Finite Automaton

A Finite Automaton is formally a tuple $A = \langle \Sigma, F, Q, \delta \rangle$ where Σ, F, Q are finite nonempty sets with $F \subset Q$. The *set of states* Q contains a special *initial state* ι. The *transition function* δ has type

$$\delta : Q \times \Sigma \longrightarrow Q$$

The interpretation of $\delta(q, s) = q'$ is that if A is in state q and receives input symbol s, then q' is the new state.

We assume the sets Q and Σ are disjoint. A *configuration* of A is a string xqy with $x, y \in \Sigma^*$, and $q \in Q$. Configuration xqy is interpreted as: A is in state q, the input already consumed is x, the remaining input is y, and the next input symbol is the left-most symbol of y. If C and C' are configurations, then $C \to C'$ if $C = xqsy$, $\delta(q, s) = q'$, and $C' = xsq'y$. A configuration xqy is *accepting* if $q \in F$ (elements of F are called *accept states*). A configuration xqy is *halting* if y is empty.

The *computation* of A on input $w \in \Sigma^*$ beginning from state q is the unique sequence $C_0, C_1, ...$ of configurations such that $C_0 = qw$, $C_i \to C_{i+1}$ for each i, and the sequence ends in a halting configuration. The *number of steps* in a computation is one less than the number of configurations. Automaton A therefore induces a function

$$\delta^* : Q \times \Sigma^* \longrightarrow Q$$

defined by $\delta^*(q, w) = q'$ where q' is the state in the halting configuration of the computation of A on input w beginning from state q. We say that A *accepts* w iff $\delta^*(\iota, w) \in F$. The *language* $\mathcal{L}(A)$ *accepted by* A is the set of all strings in Σ^* which A accepts.

A *nondeterministic* finite automaton is in some sense a generalization of a finite automaton; it has a transition function δ of type

$$\delta : Q \times \Sigma \longrightarrow 2^Q$$

The interpretation of $\delta(q, s) = S$ is that if A is in state q and receives input symbol s, then any element $q' \in S$ may be the new state. Accordingly, if C and C' are configurations, then $C \to C'$ if $C = xqsy$, $q' \in \delta(q, s)$, and $C' = xsq'y$. Moreover, a computation of A on input $w \in \Sigma^*$ beginning from state q is *any* sequence $C_0, C_1, ...$ of configurations such that $C_0 = qw$, $C_i \to C_{i+1}$ for each i, and the sequence ends in a halting configuration. Nondeterministic automaton A therefore induces a function

$$\delta^* : Q \times \Sigma^* \longrightarrow 2^Q$$

where $\delta^*(q, w)$ is the set of all q' such that q' is the state in the halting configuration of *any* computation of A on input w beginning from state q. We say that A accepts w iff $\delta^*(\iota, w)$ contains some element of F. The language $\mathcal{L}(A)$ accepted by A is the set of all strings in Σ^* which A accepts.

A nondeterministic automaton A with λ-*transitions* is in some sense a generalization of a nondeterministic finite automaton; it has a transition function δ of type

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \longrightarrow 2^Q$$

where $\lambda \notin Q \cup \Sigma$. The interpretation of $\delta(q, s) = S$ is that:

- If A is in state q and receives input symbol $s \in \Sigma$, then any element $q' \in S$ may be the new state. Accordingly, if C and C' are configurations, then $C \to C'$ if $C = xqsy$, $q' \in \delta(q, s)$, and $C' = xsq'y$.
- If A is in state q and $s = \lambda$, then any element $q' \in S$ may be the new state; this corresponds to a state transition which does not consume input. Accordingly, if C and C' are configurations, then $C \to C'$ if $C = xqy$, $q' \in \delta(q, \lambda)$, and $C' = xq'y$.

Moreover, a computation of A on input $w \in \Sigma^*$ beginning from state q is *any* sequence $C_0, C_1, ...$ of configurations such that $C_0 = qw$, $C_i \to C_{i+1}$ for each i, and the sequence ends in a halting configuration. Nondeterministic automaton A with λ-transitions therefore induces a function

