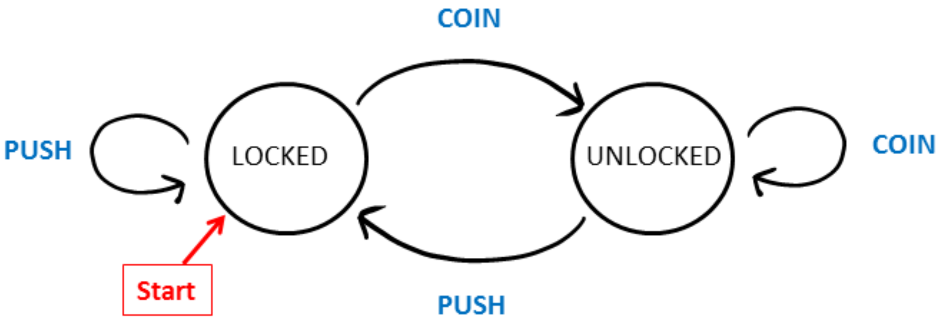
17.1 Finite state machines

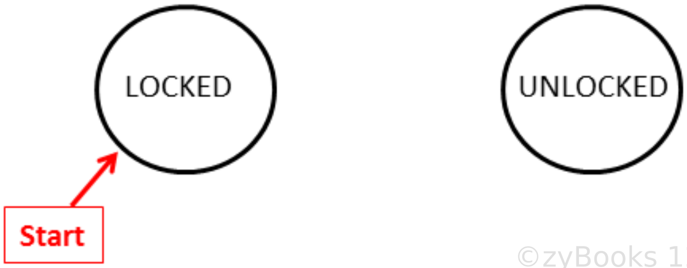
This section has been set as optional by your instructor.

Computer scientists develop abstract models for computing devices in order to reason about what can and cannot be computed by a particular kind of device. One such model is a finite state machine. A ***finite state machine*** (or ***FSM***) consists of a finite set of states with transitions between the states triggered by different input actions. Finite state machines are used in various computer science applications. Most of this section's examples show how FSMs can model logical devices. FSMs are also used in specifying network and communication protocols, compiler design, and text search. A finite state machine is sometimes called a ***finite state automaton*** (plural: ***finite state automata***).

We illustrate the components of an FSM using a small example before giving a formal definition. The diagram below shows an FSM that controls a turnstile. The turnstile starts out in a locked state and unlocks when a person inserts a coin. Additional coins inserted when the turnstile is unlocked do not change the state of the system. If the turnstile is unlocked and the person pushes it, then the turnstile transitions to the locked state, awaiting a coin from the next person. Pushing the turnstile while it is locked does not affect the system state.



Now we will introduce and define each component of the FSM. A finite state machine has, as the name suggests, a finite set of states. The FSM always resides in one of the states. Furthermore, the current state is the only information that the FSM remembers about the past. To help a human understand what the states represent, the states are given descriptive names. The first example presented here is a turnstile which is either locked or unlocked. So the set of states Q would be {LOCKED, UNLOCKED}. Although the states can be given any two distinct labels, it is good practice to give states descriptive names. The FSM also has a designated start state which is the initial state of the system before any input is received. In the case of the turnstile, the start state is the LOCKED state. In the diagram, there is a vertex (circle) labeled with each state. The start state is designated with an arrow:



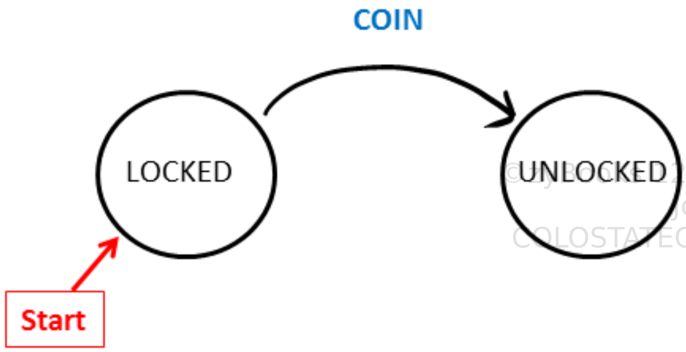
The FSM receives input from the outside in the form of a sequence of actions. Each individual action is from a finite set of inputs. In the case of the turnstile, a person can insert a coin into the turnstile or push the turnstile to pass through it. The input set I = {COIN, PUSH}. The input to the FSM is a sequence of actions from the input set. In the case of a turnstile, the input might be: COIN, COIN, PUSH, COIN, PUSH.

The FSM then reacts to the input. The reaction depends on the current state and the input action. We present a variety of different types of FSMs that have different mechanisms for reacting to the input. In the case of the turnstile, the reaction is encoded in the state: the turnstile is either LOCKED (so that the person is prevented from passing through) or UNLOCKED (so that the person is allowed to pass through).

The reaction of a finite state machine to the input received is denoted by a ***transition function***. The input to the transition function is the current state and the input action, so the domain is the Cartesian product set Q × I. The output of the transition function for this particular style of FSM is just the next state to which the FSM transitions. Therefore the target set of the transition function is the set Q. The image below shows that the FSM upon receiving input COIN while in the LOCKED state transitions to the UNLOCKED state. Thus, the output of the transition function on input (LOCKED, COIN) is UNLOCKED. The transition function is often denoted by the symbol δ.

$$\delta(\text{LOCKED}, \text{COIN}) = \text{UNLOCKED}$$

The transition is depicted in the diagram by an arrow from LOCKED to UNLOCKED and is labeled with the input COIN.



The transition function can also be specified by a state transition table. The rows represent the current state, the columns represent the possible inputs. The entry for a particular row and column indicate the new state resulting from that state/input combination. Below is the state transition table for the turnstile example:

Table 17.1.1: State transition table for the turnstile finite state machine.

-	Coin	Push
Locked	Unlocked	Locked
Unlocked	Unlocked	Locked

The table below summarizes the components of a finite state machine:

Table 17.1.2: Components of a FSM.

Notation	Description
Q	Finite set of states.
$q_0 \in Q$	q_0 is the start state.
I	Finite set of input actions.
$\delta: Q \times I \rightarrow Q$	Transition function.

The animation below shows how the turnstile FSM behaves on a particular sequence of input actions:

PARTICIPATION ACTIVITY | 17.1.1: The actions of the turnstile FSM.

Animation content:

`undefined`

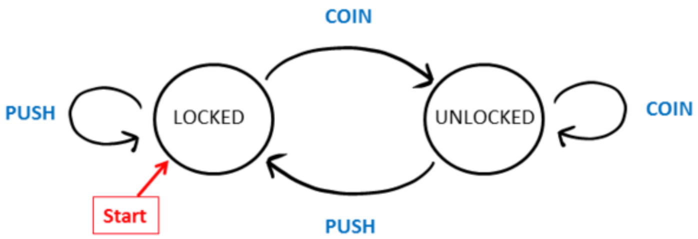
Animation captions:

1. The start state is "LOCKED". On input "COIN", the current state transitions to "UNLOCKED".
2. The next input is "COIN" and the current state remains "UNLOCKED".
3. The next input is "PUSH" and the current state transitions to "LOCKED". The last input is "PUSH" and the current state remains "LOCKED", the final state.

PARTICIPATION ACTIVITY

17.1.2: Input and output of the transition function for the turnstile FSM.

Consider the turnstile FSM below:



1) Which pair would be a valid input to the transition function δ?

○ (COIN, PUSH)

○ (LOCKED, PUSH)

○ (LOCKED, UNLOCKED)

2) What is the value of δ(UNLOCKED, COIN)?

○ UNLOCKED

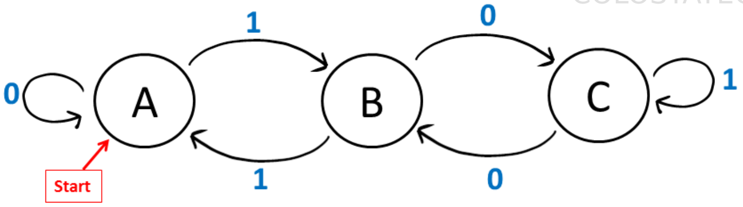
○ LOCKED

○ PUSH

PARTICIPATION ACTIVITY

17.1.3: Following the transitions of a finite state machine.

