Languages and Computation (COMP 2049) Lab 04
Regular Grammars

(1) Consider the floating-point numbers from Exercise (1) of the second lab session, except that we restrict the digits to the set $\{0, 1\}$. These binary floating-point numbers must be formed according to the following rules:

- Each number may be signed or unsigned.
  - unsigned as in `1.01`, signed as in `+1.01` or `-1.01`;
- The numerical part (also called the value field) must start with a non-empty sequence of digits.
  - For instance, in the number `+110.011`, the value field is `110.011`, which starts with the sequence of digits `110`.
- The value field may optionally include a decimal point '.', in which case it must be followed by some other digits;
  - `1` and `1.01` are acceptable, but `1.` is not acceptable.
- There may be an optional exponent field, in which case, it must contain the letter 'e', followed by a signed or an unsigned integer.
  - For instance, `101e+11` or `-1.11e101` are acceptable, but `1.01e` and `1.01e-` are not acceptable.

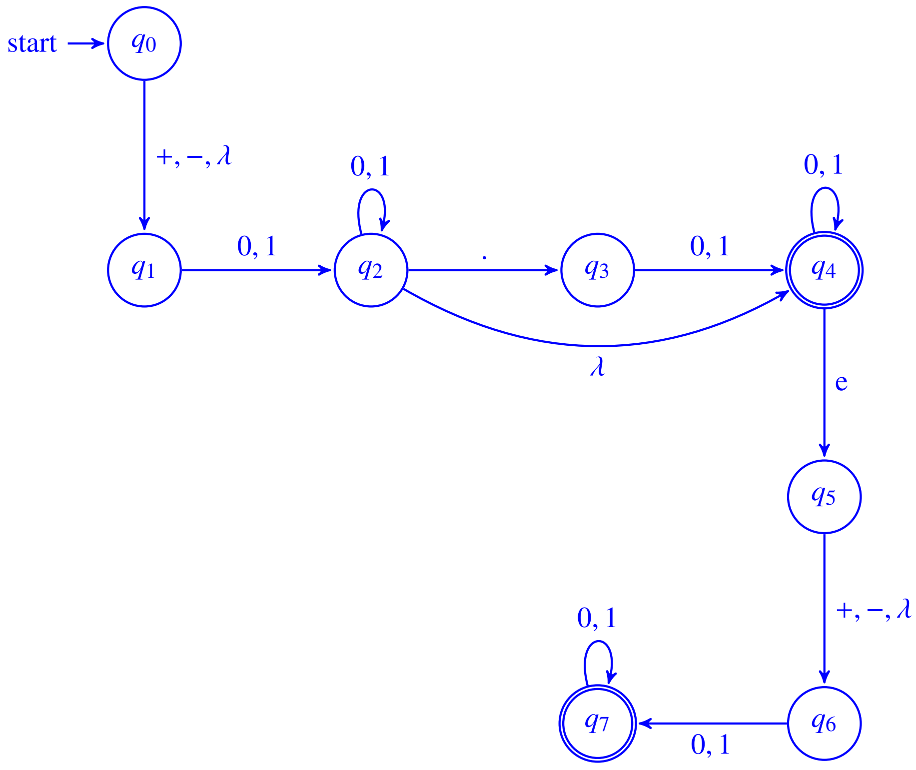
The following grammar, in which $\langle \text{ number } \rangle$ is the start symbol, generates this language:

$$
\begin{aligned}
\langle \text{ number } \rangle &\to \langle \text{ sign } \rangle \langle \text{ digits } \rangle \langle \text{ rest } \rangle \\
\langle \text{ sign } \rangle &\to + \mid - \mid \lambda \\
\langle \text{ digits } \rangle &\to \langle \text{ digit } \rangle \langle \text{ digits } \rangle \mid \langle \text{ digit } \rangle \\
\langle \text{ rest } \rangle &\to \langle \text{ exponent } \rangle \mid . \langle \text{ frac } \rangle \\
\langle \text{ frac } \rangle &\to \langle \text{ digits } \rangle \langle \text{ exponent } \rangle \\
\langle \text{ exponent } \rangle &\to \lambda \mid \mathrm{e} \langle \text{ sign } \rangle \langle \text{ digits } \rangle \\
\langle \text{ digit } \rangle &\to 0 \mid 1
\end{aligned}
$$

Design a non-deterministic finite automaton M that accepts the language of the above binary floating-point numbers. Use JFLAP to test your design.

Solution

The following is one possible solution:



(2) Consider the right-linear grammar $G = (V, T, S, P)$, in which $V = \{S, D\}$, $T = \{a, b, c\}$, and the set P of productions is as follows:

$$\begin{aligned} S &\to acD \mid bcD \\ D &\to S \mid \lambda \end{aligned} \tag{†}$$

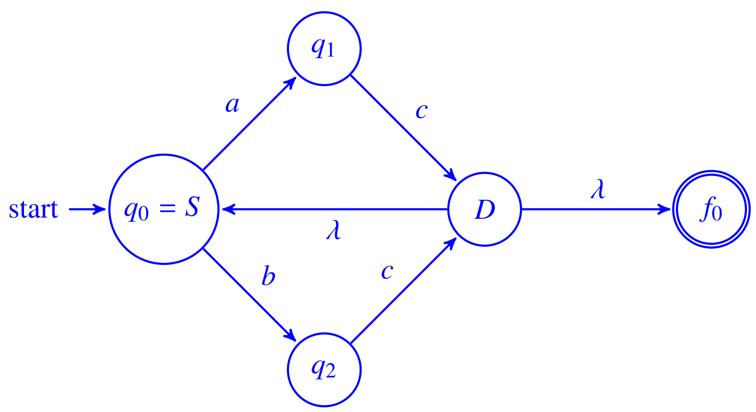
(i) Write a regular expression r such that $L(r) = L(G)$.