$$\delta^* : Q \times \Sigma^* \longrightarrow 2^Q$$

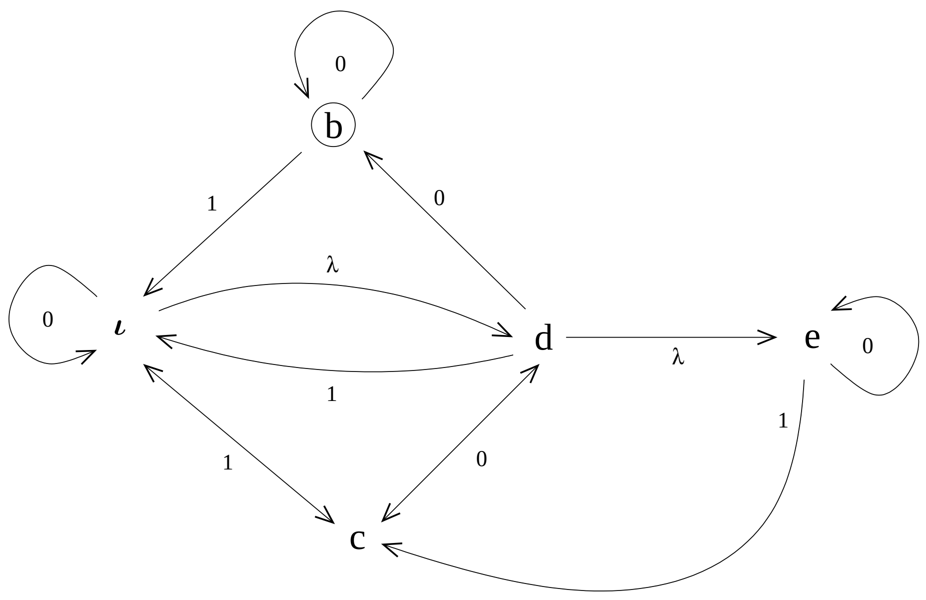
where $\delta^*(q, w)$ is the set of all q' such that q' is the state in the halting configuration of *any* computation of A on input w beginning from state q. We say that A accepts w iff $\delta^*(\iota, w)$ contains some element of F. The language $\mathcal{L}(A)$ accepted by A is the set of all strings in Σ^* which A accepts.

A nondeterministic automaton $A = \langle \Sigma, F, Q, \delta \rangle$ with λ-transitions is equivalent to some (deterministic) automaton A' in the sense that given A one can construct $A' = \langle \Sigma, F', Q', \delta' \rangle$ such that $\mathcal{L}(A') = \mathcal{L}(A)$. The following algorithm — which is described in terms of the graphical representation for automaton – implements the construction.

1. The initial state of A' is $\delta^*(\iota, \varepsilon)$, where ε is the empty string.
2. Repeat until no edges are missing:
   (a) Let v be a vertex (state) of A' that has no outgoing edge for some $s \in \Sigma$.
   (b) Let v' be a vertex (state) of A' defined by

   $$v' = \bigcup_{a \in v} \delta^*(a, s)$$

   (c) If not already present, add an s-labeled edge from v to v' (i.e., $\delta'(v, a) = v'$).
3. The accept states of A' are those that contain some element of F.



$$\iota \to d \to e$$

Initial state: $\{\iota, d, e\}$

$$\iota 0 \to 0\iota \to 0d \to 0e$$
$$\iota 0 \to d0 \to 0b$$
$$\iota 0 \to d0 \to 0c$$
$$\iota 0 \to d0 \to e0 \to 0e$$

$\{\iota, d, e\}0 \longrightarrow \{\iota, b, c, d, e\}$

$\therefore\ \{\iota, b, c, d, e\}0 \longrightarrow \{\iota, b, c, d, e\}$

$$\iota 1 \to 1c$$
$$\iota 1 \to d1 \to 1\iota \to 1d \to 1e$$
$$\iota 1 \to d1 \to e1 \to 1c$$

$\{\iota, d, e\}1 \longrightarrow \{\iota, c, d, e\}$

$\therefore\ \{\iota, c, d, e\}0 \longrightarrow \{\iota, b, c, d, e\}$

$$b1 \to 1\iota \to 1d \to 1e$$
$$c1 \to 1\iota \to 1d \to 1e$$

$\{\iota, b, c, d, e\}1 \longrightarrow \{\iota, c, d, e\}$

$\therefore\ \{\iota, c, d, e\}1 \longrightarrow \{\iota, c, d, e\}$

Final state: $\{\iota, b, c, d, e\}$

An *alphabet* Σ is a finite set. A *language* R over Σ is a subset $R \subset \Sigma^*$. The *empty string* λ has zero length and is the identity for concatenation (the concatenation rs of string r with s is their juxtaposition).

The product RS of languages R and S is

$$RS \;=\; \{rs \,|\, r \in R,\, s \in S\}$$

Note that if either R or S is empty, then $RS = \emptyset$. Language product is associative, but not commutative.

Given integer $n \geq 0$, the power R^n of language R is a language, defined recursively by

$$\begin{aligned} R^0 &= \{\lambda\} \\ R^{i+1} &= RR^i \end{aligned}$$

The *kleene closure* R^* of a language R is the language

$$R^* \;=\; \bigcup_{n \geq 0} R^n$$

The union of languages R and S is denoted by $R + S$.

A *regular expression* is defined inductively as follows:

- $\emptyset$ is a regular expression denoting the language $\emptyset$.
- $x \in \Sigma \cup \{\lambda\}$ is a regular expression, denoting the language $\{x\}$.
- Let x and y be regular expressions denoting the languages $\mathcal{L}(x)$ and $\mathcal{L}(y)$ respectively.
  - (xy) is a regular expression denoting the language $\mathcal{L}(x)\mathcal{L}(y)$.
  - $(x + y)$ is a regular expression denoting the language $\mathcal{L}(x) + \mathcal{L}(y)$.
  - (x^*) is a regular expression denoting the language $\mathcal{L}(x)^*$.

