Models of computation

Gabriel Le Dain

March 2025

1 Finite automata

1.1 Finite automata

Definition 1.1 (Finite automaton)
A **finite automaton**, or **fintite state machine** (sometimes just **state machine**) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

(i) Q is a finite set called the *states*

(ii) Σ is a finite set called the *alphabet*

(iii) $\delta : Q \times \Sigma \to \Sigma$ is the *transition function*

(iv) $q_0 \in Q$ is the *start state*

(v) $F \subseteq Q$ is the set of *final states* or *accept states*

Notation 1.2 (Words over an alphabet)
Let Σ be a set. The set of all words over Σ is denoted by Σ^*. This set of words always contains the empty string.

Notation 1.3 (Augmentation of an alphabet)
For any alphabet Σ, we define the **augmentation** of Σ as

$$\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$$

where ε is a formal symbol not in Σ. We think of ε as the empty string.

Definition 1.4 (Augmentation map)
For any alphabet Σ, we define the augmentation map by

$$\text{aug} : \Sigma_\varepsilon * \to \Sigma^*$$

$$\text{aug}(\varepsilon) = \text{the empty string}$$

$$\text{aug}(a) = a \text{ for all } a \in \Sigma$$

extended to words over Σ in the obvious way.

Definition 1.5 (Language)
A language over a set Σ is some subset of Σ^*

Definition 1.6 (Accept)
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automation and $w = w_1 w_2 \dots w_n$ be a word over Σ. We say that M **accepts** w if there exist $r_0, r_1, \dots, r_n$ in Q such that

(i) $r_0 = q_0$

(ii) $r_{i+1} = \delta(r_i, w_{i+1})$ for all $i = 0, 1, \dots, n-1$

(iii) $r_n \in F$

Definition 1.7 (Recognise)
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. We say that M **recognises** the language

$$L(M) := \{w \in \Sigma^* : M \text{ accepts } w\}$$

1.2 Regular languages

Definition 1.8 (Regular language)
We say that a language A over a set Σ is a **regular language** if there exists some finite automation M with Σ as its alphabet such that M recognises A.

Definition 1.9
Non-deterministic finite automaton A **non-deterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

(i) Q is a finite set called the *states*

(ii) Σ is a finite set called the *alphabet*

(iii) $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the *transition function*

(iv) $q_0 \in Q$ is the *start state*

(v) $F \subseteq Q$ is the set of *final states* or *accept states*

Definition 1.10 (Accept)
et $M = (Q, \Sigma, \delta, q_0, F)$ be a non-deterministic finite automaton and $w = w_1 w_2 \dots w_n$ be a word over Σ. We say that M **accepts** w if there exist $r_0, r_1, \dots, r_n$ in Q such that

(i) $r_0 = q_0$

(ii) $r_{i+1} \in \delta(r_i, w_{i+1})$ for all $i = 0, 1, \dots, n-1$

(iii) $r_n \in F$

Definition 1.11 (ε-Non-deterministic finite automaton)
An **ε-non-deterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

(i) Q is a finite set called the *states*

(ii) Σ is a finite set called the *alphabet*

(iii) $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the *transition function*

(iv) $q_0 \in Q$ is the *start state*

(v) $F \subseteq Q$ is the set of *final states* or *accept states*

Definition 1.12 (Accept)
Let $M := (Q, \Sigma, \delta, q_0, F)$ be an ε-nondeterministic finite automaton. We say that M accepts a word $w \in \Sigma^*$ if there exists $w' := w_0 w_1 \dots w_n \in \Sigma_\varepsilon^*$ and $r_0, r_1, \dots, r_n \in Q$ such that $\text{aug}(w') = w$ and

(i) $r_0 = q_0$

(ii) $r_{i+1} \in \delta(r_i, w_{i+1})$ for all $i = 0, 1, \dots, n-1$

(iii) $r_n \in F$

We can give a more compact definition of an ε-non-deterministic state machine accepting a word using the notion of the ϵ-closure of a set.