Consider the FSM defined in the state diagram below. The set of states is {A, B, C} and the set of input actions are labeled 0 or 1.



1) What is the current state after the FSM has processed the input sequence 00101? □

Check Show answer

2) What is the current state after the FSM has processed the input sequence 111010? □

Check Show answer

Finite state machines with output

In the turnstile example, the response of the FSM was entirely expressed in the state, i.e., whether the turnstile is locked or unlocked. In some applications, a more detailed response is required. A ***finite state machine with output***, or ***FSM with output***, produces a response that depends on the current state as well as the most recently received input. We illustrate an FSM with output using an example of a gumball machine. The machine sells gumballs for 20 cents and accepts only nickels and dimes. To keep the example simple, the machine will not make change. If the user has already inserted 15 cents and then inserts a dime, the machine just returns the dime and remains in the same state. Five states represent the amount of money that has been inserted so far:

$Q = \{ q_0, q_5, q_{10}, q_{15}, q_{20} \}$

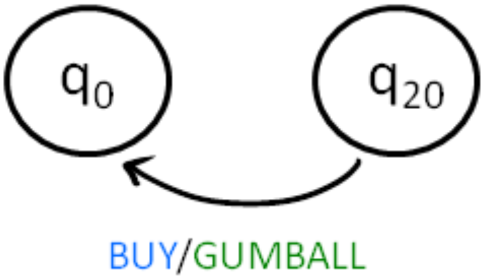
The user can insert a nickel, insert a dime, or press a button to buy a gumball:

I = { NICKEL, DIME, BUY }

The FSM has different ways of responding to the user's actions. The gumball machine can release a gumball, can return the most recently inserted coin, or can issue a message that there is not enough money to buy a gumball. The gumball may also not have an output in some situations. The responses of the gumball machine is a finite set called the output set and is denoted by O:

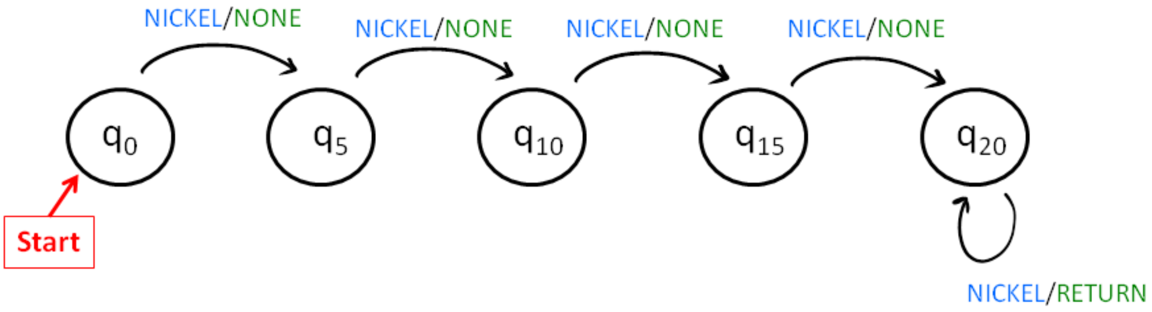
O = { GUMBALL, RETURN, MESSAGE, NONE }

The next part is to determine the transition function. The input to the transition function is a state and an input action. The output of the transition function is the new state to which the FSM transitions and a response from the set O. Therefore δ is a function whose domain is Q × I and whose target is Q × O. We specify the transition function by a state diagram which is a directed graph whose nodes represent states. As in the example of the turnstile, a directed edges represent transitions from one state to another. However, since the gumball machine example is an FSM with output, each edge is labeled with an input action from I and an output response from O. For example, a directed edge from state q_{20} to state q_0 labeled BUY/GUMBALL indicates that if the FSM is in state q_{20} and receives input BUY, then the FSM will transition to state q_0 and release a gumball:

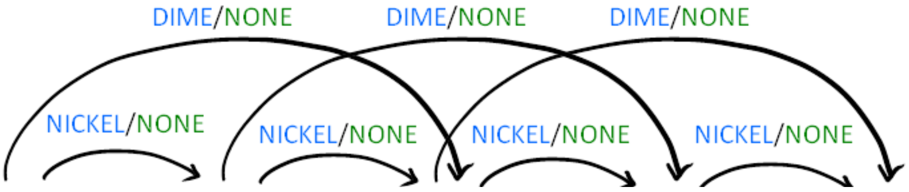


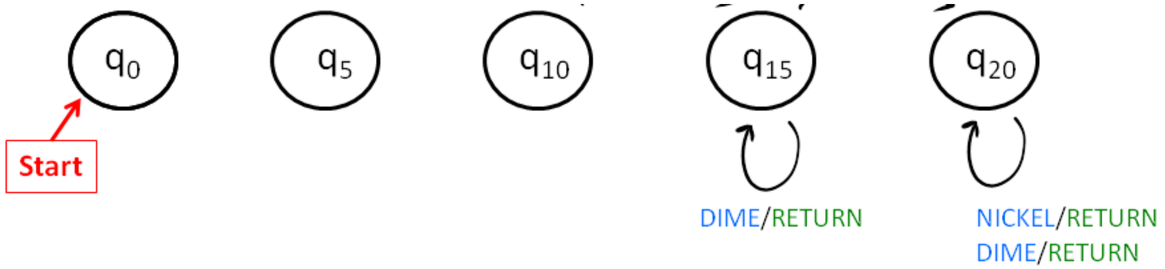
The directed edge corresponds to the fact that $\delta(q_{20}, \text{BUY}) = (q_0, \text{GUMBALL})$.

We construct the transition function by adding in edges to the state diagram in phases in order to illustrate how one might design an FSM to perform a specific function. First we show how the gumball machine responds when the user inputs a nickel. If the user has input less than 20 cents, the FSM advances to the next state (reflecting five more cents have been inserted) and produces no output. If the FSM is in state q_{20} and the user inputs a nickel, the coin is returned and the FSM stays in state q_{20}. The transitions are reflected in the diagram below:

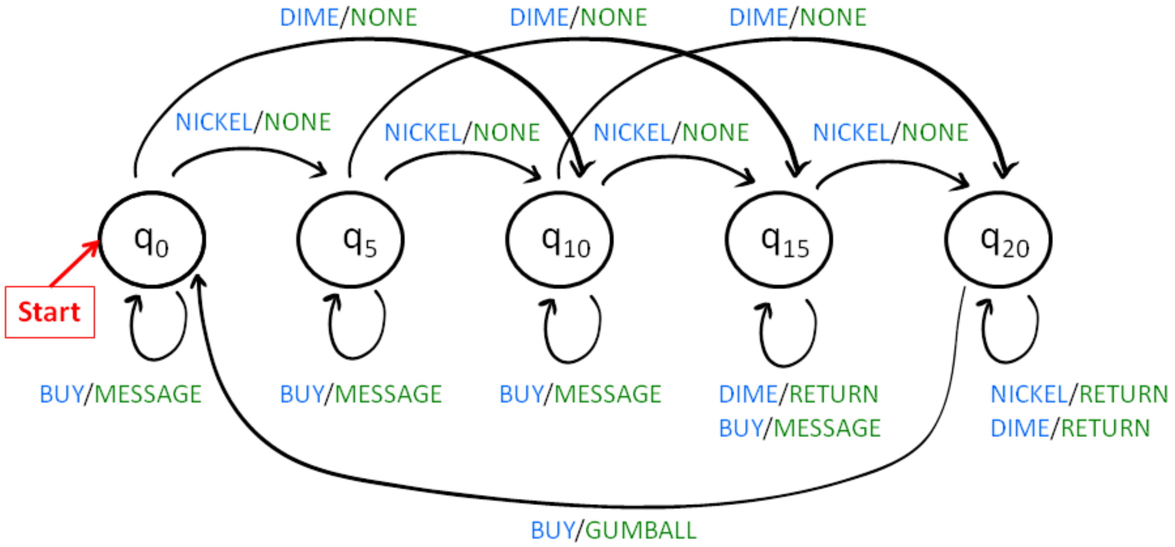


Now we add edges to the state diagram reflecting how the gumball machine responds when the user inputs a dime. If the user has input 10 cents or less (q_0, q_5, or q_{10}), the FSM transitions to a state reflecting 10 more cents have been inserted and produces no output. If the user has inserted more than ten cents (q_{15} or q_{20}), the FSM stays in the same state and returns the coin.