Given a language R over an alphabet Σ, define $\delta : R \longrightarrow \{\lambda, \emptyset\}$ by

$$\delta(R) \;=\; \begin{cases} \lambda & \text{if } \lambda \in R \\ \emptyset & \text{otherwise} \end{cases}$$

Note that

$$\begin{aligned} \delta(x) &= \emptyset \quad \text{for all } x \in \Sigma \\ \delta(\emptyset) &= \emptyset \\ \delta(\lambda) &= \lambda \\ \delta(R^*) &= \lambda \quad \text{for every language } R \\ \delta(RS) &= \delta(R)\,\delta(S) \\ \delta(R + S) &= \delta(R) + \delta(S) \quad \text{for all languages } R,\, S \end{aligned}$$

Given language R and sequence $s \in \Sigma^*$, the *derivative* of R with respect to s is

$$\mathcal{D}_s R \;=\; \{t \,|\, st \in R\}$$

Note that $s \in \Sigma^*$ is contained in a regular expression R if and only if $\lambda \in \mathcal{D}_s R$.

If R is regular and $s \in \Sigma$, then $\mathcal{D}_s R$ may be computed recursively by

$$\begin{array}{rcl}
\mathcal{D}_s(R^*) & = & (\mathcal{D}_s R)R^* \\
\mathcal{D}_s(RS) & = & (\mathcal{D}_s R)S + \delta(R)\mathcal{D}_s S \\
\mathcal{D}_s(R+S) & = & \mathcal{D}_s(R) + \mathcal{D}_s S \\
\mathcal{D}_s s & = & \lambda \\
\mathcal{D}_s a & = & \emptyset \quad \text{for } a = \lambda,\ a = \emptyset, \text{ or } a \in \Sigma \setminus \{s\}
\end{array}$$

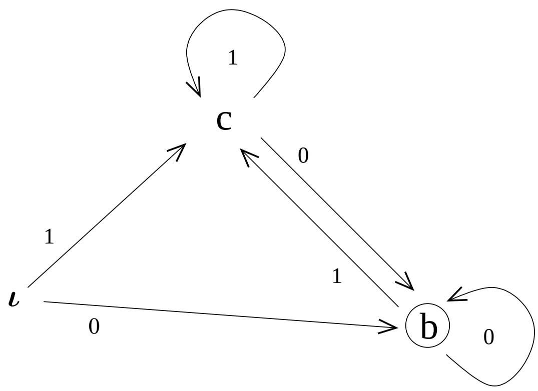
If R is regular and $s = s_1 \ldots s_{n+1} \in \Sigma^*$, then

$$\begin{array}{rcl}
\mathcal{D}_s R & = & \mathcal{D}_{s_{n+1}}(\mathcal{D}_{s_1 \ldots s_n} R) \\
\mathcal{D}_\lambda R & = & R
\end{array}$$

The language $\mathcal{L}(A)$ of a finite automaton A is regular (i.e., it is denoted by some regular expression). To obtain a regular expression R_ι denoting $\mathcal{L}(A)$,

1. Associate an equation R_q with each state q of A; if there is a transition from q to p on input a, then R_q contains the term aR_p. Moreover, λ is a term of R_q if and only if R_q is an accepting state.

2. Solve for R_ι, using the fact that S^*T is the solution to $X = SX + T$ if $\delta(S) = \emptyset$.



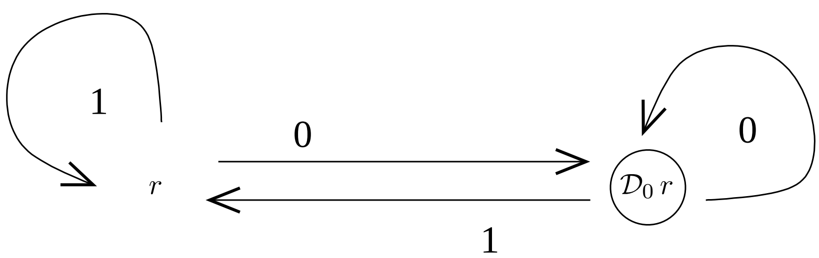
$$\begin{array}{rcl}
R_\iota & = & 1R_c + 0R_b \\
R_b & = & \lambda + 1R_c + 0R_b \\
R_c & = & 1R_c + 0R_b \quad \Longrightarrow \quad R_c = 1^*0R_b \\
\therefore R_b & = & \lambda + 11^*0R_b + 0R_b \quad \Longrightarrow \quad R_b = (11^*0 + 0)^*\lambda = (1^*0)^* \\
\therefore R_\iota & = & 11^*0(1^*0)^* + 0(1^*0)^* = (11^*0 + 0)(1^*0)^* = (1^*0)(1^*0)^*
\end{array}$$

Conversely, to obtain an automaton A from a regular expression r,

> Associate a state with each derivative $\mathcal{D}_s r$; if $\mathcal{D}_a(\mathcal{D}_s r) = \mathcal{D}_t r$ then state $\mathcal{D}_s r$ transitions to state $\mathcal{D}_t r$ on input a. A state $\mathcal{D}_s r$ is accepting if it contains λ.

