Resume : Mathematical Methods for Computer Science 2

Sylvain Julmy

June 15, 2018

1 Fibonacci and other recursive sequences

2 Generating functions

3 Partitions

4 Catalan numbers

5 Deterministic and nondeterministic finite automata

6 Automata with ϵ-transitions and regular expressions

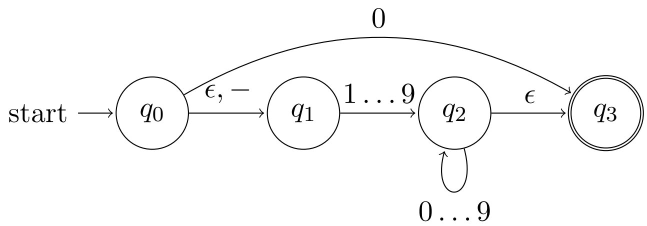
An ϵ-NFA is a NFA with spontanous transitions ϵ which deletes the empty word.

Definition 1. *A non-deterministic finite automata with ϵ-transitions (or ϵ-NFA) is $(Q, \Sigma, \delta, q_0, F)$ where*

$$
\begin{aligned}
&Q: \text{ a finite set of states}\\
&\Sigma: \text{ a finite alphabet}\\
&q_0 \in Q: \text{ the initial state}\\
&F \subset Q: \text{ the set of final states}\\
&\delta : Q \times (\Sigma \cup \{\epsilon\}) \mapsto 2^Q
\end{aligned}
$$

Definition 2. *A string ω is acceptable my an ϵ-NFA if and only if there is a sequence of transitions from q_0 to a final state corresponding to string symbols with any number of ϵ-transition in between.*

Example : ϵ-NFA that accepts all integers written in a correct decimal form.



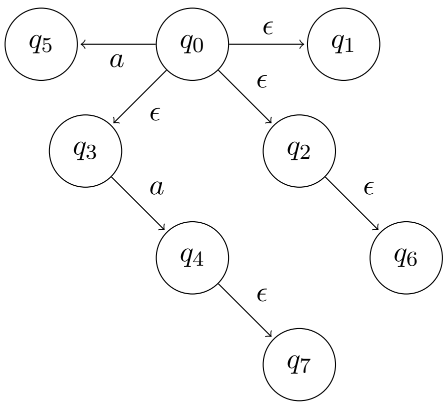
δ	ϵ	$-$	0	$1\dots 9$
q_0	q_1	q_1	q_3	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\emptyset$	q_2
q_2	q_3	$\emptyset$	q_2	q_2
q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Definition 3. *A subset $P \subset Q$ is ϵ-close if all ϵ-transitions from P leads to P, that is for all $q \in P$, $\delta(q, \epsilon) \subset P$.*

Definition 4. *The ϵ-closure $\overline{P}$ of P is the minimal ϵ-close subset containing P, the construction of $\overline{P}$ is done by induction :*
Base case : *State q is in $ECLOSE(q)$.*
Induction : *If state p is in $ECLOSE(q)$, and there is a transition from state p to state r labeled ϵ, then r is in $ECLOSE(q)$.*

Definition 5. *The extended transition function for an ϵ-NFA $\widehat{\delta} : Q \times \Sigma^* \mapsto 2^Q$ is defined recursively as follows : $\widehat{\delta}(q, \epsilon) := \{q\}$ and if $\omega \in \Sigma^*$ and $|\omega| = n$, then $\omega = \omega_0 a$ for $|\omega_0| = n - 1$, $a \in \Sigma$, $\widehat{\delta}(q, \omega) = \overline{\delta(\widehat{\delta}(q, \omega_0), Q)}$, by definition $\delta(P, a) := \bigcup_{a \in P} \delta(q, a)$, $P \subset Q$.*

Example :



$$\widehat{\delta}(q, \epsilon) = \{q_0, q_1, q_2, q_3, q_6\}$$
$$\widehat{\delta}(q, a) = \{q_5, q_4, q_7\}$$

Definition 6. *The language accepted by an ϵ-NFA A is $L(A) = \{\omega \in \Sigma^* | \widehat{\delta}(q_0, \omega) \cap F \neq \emptyset\}$*

Theorem 6.1. *If L is a language accepted by an ϵ-NFA A, then there exists a DFA D which accepts L.*

Proof.

$$A = (Q, \Sigma, \delta, q_0, F)$$
$$D = (2^Q, \Sigma, \delta', q_0', F')$$
$$q_0' = \{q_0\}$$
$$F' = \{P \subset Q | P \cap F \neq \emptyset\}$$
$$\delta'(P, a) = \overline{\delta(P, a)}$$