Now we add the edges to the state diagram reflecting how the gumball machine responds when the user presses the "buy" button. If the user has input less than 20 cents, it stays in the same state and outputs the message that the user has not input enough money. However, if the FSM is in state q_{20}, the user has input enough money for a gumball. The gumball machine releases a gumball and returns to the q_0 state:



The state diagram is complete: every state has an outgoing edge for every possible input action. Note that the only memory the FSM has is encoded in its current state. If the FSM is in state q_{10}, it knows that 10 cents have been inserted so far, but it does not know whether the user put in one dime or two nickels. Fortunately, the only information an FSM needs to know to operate correctly is the amount of money they user has inserted. The table below summarizes the components of an FSM with output. The new components that are not included in an FSM without output are highlighted in red:

Table 17.1.3: Components of a finite state machine with output.

Notation	Description
Q	Finite set of states.
$q_0 \in Q$	q_0 is the start state.
I	Finite set of input actions.
O	Finite set of output responses.
$\delta: Q \times I \rightarrow Q \times O$	Transition function.

The animation below shows how the gumball machine FSM behaves on a particular sequence of input actions:

PARTICIPATION ACTIVITY 17.1.4: The actions of the gumball machine FSM.

Animation content:

undefined

Animation captions:

1. The start state is q_0. On input "BUY", the current state remains q_0 and the output is "MESSAGE".
2. The next input is "NICKEL", the current state transitions to q_5 and the output is "NONE".
3. The next input is "DIME", the current state transitions to q_{15} and the output is "NONE".
4. The next input is "DIME", the current state remains q_{15} and the output is "RETURN".
5. The next input is "NICKEL", the current state transitions to q_{20} and the output is "NONE".
6. The final input is "BUY", the current state transitions back to q_0 and the output is "GUMBALL".

PARTICIPATION ACTIVITY 17.1.5: The gumball machine FSM.

1) What is the last output response after the gumball machine processes the sequence: DIME, DIME, NICKEL, DIME?

Check Show answer

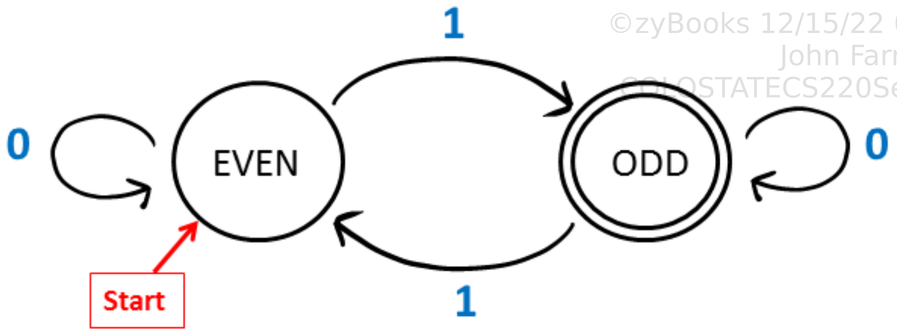
2) How many gumballs are purchased after the following input sequence: NICKEL, DIME, NICKEL, BUY, DIME, NICKEL, BUY? Write your answer in numerical form.

Check Show answer

Finite state machines that recognize properties

Finite state machines can also be used to determine whether an input string has a particular property. Instead of having input actions from a user, each input is a character from a finite alphabet. The sequence of inputs is a string over the input alphabet. An input string is accepted if the FSM ends up in an ***accepting state*** after each character in the string is processed. The set of all accepting states is a designated subset of the states.

Consider a finite state machine with input alphabet {0, 1}. There are two states: {ODD, EVEN}. The state EVEN is the start state and the state ODD is the only accepting state. The transition function for the FSM is specified by the state diagram below. As before, the start state is denoted by an arrow. The accepting state is shown with a double outline.



The ***parity of a string*** is whether the number of bits in the string is odd or even. The string 01011 has odd parity because 01011 has three 1's and three is an odd number. The string 11011 has even parity because 11011 has four 1's and four is an even number. The finite state machine shown above accepts a binary string if and only if the string has odd parity. The state encodes whether the number of 1's seen so far is odd or even. An input of 0 causes the FSM to stay in the same state. An input of 1 causes the FSM to change states.

The table below summarizes the components of an FSM that recognizes a property. The new components that are not included in an FSM without output are highlighted in red:

Table 17.1.4: Components of a finite state machine that recognizes a property.

Notation	Description
Q	Finite set of states.
$q_0 \in Q$	q_0 is the start state.
I	Finite set of input actions.
$A \subseteq Q$	Accepting states are a subset of the states.
$\delta: Q \times I \to Q$	Transition function.

Here is an animation of the action of the FSM on a particular input:

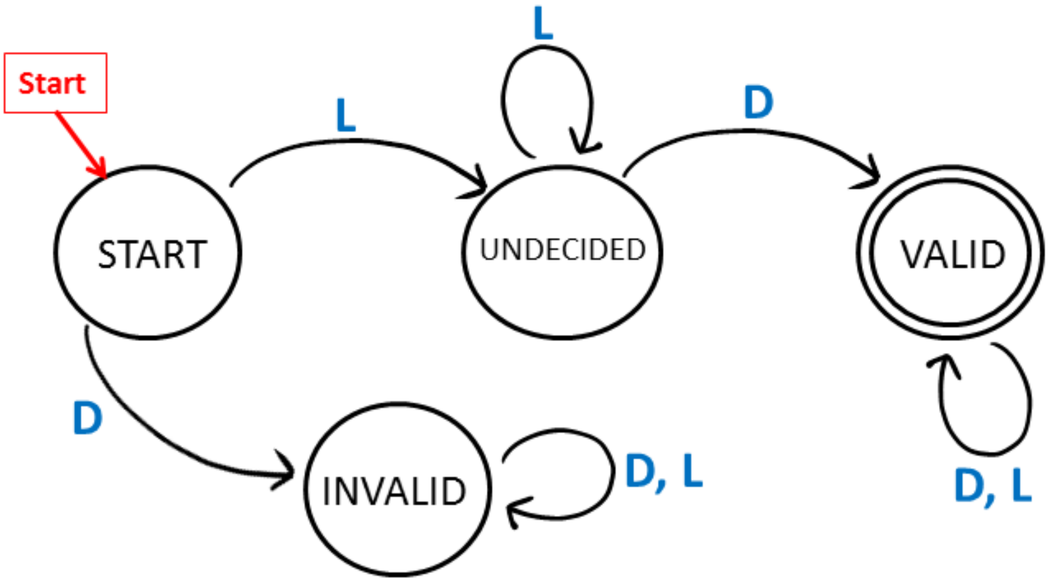
PARTICIPATION ACTIVITY | 17.1.6: The actions of the FSM that computes the parity of a string.

Animation captions:

1. The start state is "EVEN". On input 1, the current state transitions to "ODD".
2. The rest of the inputs are 1, 0, 1. The state transitions to "EVEN", "EVEN", then "ODD". Since the final state "ODD" is accepting, the input 1101 is accepted.
3. On inputs 101, the sequence of states is "EVEN", "ODD", "ODD", "EVEN". Since the final state "EVEN" is not accepting, the input 101 is not accepted.