Example

$$
\begin{aligned}
r &= 1^*0(1^*0)^* \\
\mathcal{D}_\lambda\, r &= r \\
\mathcal{D}_0\, r &= (\mathcal{D}_0\, 1^*0)(1^*0)^* + \delta(1^*0)\mathcal{D}_0(1^*0)^* \\
&= ((\mathcal{D}_0\, 1^*)0 + \delta(1^*)\mathcal{D}_0\, 0)(1^*0)^* \\
&= ((\mathcal{D}_0\, 1)1^*0 + \lambda)(1^*0)^* \\
&= (1^*0)^* \qquad\qquad \texttt{//contains } \lambda \\
\mathcal{D}_1\, r &= (\mathcal{D}_1\, 1^*0)(1^*0)^* + \delta(1^*0)\mathcal{D}_1\,(1^*0)^* \\
&= ((\mathcal{D}_1\, 1^*)0 + \delta(1^*)\mathcal{D}_1\, 0)(1^*0)^* \\
&= ((\mathcal{D}_1\, 1)1^*0)(1^*0)^* \\
&= r \\
\mathcal{D}_{00}\, r &= \mathcal{D}_0\,(\mathcal{D}_0\, r) \\
&= (\mathcal{D}_0\, 1^*0)(1^*0)^* \\
&= \mathcal{D}_0\, r \\
\mathcal{D}_{01}\, r &= \mathcal{D}_1\,(\mathcal{D}_0\, r) \\
&= (\mathcal{D}_1\, 1^*0)(1^*0)^* \\
&= r
\end{aligned}
$$



HOMEWORK: create an example of a nondeterministic automaton having three or four states, and:

- *Obtain an equivalent deterministic automaton using the algorithm on page 2 (an example is on page 3).*
- *Obtain a corresponding regular expression using the algorithm illustrated on pages 4 and 5.*
- *Beginning from the regular expression (obtained in the previous step), obtain an equivalent deterministic automaton using the algorithm illustrated on page 6.*

HOMEWORK: prove (any 5 of) the following simplification rules:

$$\begin{aligned}
\alpha &= \lambda\alpha \;=\; \alpha\lambda && (1)\\
(\alpha\beta)\gamma &= \alpha(\beta\gamma) && (2)\\
\alpha+\alpha &= \alpha && (3)\\
\alpha+\beta &= \beta+\alpha && (4)\\
(\alpha+\beta)(\gamma+\delta) &= \alpha\gamma+\alpha\delta+\beta\gamma+\beta\delta && (5)\\
\alpha\subset\alpha' \wedge \beta\subset\beta' &\implies \alpha\beta\subset\alpha'\beta' && (6)\\
\alpha\subset\beta &\implies \alpha^*\subset\beta^* && (7)\\
\alpha\beta\subset\beta \wedge \lambda\in\beta &\implies \alpha^*\subset\beta && (8)\\
\alpha^* &= (\alpha^*)^* && (9)\\
&= \alpha^*\alpha^* && (10)\\
&= \lambda+\alpha^+ && (11)\\
\alpha^+ &= \alpha\alpha^* && (12)\\
&= \alpha^*\alpha && (13)\\
(\alpha+\beta)^* &= (\alpha^*\beta^*)^* && (14)\\
&= (\beta^*\alpha^*)^* && (15)\\
&= (\beta^*\alpha)^*\beta^* && (16)\\
&= (\alpha^*\beta)^*\alpha^* && (17)\\
(\alpha+\beta)^*\alpha &= (\alpha^*\beta)^*\alpha^+ && (18)\\
&= (\beta^*\alpha)^+ && (19)
\end{aligned}$$

Hint:

To establish (8), induct on $n \geq 0$ to show $\alpha^n \subset \beta$.

To establish (17), show $(\alpha+\beta)^* \subset (\alpha^*\beta)^*\alpha^* \subset (\alpha+\beta)^*$
(consider using (8) for the first containment).

Note that (18) and (19) follow from (17) and (16) respectively.

