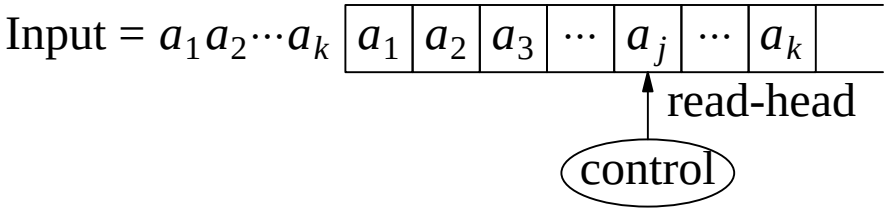
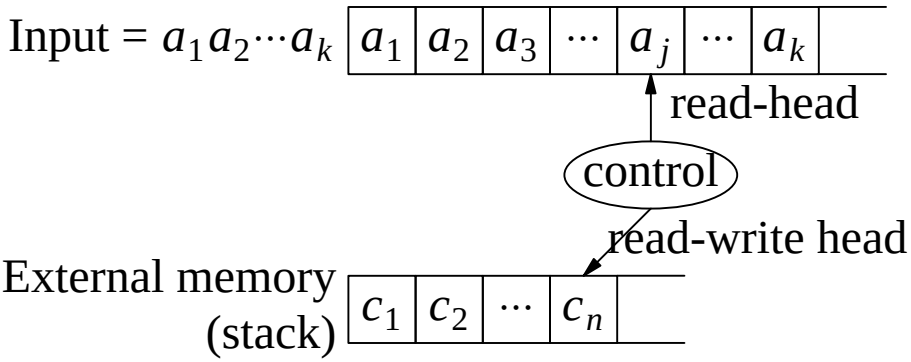
BASIC STRUCTURE OF MACHINES

Finite-state Automata: No output as such.



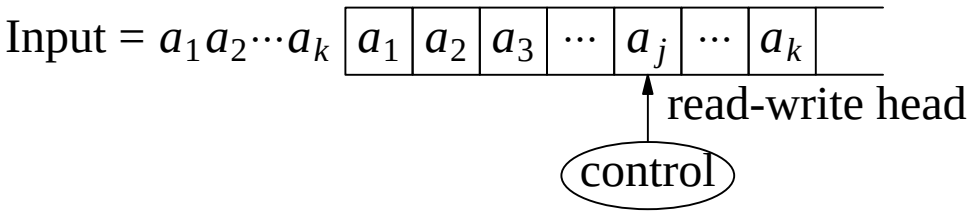
- *Control* updates "internal memory" = *state* of the machine as it reads each input symbol a_j only once.
- The read-head moves to right after reading a symbol.

Push-down Automata: No output as such.



- Control now also updates the "external memory".

Turing machine: Output = a string, written on input-output tape.



- An input/output-tape symbol may be read/written more than once.

FSA FOR EVEN NUMBER OF ZEROS IN BINARY STRINGS

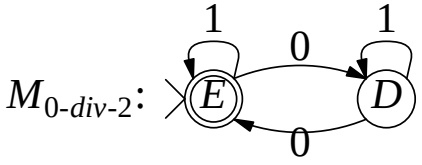
string:	1	0	0	1	0	1	0
zero-count:	0	1	2	2	3	3	4
even:	Y	N	Y	Y	N	N	Y

(before reading symbol)

Key observations for designing the algorithm:

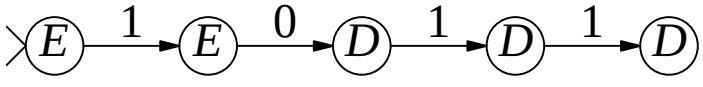
- Initially, the zero-count is even (= zero); we only need to keep track of odd/even status of the zero-count.
- Reading ’1’ does not change the odd/even status of the zero-count.
- Reading ’0’ changes the odd/even status to the opposite.
- Final desired state is ’even’.
- Suffices to look at each symbol only once, in the left to right order.

Representing the algorithm as a state-diagram:



$L_{0\text{-}div\text{-}2} = \{\lambda, 1, 00, 11, 001, 010, 100, 111, \cdots\}$

- State = odd/even status of zero-count
- Start-state indicated by ">"; final-state indicated by thick-circle
- Transitions indicate the input-symbol read and the state-update.
- Each *bad* string ($x \notin L_{0-div-2}$) ends in a non-final state starting at the start-state; each *good* string ($x \in L_{0-div-2}$) ends in a final-state.