Recognizing valid passwords

In another example, consider a computer system that accepts a password composed of digits and letters of any length. A valid password must begin with a letter and have at least one digit. We will show an FSM that accepts an input string if and only if it is a valid password. The input alphabet is {D, L} which stand for "digit" and "letter". There are four states: {START, INVALID, VALID, UNDECIDED}. The only accepting state is the VALID state. The state diagram for the FSM is given below:



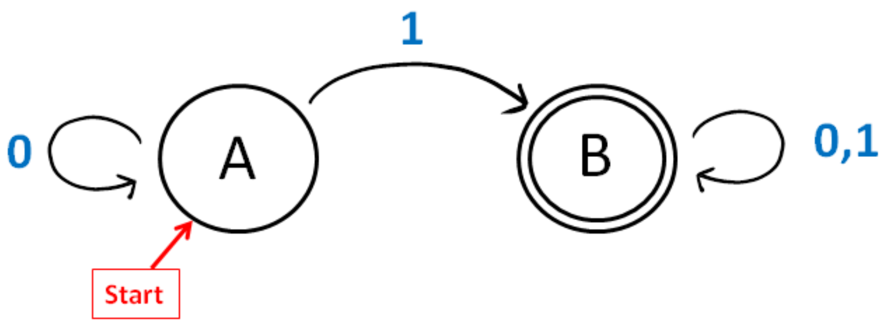
Note that if the first input character is a digit, the FSM will transition to the INVALID state because the first character is required to be a letter. If the FSM reaches the invalid state, the FSM will not accept the input regardless of what follows. On the other hand, if the first input character is a letter, the FSM will reach the UNDECIDED state. In the UNDECIDED state, the FSM is waiting to see if the input string contains a digit. If the FSM is in the UNDECIDED state and a digit appears, then all the conditions for the password have been met and the FSM will transition to the VALID state and will accept the input string regardless of what other characters follow. On the other hand, if the string ends while the FSM is in the UNDECIDED state (i.e., without having seen a digit in the input string), the FSM will not accept.

Additional Information 17.1.1: Limitations of FSMs.

While finite state machines can be used to recognize many useful properties, there are limits to the kinds of properties that can be recognized by an FSM. For example, it is impossible to design an FSM that takes as input any binary string and accepts if and only if the majority of the bits are 1. The proof that FSMs can not recognize certain properties is beyond the scope of this material, but it is not hard to see why FSMs are so limited. Essentially, the only memory that an FSM has is encoded in the current state. Since the set of states are finite, an FSM inherently has a finite memory. In order to compute whether there is a majority of 1's in a string the machine would need to keep track of how many more 1's than 0's have been observed in the input string up to a given point. Since the set of all binary strings is infinite, there is no bound to the number that a device computing majority would need to hold.

PARTICIPATION ACTIVITY 17.1.7: Finite state machines as recognizers.

The input alphabet for the FSM below is {0, 1}. The state set is {A, B}

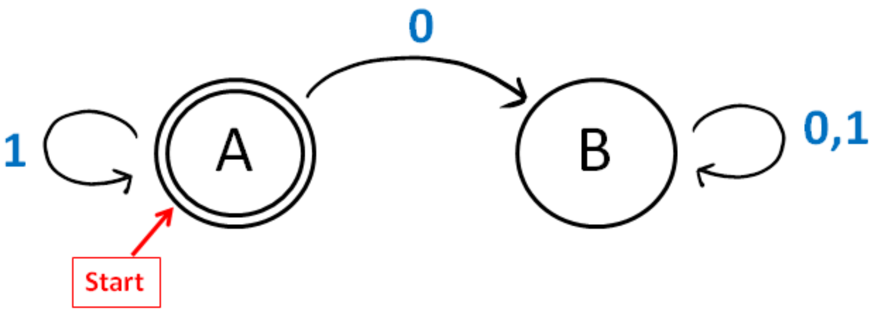


1) Which sentence describes the set of strings accepted by the FSM?

○ The FSM accepts a binary string if and only if the majority of bits are 1's.

PARTICIPATION ACTIVITY | 17.1.8: Finite state machines as recognizers.

Consider the finite state machine denoted below. The input alphabet for each is {0, 1}. The state set for each is {A, B}



1) Which sentence describes the set of strings accepted by the FSM?

○ The FSM accepts a binary string if and only if the majority of bits are 1's.

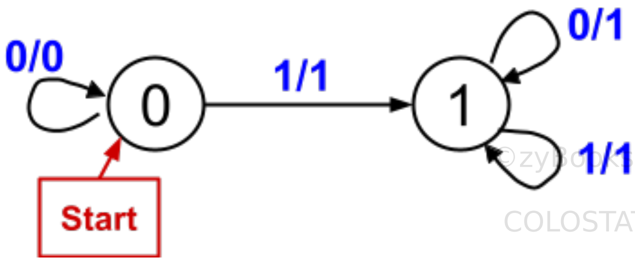
○ The FSM accepts a binary string if and only if the parity of the string is odd.

○ The FSM accepts a binary string if and only if all the input bits are 1.

○ The FSM accepts a binary string if and only if there is at least one 1 in the input sequence.

Additional exercises

EXERCISE 17.1.1: Output for an FSM on a given input - example 1.



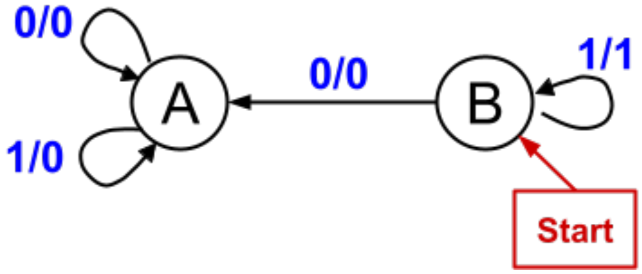
The input and output alphabet for the FSM shown above is {0, 1}. Give the output for the FSM for each input string.

(a) 000

(b) 010

(c) 111

EXERCISE 17.1.2: Output for an FSM on a given input - example 2.



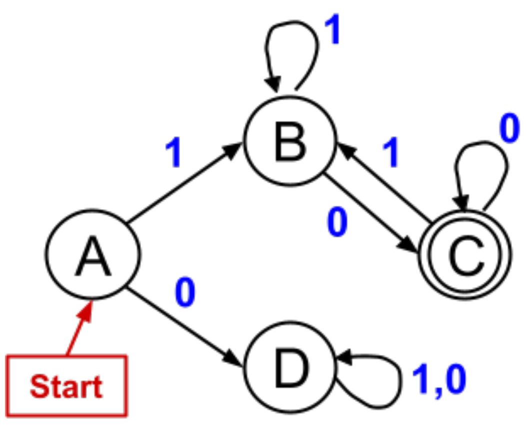
The input and output alphabet for the FSM shown above is {0, 1}. Give the output for the FSM for each input string.

(a) 000

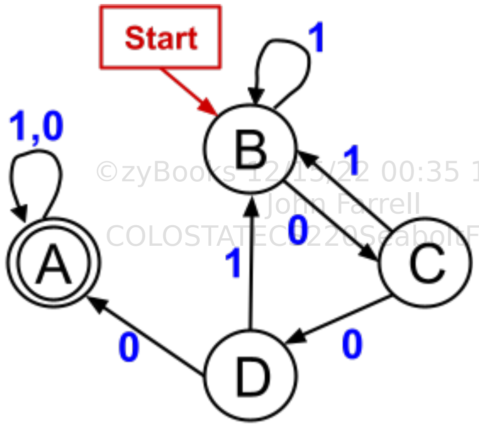
(b) 010

(c) 101

EXERCISE | 17.1.3: Identifying the set of strings accepted by an FSM.



FSM 1



FSM 2

(a) Give the final state of each FSM after each input string below. That is, start in the start state, process the string, and indicate which state (A, B, C, or D) the FSM is in at the end of the string. Then indicate whether or not the final state is an accepting state.

- 100011
- 0000
- 0010
- 1100

(b) Which one of the following statements below describes the set of strings accepted by FSM 1?

1. The FSM accepts a string x if and only if x contains the pattern "01" somewhere in the string.
2. The FSM accepts a string x if and only if x starts with a 1 and ends with a 0.
3. The FSM accepts a string x if and only if x is all 0's or all 1's (i.e. x only contains one type of character).
4. The FSM accepts a string x if and only if there are at least three consecutive 0's somewhere in x.
5. The FSM accepts a string x if and only if x ends with "000".
6. The FSM accepts a string x if and only if x ends with "00".