Definition 1.13
Let Q, Σ be sets and

$$\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$$

be a function. Let $S \subseteq Q$. Then the ε-closure of S is the set

$$\overline{S} := \{q \in Q : \exists q_1, \dots, q_n \in S, q = q_1 \wedge \forall 1 \leq i < n, q_{i+1} \in \delta(q_i, \varepsilon)\}$$

Definition 1.14 (Accept)
Let $M := (Q, \Sigma, \delta, q_0, F)$ be an ε-nondeterministic finite automaton. We say that M accepts a word $w \in \Sigma^*$ if there exists $r_0, r_1, \dots, r_n \in Q$ such that

(i) $r_0 = q_0$

(ii) $r_{i+1} \in \overline{\delta(r_i, w_{i+1})}$ for all $i = 0, 1, \dots, n-1$

(iii) $r_n \in F$

Definition 1.15 (Accept)
Let $M := (Q, \Sigma, \delta, q_0, F)$ be an ε-nondeterministic finite automaton. We say that M accepts a word $w \in \Sigma^*$ if there exists $w := w_1 w_2 \dots w_n \in \Sigma^*$, $k_1, k_2, \dots, k_n \in \mathbb{N}$ and $r_0, \dots, r_n \in Q$ such that

(i) $r_0 = q_0, k_1 = 1$

(ii) For all $i = 0, 1, \dots, n-1$, we have one of:

  (a) $r_{i+1} \in \delta(r_i, w_{k_{i+1}})$ and $k_{i+1} = k_i + 1$

  (b) $r_{i+1} \in \delta(r_i, \varepsilon)$ and $k_{i+1} = k_i$

(iii) $r_n \in F$

Remark
The definitions 1.12 and 1.15 are equivalent by thinking about them for a second or two. However 1.15 is more suited to implementation in a programming language.

There is an obvious way in which a finite automaton can be regarded as a non-deterministic finite automaton and in which a non-deterministic finite automaton can be regarded as a ε-non-deterministic finite automaton. Moreover, these interpretations leave unchanged the language that the automaton recognises.

More surprisingly, from a (ε)-non-deterministic automaton we can also construct a finite automaton which recognises the same language. Thus, all three models of computation have the same expressive power.

Definition 1.16 (Subset construction)
Let $(Q, \Sigma, \delta, q_0, F)$ be a non-deterministic finite automaton. We construct a finite automaton $(\mathcal{P}(Q), \Sigma, \delta', \{q_0\}, F')$ where

- $F' := \{S \in \mathcal{P}(Q) : S \cap F \neq \emptyset\}$
- $\delta'(S, l) = \bigcup\{\delta(s, l) : s \in S\}$

Theorem 1.17
The subset construction leaves unchanged the language recognised by the automaton.

Proof. (Of 1.17) Exercise. □

It would be nice if the equivalence finite automata $\leftrightarrow$ $\varepsilon-$non-deterministic finite automata factored as finite automata $\leftrightarrow$ non-deterministic finite automata $\leftrightarrow$ $\varepsilon-$non-deterministic finite automata. This may well be doable (either on the present definitions or by modifying the definition of ε-non-deterministic finite automata to take the ε-closure *before* the transition function) but we will go the simpler route of a direct construction $\varepsilon -$ non-deterministic finite automata $\rightarrow$ finite automata.

Definition 1.18 (Subset construction)
Let $(Q, \Sigma, \delta, q_0, F)$ be an ε-non-deterministic finite automaton. We construct a finite automaton $(\mathcal{P}(Q), \Sigma, \delta', \overline{\{q_0\}}, F')$ where

- $F' := \{S \in \mathcal{P}(Q) : S \cap F \neq \emptyset\}$
- $\delta'(S, l) = \bigcup\{\overline{\delta(s, l)} : s \in S\}$

Theorem 1.19
The subset construction leaves unchanged the language recognised by the automaton.

Proof. (Of 1.19) Exercise. □

1.3 Regular Languages

Definition 1.20 (Regular operations)
Let Σ be some alphabet. We define three **regular operations** on languages:

(i) Union: $A \cup B$

(ii) Concatenation: $A \circ B := \{ab : a \in A, b \in B\}$

(iii) Kleene star: $A^* := \{x_1 x_2 \dots x_n \text{ where } x_1, x_2, \dots, x_n \in A\}$

Note 1.21
Note that the definition of the Kleene star on a language above is compatible with its use to denote the set of words over some alphabet.

We want to prove that applying the regular languages are closed under the regular operations. In light of the equivalence of the three kinds of automata,

it is enough to construct just an ε-non-deterministic finite automaton which recognises the union/concatenation/Kleene star of the languages.

For fun, let's consider intersection before any of the three regular operations:
Proposition 1.22 (Intersections of regular languages are regular)
Let A, B be regular languages over the same alphabet Σ. Then $A \cap B$ is also regular.