Claim : $\widehat{\delta'}(q'_0, \omega) = \widehat{\delta}(q_0, \omega)$

$$\begin{aligned} \omega \in L(A) &\iff \widehat{\delta}(q_0, \omega) \cap F \neq \emptyset \\ &\iff \widehat{\delta'}(q'_0, \omega) \cap F \neq \emptyset \\ &\iff \widehat{\delta}(q_0, \omega) \cap F' \neq \emptyset \\ &\iff \omega \in L(D) \end{aligned}$$

Then $L(A) = L(D)$.
Induction on the length :

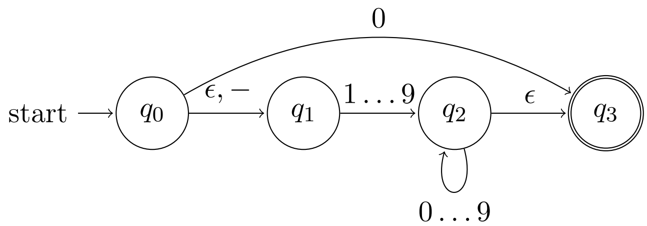
Base case : $|\omega| = 0$, then $\omega = \epsilon$ $\widehat{\delta'}(q'_0, \epsilon) = q'_0 = \{q_0\}$.

Inductive step : $|\omega| = n$, then $\omega = \omega_0 a$, $|\omega_0| = n - 1$, $a \in \Sigma$

$$\begin{aligned} \widehat{\delta'}(q'_0\omega) &= \delta'(\widehat{\delta'}(q'_0, \omega_0), a) \\ &= \delta'(\widehat{\delta}(q_0, \omega_0), a) \\ &= \delta(\widehat{\delta}(q_0, \omega_0), a) \\ &= \widehat{\delta}(q_0, \omega) \end{aligned}$$

□

We transform



$$q'_0 = \{q_0\} = \{q_0, q_1\}$$

δ	$-$	0	$1 \ldots 9$
$\{q_0, q_1\}$	$\{q_1\}$	$\{q_3\}$	$\{q_2, q_3\}$
$\{q_1\}$	$\emptyset$	$\emptyset$	$\{q_2, q_3\}$
$\{q_3\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\{q_2, q_3\}$	$\emptyset$	$\{q_2, q_3\}$	$\{q_2, q_3\}$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

7 Regular expressions and regular languages

Definition 7. *Regular expressions (RE) denote languages*

1. *$\emptyset$ is a RE generating the empty language $\emptyset$ (two state, no transition, the initial state is not accepting).*

2. *ϵ is a RE generating $\{\epsilon\}$ (one state, initial and final).*

3. *$a \in \Sigma$ is a RE generating $\{a\}$.*

4. *if r and s are RE generating R and S, then $r + s$ is a RE generating the language $R \cup S$.*

5. *if r and s are RE generating R and S, then $r \cdot s$ is a RE generating $RS = \{uv | u \in R \wedge v \in S\}$*

6. *if r is a RE generating R, then r^* is a RE generating $R^* = \bigcup_{i=0}^{\infty} R^i$, $R^i = \underbrace{RRR \ldots R}_{i \text{ times}}$, $R^0 = \epsilon$. Its called the Kleene closure of R.*

7. *Priority operation : $* > \cdot > +$*

Theorem 7.1. *A language L is accepted by some DFA if and only if it is denoted by a regular expression.*

Lemma 7.1. *For a regular expression r, there is an ϵ-NFA M such that it accepts $R = L(r)$ and it has only one final state without any transition from it.*

Proof. Induction on the number of operation in r :
Base :

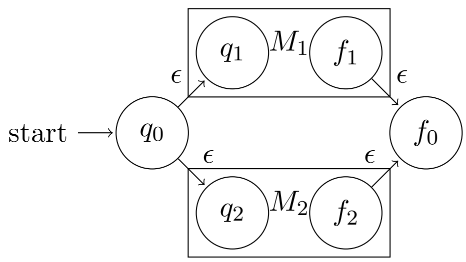
1. $r = \emptyset$: two state and no transition, the initial state is not final.

2. $r = \epsilon$: one state which is final.

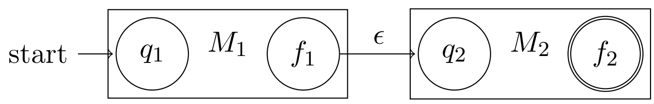
3. $r = a \in Sigma$: two state and one transition labeled a, the initial state is not final but the other is.

Inductive step :

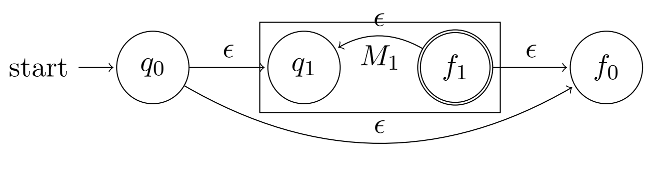
1. $r = r_1 + r_2$: $R = R_1 \cup R_2 \rightarrow$ if ω is accepted by $M \iff \omega \in R_1 \vee \omega \in R_2$.