Example:

$$\begin{aligned}
&1(0^*1)^*0^+ + 0^+ + 0^+1(0^*1)^*0^+ && \\
&1(1^*0)^+ + 0^+ + 0^+1(1^*0)^+ && \text{via } (\alpha^*\beta)^*\alpha^+ \to (\beta^*\alpha)^+\\
&0^+ + (\lambda+0^+)1(1^*0)^+ && \text{via } (4), (5)\\
&0^+ + 0^*1(1^*0)^+ && \text{via } \lambda+\alpha^+ \to \alpha^*\\
&0^+ + (0^*1)(0^*1)^*0^+ && \text{via } (2),\ (\beta^*\alpha)^+ \to (\alpha^*\beta)^*\alpha^+\\
&0^+ + (0^*1)^+0^+ && \text{via } (12)\\
&(\lambda+(0^*1)^+)0^+ && \text{via } (5)\\
&(0^*1)^*0^+ && \text{via } \lambda+\alpha^+ \to \alpha^*\\
&(1^*0)^+ && \text{via } (\alpha^*\beta)^*\alpha^+ \to (\beta^*\alpha)^+
\end{aligned}$$

Given automaton $A = \langle \Sigma, F, Q, \delta \rangle$ having n states, let $s = s_1 \dots s_n \in \Sigma^*$ and consider

$$\begin{aligned} f : \{p_0, \dots, p_n\} &\longrightarrow Q \\ x &\longmapsto \delta^*(\iota, x) \end{aligned}$$

where p_i is the length i prefix of s. Since the domain of f (the set of all prefixes of s) has greater cardinality than the range of f (the set of states of A), f cannot be injective; let $i < j$ be minimal such that

$$\delta^*(\iota, p_i) \;=\; \delta^*(\iota, p_j)$$

Therefore, if $x = p_i$, $y = s_{i+1} \dots s_j$, $z = s_{j+1} \dots s_n$ (where $z = \lambda$ if $j = n$), then

$$\begin{aligned} s &= xyz \\ |y| &> 0 \\ |xy| &\leq n \\ \delta^*(\iota, x) &= \delta^*(\delta^*(\iota, x), y) \end{aligned}$$

It follows that for any $i \in \mathbb{Z}^{\geq 0}$,

$$\delta^*(\iota, xz) \;=\; \delta^*(\delta^*(\iota, x), z) \;=\; \delta^*(\delta^*(\delta^*(\iota, x), y^i), z) \;=\; \delta^*(\iota, xy^i z)$$

The above is the *Pumping lemma for finite automaton.*

If R is an infinite regular language, then

$$\sum_{s \in R} [|s| \leq t] \;=\; \Omega(t)$$

> *Proof:* Let A be an automaton with n states such that $\mathcal{L}(A) = R$. Let $s \in R$ have length greater than n. Appealing to the the pumping lemma, $s = xyz$ for some x, y, z where $|y| > 0$ and for all $i \in \mathbb{Z}^{\geq 0}$,
>
> $$xy^i z \;\in\; R$$
>
> It follows that for all $i \in \mathbb{Z}^+$,
>
> $$i + 1 \;\leq\; \sum_{s \in R} [|s| \leq |x| + |z| + i|y|]$$
>
> Let $t > |s|$, and determine i by $|x| + |z| + i|y| \leq t < |x| + |z| + (i+1)|y|$. Then
>
> $$\begin{aligned} \sum_{s \in R} [|s| \leq t] &\geq i + 1 \\ &\geq \frac{t - |x| - |z|}{|y|} \\ &\geq t \, \frac{1 - (|s| - |y|)/|s|}{|y|} \end{aligned}$$

Let $\pi(x)$ denote the number of primes less than or equal to x. The *prime number theorem* is the result that

$$1 \;=\; \lim_{x\to\infty} \frac{\pi(x)}{x/\ln x}$$

Let $R = \{1^p \,:\, p \text{ is a prime number }\}$. Note that if R were regular, then

$$\pi(t) \;=\; \sum_{s\in R} [|s| \le t] \;=\; \Omega(t)$$

which leads to the contradiction

$$1 \;=\; \lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} \;\ge\; \lim_{x\to\infty} \frac{\Omega(x)}{x/\ln x} \;=\; \infty$$

Finite Automaton with I/O

A Finite Automaton with input/output is formally a tuple $A = \langle \mathcal{Q}, \Sigma, \mathcal{O}, \delta, \omega \rangle$ where $\mathcal{Q}$, Σ, $\mathcal{O}$ are finite nonempty sets; $\mathcal{Q}$ is the set of states, Σ is the input alphabet, and $\mathcal{O}$ is the output alphabet (we assume Q and Σ are disjoint). The *transition function* δ has type

$$\delta : Q \times \Sigma \longrightarrow Q$$

The interpretation of $\delta(q, s) = q'$ is that if A is in state q and receives input symbol s, then q' is the new state. The *output function* ω has type

$$\omega : Q \times \Sigma \longrightarrow \mathcal{O}$$

The interpretation of $\omega(q, s)$ is that if A is in state q and receives input symbol s, then $\omega(q, s)$ is output as the automaton transitions from state q to $\delta(q, s)$.