(c) Refer to the same set of statements in the previous question and indicate which statement describes the set of strings accepted by FSM 2.

EXERCISE 17.1.4: Designing FSMs that accept a given set of strings.

For each property, design an FSM with input alphabet {0, 1} that accepts a string x if and only if the string has the property described.

(a) There are no occurrences of "00" or "11" in the string. (The empty string has no occurrences of "00" or "11".)

(b) The number of 1's is a multiple of 3. (Zero is a multiple of 3).

(c) There is at least one 0 and at least one 1.

17.2 Turing machines

This section has been set as optional by your instructor.

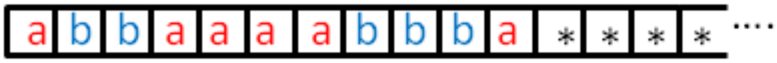
Finite state machines are effective in some settings, such as modeling devices with limited input and output capabilities. However, finite state machines are unable to solve even simple computational tasks such as determining whether a binary string has more 0's than 1's.

Mathematician Alan Turing developed a model of a computer, called a ***Turing machine***, in order to reason about general purpose computers. Turing machines are much simpler than modern processors, but Turing machines are believed to be every bit as powerful. The ***Church-Turing conjecture*** says that any problem that can be solved efficiently on any computing device can be solved efficiently by a Turing machine.

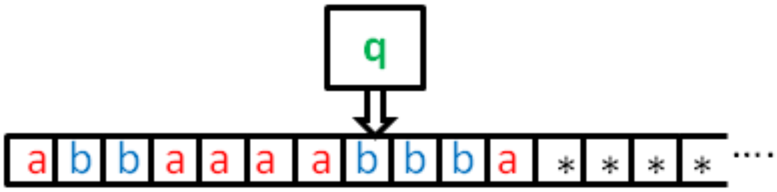
There is often a tradeoff between the complexity of a device and how easy that device is to program. A modern computer with its layers of hardware, operating system and compilers, provides an intuitive user interface so that people can easily write programs to solve complex tasks. Turing machines, on the other hand, are simple to describe in the abstract but are very cumbersome to use in solving specific computational problems. Nonetheless, the Turing machine is a useful abstraction for reasoning about the nature and limits of computation.

The definition of a Turing machine

The memory of a Turing machines is a one-dimensional tape. The tape is divided into individual cells, each of which can hold one symbol from a finite ***tape alphabet*** denoted by Γ. The tape is infinite to the right but has a leftmost cell. The tape alphabet must contain a special symbol called the blank symbol (represented by a * symbol) and at least one other symbol. Here is what the tape of a Turing machine might look like for tape alphabet {a, b, *}:



The Turing machine has a finite set of states, denoted by Q. The set Q includes three special states: q_0 is the start state, q_{acc} is the accept state, and q_{rej} is the reject state. The Turing machine also has a head that always points to a cell of the tape. The symbol in the cell to which the head is currently pointing is called the current symbol. A ***configuration*** of a Turing machine consists of the contents of the tape, the current state, and the tape cell to which the head is currently pointing. Here is a drawing of a configuration of a Turing machine that is in state q and whose current symbol is b:



The action of the Turing machine is defined by a transition function δ. The input to the transition function is the current state and the current symbol. The output of the transition function is a state, a tape symbol and a direction for the tape head to move. The direction is an element from the set {L, R}. L denotes a move to the left and R denotes a move to the right. The Turing machine proceeds in a series of steps. Each step is dictated by the transition function. Suppose, for example, that the state of the Turing machine is as pictured above and δ(q, b) = (q', a, L). Then the Turing machine will write an 'a' in the current cell, transition to state q' and the head will move one cell to the left. If the head is supposed to move left and is already in the leftmost tape cell, then the head stays in the same location after the step. Here is an animation showing several steps of the Turing machine:

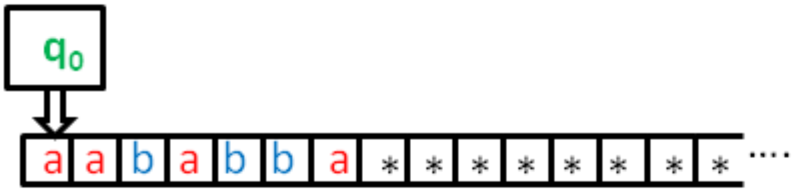
PARTICIPATION ACTIVITY 17.2.1: Steps of a Turing machine.

Animation captions:

1. The state is q_1 and the current symbol is a. δ (q_1, a) = (q_2, *, L). The state becomes q_2, the symbol a is replaced by *, and the head moves left.
2. The state is q_2 and the current symbol is b. δ (q_2, b) = (q_1, b, R). The state becomes q_1, the symbol b is unchanged, and the head moves right.
3. The state is q_1 and the current symbol is *. δ (q_1, *) = (q_3, b, R). The state becomes q_3, the symbol * is replaced by b, and the head moves right.

The transition function is defined for every (state, tape symbol) pair, except for the states q_{acc} and q_{rej}. If the Turing machine transitions to q_{acc} or q_{rej} during a step, then the Turing machine completes the step and then stops.

The input to the Turing machine is specified as a string over the ***input alphabet***. The input alphabet, denoted by Σ, must be a subset of the tape alphabet ($\Sigma \subset \Gamma$) and can not contain the blank symbol ($* \notin \Sigma$). A Turing machine starts out an execution in state q_0 with the head in the leftmost position. The input to the problem to be solved is written on the tape as a string of input symbols starting in the leftmost cell. The rest of the tape is all blank symbols that extend infinitely to the right. Here is a drawing of the initial configuration of a Turing machine on input "aababba":



The table below lists all the components in the definition of a Turing machine:

Table 17.2.1: Components of a Turing machine.

Notation	Description
Q	Finite set of states.
Γ	Finite set of tape symbols.
$\Sigma \subset \Gamma$	A subset of the tape symbols are input symbols.
$q_0 \in Q$	q_0 is the start state.
$q_{acc} \in Q$	q_{acc} is the accept state.
$q_{rej} \in Q$	q_{rej} is the reject state.
$\delta: (Q - \{q_{acc}, q_{rej}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$	Transition function.

The Turing machine proceeds according to the rules designated by the transition function. If the Turing machine reaches the accept state after starting with a particular input string x, then the ***Turing machine accepts*** x. If the Turing machine reaches the reject state after starting with a particular input string x, then the ***Turing machine rejects*** x. If a Turing machine reaches either the accept or reject state on input x, we say that the ***Turing machine halts*** on input x. As with other computer programs, the Turing machine may never halt on a particular input in which case the Turing machine neither accepts or rejects the input.

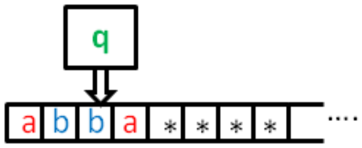
PARTICIPATION ACTIVITY | 17.2.2: Identifying the next configuration of a Turing machine.

Consider a Turing machine whose transition function is partially specified below:

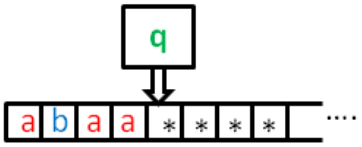
δ(q, b) = (q', a, R)
δ(q', a) = (q, a, R)
δ(q, *) = (q', b, L)

Below are four Turing machine configurations:

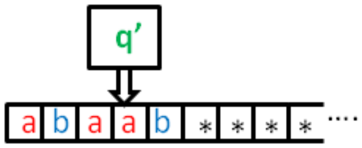
1:



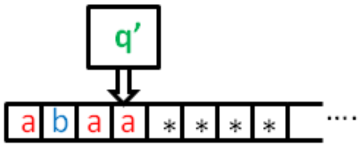
2:



3:



4:



1) The current configuration of a Turing machine is shown in drawing 1. Which drawing shows the configuration of the Turing machine after one step? Enter your answer as a number from 1 to 4.

Check Show answer

2) The current configuration of a Turing machine is shown in drawing 1. Which drawing shows the configuration of the Turing machine after two steps?