2. $r = r_1 \cdot r_2$: $R = R_1 R_2 \rightarrow$ if $\omega \in R_1 R_2 \iff \omega = \omega_1 \omega_2 \iff \omega$ accepted by M.



3. $r = r_1 *$: $R = R_1^* = \bigcup_{i=0}^{\infty} R^i$, if $\omega \in R_i^* \iff \omega = \omega_1 \ldots \omega_k \iff \omega$ accepted by M.



□

Lemma 7.2. *For a DFA M, there is a regular expressions r describing the language $R = L(M)$.*

Proof. Assume that $M's$ states are $\{1, 2, \ldots, n\}$ for some integer n. No matter what the states of A actually are, there will be n of them for some finite n , and by renaming the states, we can refer to the states in this manner, as if they were the first n positive integers.
Denote by $R_{ij}^{(k)}$ the name of a regular expression whose language is the set of strings ω such that ω is the label of a path from state i to state j in A, and that path has no intermediate node whose number is greater than k. Note that the beginning and end points of the path are not "intermediate", so there is no constraint that i and/or j be less than or equal to k.
To construct the expressions $R_{ij}^{(k)}$, we use the following inductive definition, starting at $k = 0$ and finally reaching $k = n$. When $k = n$, there is no restriction at all on the paths represented, since there are no states greater than n.
Base : $k = 0$

- $i \neq j$ and $R_{ij}^0 = \emptyset \implies r_{ij}^0 = \emptyset$.
- $i \neq j$ and $R_{ij}^0 = \{a_1, \ldots, a_m\} \implies r_{ij}^0 = a_1 + a_2 + \cdots + a_m$.
- $i = j$ and $R_{ij}^0 = \{\epsilon\} \implies r_{ij}^0 = \epsilon$
- $i = j$ and $R_{ij}^0 = \{a_1, \ldots, a_m\} \implies r_{ij}^0 = a_1 + a_2 + \cdots + a_m + \epsilon$

Inductive step : Suppose there is a path from state i to state j that goes through no state higher than k. There are two possible cases to consider :

1. The path does no go through state k at all. In this case, the label of the path is in the language if $R_{ij}(k-1)$.
2. The path goes through state k at least once, then we can break the path into several pieces. The first piece goes from state i to k and the last piece goes from k to j without passing through k, and all the pieces in the middle go from k to itself, without passing through k. The set of labels for all paths of this type is represented by the regular expression $R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$.

When we combine the expressions for the paths of the two types above, we have the expression

$$R_{ik}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}(R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$$

The regular expression for the language of the automaton is then the sum (union) of all expressions $R_{1j}^{(n)}$ such that j is an accepting state. □

Theorem 7.2. *Assume that L is a regular language, then $\Sigma^* \setminus L$ is a regular language.*

Proof.

$$\begin{aligned}
&\exists M : DFA, L = L(M), M = (Q, \Sigma, \delta, q_0, F) \\
&M' = (Q, \Sigma, \delta, q_0, Q \setminus F) \text{ accepts } \Sigma^* \setminus L \\
&\omega \in \Sigma^* \setminus L \iff \omega \notin L \iff \widehat{\delta}(q_0, \omega) \notin F \iff \widehat{\delta}(q_0, \omega) \in Q \setminus F
\end{aligned}$$

□

Corollary 7.1. *If L_1 and L_2 are regular languages, then $L_1 \cap L_2$ is a regular language and $L_1 \setminus L_2$ is a regular language.*

Proof. ($L_1 \cap L_2$ is a regular language) We know that $L_1 \cup L_2$ is a regular language. Therefore, we have

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}, \text{where } \overline{L} = \Sigma^* \setminus L$$

□

Proof. ($L_1 \setminus L_2$ is a regular language)

$$L_1 \setminus L_2 = L_1 \cap \overline{L_2}$$

□

8 The pumping lemma and homomorphisms

Can we decide algorithmically whether two given automata or two given RE define the same language ? It suffices to give a positive answer to one of those questions, because we have algorithms for automaton $\leftrightarrow$ RE.