A Finite Automaton with input/output is often represented by a *state table*. For example, the following table

	δ		ω	
	0	1	0	1
q_1	q_6	q_3	0	0
q_2	q_3	q_1	0	0
q_3	q_2	q_4	0	0
q_4	q_7	q_4	0	0
q_5	q_6	q_7	0	0
q_6	q_5	q_2	1	0
q_7	q_4	q_1	0	0

indicates

$$\begin{aligned} Q &= \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\} \\ \Sigma &= \{0, 1\} \\ \mathcal{O} &= \{0, 1\} \end{aligned}$$

Moreover,

- $\delta(q_i, 0)$ is in the row labaled by q_i and column labeled (in the δ section) by 0.
- $\delta(q_i, 1)$ is in the row labaled by q_i and column labeled (in the δ section) by 1.
- $\omega(q_i, 0)$ is in the row labaled by q_i and column labeled (in the ω section) by 0.
- $\omega(q_i, 1)$ is in the row labaled by q_i and column labeled (in the ω section) by 1.

For example, $\delta(q_5, 1) = q_7$, and $\omega(q_6, 0) = 1$.

The following *minimization process* takes a finite state machine as input, and produces an equivalent machine — one having the same I/O behavior — which has a minimal number of states.

1. $k = 1$: determine *k-equivalent* states q, q' defined by

   $$q \sim_k q' \iff \forall x \in \Sigma^k \,.\, \omega^*(q, x) = \omega^*(q', x)$$

   where ω^* denotes the extension of ω from Σ to Σ^*,

   $$\begin{aligned} \omega^*(q,\, \lambda) &= \lambda \\ \omega^*(q,\, s_1 \ldots s_{n+1}) &= \omega^*(q,\, s_1 \ldots s_n)\, \omega(\delta^*(q,\, s_1 \ldots s_n), s_{n+1}) \\ \delta^*(q,\, \lambda) &= q \\ \delta^*(q,\, s_1 \ldots s_{n+1}) &= \delta(\delta^*(q,\, s_1 \ldots s_n), s_{n+1}) \end{aligned}$$

   Let P_k be the partition of Q corresponding to the equivalence classes of $\sim_k$.

2. determine *k+1-equivalent* states;

   $$q \sim_{k+1} q' \iff q \sim_k q' \ \wedge\ \forall s \in \Sigma \,.\, \delta(q, s) \sim_k \delta(q', s)$$

   Let P_{k+1} be the partition of Q corresponding to the equivalence classes of $\sim_{k+1}$.

3. If $P_{k+1} \neq P_k$, then increment k and goto step 2.

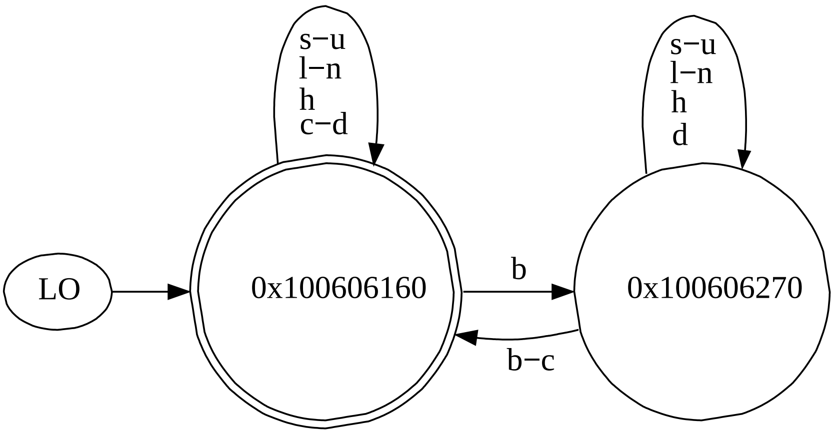
At termination ($P_{k+1} = P_k$) the desired result is obtained by restricting the automaton to a set of equivalence class representatives.

$$\begin{aligned} P_1 &= \{\{q_1, q_2, q_3, q_4, q_5, q_7\}, \{q_6\}\} \\ P_2 &= \{\{q_1, q_5\}, \{q_2, q_3, q_4, q_7\}, \{q_6\}\} \\ P_3 &= \{\{q_1, q_5\}, \{q_2, q_7\}, \{q_3, q_4\}, \{q_6\}\} \\ P_4 &= \{\{q_1\}, \{q_5\}, \{q_2, q_7\}, \{q_3, q_4\}, \{q_6\}\} \\ P_5 &= \{\{q_1\}, \{q_5\}, \{q_2, q_7\}, \{q_3, q_4\}, \{q_6\}\} \end{aligned}$$

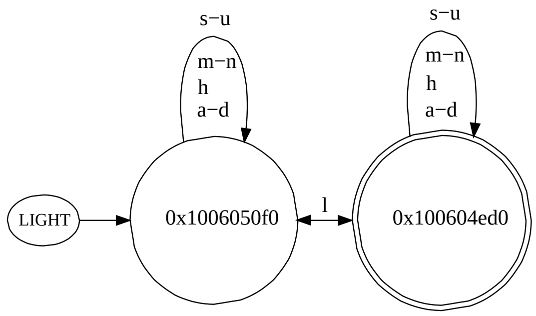
	ν	ω
	0 1	0 1
q_1	q_6 q_3	0 0
q_2	q_3 q_1	0 0
q_3	q_2 q_3	0 0
q_5	q_6 q_2	0 0
q_6	q_5 q_2	1 0

a	bake
b	broil
c	clear
d	down
h	hour
l	light
m	minute
n	on
o	off
s	start
t	temp
u	up

