



Formal Definition of Computation

Computation model

- The model of computation considered so far is the work performed by a finite automaton

Computation model

- The model of computation considered so far is the work performed by a finite automaton
- Finite automata were described informally, using state diagrams, and formally, as a 5-tuple

Computation model

- The model of computation considered so far is the work performed by a finite automaton
- Finite automata were described informally, using state diagrams, and formally, as a 5-tuple
- Informal description is easier to grasp first, but the formal definition is useful for making this notion precise, resolving any ambiguities that may occur in formal description

Formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1w_2 \dots w_n$ be a string over Σ

Formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1w_2 \dots w_n$ be a string over Σ

Then M accepts w if a sequence of states r_0, r_1, \dots, r_n exist in Q such that the following hold:

Formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1w_2 \dots w_n$ be a string over Σ

Then M accepts w if a sequence of states r_0, r_1, \dots, r_n exist in Q such that the following hold:

1. $r_0 = q_0$

Formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1w_2 \dots w_n$ be a string over Σ

Then M accepts w if a sequence of states r_0, r_1, \dots, r_n exist in Q such that the following hold:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, 1, \dots, n - 1$

Formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1w_2 \dots w_n$ be a string over Σ

Then M accepts w if a sequence of states r_0, r_1, \dots, r_n exist in Q such that the following hold:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, 1, \dots, n - 1$
3. $r_n \in F$

Interpretation

- **Condition (1) says** that the machine starts its computation in the start state

Interpretation

- **Condition (1) says** that the machine starts its computation in the start state
- **Condition (2) says** that as long as input is available the machine goes from state to state according to its transition function δ

Interpretation

- **Condition (1) says** that the machine starts its computation in the start state
- **Condition (2) says** that as long as input is available the machine goes from state to state according to its transition function δ
- **Condition (3) says** that the machine accepts its input if it ends up in an accept state

Interpretation

- **Condition (1) says** that the machine starts its computation in the start state
- **Condition (2) says** that as long as input is available the machine goes from state to state according to its transition function δ
- **Condition (3) says** that the machine accepts its input if it ends up in an accept state
- **Can you translate this** interpretation into the computation performed by a program?

Translation

A program behaves like a finite automaton:

Translation

A program behaves like a finite automaton:

- Its **start state** is a function mapping program variables to their initial values

Translation

A program behaves like a finite automaton:

- Its **start state** is a function mapping program variables to their initial values
- **Program execution** goes from state to state by transitions performed according to program control flow

Translation

A program behaves like a finite automaton:

- Its **start state** is a function mapping program variables to their initial values
- **Program execution** goes from state to state by transitions performed according to program control flow
- The **language** of this machine is the class of problems solved by program execution

Difference

A program **computation differs** from FA computation because:

Difference

A program **computation differs** from FA computation because:

- A **program** may have a potential infinite set of states and can run forever

Difference

A program **computation differs** from FA computation because:

- A **program** may have a potential infinite set of states and can run forever
- During **execution** a program may interact with its environment

Difference

A program **computation differs** from FA computation because:

- A **program** may have a potential infinite set of states and can run forever
- During **execution** a program may interact with its environment
- The **accepting state** of the program has a larger interpretation

Note

We say that a machine M recognizes the language A if
 $A = \{w \mid M \text{ accepts } w\}$

How is this interpreted in terms of a program execution?

Regular language

Definition 1.7 A language is called a **regular language** if some finite automaton recognizes it