Theorem 8.1. *There is an algorithm that decide whether two given automata are equivalent.*

Proof. Let M_1 and M_2 be two automata, and $L_1 = L(M_1)$, $L_2 = L(M_2)$. Consider the symmetric difference : $L_1 \triangle L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$, L_1, L_2 regular $\implies L_1 \triangle L_2$ regular.
Let M be a finite automaton that accepts the language $L_1 \triangle L_2$, it suffices to decide if $L(M)$ is empty or not, but $L(M) = \emptyset \iff$ there is no path from q_0 to F □

Theorem 8.2 (The pumping lemma for regular languages)**.** *Let L be a regular language, then there is a positive integer n such that any word $z \in L$ of length $|z| \geq n$ can be written as $z = uvw$ in such a way that $|uv| \leq n$, $|v| \geq 1$ and $\forall k \geq 0 : uv^k w \in L$.*

Proof. Take a DFA that accepts L, let n be the number of its states. Take for i, j tje first pair where the coincidence occurs. Then $j \leq n$. □

Example : $\Sigma = \{0\}, L = \{0^P \mid p \; prime\}$, then L is non-regular.
There are large gaps between the primes : $n! + 2, n! + 3, \dots, n! + n$ are all composite (non-prime). Take k such that $p_{k+1} > p_k + n$, $0^{p_k} \in L$ (assume $L = L(M)$ and $|Q| = n$), $0^{p_k} = \underbrace{uvw}_{v=0^l, l \leq n}$, $uv^2w = o^{p_k+p}$, $p_k + p \leq p_k + n < p_k + 1 \implies uv^2w \notin L$.

Theorem 8.3. *A language accepted by a DFA with n states is*

1. *non-empty if an only if it contains a word of length $< n$.*
2. *infinite if and only if it contains a word of length l, where $n \leq l \leq 2n$.*

Proof.

1. A shortest word in L has length $< n$, otherwise it revisits some state and can be shortened.
2.
   - Assume $z \in L$ and $|z| \geq n$, then $z = uvw \implies uv^k w \in L \implies L \; infinite$.
   - Assume L is infinite, then $\exists z \in L$ such that $|z| \geq n$.
     - If $|z| < 2n$, then done.

– If $|z| \geq 2n$, tjen by the pumping lemma

$$z = uvw,\ 1 \geq |v| \geq n,\ uw \in L$$

We have $|uw| = |z| - |v| \geq |z| - n \geq n$

□

Definition 8. *Let Σ, Δ be finite alphabets, a homomorphisms is a map $h : \Sigma^* \mapsto \Delta^*$ such that $h(xy) = h(x)h(y) \forall x, y \in \Sigma^*$.*

Lemma 8.1. *A homomorphisms is uniquely determined by the images of letters of Σ. That is, any $h : \Sigma \mapsto \Delta^*$ extends to a unique homomorphisms.*

Proof.

- Uniqueness : assume $h(a)$ is given $\forall a \in \Sigma$, then for $\omega = a_1a_2 \ldots a_n$, we have no other choice but $h(\omega) = h(a_1)h(a_2) \ldots h(a_n)$, $h(\epsilon) = \epsilon$ and $h(x) = h(x\epsilon) = h(x)h(\epsilon) \implies h(\epsilon) = \epsilon$.
- Existence : $h(\omega) = h(a_1)h(a_2) \ldots h(a_n)$ defines a homomorphisms.

□

Example : $\Sigma = \Delta = \{0\}$, $L = \Sigma^*$

$$h(0) = 00 \implies h(L) = \{\text{all words of even length}\}$$

Example : $\Sigma = \Delta = \{0, 1\}$, $L = \Sigma^*$

$$\begin{gathered} h(0) = 0 \\ h(1) = 10 \\ h(L) = ? \\ L = (0+1)^* \\ h(L) = (0+10)^* \end{gathered}$$

Theorem 8.4. *A homomorphic image of a regular language is regular*

Proof. Let L be a regular language and r a regular expression generating L. Replace in r every letter by its image under h. The result is a regular expression. This expression represents the language $h(L)$. Proof by induction on the complexity if r. □

Note : a homomorphisms image of a non-regular language might be regular.

Definition 9. *Given a homomorphisms $h : \Sigma^* \mapsto \Delta^*$ and $L \cup \Delta^*$, the inverse homomorphic image of L is :*

$$h^{-1}(L) = \{\omega \in \Sigma^* | h(\omega) \in L\}$$

Example : $\Sigma = \{a, b\}$ $\Delta = \{0, 1\}$

$$L = (00+1)^*$$
$$h(a) = 01$$
$$h(b) = 10$$
$$h^{-1}(1001) = \{ba\}$$
$$h^{-1}(L) = \text{all words in } a, b \text{ such that after the 0's come in pairs}$$

- cannot begin with a
- it has to begin with b

$$\implies (ba)^* = h^{-1}(L)$$

Theorem 8.5. *Regular language are closed under inverse homomorphisms.*

Proof. Let $M = (Q, \Delta, \delta, q_0, F)$ be a DFA for L, then construct $M' = (Q, \Sigma, \delta', q_0, F)$ such that $L(M') = h^{-1}(L)$.