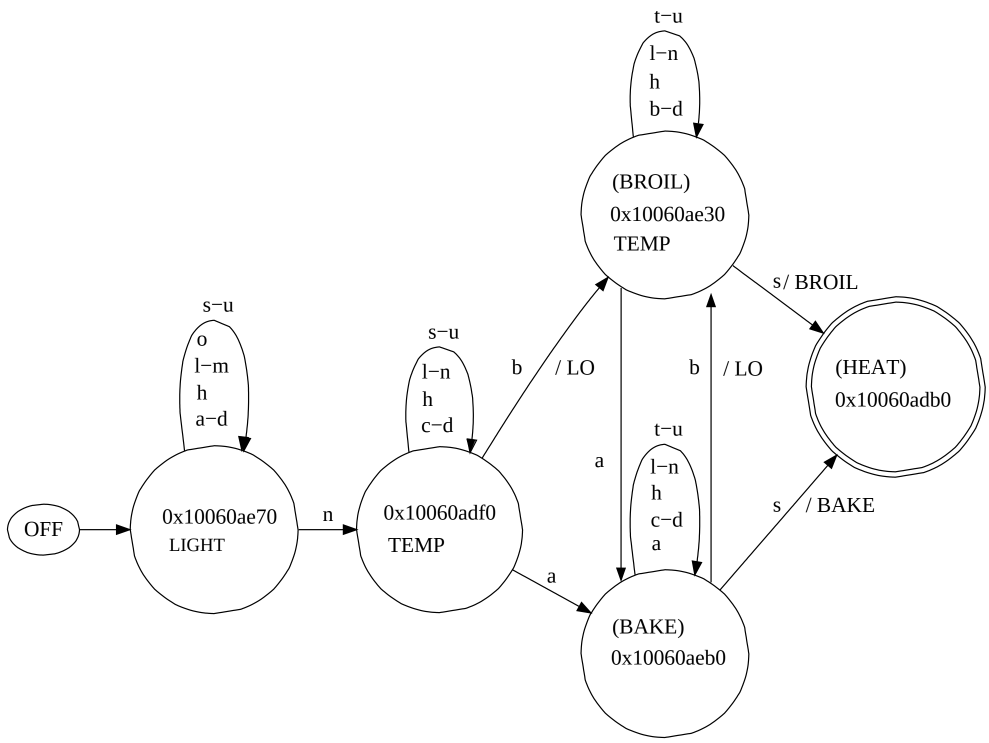
```
> even_b='[cdhlmnstu]*(b[dhlmnstu]*(c[cdhlmnstu]*|b[cdhlmnstu]*))*'

> ~/c/fa $even_b  '' 2>/dev/null
'c'  :  [cdhl-ns-u]*b([bc][cdhl-ns-u]*b|[dhl-ns-u])*[bc][cdhl-ns-u]*|[cdhl-ns-u]* >
        (b[dhl-ns-u]*[bc]|[cdhl-ns-u])(b[dhl-ns-u]*[bc]|[cdhl-ns-u])*
```



```
> no_light='[abcdhmnstu]*(l[abcdhmnstu]*l[abcdhmnstu]*)*'
> ~/c/fa '[abcdhlmnstu]*' $no_light  2>/dev/null
'al'  :  [a-dhl-ns-u]* > [a-dhmns-u]*l(l[a-dhmns-u]*l|
                          [a-dhmns-u])*l[a-dhmns-u]*|[a-dhmns-u]*
[a-dhmns-u]*l(l[a-dhmns-u]*l|[a-dhmns-u])*
```

