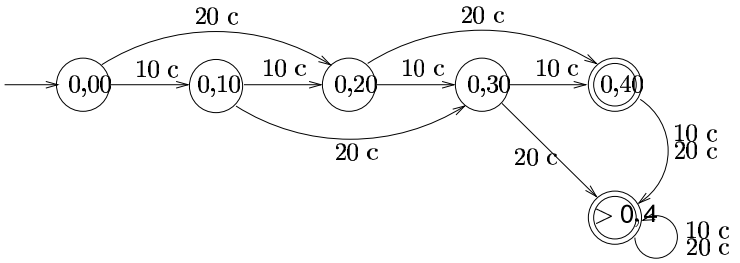
FINITE STATE AUTOMATA AND REGULAR LANGUAGES

2.1 State graphs and state tables

First we examine computing systems, with only a finite number of possible states. Such a system may be described as a *finite state automaton* (*finite state machine*).

There are many ways to describe a finite state automaton: state graphs, state tables, etc.

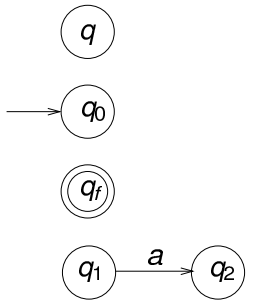
Example 1: Coffee automaton.



The automaton represented by the previous state graph solves the decision problem "is there enough money to buy coffee?"

In general, finite state automata can be used to model solutions to simple decision problems. There are automata models that give output other than yes/no (Moore automata and Mealy automata), but this course does not cover them.

Notation in state graphs:



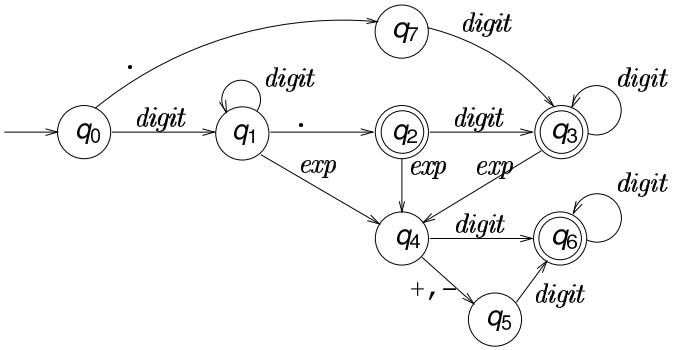
State q of the automaton

Initial state

Final state: the automaton "accepts" the input string iff it is in a final state at the end of input

A transition from state q_1 to state q_2 caused by input symbol a

Example 2: Unsigned real numbers of programming language C.



Abbreviations used: *digit* $= \{0, 1, \ldots, 9\}$, *exp* $= \{\mathtt{E}, \mathtt{e}\}$.

A finite state automaton presented as a *state table*: the new state of the automaton as a function of the old state and the input symbol.

E.g. the state table of the real number automaton:

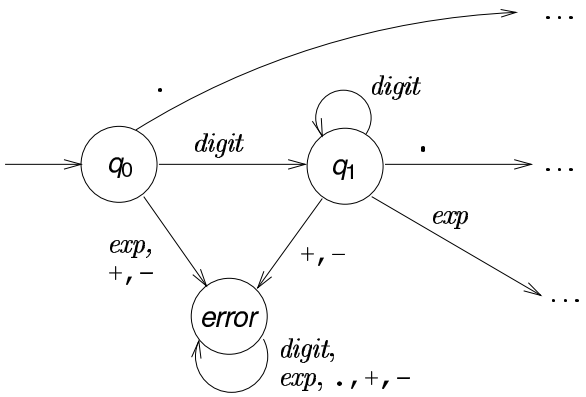
		digit	.	*exp*	+	−
→	q_0	q_1	q_7			
	q_1	q_1	q_2	q_4		
←	q_2	q_3		q_4		
←	q_3	q_3		q_4		
	q_4	q_6			q_5	q_5
	q_5	q_6				
←	q_6	q_6				
	q_7	q_3				

Q: What are the empty positions in a state table?

A: The empty positions in a state table, or analogously the "missing" arcs in a state graph, represent error situations. If the automaton moves here, the input string will be rejected.

Formally it is thought that the automaton has a special error state, which is omitted from the graph for clarity.

