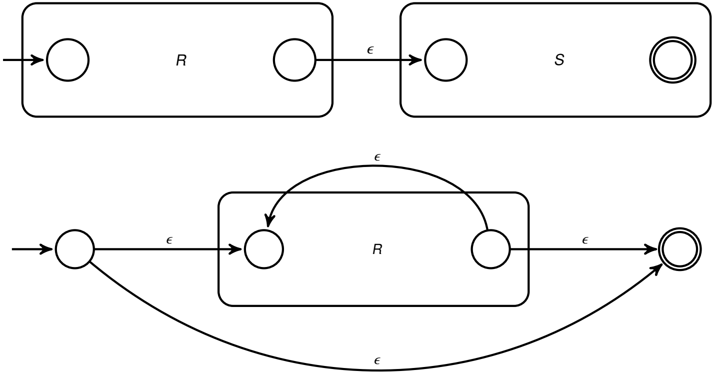
Basis step: Automata for $\epsilon, \emptyset, \mathbf{a}$.



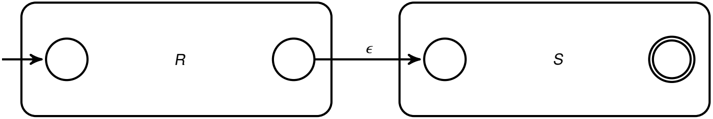
Inductive step: Automata for $R + S$.



Automata for $R.S, R^*$.



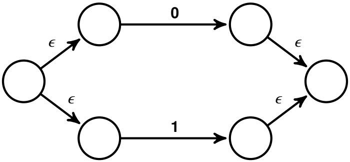
A formal proof for inductive step (b):

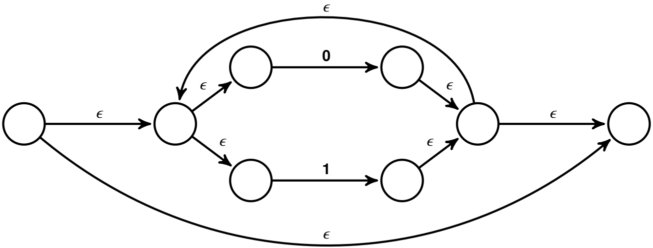


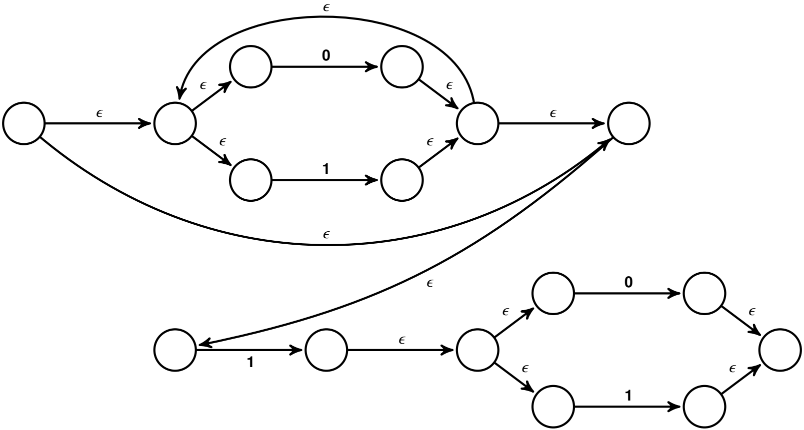
Let $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $L(R)$ and $L(S)$, respectively. Construct $A = (Q_1 \cup Q_2, \Sigma, \delta, q_1, F_2)$ recognize $L(R.S)$, where

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & q \in Q_1 \setminus F_1 \\ \{q_2\}, & q \in F_1 \wedge a = \epsilon \\ \delta_2(q, a), & q \in Q_2 \end{cases}$$

Convert $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$ to an ϵ-NFA.






ϵ
0
ϵ
ϵ
ϵ
ϵ
ϵ
ϵ
1
ϵ
ϵ
0
ϵ
ϵ
1
ϵ
ϵ
ϵ
1

Regular Expressions in UNIX

Regular Expressions in UNIX

We introduce the UNIX notation for extended regular expressions. This notation gives us a number of additional capabilities.
UNIX regular expressions allow one to write character classes to present ASCII characters as succinctly as possible. The rules for character classes are:

- The symbol · (dot) stands for any character.

- The sequence $[a_1 a_2 \cdots a_k]$ stands for the regular expression $\mathbf{a}_1 + \mathbf{a}_2 + \cdots + \mathbf{a}_k$.
- Between the square brace one can put a range of the form x-y to mean all the characters from x to y in the ASCII sequence. e.g., the digits can be expressed [0-9], the upper-case letters [A-Z].
- Some special notations for most common classes of characters are:

    [:digit:] for [0-9];
    [:alpha:] for [A-Za-z];
    [:alnum:] for [A-Za-z0-9].

There are several operators that are used in UNIX regular expressions.

- The operator | is used in place of $+$ to denote union.
- The operator ? means "zero or one of". Thus, $R?$ in UNIX is the same as $\epsilon + R$.
- The operator $+$ means "one or more of". Thus, $R+$ in UNIX is short for $R^+ = RR^*$.
- The operator $\{n\}$ means "n copies of". Thus, $R\{5\}$ in UNIX is short for $RRRRR$.

Homework

Exercise 3.2.8