Solution

$r = (ac + bc)(ac + bc)^*$.

(ii) Draw the transition graph of a non-deterministic finite automaton (NFA) $M = (Q, T, \delta, q_0, F)$ such that $L(M) = L(G)$. Use the construction of [LR23, Theorem 3.3], and make sure that there is only one final state f_0, i. e., $F = \{f_0\}$, and $f_0 \neq q_0$.

Solution

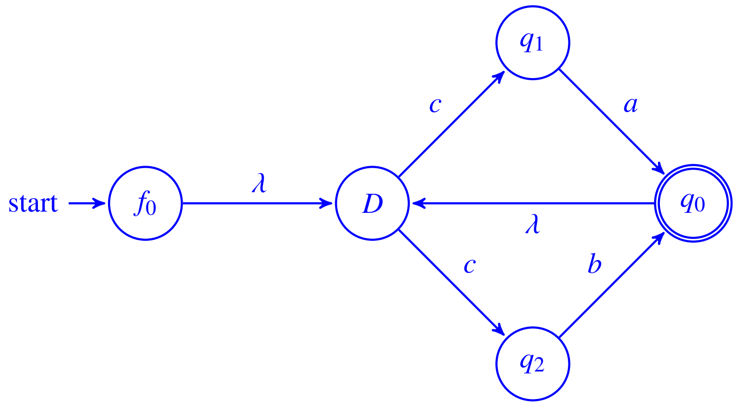


(iii) Draw the transition graph of the NFA $M^R = (Q, T, \delta^R, f_0, \{q_0\})$, obtained as follows:

- The set of states Q and alphabet T are the same as the ones for M;
- The final state q_0 of M^R is the initial state of M;
- The initial state f_0 of M^R is the only final state of M;
- The transitions of δ^R are the reverses of those in δ, i. e., whenever a transition labeled x goes from q_i to q_j in M, a transition labeled x goes from q_j to q_i in M^R (and vice versa).

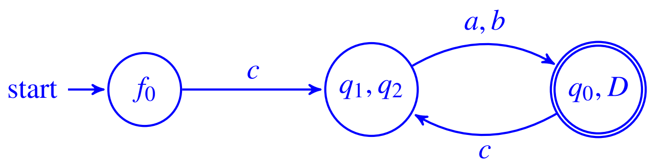
Note that, by this construction, $L(M^R) = L(M)^R$.

Solution



(iv) Use the subset construction to obtain a deterministic finite automaton (DFA) M_1^R such that $L(M_1^R) = L(M^R)$. To make it easier to read the results, you do not need to draw the trap state.

Solution



(v) Write the details of a right-linear grammar $G' = (V', T', S, P')$ such that $L(G') = L(M_1^R)$, by using the construction of [LR23, Theorem 3.4].

Solution

$G' = (V', T', S, P')$, in which:

- $V' = \{S, A, B\}$;
- $T' = \{a, b, c\}$;
- The production rules are as follows:

$$\begin{aligned} S &\to cA \\ A &\to aB \mid bB \\ B &\to cA \mid \lambda \end{aligned}$$

(vi) Now consider the grammar $\tilde{G} = (\tilde{V}, \tilde{T}, S, \tilde{P})$ which is obtained from G' as follows:

- $\tilde{V} = V'$;
- $\tilde{T} = T'$;
- The start variable S is the same as that of G';

- For each variable $U \in \tilde{V}$ and string $\alpha \in (\tilde{V} \cup \tilde{T})^*$, the production rule $U \to \alpha$ is in $\tilde{P}$ if and only if $U \to \alpha^R$ is in P'. In simple terms, each production in $\tilde{P}$ is obtained from reversing the right side of a production in P'.

Write down the production rules for $\tilde{G}$ and notice that, $\tilde{G}$ is a left-linear grammar which generates the same languages as the original right-linear grammar G in (†).

Solution

$$
\begin{aligned}
S &\to Ac \\
A &\to Ba \mid Bb \\
B &\to Ac \mid \lambda
\end{aligned}
$$

(3) Suggest a construction by which a left-linear grammar G can be obtained from an NFA M such that $L(G) = L(M)$.

Solution

1. Convert the NFA M to another NFA M_1 which has only one final state.
2. Exchange the initial and final state and reverse direction of all edges in M_1 to obtain M'.
3. Convert the NFA M' to an equivalent DFA M''.
4. Follow the construction of [LR23, Theorem 3.4] to obtain a right-linear grammar G' such that $L(G') = L(M'') = L(M)^R$.
5. Reverse the right-side of all the productions to obtain the left-linear grammar G, which must satisfy $L(G) = L(G')^R = (L(M)^R)^R = L(M)$.

Note that, although the proof of [LR23, Theorem 3.4] is given for DFAs, the construction also works if we start with an NFA. So, step 3. is optional.

References

[LR23] Linz, P. and Rodger, S. H. An Introduction to Formal Languages and Automata. 7th ed. Jones & Bartlett Learning, 2023.