Example computation

Take the machine M_5 from Example 1.5 and

$w = 10\langle RESET \rangle 22\langle RESET \rangle 012$

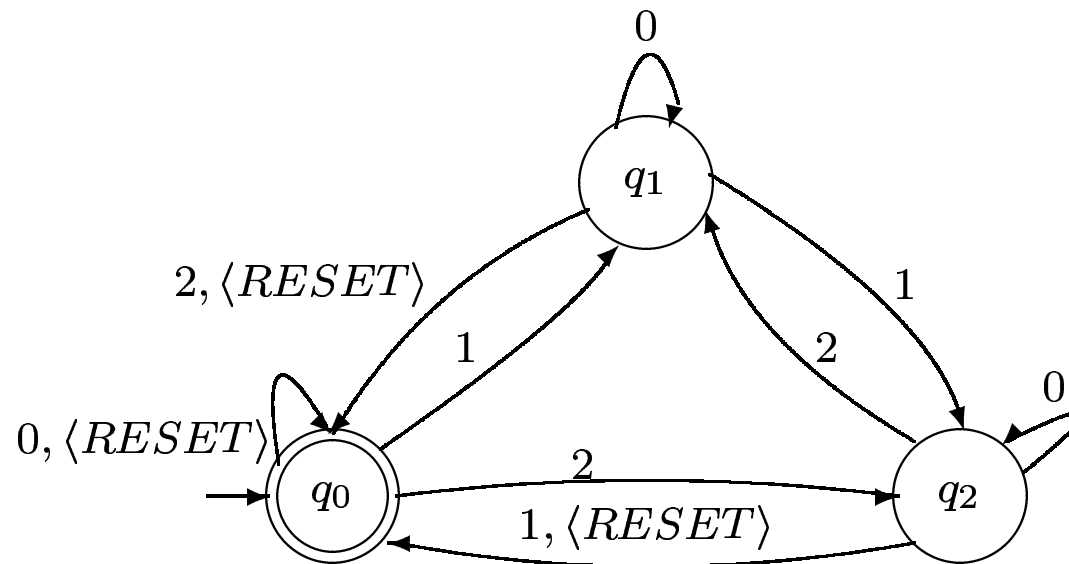


Figure 1: Finite automaton M_5

Running M_5

M_5 accepts w according to the formal definition of computation because the sequence of states it enters when computing on w is $q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$:

Running M_5

M_5 accepts w according to the formal definition of computation because the sequence of states it enters when computing on w is $q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$:

- q_0 is the first state by definition

Running M_5

M_5 accepts w according to the formal definition of computation because the sequence of states it enters when computing on w is $q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$:

- q_0 is the first state by definition
- $\delta(q_0, 1) = q_1, \delta(q_1, 0) = q_1, \delta(q_1, \langle RESET \rangle) = q_0$

Running M_5

M_5 accepts w according to the formal definition of computation because the sequence of states it enters when computing on w is $q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$:

- q_0 is the first state by definition
- $\delta(q_0, 1) = q_1, \delta(q_1, 0) = q_1, \delta(q_1, \langle RESET \rangle) = q_0$
- $\delta(q_0, 2) = q_2, \delta(q_2, 2) = q_1, \delta(q_1, \langle RESET \rangle) = q_0$

Running M_5

M_5 accepts w according to the formal definition of computation because the sequence of states it enters when computing on w is $q_0, q_1, q_1, q_0, q_2, q_1, q_0, q_0, q_1, q_0$:

- q_0 is the first state by definition
- $\delta(q_0, 1) = q_1, \delta(q_1, 0) = q_1, \delta(q_1, \langle RESET \rangle) = q_0$
- $\delta(q_0, 2) = q_2, \delta(q_2, 2) = q_1, \delta(q_1, \langle RESET \rangle) = q_0$
- $\delta(q_0, 0) = q_0, \delta(q_0, 1) = q_1, \delta(q_1, 2) = q_0$

The language $L(M_5)$

$L(M_5) = \{w \mid \text{the sum of the symbols in } w \text{ is 0 modulo 3, except that } \langle RESET \rangle \text{ resets the count to 0}\}$

Since M_5 is a finite automaton $L(M_5)$ is a regular language

Designing finite automata

- Whether it be of **automata or artwork**, design is a creative process

Designing finite automata

- Whether it be of **automata or artwork**, design is a creative process
- Consequently it **cannot be reduced to a simple recipe or formula**

Designing finite automata

- Whether it be of **automata or artwork**, design is a creative process
- Consequently it **cannot be reduced to a simple recipe or formula**
- However, **for a given design task** one can find a particular approach useful; here we consider a useful approach for designing various types of automata

-
-
-

The approach

Be yourself the machine you are trying to design!

The approach

Be yourself the machine you are trying to design!

- As the machine you try to design, **answer the question:** how would you go about performing the machine's task?

The approach

Be yourself the machine you are trying to design!