$$\delta'(q_0, a) = \widehat{\delta}(q_0, h(a)) \implies \forall w \in \Sigma^* : \widehat{\delta'}(q_0, \omega) = \widehat{\delta}(q_0, h(w))$$

$$\omega \in L(M') \iff \widehat{\delta}(q_0, h(\omega)) \in F \iff h(\omega) \in L(M)$$

□

9 The Myhill-Nerode theorem

Definition 10. *Let $L \subset \Sigma^*$ be any language. We say that $u, v \in \Sigma^*$ are $L-$equivalent ($u \sim_L v$) if $\forall x \in \Sigma^* : (ux, vx \in L) \vee (ux, vx \notin L)$.*
Note : *or $u \not\sim_L v \iff \exists$ distinguisting extension $x \in \Sigma^*$,that is exactly one of ux, vx is in L.*

Lemma 9.1. *$\sim_L$ is an equivalence relation :*

- *reflexive : $u \sim_L u$.*
- *symetric : $u \sim_L v \implies v \sim_L u$.*
- *transitive : $u \sim_L v \wedge v \sim_L w \implies u \sim_L w$.*

Proof. (transitivity)
Assume $u \not\sim_L w$.
Take x such that, without lost of generality, $ux \in L$ and $wx \in L$, now

- $vx \in L \implies v \not\sim_L w$
- $vx \notin L \implies u \not\sim_L v$

□

Corollary 9.1. *Σ^* splits into equivalence classes :*

$$\Sigma^* = \bigcup_i S_i$$

where $u \sim_L v \iff \exists i$ s.t. $u, v \in S_i$

Example : $L\ subset\{0,1\}^* \quad L = \{w | \underbrace{l_0(w)}_{\text{number of 0 in } w} \quad \text{is not divisible by 3}\}$

Then

$$u \sim_L v \iff l_0(u) \equiv l_0(v) \mod 3$$

$$\begin{aligned} l_0(u) &= 2l_0(ux) && = l_0(u) + l_0(x) = l_0(x) + 2 \\ l_0(v) &= 5l_0(vx) && = l_0(v) + l_0(x) = l_0(x) + 5 \end{aligned}$$

$$\begin{aligned} \Sigma^* &= S_0 \cup S_1 \cup S_2 \\ S_i &= \{w | l_0(w) \equiv i \mod 3\} \\ L &= S_1 \cup S_2 \end{aligned}$$

Lemma 9.2.

$$u \sim_L v \implies u, v \in L \vee u, v \notin L$$

The converse is not true.

Proof. Put $x = \epsilon$ in the definition. □

Corollary 9.2.

$$\forall S_i : either S_i \subset L \ or \ S_i \cap L = \emptyset$$

Example : $L = \{w \mid l_0(w) = l_1(w)\}$

$$u \sim_L v \iff l_0(u) - l_1(u) = l_0(v) - l_1(v)$$

follows from $l_0(ux) = l_0(u) + l_0(x) \dots$

Lemma 9.3. *If $u \sim_L v$, then $\forall a : \Sigma$, we have $ua \sim_L va$ ($\sim_L$ is right invariant).*

Proof. Assume $ua \sim_L va$, take a distinguishing extension x, without lost of generality, we have $vax \in L, vax \notin L$. Then ax is a distinguishing extension for u, v. □

Remark : the converse is false : $ua \sim_L va \not\Longrightarrow u \sim_L v$

Example : $L = (0+1)^*0$, $10 \sim_L 00$, but $1 \not\sim_L 0$.

$$x = \epsilon \implies u \notin L \wedge v \in L$$

Theorem 9.1 (Myhill-Nerode)**.** *A language L is regular if and only if the number of L-equivalence classes is finite.*

Proof. Assume $\Sigma^* = S_1 \cup S_2 \cup \cdots \cup S_n, \epsilon \in S_1$. We will construct a DFA with n states accepting L. $Q = \{q_1, \dots, q_n\}$ where q_1 is the initial state and q_i a final state such that $q_i(final) \iff S_i \subset L$. To find $\delta(q_i, a)$, take any $v \in S_i$ and look where va is :

$$\delta(q_i, a) = q_j, \text{ where } va \in S_j$$

$\widehat{\delta}(q_1, w) = q_i$ where $w \in S_i$ (choosing $\epsilon \in S_1$) $w = a_1 \dots a_n$, $w \in L(M) \iff q_i$ final $\iff S_i \subset L \iff w \in L$.
Assume L is regular and take a DFA M accepting L. Put $T_i = \{w \in \Sigma^* | \widehat{\delta}(q_1, w) = q_i\}$. These are equivalence classes with respect to $u \sim_M v \iff \widehat{\delta}(q_1, u) = \widehat{\delta}(q_1, v)$.