E.g. a complete state graph of the real number automaton would be:



and a complete state table of the real number automaton would be

		digit	.	*exp*	+	−
→	q_0	q_1	q_7	*error*	*error*	*error*
	q_1	q_1	q_2	q_4	*error*	*error*
⋮	⋮	⋮	⋮	⋮	⋮	⋮
←	q_6	q_6	*error*	*error*	*error*	*error*
	error	*error*	*error*	*error*	*error*	*error*

2.2 Programming based on finite state automata

Given a finite state automaton it is simple to design a program that functions like the automaton. E.g. syntax testing an input string based on the real number automaton:

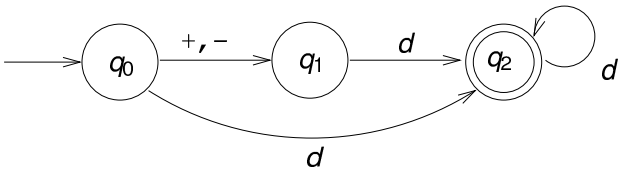

```
#include <stdio.h>
#include <ctype.h>
main() {
   int q, c;
   q = 0;
   while ((c = getchar()) != '\n')
      switch (q) {
      case 0:
         if (isdigit(c)) q = 1;
         else if (c == '.') q = 7;
         else q = 99;
         break;
      case 1:
         if (isdigit(c)) q = 1;
         else if (c == '.') q = 2;
         else if (c == 'E' || c == 'e') q = 4;
         else q = 99;
         break;
```



```
      ...
      case 99:
         break;
      }
   if (q == 2 || q == 3 || q == 6)
      printf("INPUT IS A REAL NUMBER.\n");
   else
      printf("INPUT IS NOT A REAL NUMBER.\n");
}
```


Adding semantics to finite state automata

Example. An automaton that recognizes octal numbers and computing the value of the input string (“conversion to base 10”).



Abbreviation $d = \{0, 1, \ldots, 7\}$.

Just the syntax testing:

```
#include <stdio.h>
#include <ctype.h>

main()
{
   int q, c;
   q = 0;
   while ((c = getchar()) != '\n') {
      switch (q) {
      case 0:
         if (c == '+' || c == '-') q = 1;
         else if ('0' <= c && c <= '7') q = 2;
         else q = 99;
         break;
```



```
      case 1:
         if ('0' <= c && c <= '7') q = 2;
         else q = 99;
         break;
      case 2:
         if ('0' <= c && c <= '7') q = 2;
         else q = 99;
         break;
      case 99:
         break;
      }
   }
   if (q == 2)
      printf("INPUT OK.\n");
   else
      printf("ILLEGAL NUMBER.\n");
}
```


Adding operations to compute value of input:

```
#include <stdio.h>

int main(void)  {
  int q, c;
  int sgn, val;       /* SEM: sgn = sign  */
  sgn = 1; val = 0;   /* SEM: val = absolute value */
  q = 0;
```



```
  while ((c = getchar()) != '\n') {
    switch (q) {
    case 0:
      if (c == '+') q = 1;
      else if (c == '-') {
        sgn = -1;                          /* SEM */
        q = 1;
      }
      else if ('0' <= c && c <= '7') {
        val = c - '0';                     /* SEM */
        q = 2;
      }
      else q = 99;
      break;
```

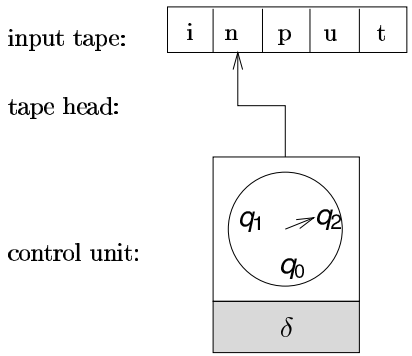
```
case 1:
  if ('0' <= c && c <= '7') {
    val = c - '0';                    /* SEM */
    q = 2;
  }
  else q = 99;
  break;
case 2:
  if ('0' <= c && c <= '7') {
    val = 8 * val + (c - '0');        /* SEM */
    q = 2;
  }
  else q = 99;
  break;
case 99:
  break;
}
```



```
  }
  if (q == 2)
  { printf("VALUE OF INPUT IS %d.\n", sgn*val);/*SEM*/
    exit(0); }
  else
  { printf("ILLEGAL INPUT.\n"); /* SEM */
    exit(1); }
}
```


2.3 Formalizing the concept of finite state automata

A mechanistic model:



A finite state automaton *M* consists of a *control unit* with a finite number of states; the control unit is regulated by the *transition function* δ of the automaton; and an *input tape* divided into symbol positions; and a *tape head* that connects the control unit and tape and points at a single tape symbol at a time.