Check Show answer

3) The current configuration of a Turing machine is shown in drawing 1. Which drawing shows the configuration of the Turing machine after three steps?

Check Show answer

A simple Turing machine

The first example of a Turing machine takes as input any string over the alphabet {a, b} and accepts the string if and only if the string has at least two b's anywhere in the string. The two b's do not have to be consecutive, so the Turing machine should accept the string baba, for example. The input alphabet is {a, b} and the tape alphabet is {a, b, *}. The set of state is {q_0, q_1, q_{acc}, q_{rej}}. The head starts in the left-most cell and moves from left to right. In each move the Turing machine writes the same symbol that was read, so the tape never changes content. The number of b's encountered so far is encoded in the state. q_0 represents the condition that no b's have been seen. q_1 represents the condition that one b has been seen. If the Turing machine is in state q_0 and the tape symbol b is encountered, the Turing machine transitions from q_0 to q_1 and continues moving to the right. If the Turing machine is in state q_1 and the tape symbol b is encountered, the Turing machine transitions from q_1 to q_{acc}, reflecting the fact that two b's have been seen. If the Turing machine encounters a * symbol in state q_0 or q_1, then the Turing machine has reached the end of the input string without having seen two b's and therefore transitions to q_{rej}.

The transition function is given in the table below. The animation illustrates the execution of the Turing machine on two different input strings.

Table 17.2.2: Transition function for the Turing machine that recognizes strings with two b's.

-	a	b	*
q_0	(q_0, a, R)	(q_1, b, R)	(q_{rej}, *, L)
q_1	(q_1, a, R)	(q_{acc}, b, R)	(q_{rej}, *, L)

PARTICIPATION ACTIVITY 17.2.3: A Turing machine that accepts strings with two b's.

Animation content:

undefined

Animation captions:

1. Input abba is written on the tape, followed by *'s. The start state is q_0 and the head is at the first a. In step 1, the head moves right. The state is unchanged.
2. In the next step, the head moves right, the state transitions to q_1 and the tape does not change.
3. In the next step, the state transitions to q_{acc}. The Turing machine halts and accepts.
4. Input abaa is written on the tape, followed by *'s. The start state is q_0 and the head is at the first a. In step 1, the head moves right. The state is unchanged.
5. In the next step, the head moves right, the state transitions to q_1 and the tape does not change.
6. After the next two steps, the current symbol is * and the state is q_1. In the next step, the state goes to q_{rej}. The Turing machine halts and rejects.

A more complex Turing machine

The next example of a Turing machine has an input alphabet with just one symbol: {a}. The Turing machine accepts if the number of a's is a power of two. The tape alphabet contains three symbols: {a, x, *}. Although there are no x's on the tape at the beginning of the execution of the Turing machine, x symbols are written on the tape to help keep track of what is going on in the execution.

The Turing machine uses the fact that if a positive integer n is a power of two, then it is possible to repeatedly divide the number evenly by 2 until the number gets down to 1. The head of the Turing machine shuttles back and forth between the left end and the right end of the Turing machine. As the head moves from left to right, every other a is changed to an x. The Turing machine keeps track of whether an even or an odd number of a's have been encountered. If the head reaches the right end of the string after having seen an odd number of a's, the string is rejected. If the Turing machine reaches the end of the the input string after having encountered an even number of a's, the head moves back to the left end of the string and continues. The Turing machine loops, changing every other a to an x, until there is only one a in the string.

At the first step, the Turing machine changes the leftmost 'a' to a * symbol in order to mark the leftmost end of the input string. When the head moves left and encounters the * symbol, that triggers the head to change directions. The leftmost * symbols counts as an 'a' in determining whether there are an odd or even number of a's in the string.

The transition function for the Turing machine is given below so that the definition is precise. However, the transition function is not very helpful in understanding how and why the Turing machine works. It's best to understand the action of the Turing machine by actually running the Turing machine on different inputs. The tool below allows you to experiment with the execution of the Turing machine on different inputs. Remember that the input alphabet is just the single symbol a, so the tape has to be initialized to a sequence of a's following by blank symbols.

Table 17.2.3: Transition function for the Turing machine that recognizes powers of 2.

-	a	x	*
q_0	$(q_{first}, *, R)$	$(q_{rej}, *, R)$	(q_{rej}, x, R)
q_{first}	(q_{even}, x, R)	(q_{first}, x, R)	$(q_{acc}, *, R)$
q_{even}	(q_{odd}, a, R)	(q_{even}, x, R)	$(q_{ret}, *, L)$
q_{odd}	(q_{even}, x, R)	(q_{odd}, x, R)	$(q_{rej}, *, L)$
q_{ret}	(q_{ret}, a, L)	(q_{ret}, x, L)	$(q_{first}, *, R)$

PARTICIPATION ACTIVITY | 17.2.4: Turing machine that recognizes powers of 2.

Start Modify tape Next Reset

Symbol tape	a	a	a	a	*	*	*	*	*	*

Transition table for the Turing machine

	a	x	*
q_0	(q_{first}, *, R)	(q_{rej}, x, R)	(q_{rej}, *, R)
q_{first}	(q_{even}, x, R)	(q_{first}, x, R)	(q_{acc}, *, R)
q_{even}	(q_{odd}, a, R)	(q_{even}, x, R)	(q_{ret}, *, L)
q_{odd}	(q_{even}, x, R)	(q_{odd}, x, R)	(q_{rej}, *, L)
q_{ret}	(q_{ret}, a, L)	(q_{ret}, x, L)	(q_{first}, *, R)

PARTICIPATION ACTIVITY | 17.2.5: Power of 2 Turing machine.

The questions below refer to the Turing machine from the tool above whose input alphabet is {a}. The Turing machine accepts a string of a's if and only if the number of a's is a power of 2. Use the tool to answer the following questions.

1) If the Turing machine starts with input "aaaa", what are the contents of the tape after four steps?

○ axax

○ *xax

○ xaxa

2) If the Turing machine starts with input "aaaaa", what is the state of the Turing machine after 6 steps?

○ q_{rej}

○ q_{acc}

○ q[illegible]

3) How many times does the head move from left to right on input "aaaa" before it halts?

○ 1

○ 2

○ 3

CHALLENGE ACTIVITY | 17.2.1: Turing machine transitions.

422102.2723990.qx3zqy7

Start

Use the input and table to execute.

Input: babbbb

	a	b	*
q0	(q2, a, R)	(q0, b, R)	(q3, b, L)
q1	(q3, a, R)	(q0, *, R)	(q3, b, R)
q2	(q1, a, R)	(q1, a, R)	(q2, *, L)
q3	(q0, *, L)	(q1, *, L)	(q2, a, L)

First 6 characters of the tape after step 1: Ex: *abb*b

Select the state of the Turing Machine after each step:

q0 ⌄

1 2 3 4

Check Next

Additional exercises

EXERCISE 17.2.1: Simulating a Turing machine - example 1.

Here is a description of a Turing machine. The input alphabet is {a, b}. The state set is:

$$\{ q_0, q_1, q_2, q_3, q_4, q_{acc}, q_{rej} \}$$

The transition function is given in the table below:

	q_0	q_1	q_2	q_3	q_4
a	(q_1, a, R)	(q_1, a, R)	(q_2, a, R)	(q_{acc}, a, R)	(q_{rej}, a, R)
b	(q_2, b, R)	(q_1, b, R)	(q_2, b, R)	(q_{rej}, b, R)	(q_{acc}, b, R)
*	$(q_{rej}, *, R)$	$(q_3, *, L)$	$(q_4, *, L)$	$(q_{rej}, *, R)$	$(q_{rej}, *, R)$

(a) Draw the configuration of the Turing machine after 3 steps on input "aabba". When the Turing machine is in the initial configuration, it has executed zero steps.

(b) For how many steps does the Turing machine run on input string "a" before the Turing machine halts? Does the Turing machine accept the string "a"?

(c) Simulate the Turing machine on input "aaba". Does it accept? Draw the final two configurations of the Turing machine computation. In the last configuration, the Turing machine is either in the accept or reject state.

(d) Simulate the Turing machine on input "aabb". Does it accept? Draw the final two configurations of the Turing machine computation. In the last configuration, the Turing machine is either in the accept or reject state.