EXERCISE

1. Show that there are 2^{n-1} strings of length n (≥ 1) in $L_{0\text{-}div\text{-}2}$ by giving an 1-1 and onto mapping $f(x)$ from $\{x \in L_{0\text{-}div\text{-}2}: |x| = n\}$ to $\{y \in \{0, 1\}^*: |y| = n - 1\}$.
2. Give the state-diagram for an FSA $M_{0\text{-}div\text{-}3}$ which accepts the binary strings $L_{0\text{-}div\text{-}3} = \{x \in \{0, 1\}^*: \#(0, x)$ is divisible by $3\}$. (Hint: First, write down some of the strings in $L_{0\text{-}div\text{-}3}$.)
3. Show the state-diagram of an FSA which accepts the complement language $L^c_{0\text{-}div\text{-}3}$.
4. Show the state-diagrams of all FSA's with input alphabet $\Sigma = \{0, 1\}$, two states (both must be reachable from the start-state), and at least one final-state. Give an English descriptions of the set of strings accepted by each FSA.

BINARY STRINGS WITH EVEN #(0, x): A COMPUTABLE PATTERN

Pattern-description: Binary strings with even number of zeros.

- This is a mathematically good description; it states a precise condition (a property) that a string must satisfy.
- It does not say how to go about in verifying the required condition (property) for a given string x.
- Some possible methods:

  (1) Count #(0, x), divide it by 2, and verify that the remainder is zero. (Also, check that each symbol in x is 0 or 1.)

  (2) Find $|x|$, count #(1, x), subtract it from $|x|$ to get #(0, x), etc.

Computational-description: The finite-state automaton $M_{0\text{-}div\text{-}2}$.

- It describes *an algorithm* to verify the required property.
  - Where/how to start (set initial count = even, i.e., position yourself at the start-state and at the first input symbol).
  - When to stop (at the end of input string).
  - What to do at each step (what to read, where to find it, how to update the state).
- The description is finite: the finite alphabet, the finite number of states (and the transitions), the start-state, and the final-states.
- How does $M_{0\text{-}div\text{-}2}$ describe infinitely many strings in a finite way?
  - Via the *dynamics* of FSA: successive applications of transitions. Longer strings give longer chains of transitions.

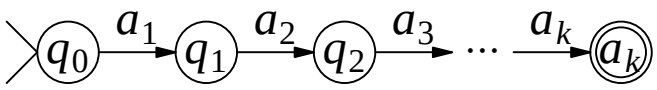
FORMAL DEFINITION OF AN FSA

$M = (Q, q_0, F, \Sigma, \delta)$, where

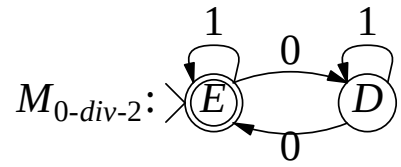
- **Static-part:**

Q = A finite, non-empty set of states
q_0 = The start-state; $q_0 \in Q$
F = The set of final-state(s) $\subseteq Q$
Σ = A finite, non-empty input alphabet
$\delta(q_i, a_j)$ = q_k, the next-state after reading the input symbol a_j in state q_i; δ is called the *transition function*. We sometimes write $\delta(q_i, a_j) = q_k$ as the triplet (q_i, a_j, q_k).

- **Dynamic-part:** For input $x = a_1 a_2 \cdots a_k$, apply the transitions successively as indicated below:



Example:



$Q = \{E, D\}$, $q_0 = E$, $F = \{E\}$, $\Sigma = \{0, 1\}$,
Transitions = $\{(E, 0, D), (E, 1, E), (D, 0, E), (D, 1, D)\}$

- An input string $x = a_1 a_2 \cdots a_k$ is *accepted* by M if the state reached after reading/processing *all* of x is a final-state.

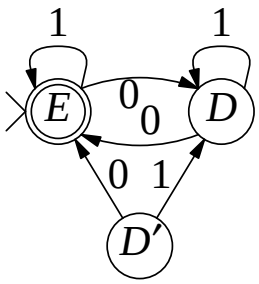
Language defined by M: $L(M) = \{x \in \Sigma^* : x \text{ accepted by } M\}$.

$$L(M_{0-div-2}) = \{\lambda, 1, 00, 11, 001, \cdots\}$$

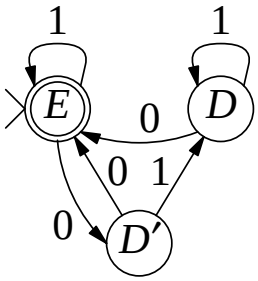
Question: What is the language if both states E and D were final-states in $M_{0-div-2}$?