Claim. $u \sim_M v \implies u \sim_L v$

Have $\widehat{\delta}(q_1, u) = \widehat{\delta}(q_1, v)$, then

$$\begin{aligned}
\widehat{\delta}(q_1, ux) &= \widehat{\delta}(\widehat{\delta}(q_1, u), x) \\
&= \widehat{\delta}(\widehat{\delta}(q_1, v), x) \\
&= \widehat{\delta}(q_1, vx)
\end{aligned}$$

$$\implies \text{ either both } ux, vx \in L \text{ or both } ux, vx \notin L$$

This holds $\forall x \implies u \sim_L v$. It follows that every T_i is contained in some $S_j \implies$ the number of L-equivalence classes is finite and $\leq m$. ☐

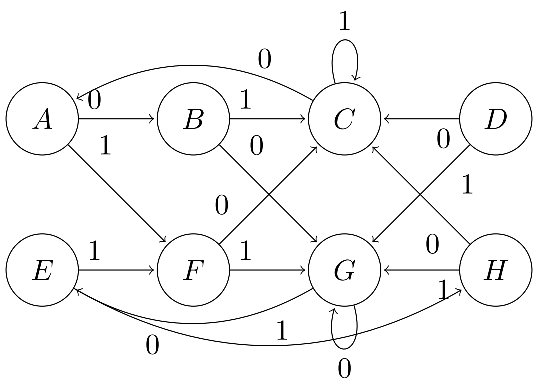
Theorem 9.2. *The minimum number of states in a DFA accepting L is equal to $ind(L)$ (index of L, its number of equivalence classes). The minimal automaton is unique (up to renaming the states).*

Proof. From the end of the last proof, $|Q| \geq ind(L)$. For any DFA accepting L :

- From the 1^{st} half of the Myhill-Nerode theorem, there is a DFA M with $ind(L)$ states.
- For any DFA with $|Q| = ind(L)$, every T_i is equal to some S_j.
- One can see that δ for M coincides with δ defined for $S_1, \ldots, S_n$

☐

Algorithm to minimize DFA



B	×						
C	×	×					
D	×	×	×				
E		×	×	×			
F	×	×	×		×		
G	×	×	×	×	×	×	
H	×		×	×	×	×	×
	A	B	C	D	E	F	G

Algorithm.
Base : if $p \in F$ and $q \notin F$, then mark (p, q).
Inductive step : $\forall (p, q)$ *s.t.* (p, q) not marked and $\forall a$ if $(\delta(p, a), \delta(q, a))$, then mark (p, q). Stop when no more pairs are marked. ☐

10 Context-free grammars

Definition 11. *A context-free grammar (CFG, or just grammar) is $G = (V, T, P, S)$ where*

- *V is a finite set of variables.*
- *T is a finite set of terminals.*
- *P is a finite set of productions of the form $A \to \alpha$, where $A \in V$, $\alpha \in (V \cup T)^*$.*
- *$S \in V$ a special variable called start symbol.*

Convention on notation :

- Capitals $A, B, C, D, \dots$ are variables.
- lowercase $a, b, c, d, \dots$ and digits are terminals.
- $\alpha, \beta, \gamma \in (V \cup T)^*$ are strings of variables and terminals.
- $u, v, w, x, y, z \in T^*$ are strings of terminals.

Definition 12. *If $A \to \beta$ is a production in G, then we write $\alpha A \gamma \Rightarrow_G \alpha\beta\gamma$ and say : "$\alpha A \gamma$ directly derives $\alpha\beta\gamma$".*
If $\alpha_1, \dots, \alpha_n \in (V \cup T)^*$ such that

$$\alpha_1 \Rightarrow_G \alpha_2, \alpha_2 \Rightarrow_G \alpha_3, \dots, \alpha_{n-1} \Rightarrow_G \alpha_n$$

then $\alpha_1 \Rightarrow_G^* \alpha_n$, "$\alpha_1$ derives α_n" $\implies \Rightarrow^*$ if G is understood.

Definition 13. *The language generated by G is the set of all strings of terminals that can be derived from the start symbol :*

$$L(G) = \{\omega \in T^* | S \Rightarrow_G^* \omega\}$$

Example : $S \to (S \wedge S) \mid (S \vee S) \mid (\neg S) \mid p \mid q$ generates propositional formulas in p and q.

Definition 14. *Grammars G and G' are called equivalent if $L(G) =)L(G')$.*

Definition 15. *A symbol $X \in V \cup T$ is called useful if it is used in a derivation of some word in $L(G)$, if this is a derivation of the form :*

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* \omega, \text{ where } \omega \text{ is composed of words and terminals only}$$

A symbol X is called generating if $X \Rightarrow^* \omega$ for some ω. Any $a \in T$ is generating.
A symbol X is called reachable if

$$S \Rightarrow^* \alpha X \beta \text{ for some } \alpha, \beta \in (V \cup T)^*$$

Then, useful means that the symbol is reachable and generating, $\alpha X \beta \Rightarrow^* \omega \implies X \Rightarrow^* v$, where v is a subword of ω.