Proof. Since A, B are regular, they are recognised by finite automata $M^1 := (Q^1, \Sigma, \delta^1, q_0^1, F^1)$ and $M^2 := (Q^2, \Sigma, \delta^2, q_0^2, F^2)$. Define $M^3 := (Q^3, \Sigma, \delta^3, q_0^3, F^3)$ where:

- $Q^3 := Q^1 \times Q^2$
- $\delta^3((s^1, s^2), a) = (\delta^1(s^1, a).\delta^2(s^2, a))$
- $q_0^3 = (q_0^1, q_0^2)$
- $F^3 := F^1 \times F^2$

Then M_3 recognises $A \cap B$. □

For closure under union, we can actually construct a deterministic finite automaton.
Proposition 1.23 (Unions of regular languages are regular)
Let A, B be regular languages over the same alphabet Σ. Then $A \cup B$ is also regular.

Proof. (Of 1.23) Since A, B are regular, they are recognised by finite automata $M^1 := (Q^1, \Sigma, \delta^1, q_0^1, F^1)$ and $M^2 := (Q^2, \Sigma, \delta^2, q_0^2, F^2)$. Define $M^3 := (Q^3, \Sigma, \delta^3, q_0^3, F^3)$ where:

- $Q^3 := Q^1 \times Q^2$
- $\delta^3((s^1, s^2), a) = (\delta^1(s^1, a).\delta^2(s^2, a))$
- $q_0^3 = (q_0^1, q_0^2)$
- $F^3 := \left(F^1 \times Q^2\right) \cup \left(Q^1 \times F^2\right)$

Then M_3 recognises $A \cup B$. □

We can also construct an ε-non-determinstic automaton using a slightly different technique.

Proof. (Of 1.23) Since A, B are regular, they are recognised by ε-non-deterministic finite automata $M^1 := (Q^1, \Sigma, \delta^1, q_0^1, F^1)$ and $M^2 := (Q^2, \Sigma, \delta^2, q_0^2, F^2)$. Define $M^3 := (Q^3, \Sigma, \delta^3, q_0^3, F^3)$ where:

- $Q^3 := Q^1 \coprod Q^2 \coprod \{*\}$

- $\delta^3(q,l) = \begin{cases} \{q_0^1, q_0^2\} & \text{if } q = * \text{ and } l = \varepsilon \\ \delta^1(q,l) & \text{if } q \in Q^1 \\ \delta^2(q,l) & \text{if } q \in Q^2 \\ \emptyset & \text{otherwise} \end{cases}$
- $q_0^3 = *$
- $F^3 := F^1 \coprod F^2$

Then M_3 recognises $A \cup B$. □

The other two operations are harder to construct deterministic automata for, so we instead construct ε-non-deterministic automata for recognising them.

1.4 Pushdown automata

Definition 1.24 (Pushdown Automata)
A **pushdown automata** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

(i) Q is a finite set of *states*

(ii) Σ is a finite set called the *input alphabet*

(iii) Γ is a finite set called the *stack alphabet*

(iv) $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the *transition function*

(v) $F \subseteq Q$ is the (finite) set of *accept states*

One can define a deterministic version of a pushdown automata but it is *not* equivalent to the non-deterministic version, unlike for finite state automata or Turing machines. I won't bother, as it doesn't seem like a classic part of the theory of computation (and also I can't be bothered).
Definition 1.25 (Accept)
A pushdown automata $(Q, \Sigma, \Gamma, \delta, q_0, F)$ a word $w \in \Sigma^*$ and there exists $w_1, \ldots, w_n \in \Sigma_\varepsilon, r_0, \ldots, r_n \in Q, s_0, \ldots, s_n \in \Gamma^*$ such that

(i) $w = \text{aug}(w_1 \ldots w_n)$

(ii) $r_0 = q_0, s_0 = \varepsilon$

(iii) For all $0 \leq i \leq m-1$, there exists $a, b \in \Gamma_\varepsilon, t \in \Gamma^*$ such that $(r_{i+1}, b_i) \in \delta(r_i, w_{i+1}, a), s_i = \text{aug}(at), s_{i+1} = \text{aug}(bt)$

(iv) $r_n \in F$

The case where $a = \varepsilon$ indicates that the pushdown automata moves without reading from the stack and the case where $b = \epsilon$ indicates that the pushdown automata moves without writing to the satck. The above definition is a bit horrible, but I guess I'll get used to it.