The "working" of the automaton:

The automaton starts in a special *initial state* q_0, so that the input is written on the input tape and the tape head points at the first symbol of the input.

In one step the automaton reads the symbol at the tape head, decides the new state of the control unit based on the old state and the symbol it just read, and moves the tape head one symbol forwards.

The automaton stops when the last input symbol has been handled. At this point the automaton *accepts* the input, if it is in one of the *accepting final states*, and *rejects* the input otherwise.

The automaton *recognizes a language* that consists of all the strings it accepts.

An exact formulation:

A finite state machine is a 5-tuple

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

where

- Q is the finite set of *states*;
- Σ is the *input alphabet*;
- $\delta : Q \times \Sigma \to Q$ is the *transition function*;
- $q_0 \in Q$ is the *initial state*;
- $F \subseteq Q$ is the set of *accepting final states*.

Example. A formal presentation of the real number automaton:

$$\begin{aligned} M \;=\; & (\{q_0, \ldots, q_7, \mathit{error}\}, \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}, \mathtt{.}, \mathtt{E}, \mathtt{e}, \mathtt{+}, \mathtt{-}\}, \\ & \delta, q_0, \{q_2, q_3, q_6\}), \end{aligned}$$

where δ is as in the table previously; e.g.

$$\begin{aligned} & \delta(q_0, \mathtt{0}) = \delta(q_0, \mathtt{1}) = \cdots = \delta(q_0, \mathtt{9}) = q_1, \\ & \delta(q_0, \mathtt{.}) = q_7, \quad \delta(q_0, \mathtt{E}) = \delta(q_0, \mathtt{e}) = \mathit{error}, \\ & \delta(q_1, \mathtt{.}) = q_2, \quad \delta(q_1, \mathtt{E}) = \delta(q_1, \mathtt{e}) = q_4, \end{aligned}$$

jne.

The *configuration* of an automaton $(q, w) \in Q \times \Sigma^*$; in particular, *the initial configuration with input x* is the pair (q_0, x).

Intuition: q is the state of the automaton and w is the remaining part of the input, i.e., the part of the input on or to the right of the tape head.

The configuration (q, w) *directly leads to* configuration (q', w'), denoted by

$$(q, w) \underset{M}{\vdash} (q', w'),$$

if $w = aw'$ ($a \in \Sigma$) and $q' = \delta(q, a)$. It can also be said that (q', w') is an *immediate successor* of (q, w).

Intuition: in the state q upon reading the first symbol a of the string $w = aw'$ from the tape the automaton moves to state q' and moves the tape head one step forward, and leaves the string w' on the tape.

If the automaton M is obvious from the context, the relation can be denoted by

$$(q, w) \vdash (q', w').$$

The configuration (q, w) *leads to* (q', w') or the configuration (q', w') is a *successor* of (q, w), denoted with

$$(q, w) \vdash_M^* (q', w'),$$

if there exists a sequence of configurations (q_0, w_0), (q_1, w_1), ..., (q_n, w_n), $n \geq 0$, such that

$$(q, w) = (q_0, w_0) \vdash_M (q_1, w_1) \vdash_M \cdots \vdash_M (q_n, w_n) = (q', w')$$

As a special case with $n = 0$ we have $(q, w) \vdash_M^* (q, w)$ for all (q, w).

Again, if M is obvious from the context, we denote

$$(q, w) \vdash^* (q', w').$$

The automaton M *accepts* the string $x \in \Sigma^*$, if we have

$$(q_0, x) \vdash_M^* (q_f, \varepsilon) \qquad \text{for some } q_f \in F;$$

otherwise M *rejects* x.

In other words: the automaton accepts x, if its initial configuration with input x leads, at the end of the input, to some accepting final configuration.

The language M *recognizes* is defined by:

$$L(M) = \{x \in \Sigma^* \mid (q_0, x) \vdash_M^* (q_f, \varepsilon) \quad \text{for some } q_f \in F\}.$$

Example: handling the input string "`0.25E2`" with the real number automaton:

$$\begin{array}{lllll} (q_0, 0.25\text{E}2) & \vdash & (q_1, .25\text{E}2) & \vdash & (q_2, 25\text{E}2) \\ & \vdash & (q_3, 5\text{E}2) & \vdash & (q_3, \text{E}2) \\ & \vdash & (q_4, 2) & \vdash & (q_6, \varepsilon). \end{array}$$

Since $q_6 \in F = \{q_2, q_3, q_6\}$, we have $0.25\text{E}2 \in L(M)$.