- As the machine you try to design, **answer the question:** how would you go about performing the machine's task?
- **Pretending that you are the machine** is a psychological trick that helps engage your whole mind in the design process

Using the approach

Suppose that you are given some language and want to design a finite automaton that recognizes it

Remember: a language is a set of strings over a given alphabet!

Pretending to be the automaton

- Pretending that you are the automaton, you receive an input string and must determine whether it belongs to the language

Pretending to be the automaton

- Pretending that you are the automaton, you receive an input string and must determine whether it belongs to the language
- You got to see the symbols in the string one by one. After each symbols you must decide whether the string seen so far is in the language.

Pretending to be the automaton

- Pretending that you are the automaton, you receive an input string and must determine whether it belongs to the language
- You got to see the symbols in the string one by one. After each symbols you must decide whether the string seen so far is in the language.
- The reason is that you, like the machine, don't know when the end of the string is coming; so you must always be ready with the answer.

Making decisions

- In order to make decisions, you have to figure out what you need to remember about the input string as you are reading it

Making decisions

- In order to make decisions, you have to figure out what you need to remember about the input string as you are reading it
- Bear in mind that as you are pretending to be a finite automaton, you have a finite number of states and thus a finite memory, say a single sheet of paper

Making decisions

- In order to make decisions, you have to figure out what you need to remember about the input string as you are reading it
- Bear in mind that as you are pretending to be a finite automaton, you have a finite number of states and thus a finite memory, say a single sheet of paper
- Fortunately, you don't need to remember entire input, you only need to remember certain crucial information

Note

Which information is crucial depends on the particular language considered

Example design

Suppose $\Sigma = \{0, 1\}$ and the language consists of all strings with an odd number of 1s. You want to construct a finite automaton E_1 to recognize this language

Constructing the states

- Start getting an input string of 0s and 1s symbol by symbol

Constructing the states

- Start getting an input string of 0s and 1s symbol by symbol
- Remember whether the number of 1s seen so far is even or odd;

Constructing the states

- Start getting an input string of 0s and 1s symbol by symbol
- Remember whether the number of 1s seen so far is even or odd;
- Represent the information you need as a finite list: (1) even so far, (2) odd so far

Constructing the states

- Start getting an input string of 0s and 1s symbol by symbol
- Remember whether the number of 1s seen so far is even or odd;
- Represent the information you need as a finite list: (1) even so far, (2) odd so far
- Assign a state to each possibility, Figure 2

Constructing transitions

To construct transitions you need to observe the way of going between possibilities while reading the input

Example: the transition to flip state on a 1 and stay put on 0 is shown in Figure 3

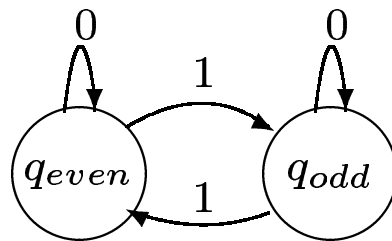


Figure 3: Adding the start and accept state

Start state and final states

- **Set the start state** to the state corresponding to the possibility associated with having seen 0 symbols so far (i.e., the empty string ϵ)

Start state and final states

- **Set the start state** to the state corresponding to the possibility associated with having seen 0 symbols so far (i.e., the empty string ϵ)
- **In our example** this correspond to q_{even}

Start state and final states

- **Set the start state** to the state corresponding to the possibility associated with having seen 0 symbols so far (i.e., the empty string ϵ)
- **In our example** this correspond to q_{even}
- **Set the accept** states to be those corresponding to possibilities where you want to accept the input

Start state and final states

- **Set the start state** to the state corresponding to the possibility associated with having seen 0 symbols so far (i.e., the empty string ϵ)
- **In our example** this correspond to q_{even}
- **Set the accept** states to be those corresponding to possibilities where you want to accept the input
- **In our example** this is the set $\{q_{odd}\}$

The result

The result of the example considered above is in Figure 4

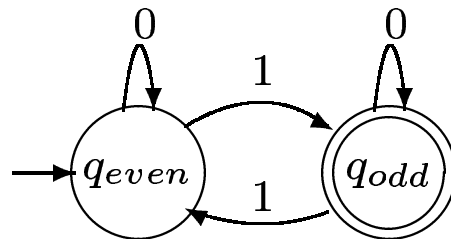


Figure 4: Adding the start and accept state

Example 1.9

This example shows how to design a finite automaton E_2 to recognize the regular language of all strings over the alphabet $\Sigma = \{0, 1\}$ that contain the string 001 as a substring