Just as finite automata have a grammatical analogue in the form of regular expressions, so too do pushdown automata have a grammatical analogue in the form of *context-free grammars*.

Definition 1.26 (Context-free grammar)
A **context-free grammar** (or **CFG**) is a 4-tuple (V, Σ, R, S) where

(i) V is a finite set called the *variables*

(ii) Σ is a finite set disjoint from V called the *terminals*

(iii) $R \subseteq V \times (V \coprod \Sigma)^*$ is a (finite) set of rules

(iv) $S \in V$ is the *start variable*

[Yields, Derives] Let (V, Σ, R, S) be a CFG, x, y be strings of variables and terminals. We say that x **yields** y if there exist sequences of strings and variables u, v a rule $A \to w$ and a variable A such that $x = uAv$ and $y = uwv$. In this case we write

$$x \Rightarrow_R y$$

or sometimes just

$$x \Rightarrow y$$

We say that x **derives** y, written $x \overset{*}{\Rightarrow} y$, if there exist $w_1 \ldots, w_n$ such that $x \Rightarrow w_1 \Rightarrow_{R_1} \ldots \Rightarrow_{R_n} w_n \Rightarrow y$.

Definition 1.27 (Language of a CFG)
The **language** of a context-free grammar (V, Σ, R, S) is the set

$$\{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}$$

Definition 1.28 (Context-free language)
A language which is generated by some context-free grammar is called a **context-free language**.

Definition 1.29 (Tree)
Let T, I be sets and let Leaf, Node be formal symbols not in either set. A **tree** with *terminal nodes* (also called *leaves*) in T, *internal nodes* in I is defined recurssively as any of the following:

- (Leaf, t) where $t \in T$
- $(\text{Node}, i, x_1, \ldots, x_n)$ where $i \in I$ $x_1, \ldots, x_n$ are trees which has already been constructed

The formal symbols Leaf, Node are needed in case e.g. the set T contains (i, t) for some $t \in T$, $i \in I$; then we wouldn't be able to distinguish betwene the single-node tree $(i, t) \in T$ and the tree with root node i and child node i. A tree can also be defined as a special kind of graph, but this definition makes the inherent recursive structure clear.

Definition 1.30 (Root)
Let X be a tree with terminal nodes in T and intermediate nodes in I. The root of X is:

- t if $X = (\text{Leaf}, t) \in T$
- i if $X = (\text{Node}, i, t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are trees over T, I

We call X a (T, I)-tree.

Note that the set of (T, I)-trees forms a directed acyclic graph whose edges are given by pairs of trees where one tree can be obtained from the other by the addition of some number of children to a single leaf node.

Definition 1.31 (Parse tree)
Let (V, Σ, R, S) be a CFG. A parse tree for this CFG is a $(T, V \times R)$-tree such that the root of T is S and T follows the following recursive condition C:

- Any tree of the form (Leaf, x) for $x \in \Sigma \coprod V$ satisfies C
- A tree of the form $(\text{Node}, (v, r), t_1, \ldots, t_n)$ satisfies C if r has the form $v \to l_1 \ldots l_n$ and for each $1 \leq k \leq n$ we have $t_i = (\text{Leaf}, l_i)$ or $t_i = (\text{Node}, l_i, t_i')$ for some tree t_i'

Traversing the leaves of the tree in left-to-right order gives an element of Σ^*. We say that the parse tree is a parse tree for this element of Σ^*.

Definition 1.32 (Derivation)
Let (V, Σ, R, S) be a CFG and x, y be strings of variables and terminals. A derivation of y from x is a path through the directed graph of parse trees from a parse tree for x to a parse tree for y.

Definition 1.33 (Leftmost derivation)
A leftmost derivation is a derivation where the rule applied is always to the leftmost variable for which a rule exists.

Definition 1.34 (Ambiguous grammar)
A context-free grammar is called **ambiguous** if there exists a word in the associated language with more than one leftmost derivation.

The definition of ambiguous grammar isn't "morally correct" in the sense that one should really care whether there are any duplicate derivations with the same structure, rather than just leftmost ones. However, this should be equivalent, because you can just chop off the start of all the strings involved.

Note that amiguity is a property of the grammar, *not* the language (i.e. a single language can have several grammars which generate it, some of which are ambiguou and some of which are not). There do exist so-called **inherently ambiguous languages** - that is, context-free languages which are not generated by any non-ambiguous context-free grammar.