Theorem 10.1. *Let G be a CFG, then there is a CFG G'' such that $L(G'') = L(G)$ and G'' has no useless symbols.*
Construction : *Let $G = (V, T, P, S)$*

1. *Construct* $G' = (V', T', P', S)$
   - *elimination from* V *and* T *of all non-generating symbols.*
   - *elimination from* P *of all production that contains non-generating symbols.*
2. *Construct* G'' *by elimination from* G' *of all symbols non-reachable in* G' *and all production with these symbols.*

Proof. Suppose X is a symbol that remains ($X \in V_1 \cup T_1$). We know that $X \Rightarrow^*_G \omega$ for some ω in T^*. Moreover, every symbol used in the derivation of ω from X is also generating. Thus, $X \Rightarrow^*_{G''} \omega$
Since X was not eliminated in the second step, we also know that there are α and β such that $S \Rightarrow^*_{G''} \alpha X \beta$. Further, every symbol used in this derivation is reachable, so $S \Rightarrow^*_{G'} \alpha X \beta$.
We know that every symbol in $\alpha X \beta$ is reachable, and we also know that all these symbols are in $V_2 \cup T_2$, so each of them is generating in G''. The derivation of some terminal string, say $\alpha X \beta \Rightarrow^*_{G''} xwy$, involves only symbols that are reachable from S, because they are reached by symbols in $\alpha X \beta$. Thus, this derivation is also a derivation of G'; that is,

$$S \Rightarrow^*_{G'} \alpha X \beta \Rightarrow^*_{G'} xwy$$

We conclude that X is useful in G'. Since X is an arbitrary symbol of G', we conclude that G' has no useless symbols.
The last detail is that we must show $L(G_1) = L(G)$. As usual, to show two sets the same, we show each is contained in the other.

- $L(G_1) \subseteq L(G)$: Since we have only eliminated symbols and productions from G to get G', it follows that $L(G_1) \subseteq L(G)$.
- $L(G_1) \supseteq L(G)$: We must prove that if $\omega \in L(G)$, then $\omega \in L(G')$. If $\omega \in L(G)$, then $S \Rightarrow^*_G \omega$. Each symbol in this derivation is evidently both reachable and generating, so it is also a derivation of G'. That is, $S \Rightarrow^*_{G'} \omega$, and thus $\omega \in L(G_1)$.

□

Theorem 10.2. *Let* G *be a context free grammar, then* $\exists G' : L(G') = L(G) \setminus \{\epsilon\}$ *and* G' *has no* ϵ*-productions.*

Algorithm.

1. Identify nullable variables, those A for which $A \Rightarrow^*_G \epsilon$, by recursion :
   - $A \to \epsilon \implies A$ is *nullable.*
   - $A \to B_1, \ldots, B_n \wedge B_1, \ldots, B_n$ are *nullable* $\implies A$ is *nullable.*
2. Remove all ϵ-productions and add new productions : Let $A \to X_1 \ldots X_n$ be a production from G with n nullable variables among $X_1, \ldots, X_n$. Add production of the form $A \to X_1 \ldots X_n$, any subset of nullable variables is removed. There are 2^m productions.

Exception : if all X_i are nullable variable, then don't remove all of them at the same time. □

Example : $S \to AB \quad A \to aAA \mid \epsilon \quad B \to bBB \mid \epsilon$, nullable : S, A, B :

$$S \to AB \mid A \mid B \quad A \to aAA \mid aA \mid a \quad B \to bBB \mid bB \mid b$$

Theorem 10.3. *Let G be a context free grammar, then $\exists G'$ such that $L(G') = L(G)$ and G' has no unit production, where an unit production is a production of the form $A \to B$.*

Algorithm.

1. Identify unit pairs : (A, B) such that $A \Rightarrow_G^* B$, note that (A, A) is a unit pair.
2. $\forall$ unit pair (A, B) and $\forall$ non-unit pair $B \to \alpha$, form $A \to \alpha$. Let P' be the set of all productions formed in this way, construct $G' = (V, T, P', S)$. P' contains all non-unit production from P.

□

Theorem 10.4. *Let G be a context free grammar that contains at least one non-empty word, then $\exists G' : L(G') = L(G) \setminus \{\epsilon\}$ and G' has no useless symbols, no ϵ-production and no unit-productions.*

Algorithm.

1. Eliminate ϵ-productions.
2. Eliminate unit-productions.
3. Eliminate useless symbols.

□

Definition 16. *A grammar G is in a Chomsky normal form is all its productions are of the form $A \to BC$ and $A \to a$.*

Theorem 10.5. *Any context free language without ϵ can be generated by a grammar in a Chomsky normal form.*

Proof. By previous theorem, $\exists G' : L(G') = L(G)$ without ϵ-productions and unit-productions. Any productions with one symbol at the right is $A \to a$, thus admissible.