(e) Determine whether the Turing Machine accepts or rejects strings "bab" and "bba".

(f) Describe in words the conditions under which the Turing machine accepts the input string.

EXERCISE 17.2.2: Simulating a Turing machine - example 2.

Here is a description of a Turing machine. The input alphabet is {a, b}. The state set is:

$$\{ q_0, q_a, q_b, q_{ca}, q_{cb}, q_{left}, q_{acc}, q_{rej} \}$$

The transition function is given in the table below:

	q_0	q_a	q_b	q_{ca}	q_{cb}	q_{left}
a	$(q_a, *, R)$	(q_a, a, R)	(q_b, a, R)	$(q_{left}, *, L)$	$(q_{rej}, *, R)$	(q_{left}, a, L)
b	$(q_b, *, R)$	(q_a, b, R)	(q_b, b, R)	$(q_{rej}, *, R)$	$(q_{left}, *, L)$	(q_{left}, b, L)
*	$(q_{acc}, *, R)$	$(q_{ca}, *, L)$	$(q_{cb}, *, L)$	$(q_{acc}, *, R)$	$(q_{acc}, *, R)$	$(q_0, *, R)$

(a) Draw the configuration of the Turing machine after 3 steps on input "aabba". When the Turing machine is in the initial configuration, it has executed zero steps.

(b) For how many steps does the Turing machine run on input string "a" before the Turing machine halts? Does the Turing machine accept the string "a"?

(c) Simulate the Turing machine on input "aba". Does it accept? Draw the configuration of the Turing machine after the Turing machine executes 4 steps on input "aba".

(d) Simulate the Turing machine on input "abba". Does it accept? Draw the configuration of the Turing machine after the Turing machine executes 5 steps on input "abba".

(e) Simulate the Turing machine on input "abbbaa". Does it accept? Draw the final two configurations of the Turing machine computation. In the last configuration, the Turing machine is either in the accept or reject state.

(f) Describe in words the conditions under which the Turing machine accepts the input string.

EXERCISE 17.2.3: Turing machine design.

Design a Turing machine with input alphabet {a, b} that accepts a string if and only if the string has the property described.

(a) The input string has an even number of b's.

(b) The input string has the same number of a's and b's.

17.3 Decision problems and languages

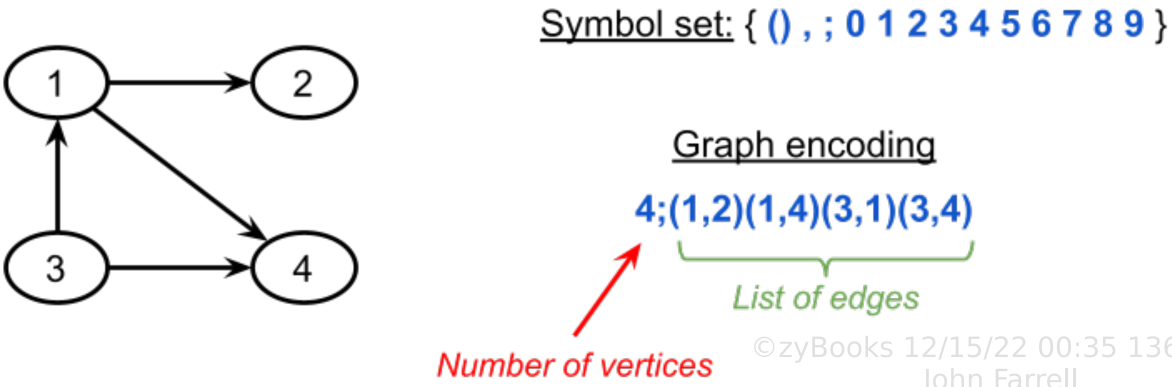
This section has been set as optional by your instructor.

At first glance Turing machines seem limited in terms of the kinds of problems they can solve. For example, Turing machines can only handle inputs in the form of strings, whereas many important computational problems operate on different kinds of mathematical objects such as graphs, boolean expressions, or numbers. Of course, any computer program receives input in the form of a string of 0's and 1's, so the fact that Turing machines require every input to be encoded as a string over a finite alphabet is not a special requirement of Turing machines alone.

Consider, for example, a problem whose input is a directed graph. A computer program that solves a graph problem might read the input graph from a text file. The program would read the characters in the file and interpret them in a particular way. One way to encode a graph would be to first indicate the number of vertices in the graph, followed by a semicolon. The labels for the vertices are the integers in the range from 1 up to and including the number of vertices in the graph. The encoding then lists each edge as an ordered pair in the format:

(*Vertex label of the tail of the edge*, *Vertex label of the head of the edge*)

There are many other perfectly acceptable ways to encode a graph as a string of symbols. The important point is that the input file is prepared using the same encoding as the program that will use the file, so that the contents of the input file are interpreted correctly. The picture below shows a small directed graph and a string encoding of the graph according to the encoding scheme described above.



Once a symbol set and method of encoding a graph are decided, there will be many strings that do not correspond to a valid graph. The Turing machine should be able to recognize if an input string is not a valid encoding of a graph and reject all such inputs immediately, just as a computer program will issue an error message and stop if an error is detected in the formatting of an input file.

PARTICIPATION ACTIVITY 17.3.1: Recognizing valid graph encodings.

1) According to the graph encoding scheme described above, which string corresponds to a valid encoding of a graph?

○ 4;(1,3)(2,3)(1,4)(5,3)(5,2)

○ (1,3)(2,3)(1,4)(5,3)(5,2)

○ 5;(1,3)(2,3)(1,4)(5,3)(5,2)

○ 5(1,3)(2,3)(1,4)(5,3)(5,2)

Another limitation of Turing machines as described is that their only output is to accept or reject an input string. The class of problems for which the answer is "Yes" or "No" are called ***decision problems***. Consider the following two problems:

- Decision problem: Given a Boolean expression, is there an assignment to the Boolean variables that causes the expression to evaluate to 1?
- Search problem: Given a Boolean expression, find an assignment to the Boolean variables that causes the expression to evaluate to 1 if one exists, or output that no such assignment exists.

Applications typically solve search problems rather than decision problems. However, in most cases, a search problem has a corresponding decision problem that captures the essential difficulty of the problem. That is, an efficient algorithm that solves the decision problem can be used to solve the corresponding search problem efficiently. Since decision problems are easier to reason about, a discussion of the computational complexity of a problem typically focuses on the decision version of the problem.

PARTICIPATION ACTIVITY | 17.3.2: Decision vs. Search problems.

Indicate whether the following problems are decision or search problems.

1) Given a list of cities and distances between each pair of cities, what is the shortest route that reaches every city?

○ Decision

○ Search

2) Given a list of cities and distances between each pair of cities, how long is the shortest route that reaches

every city?

○ Decision

○ Search

3) Given a list of cities and distances between each pair of cities, is there a route shorter than 1000 miles that visits every city?

○ Decision

○ Search

4) Given an integer N such that N > 1, is there an integer greater than one and less than N that evenly divides N?

○ Decision

○ Search

5) Given an integer N such that N > 1, find an integer greater than one and less than N that evenly divides N or indicate that no such integer exists.

○ Decision

○ Search

Consider a finite alphabet Σ. The set of all finite strings over Σ is denoted by Σ^*. Note that the strings themselves are finite but the set of all possible strings is infinite since there is no limit to the length of a string. For example, if B = {0, 1}, the set B^+ is: {0, 1, 00, 01, 10, 11, 000, ...}.

If Σ is a finite alphabet, then a subset of Σ^* is called a ***language*** over Σ. Every decision problem (along with an encoding of the input objects to strings in Σ^*) defines a language over Σ as follows: $x \in \Sigma^*$ is in the language if x is a valid encoding of an input and the answer to the decision problem on input x is "Yes". The string x is not in the language if either x does not correspond to a valid input or the answer on input x is "No".

Definition 17.3.1: Language computed by a Turing machine.