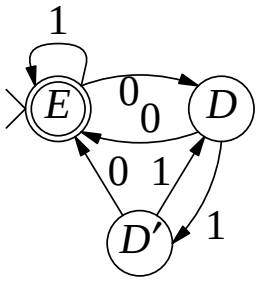
SOME OTHER FSA FOR $L_{0\text{-}div\text{-}2}$

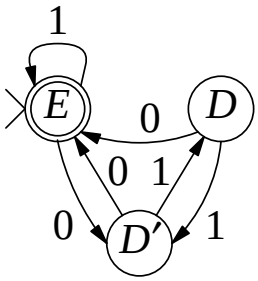
- A variant of $M_{0\text{-}div\text{-}2}$.
  - State D' is a *duplicate* of D in the sense that $\delta(D, a) = \delta(D', a)$ for $a = 0, 1$.
  - D' does not participate in accepting any input string since it is not *reachable* from the start-state. (Changing $\delta(D', a)$ for any of $a = 0, 1$ does not alter the language accepted either.)



  - To make D' reachable from the start-state *without* changing the language, we can change one or more transitions going to D (from E or D) to go to D', making sure that this does not make D itself unreachable. This can be done in three ways:







Both D and D' correspond to "odd number of 0's".

- There are countably many different FSAs for $L_{0\text{-}div\text{-}2}$.
- $M_{0\text{-}div\text{-}2}$ is the only one with the *smallest* number of states among all of them.

EXERCISE

1. Consider duplicating the state E in $M_{0\text{-}div\text{-}2}$ and making sure that the new state E' (and all other states) is reachable from the start-state. Carefully determine whether the new state E' should be a final-state since the original state E is a final-state. (We should not have two start-states, however.) Show the state-diagram of your new FSA(s).
2. Modify the original $M_{0\text{-}div\text{-}2}$ so that λ is not accepted but all other strings in $L_{0\text{-}div\text{-}2}$ are accepted. (Hint: Its start-state should be similar to the state E in $M_{0\text{-}div\text{-}2}$, but it should not be a final-state.)

FSA FOR BINARY STRINGS CONTAINING "11"

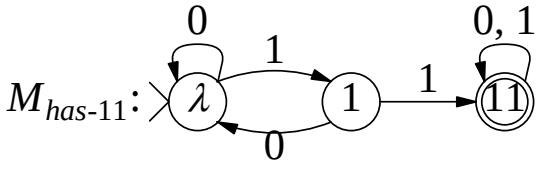
$L_{has\text{-}11}$ = { 11, 011, 110, 111,
0011, 0110, 1100, 0111, 1011, 1101, 1111, ···}

Key Steps:

- Write down systematically some of the strings in the language.
- Determine what is to be remembered as you process each symbol in an input string in order to decide if the part processed so far is "good" or "bad". The things remembered must not grow arbitrarily large as more and more input symbols are processed.
- Determine the initial and the final value(s) of the thing being remembered. The final value(s) for good strings must be different from those for the bad strings.
- Determine how this memory (= the *control* memory) is updated as each input symbol is processed.

For $L_{has\text{-}11}$:

- The memory has three possible values: No part of "11" is seen, the first 1 in a possible "11" is seen, and the second 1 in an "11" is seen. We can represent them as "λ", "1", and "11".
- Initial value = λ; final value = 11.
- The update rules for the memory are the transitions below.



Question: How to modify $M_{has\text{-}11}$ so that it accepts the complement language $L^{c}_{has\text{-}11}$?

