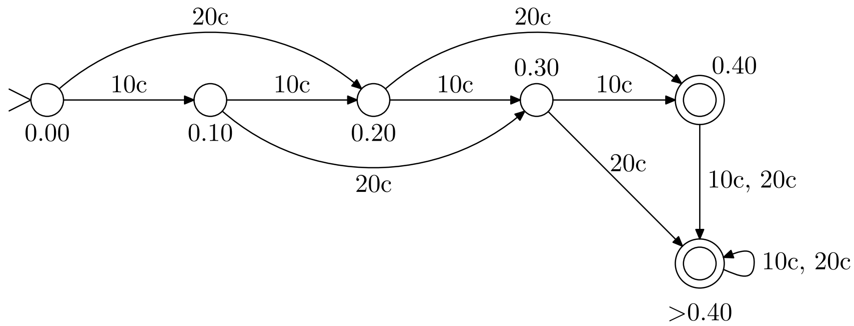
Solutions to Supplementary Problems

Problem S1.1 Formulate the model of a simple coffee machine presented at Lecture 2 (slide 2 of Sec. 2.1) precisely according to the mathematical definition of a finite automaton (slides 3–4 of Sec. 2.3). What is the formal language recognised by this automaton?

Solution. Recall the state diagram of the coffee machine automaton:



Formally, a finite automaton is represented as a tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ a finite alphabet, δ a function $Q \times \Sigma \to Q$, $q_0 \in K$ the initial state and $F \subseteq Q$ the set of accepting states.

In the coffee machine automaton the parts are defined as follows:

$$
\begin{aligned}
Q &= \{0.00, 0.10, 0.20, 0.30, 0.40, {>}0.40\} \\
\Sigma &= \{10\text{c}, 20\text{c}\} \\
q_0 &= \{0.00\} \\
F &= \{0.40, {>}0.40\}
\end{aligned}
$$

The easiest way to display the transition function δ is as a table.

q	$\delta(q, 10\text{c})$	$\delta(q, 20\text{c})$
0.00	0.10	0.20
0.10	0.20	0.30
0.20	0.30	0.40
0.30	0.40	>0.40
0.40	>0.40	>0.40
>0.40	>0.40	>0.40

A configuration $\omega \in Q \times \Sigma^*$ of a finite automaton M contains its current state and the unprocessed part of the input string. In each step, the automaton reads one symbol of the input and moves to a new state according to this symbol and the current state, as determined by the transition function δ.

If the automaton is in an accepting state when the input ends, the input is accepted, otherwise it is rejected. The language $\mathcal{L}(M)$ accepted by M consists of all the input strings it accepts. For the coffee machine automaton this set is:

$$\mathcal{L}(M) = \{w_1 w_2 \ldots w_n \mid w_i \in \Sigma \text{ for all } 1 \leq i \leq n \text{ and } \sum_{i=1}^{n} w_i \geq 40\ c\}$$

So the machine accepts all the strings (in this case maybe more naturally called 'sequences') in which the sum of the given coins is 40 c or more.

Let us consider a few input sequences and how the machine operates on them.

- $w =$ 0.10c 0.10c 0.20c:

  $$\begin{aligned}(0.00; 0.10\text{c } 0.10\text{c } 0.20\text{c}) &\vdash_M (0.10; 0.10\text{c } 0.20\text{c}) \\ &\vdash_M (0.20; 0.20\text{c}) \vdash_M (0.40; \varepsilon)\end{aligned}$$

  Because $0.40 \in F$ the sequence is accepted. Here the notation $\omega \vdash_M \omega'$ means that machine M moves from configuration $\omega = (q, w)$ to configuration $\omega' = (q', w')$ in one step.

- $w =$ 0.20c 0.10c:

  $$(0.00; 0.20\text{c } 0.10\text{c}) \vdash_M (0.20; 0.10\text{c}) \vdash_M (0.30; \varepsilon)$$

  Because $0.30 \notin F$, the sequence is rejected.

- $w =$ 0.20c 0.20c 0.20c:

  $$(0.00; 0.20\text{c } 0.20\text{c } 0.20\text{c}) \vdash_M^* (>0.40; \varepsilon)$$

  The sequence is accepted. The notation $\omega \vdash_M^* \omega'$ means that machine M moves from configuration ω to configuration ω' in some number of steps (where the number of steps can also be zero).