Example:

- strings 0010, 1001, 001, 11111110011111 are in that language

Pretending to be E_2

- As symbols come in, you would initially skip over all 1s

Pretending to be E_2

- As symbols come in, you would initially skip over all 1s
- If you come to a 0, you note that you may have just seen the first of the three symbols in the pattern 001

Pretending to be E_2

- As symbols come in, you would initially skip over all 1s
- If you come to a 0, you note that you may have just seen the first of the three symbols in the pattern 001
- If after the first 0 you see a 1, then there were too few 0s so you back to skipping over 1s

Pretending to be E_2

- As symbols come in, you would initially skip over all 1s
- If you come to a 0, you note that you may have just seen the first of the three symbols in the pattern 001
- If after the first 0 you see a 1, then there were too few 0s so you back to skipping over 1s
- If after the first 0 you see another 0 then you should remember that you have seen two symbols of the pattern

Pretending to be E_2

- As symbols come in, you would initially skip over all 1s
- If you come to a 0, you note that you may have just seen the first of the three symbols in the pattern 001
- If after the first 0 you see a 1, then there were too few 0s so you back to skipping over 1s
- If after the first 0 you see another 0 then you should remember that you have seen two symbols of the pattern
- Once you have seen two symbols of the pattern you need to scan the input until you see a 1; if you find it, remember that you succeeded to find the pattern and

The list of possibilities

1. Haven't see any symbol of the pattern
2. Have just seen a 0
3. Have just seen 00
4. Have seen the entire pattern 001

Designing states and transitions

- Assign the states q, q_0, q_{00}, q_{001} to the four possibilities

Designing states and transitions

- Assign the states q, q_0, q_{00}, q_{001} to the four possibilities
- In state q : while reading 1 remain in q , but reading 0 you move to q_0

Designing states and transitions

- Assign the states q, q_0, q_{00}, q_{001} to the four possibilities
- In state q : while reading 1 remain in q , but reading 0 you move to q_0
- In state q_0 : reading 0 you transit to q_{00} but reading 1 you transit to q

Designing states and transitions

- Assign the states q, q_0, q_{00}, q_{001} to the four possibilities
- In state q : while reading 1 remain in q , but reading 0 you move to q_0
- In state q_0 : reading 0 you transit to q_{00} but reading 1 you transit to q
- In state q_{00} : reading 1 you move to q_{001} while reading 0 you remain in state q_{00}

Designing states and transitions

- Assign the states q, q_0, q_{00}, q_{001} to the four possibilities
- In state q : while reading 1 remain in q , but reading 0 you move to q_0
- In state q_0 : reading 0 you transit to q_{00} but reading 1 you transit to q
- In state q_{00} : reading 1 you move to q_{001} while reading 0 you remain in state q_{00}
- In state q_{001} : whatever you read remain in q_{001} .

Start and final states

- Start state is q

The result is in Figure 5

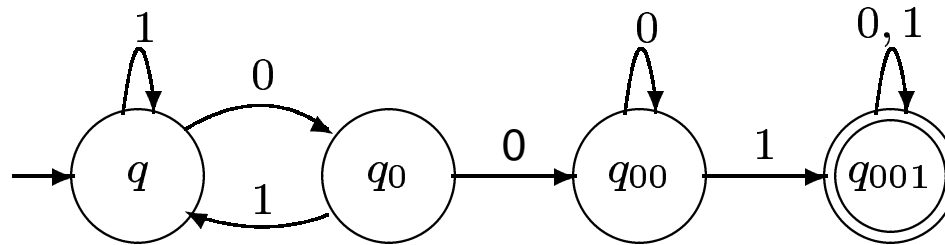


Figure 5: Accepting strings containing 001

Another example

Design a DFA that recognizes the languages:

$$L_1 = \{w \mid w \text{ begins with 1 and ends with 0} \}$$

$$L_2 = \{w \mid w \text{ is any string except a and b}\}$$