1. Take any $A \to X_1 \dots X_n$, where $n \geq 2, X \in V \cup T$. Get ride of the terminals : if $X_i = a$, then introduce a new variable C_a and $C_a \to a$. In $A \to X_1 \dots X_n$, replace X_i by C_a.
2. Now all productions are either $A \to a$ or $A \to B_1 \dots B_n, B_i \in V$, for all productions of the form $A \to B_1 \dots B_n$, introduce new variables $D_1, \dots, D_{n-2}$ and replace $A \to B_1 \dots B_n$ by $A \to B_1 D_1, D_1 \to B_2 D_2, \dots, D_{n-2} \to B_{n-1} B_n$.

□

11 Pushdown automata

Theorem 11.1. *Context-free languages are closed under union, concatenation and Kleene closure.*

Proof. Take $L_1 = L(G_1)$ and $L_2 = L(G_2)$, $G_i = (V_i, T_i, P_i, S_i)$. We can assume that $V_1 \cap V_2 = \emptyset$ (otherwise rename variables, but don't rename terminals). New alphabet of terminals : $T = T_1 \cup T_2$.

- *Union* : $G' = (V_1 \cup V_2 \cup \{S'\}, T, P_1 \cup P_2 \cup \{S' \to S_1 \mid S_2\}, S')$. Clearly, $L(G') = L(G_1) \cup L(G_2)$.
- *Concatenation* : $G'' = (V_1 \cup V_2 \cup \{S''\}, T, P_1 \cup P_2 \cup \{S'' \to S_1 S_2\}, S'')$.
- *Kleene closure* : $G''' = (V \cup \{S'''\}, T, P \cup \{S''' \to SS''' \mid \epsilon\}, S''')$

□

Corollary 11.1. *Every regular language is context-free.*

Proof. Basic languages : $\emptyset$, $\{\epsilon\}$, $\{a\}$, and any regular languages is obtained from basic languages by union, concatenation and Kleene closure. Thus, it suffices to show : basic languages are context-free.

$$S : variables$$

$$P = \begin{cases} \epsilon & L = \emptyset \\ S \to \epsilon = & L = \{\epsilon\} \\ S \to a = & L = \{a\} \end{cases}$$

□

Definition 17. *A pushdown automaton is $(Q, \Sigma, \Gamma, \delta, q_o, Z_0, F)$ where*

- *Q : set of states.*
- *$F \subset Q$: set of final states.*
- *Σ : the input alphabet.*
- *Γ : the stack alphabet.*
- *$Z_0 \in \Gamma$: the start symbol.*
- *$q_0 \in Q$: the initial state.*
- *δ : is a map from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to a finite subsets of $Q \times \Gamma^*$.*

At the beginning, the state is q_0 and the stack Z_0.
If we have $\delta(q, a, A) = \{(P_1, \phi_1, \ldots, (P_m, \phi_m))\}$, then being in the state q, reading a and seeing A on the top of the stack, one can move to thje state P_i and replace the top symbol of the stack by the word ϕ_i.
If we have $\delta(q, \epsilon, A) = \{(P_1, \phi_1, \ldots, (P_m, \phi_m))\}$, then being in the state q and seeing the A on the top of the stack (without reading the input), one can go to the state P_i and replace A by ϕ_i.

Definition 18. *A word ω is accepted by a pushdown automata :*

1. *by final states : if reading a word ω, we can reach a final state, then ω is accepted.*
2. *by empty stack : if reading a word ω, we can empty the stack, then ω is accepted.*

Theorem 11.2. *Let M be a PDA, $L(M) = \{w \mid w \text{ is accepted by final state}\}$ and $N(M) = \{w \mid w \text{ is accepted by empty stack}\}$.*

1. *If $L = N(M)$, then $\exists M'$ s.t. $L = L(M')$.*
2. *If $L = L(M)$, then $\exists M'$ s.t. $L = N(M')$.*

Proof. (Idea of)

1. $L = N(M)$: create a new state q_f, add a new starting symbol X_0 and a new initial state q_0'. Finally, we add two transition to δ :

$$\delta(q_0', X_0) \mapsto_\epsilon (q_0, Z_0 X_0)$$
$$\delta(q, X_0) \mapsto_\epsilon (q_f, \epsilon)$$

2. $L = L(M)$: from the final states, empty the stack. We add a new state q_e and the following transition :

$$(q, A) \to (q_e, \epsilon) \to (q_e, \epsilon) \circlearrowleft$$

$\square$

Theorem 11.3.

1. *For every context-free language L, there is a PDA M such that $N(M) = L$.*
2.

12 Properties of context-free languages

13 Turing machine