Let Σ denote a finite alphabet and let L be a language over Σ. A Turing machine M ***computes language L*** (or ***decides language L***) if for every $x \in \Sigma^*$, if $x \in L$, then M accepts x in a finite number of steps and if $x \notin L$, then M rejects x in a finite number of steps.

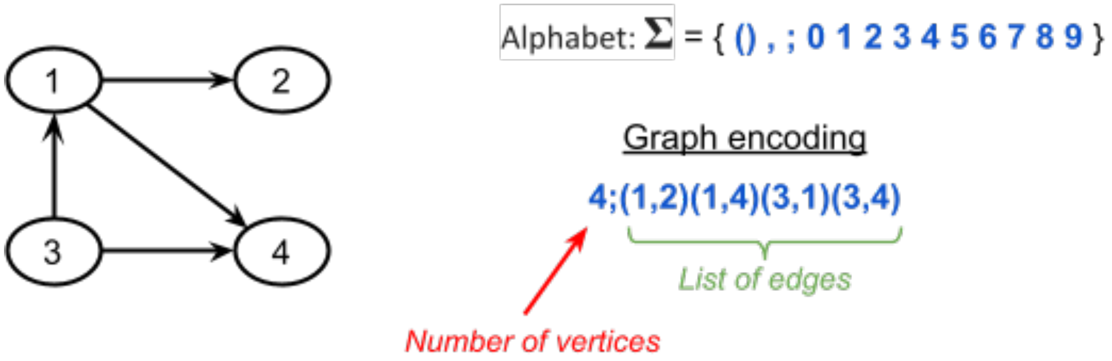
The model of Turing machines and the formulation of decision problems as languages provide a mathematically well-defined way to study what kinds of problems can be solved using any kind of computing device. Turing machines also provide a way to measure the resources used to solve a problem. ***Time complexity*** is measured by the number of steps taken by a Turing machine on a particular input. ***Space complexity*** is measured by the number of tape cells that the Turing machine uses in the course of its execution on a particular input.

PARTICIPATION ACTIVITY 17.3.3: Decision problems to languages.

Consider the following decision problem:

Given a graph G, does G have an even number of edges?

Define a language L corresponding to the decision problem that uses the graph encoding illustrated in the diagram below. The alphabet Σ for L consists of a comma, semicolon, left and right parentheses and all ten digits. Recall that the graph encoding indicates the number of vertices, followed by a semicolon, followed by a list of the edges in the graph:



A string is in the language if and only if the string corresponds to a valid encoding of a graph and the graph has an even number of edges. Indicate whether the following strings are in the language L.

1) 4;(1,2)(2,3)(3,4)(3,2)(1,4)

○ in L

○ not in L

2) 4;(a,b)(a,c)(c,d)(a,d)

○ in L

○ not in L

3) 4;(1,2)(2,3)(3,4)(3,2)

○ in L

○ not in L

4) (1,2)(2,3)(3,4)(3,2)

○ in L

○ not in L

5) 09(;;)23,4,3()3,4,5()));;;

○ in L

○ not in L

Additional information 17.3.1: Uncomputable problems.

Turing machines may not be the preferred model for devising algorithms to solve real problems, but they are useful for reasoning about what we can and cannot hope to solve with a computer. As observed earlier, the description of a Turing machine contains 6 components: the tape alphabet, the set of states, the start state, the accept state, the reject state, and the transition function. Each component can be expressed as a string of symbols over a finite alphabet. For example, the states could be denoted as: q0, q1, q2, etc. The tape characters could be denoted as s0, s1, s2, etc. Each transition rule can be expressed as a sequence, e.g., (q3, s2) → (q1, s17, L). Therefore, we can fix in advance a finite set of letters, digits, and punctuation symbols, and encode any Turing machine as a string of the selected characters. Encoding a Turing machine as a string is similar to representing a computer program as a string of symbols in a text file. Denote the encoding of Turing machine M as <M>. Since <M> is just a string over a finite alphabet, <M> can be used as the input to another Turing machine. Consider the following problem:

The Halting Problem

- Input: <M> the description of a Turing machine and an input x.
- Output: Accept if M halts on input x. Reject if M runs forever on input x.

It can be proven that there is no Turing machine that can compute the language defined by the Halting problem. A language is ***uncomputable*** if there is no Turing machine that computes the language. The fact that the halting problem is uncomputable has profound implications for areas in computer science such as program verification. For example, it is impossible to write an automatic verifier that takes as input a computer program and determines whether the program always terminates, or whether the program computes the correct answer to a particular problem.

Additional exercises

EXERCISE 17.3.1: Decision problems from search problems.

Give a decision problem corresponding to each of the search problems given below.

(a)

- Input: A set of classes to be scheduled. A list of pairs of the classes which can not be scheduled during the same period.
- Output: The largest set of classes that can all be scheduled during the same period.

(b)

- Input: A set of classes to be scheduled. A list of pairs of the classes which can not be scheduled during the same period.
- Output: A schedule for the classes that uses the smallest number of periods.

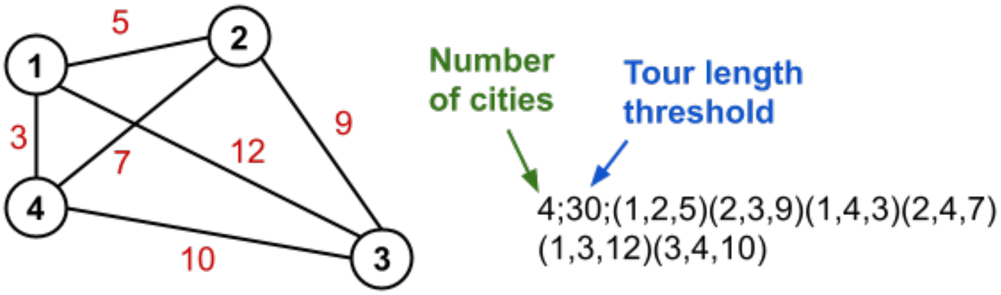
(c)

- Input: A list of items, each with a value and a weight. The values and weights of the items are positive integers. A positive integer W.
- Output: A subset of the items whose total weight is at most W and whose total value is as large as possible.

EXERCISE 17.3.2: Valid encodings of the Traveling Salesman Problem.

The Traveling Salesman Problem (TSP) is defined as follows: the input is a set of cities and a distance between each pair of cities. The problem is to find a tour of all the cities so that the salesman ends up at the same place he started so that he visits each city exactly once and travels the shortest possible distance. In the decision version of the problem, the input also includes a threshold k and the question is whether there is a tour that visits every city exactly once and whose total distance is at most k.

Here is an encoding of the Traveling Salesman Problem. The input alphabet is {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} along with a comma, semicolon, left and right parentheses. A valid encoding of a Traveling Salesman Problem will start with two numbers, each followed by a semicolon. The first number is the number of cities and the second number is the threshold for the tour. (Note that since we don't have the character "-" or ".", all numbers are non-negative integers). Each city will have a unique name which is a number in the range from 1 through n, where n is the number of cities. The encoding then lists triplets of the form (i, j, d) indicating that the distance between cities i and j is d. If the distance between two cities is given more than once or not at all, the encoding is not valid. The diagram below shows a picture of 4 cities and their distances along with a valid encoding for that particular input to the Traveling Salesman Problem:



The Traveling Salesman language is the set of all strings that correspond to valid encodings and are "yes" inputs for the Traveling Salesman decision problem. Which of the following strings are in the language and why?

(a) 4;10;(1,2,1)(1,3,5)(2,3,4)(1,4,5)(2,4,2)(3,4,3)

(b) 5;10;(1,2,1)(1,3,5)(2,3,4)(1,4,5)(2,4,2)(3,4,3)

(c) 4;11;(1,2,1)(1,3,5)(2,3,4)(1,4,5)(2,4,2)(3,4,3)

(d) 4;10;(1,2,1)(1,3,5)(1,4,5)(2,4,2)(3,4,3)

(e) 4;10;(1,2,1)(1,3,5)(2,3,4)(1,4,5)(2,4,2)(3,4,3)(2,4,2)

(f) 4;11;(((1,2,1)(1,3,5)8(2,3,4)(1,4,5)(2,4,2)(3,4,3)