EXERCISE

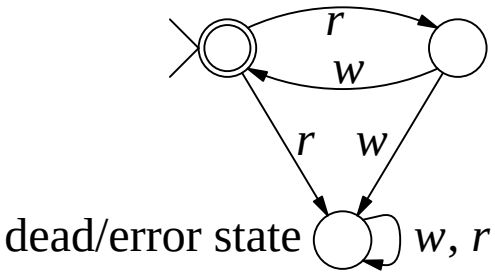
1. Let $f(n)$ = the number of strings of length n which are in $L_{has\text{-}11}$ and $g(n)$ = the number of strings of length n which are not in $L_{has\text{-}11}$. Then show that $g(n) = g(n-1) + g(n-2)$ for $n \geq 2$, and $f(n) = 2^n - g(n)$.
2. Delete the transition from state '1' to the state 'λ' (labeled by the input symbol '0') from each of $M_{has\text{-}11}$ and $M^c_{has\text{-}11}$ above, and call the new automatons M_1 and M_2. Then verify that the following is false: $L(M_2) = L(M_1)^c$.
3. Describe in English the language $L(M_1)$. Then, modify M_1 to make it completely defined, without changing its language and show the complement FSA for the modified M_1.

COMPLETELY DEFINED FSA

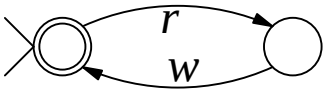
Complete Definition:

For eash state $q \in Q$ and each input sysmbol $a \in \Sigma$, give the transition $\delta(q, a)$ for the next-state.

Example. For $L = \{(rw)^n : n \geq 0\}$, the completely defined FSA is



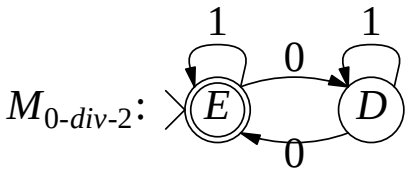
We often avoid writing dead/error states and transitions to/from them to simplify the state-diagram:



PAST- AND FUTURE-BASED STATE DESCRIPTIONS

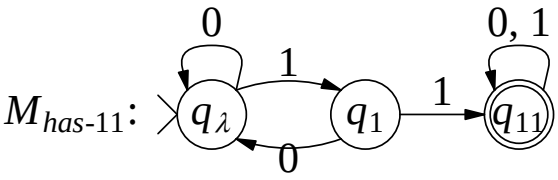
Past(q): The property of inputs $x = a_1 a_2 \cdots a_n$ that *brings the FSA to* the state q, i.e., when do we arrive at q.

Future(q): The property of inputs $x = a_1 a_2 \cdots a_n$ that *takes the FSM from q* to a final state, i.e., what remains to do at q to reach a finite state.



$L_{0\text{-}div\text{-}2} = \{\lambda, 1, 00, 11, 001, 010, 100, \cdots\}$

State	Past-based description	Future-based description
E	Have seen even many 0's	Need to see even many 0's
D	Have seen odd many 0's	Need to see odd many 0's



$L_{has\text{-}11} = \{11, 011, 110, 111, \cdots\}$

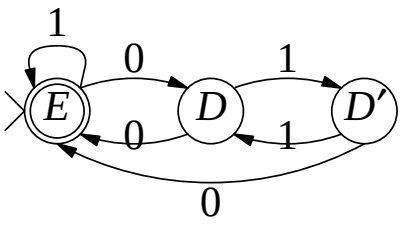
State	Past-based description	Future-based description
q_λ	Have not seen any part of "11"	Need to see "11"
q_1	Have only seen "1", or the last two symbols seen are "01"	Need to see an immediate 1 or a future "11"
q_{11}	Have seen "11"	Any thing is fine

Question: What is wrong with Past(q_1) = Have seen '1'?

Future-based state descriptions are often more useful.

EXERCISE

1. Show an FSA for $L_{bin-even} = \{x: x \in \{0, 1\}^* \text{ represents a binary even number}\}$. Then, give the past and future based descriptions of its states.
2. Give the past and future descriptions of the states of the FSA below.



3. Give the future based state descriptions for each of the three alternative FSA (with 3 states in each) shown earlier for $M_{0\text{-}div\text{-}2}$.

A PROBLEM INVOLVING COUNTING IS SOLVED WITHOUT COUNTING

Problem:

- Consider the language {*x*: *x* is a binary string with equal number of "01" and "10"}.

Examples of Good Strings of Length 4:

0000 (y)	0100 (y)	1000 (n)	1100 (n)
0001 (n)	0101 (n)	1001 (y)	1101 (y)
0010 (y)	0110 (y)	1010 (n)	1110 (n)
0011 (n)	0111 (n)	1011 (y)	1111 (y)

Equivalent Property:

- The binary string *x* begins and ends with the same symbol.

An FSA: