Summary

This document collects the various personal notes from the course "Formal Languages and Compilers" (2012), prof. Silvano Rivoira. The LaTeX source code is available in a dedicated GitHub repository.

Contents

Part I

Formal Languages

Chapter 1

Classification (FLC)

1.1 Grammars

A grammar is a 4-tuple $G = (N, T, P, S)$ where:

N : alphabet of non-terminal symbols;

T : alphabet of terminal symbols:

- $N \cap T = 0$ (two alphabets are disjoined),
- $V = N \cup T$ (alphabet of the grammar);

P : finite set of rules (productions);

S : start (non-terminal) symbol.

A language produced by $G = (N, T, P, S)$ is:

$$L(G) = \{w|w \in T^*; S \Rightarrow^* w\}$$

Grammars that produce the same languages are said "equivalent".

1.2 Types of Grammars

Type 0 grammars (phase-structure)

$$P = \left\{\alpha \to \beta \middle| \alpha \in V^+; \alpha \notin T^+; \beta \in V^*\right\}$$

Type 1 grammars (context-sensitive)

$$P = \left\{\alpha \to \beta \middle| \alpha \in V^+; \alpha \notin T^+; \beta \in V^+; |\alpha| \leq |\beta|\right\}$$

Type 2 grammars (context-free)

$$P = \left\{A \to \beta \middle| A \in N; \beta \in V^+\right\}$$

1.3 Linear Grammars

$$P = \left\{A \to xBy, A \to x \middle| A, B \in N; x, y \in T^+\right\}$$

Type 3 grammars (right/left - linear)

- Right-Linear grammars

$$P = \left\{A \to xB, A \to x \middle| A, B \in N; x \in T^+\right\}$$

- Left-Linear grammars

$$P = \left\{A \to Bx, A \to x \middle| A, B \in N; x \in T^+\right\}$$

Type 3 grammars (right/left - regular)

- Right-Regular grammars

$$P = \{A \to aB, A \to a | A, B \in N; a \in T\}$$

- Left-Regular grammars

$$P = \{A \to Ba, A \to a | A, B \in N; a \in T\}$$

Chapter 2

Regular Languages (RL)

2.1 Deterministic Finite Automata (DFA)

A DFA is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where:

Q : finite (non-empty) set of states;

Σ : alphabet of input symbols;

δ : transition function:

$$\delta : Q \times \Sigma \to Q$$

q_0 : start state:

$$q_0 \in Q$$

F : set of final states:

$$F \subseteq Q$$

2.1.1 Transition Table

Transitional Table is a tabular representation of this transition function.

2.1.2 Transition Diagram

Transitional Diagram is a graph where:

- for each state in the automaton there a node;
- for each transition $\delta(p, a) = q$ there is an arc from p to q labelled a.

The start state has an entering non-labelled arc and the final states are marked by a double circle.

2.2 Non-Deterministic Finite Automata (NFA)

An NFA is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where:

Q : finite (non-empty) set of states;

Σ : alphabet of input symbols;

$\delta:$ transition function:

$$\delta : Q \times \Sigma \to \mathscr{P}(Q)$$

$\mathscr{P}(Q)$: powerset of Q (the set of all subsets)

$$\|\mathscr{P}(Q)\| = 2^{\|Q\|}$$

$q_0:$ start state:

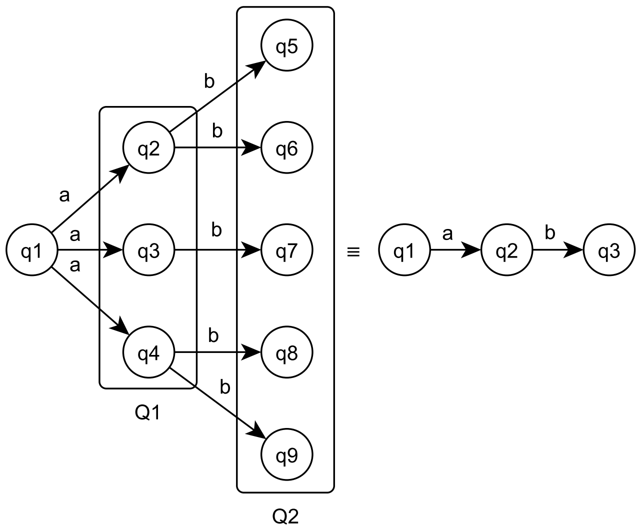
$$q_0 \in Q$$

$F:$ set of final states:

$$F \subseteq Q$$

NB: a DFA is a special case of NFA.

2.3 Equivalence of NFA and DFA



$$Q_1 = \{q_2, q_3, q_4\}$$

$$Q_2 = \{q_5, q_6, q_7, q_8, q_9\}$$

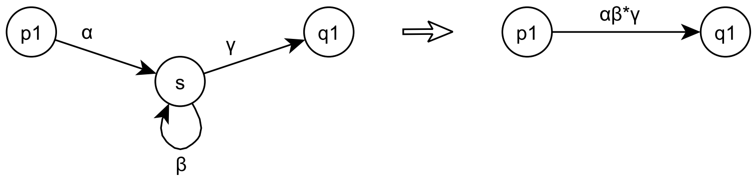
Let $N = (Q_n, \Sigma, \delta_n, q_0, F_n)$ be an NFA; let us construct a DFA $D = (Q_d, \Sigma, \delta_d, \{q_0\}, F_d)$ where:

- $Q_d \subseteq \mathscr{P}(Q_n)$;
- $\delta_d(S, a) = \cup_i \delta_n(p_1, a)$ where $p_i \in S \in Q_d$;
- $F_d = \{S | S \in Q_d; S \cap F_n \neq 0\}$.

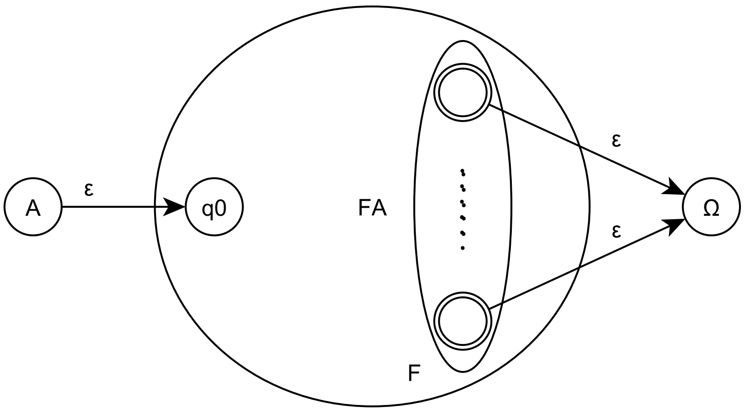
By construction $L(D) = L(N)$, so $NFA \equiv DFA$

2.4 From Finite Automata to Regular Expression

It is possible to eliminate states in a Finite Automata by maintaining all the paths and by labelling the transitions with regular expressions:



Given a finite state automaton $FA = (Q, \Sigma, \delta, q_0, F)$, add an initial state A and a final state Ω:

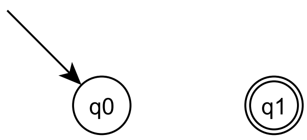


- eliminate all the states in FA;
- the union of the labels on the transitions from A to Ω gives the regular expression of the language $L(FA)$.

2.5 From Regular Expression to Finite Automata

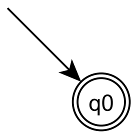
2.5.1 Regular Sets

The regular sets: 0, $\{\varepsilon\}$, $\{a\}$, $a \in \Sigma$ are accepted by finite state automata.



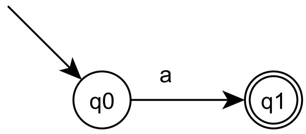
A1

$$L(A_1) = 0$$



A2

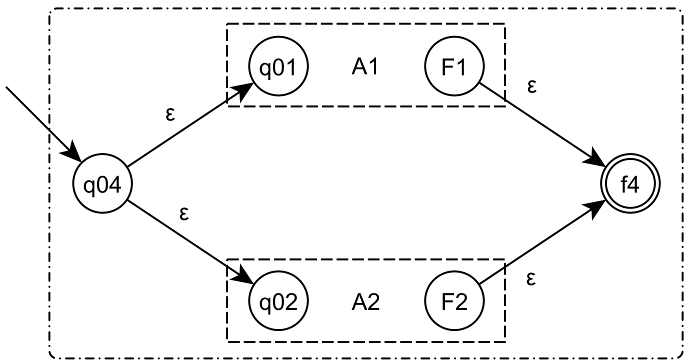
$$L(A_2) = \{\varepsilon\}$$



A3

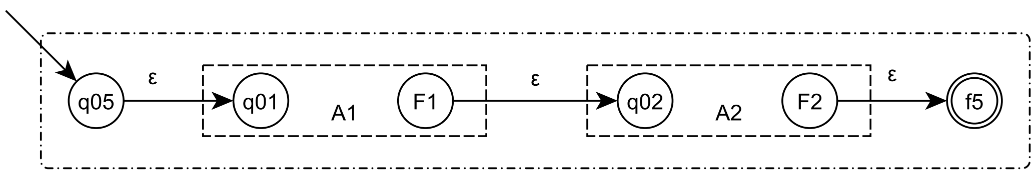
$$L(A_3) = \{a\}\,, a \in \Sigma$$

Let $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be finite state automata; the language $L(A_1) \cup L(A_2)$ is accepted by a finite state automaton A_4:



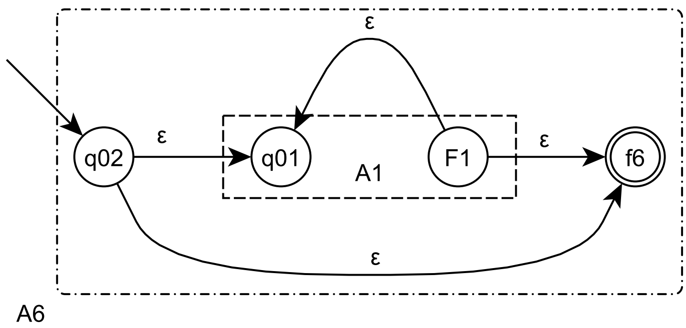
A4

The language $L(A_1)L(A_2)$ is accepted by a finite state automaton A_5:



A5

The language $L(A_1)^*$ is accepted by a finite state automaton A_6:



2.6 Non-Deterministic Finite State Automata with ε-transition (ε-NFA)

In the construction of a Finite State Automaton from regular expressions, the ε-transitions make the automata *non-deterministic*. The function ε-closure(q) gives the set of states that can be reached (recursively) from state q with empty string.

2.6.1 Equivalence of ε-NFA and DFA

Let $N = (Q_n, \Sigma, \delta_n, q_0, F_n)$ be an ε-NFA; let us construct a DFA $D = (Q_d, \Sigma, \delta_d, \varepsilon\text{-closure}(q_0), F_d)$ where:

- $Q_d \subseteq \mathscr{P}(Q_n)$;
- $\delta_d(S, a) = \varepsilon\text{-closure}(\cup_i \delta_n(p_i, a))$ where $p_i \in S \in Q_d$;
- $F_d \,\{S | S \in Q_d; S \cap F_n \neq 0\}$

By construction $L(D) = L(N)$.

2.7 Finite Automaton ≡ Regular Languages

- Let $G = (N, T, P, S)$ be a Right-Regular grammar; let us construct an FA $A = (Q, T, \delta, S, F)$ where:
  - $Q = N \cup \{\Omega\}$ with $\Omega \in N$;
  - $F = \{\Omega\}$;
  - $\delta = \begin{Bmatrix} \delta(A,a) = B & \text{if} & A \to aB \in P \\ \delta(A,a) = \Omega & \text{if} & A \to a \in P \end{Bmatrix}$

  By construction $L(G) = L(A)$.
- Let $G = (N, T, P, S)$ be a Left-Regular grammar; let us construct an FA $A = (Q, T, \delta, I, \{S\})$ where:
  - $Q = N \cup \{I\} \quad \text{with} \quad I \notin N$;
  - $F = \{S\}$;
  - $\delta = \begin{Bmatrix} \delta(B,a) = B & \text{if} & A \to Ba \in P \\ \delta(I,a) = \Omega & \text{if} & A \to a \in P \end{Bmatrix}$

  By construction $L(G) = L(A)$.

2.8 Minimum-State DFA

Let $DFA = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton, then:

- two states p and q of DFA are distinguishable if there is a string $w \in \Sigma^*$ such that $\delta(p, w) \in F$ and $\delta(q, w) \in F$;
- two states p and q of DFA are equivalent ($p \equiv q$) if they are *non-distinguishable* for any string $w \in \Sigma^*$.

A DFA is *minimum-state* if it does not contain equivalent states.

Two states p and q of a DFA are m-equivalent ($p \equiv_m q$) if they are non-distinguishable for all strings $w \in \Sigma^*$ with $\|w\| \leq m$. The equivalent states can be determined by partitioning the set Q in classes of m-equivalent states, for $m0, 1, \ldots, \|Q\| - 2$.

2.8.1 Complement of a Regular Language

The complement of a regular language is a regular language.

Let

Chapter 3

Context-Free Languages (CFL)

Chapter 4

Turing Machines (TM)

Part II

Compilers

Chapter 5

Compiler Structure (CS)

Chapter 6

Lexical Analysis (LA)

Chapter 7

Syntax Analysis (SA)

Chapter 8

Syntax-Directed Translation (SDT)

Chapter 9

Semantic Analysis and Intermediate-Code Generation (SA/ICG)