Problem S2.2 Design finite automata that recognise the following languages:

(i) $\{a^m b^n \mid m = n \mod 3\}$;

(ii) $\{w \in \{a, b\}^* \mid w \text{ contains equally many } a\text{'s and } b\text{'s, modulo } 3\}$.

(The notation "$m = n \mod 3$" means that the numbers m and n yield the same remainder when divided by three.)

Solution.

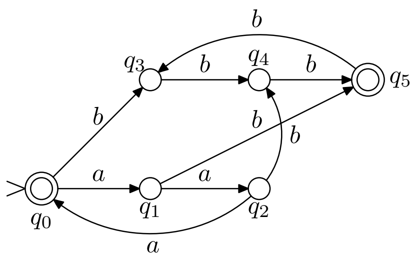
(i) The language $L = \{a^m b^n \mid m = n \mod 3\}$ can be recognised by the finite automaton:

$$
\begin{aligned}
M &= (Q, \Sigma, \delta, q_0, F) \\
Q &= \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\} \\
\Sigma &= \{a, b\} \\
F &= \{q_0, q_5\}
\end{aligned}
$$

The state transition function δ is:

q	$\delta(q,a)$	$\delta(q,b)$
q_0	q_1	q_3
q_1	q_2	q_5
q_2	q_0	q_4
q_3	q_6	q_4
q_4	q_6	q_5
q_5	q_6	q_3
q_6	q_6	q_6

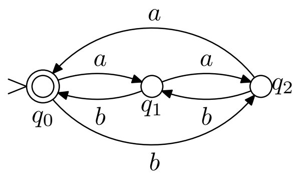
The state q_6 is here used as a "rejecting" state. This is a state where the automaton moves when it becomes clear that the input string cannot belong to the target language (in this case when a substring ba is encountered), and stays there until the end of the input. Such states are often left out when an automaton is represented as a state diagram. This is also the case in the diagram below:



Intuition for the automaton: With the states in the lower row the automaton counts 'up' on the number of a's encountered (modulo 3), and when a b occurs, it changes to the upper row to count 'down' on the number of b's encountered (modulo 3). The input string is accepted if in the end the up-counts and down-counts are equal (modulo 3).

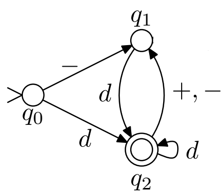
For instance, from state q_2 on the lower row ("we have seen $k * 3 + 2$ a's for some $k \in \mathbb{N}$"), the automaton moves on a b to state q_4 on the upper row ("we need to see exactly $\ell * 3 + 1$ more b's for some $\ell \in \mathbb{N}$").

(ii) The language $L = \{w \in \{a,b\}^* \mid w$ contains equally many a's and b's (modulo 3)$\}$ is recognised by the following finite automaton, designed using the same idea as in part (i):



Problem S2.3 Design a finite automaton that recognises sequences of nonnegative integers separated by plus and minus signs, where in addition the sequence may start with a minus sign (e.g. `11+20-9`, `-5+8`). Implement your automaton as a computer program that also calculates the numerical value of the input expression.

Solution.
The syntactically correct input sequences can be recognised with the following automaton:



Here d is a shorthand notation for "any numeral from the set $\{0, \ldots, 9\}$".

Below is a C program that reads user input and performs the intended summation.

```
#include <stdio.h>
#include <ctype.h>
int main (void)
{
  int q = 0; /* The current state */
  int c;     /* The current input character */
  int sgn = 1, val = 0, sum = 0; /* Some auxiliary variables */
  while ((c = getchar()) != '\n') {
    switch (q) {
    case 0:
      if (c == '-') {
        sgn = -1;
        q = 1;
      }
      else if (isdigit(c)) {
        val = c - '0';
        q = 2;
      }
```

```
      else q = 99;
      break;
    case 1:
      if (isdigit(c)) {
        val = c - '0';
        q = 2;
      }
      else q = 99;
      break;
    case 2:
      if (isdigit(c)) {
        val = 10 * val + (c - '0');
        q = 2;
      }
      else if (c == '+') {
        sum = sum + val*sgn;
        val = 0;
        sgn = 1;
        q = 1;
      }
      else if (c == '-') {
        sum = sum + val*sgn;
        val = 0;
        sgn = -1;
        q = 1;
      }
      else q = 99;
      break;
    case 99:
      break;
    }
  }
  sum = sum + sgn*val;
  if (q == 2) printf("The value of the expression is %d.\n", sum);
  else printf("The expression is not well-formed.\n");
  return 0;
}
```