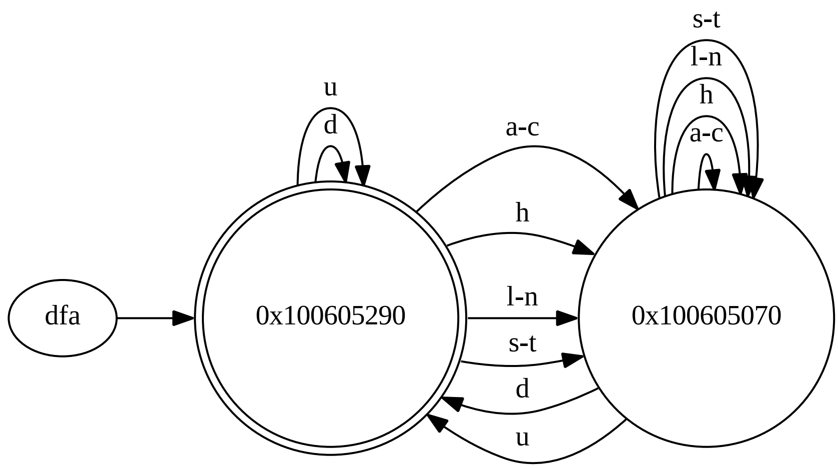


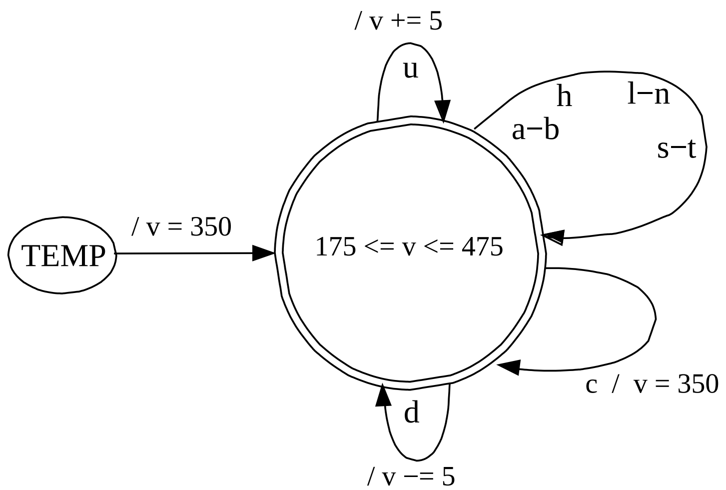

```
> on='[abcdhlmostu]*n'
> broil='[cdhlmnstu]*b[bcdhlmntu]*s'
> bake='[cdhlmnstu]*a[acdhlmntu]*s'
> ~/c/fa "$on($broil|$bake)" ''  2>/dev/null
'nbs'  :  [a-dhlmos-u]*n[cdhl-ns-u]*(b[b-dhl-ntu]*s|a[acdhl-ntu]*s) ><   :  ''
(null)
```

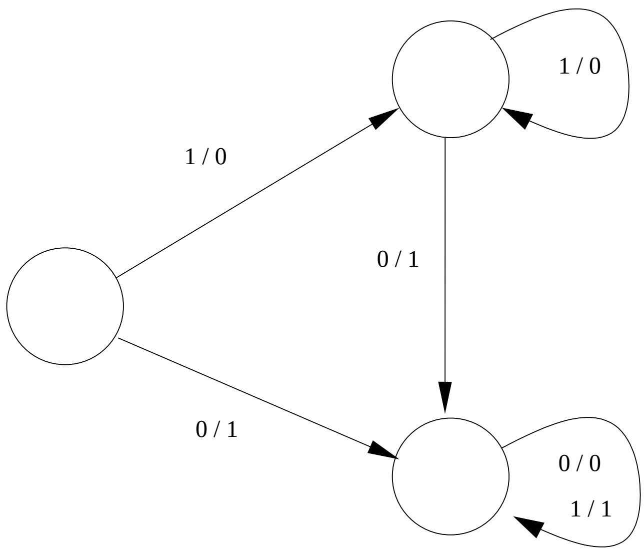


(add self-loops to 0x10060adb0 and transition — on input c — to 0x10060adf0)

```
> temp='([abchlmnst]*(d|u))*'
> ~/c/fa $temp ''  2>/dev/null
'd'  :  [du]*[a-chl-nst]([du][du]*[a-chl-nst]|[a-chl-nst])*[du][du]*|[du]* >
        ([a-chl-nst][a-chl-nst]*[du]|[du])([a-chl-nst][a-chl-nst]*[du]|[du])*
```

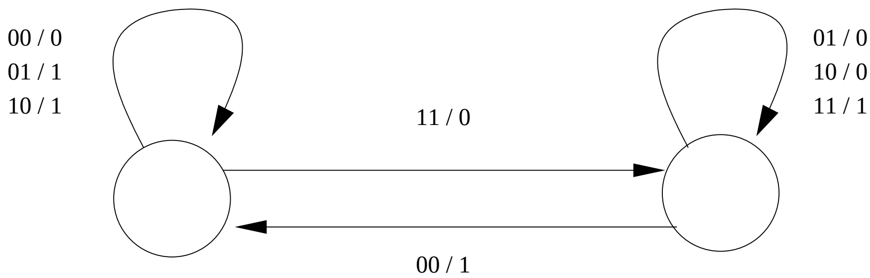






Increment, binary representation (bits in reverse order, 0^k represents 2^k).

1 / 0



Add, binary representation (bits in reverse order, 0 padded).

HOMEWORK: Let "addition check" refer to the task of checking whether $x + y = z$, where x, y, z are positive integers in unary representation. Give a finite automaton which solves the "addition check"problem (provide complete details). Show that if the input alphabet is $\Sigma = \{0, 1\}$ and the representation is of the form $x0y0z0$ — here 0 is used to terminate inputs x, y, z (respectively), then the "addition check" problem can not be solved by a finite automaton.