Languages & Machines

Koen Oostveen

April 7, 2026

Contents

1 Preliminaries

Definition 1.1. An **alphabet** is any finite set Σ. We say that the **language generated** by Σ, Σ^*, is the set of all finite sequences in Σ, that is,

$$\Sigma^* := \{(a_i)_{i=1}^n \mid n \in \mathbb{N}, a_i \in \Sigma\}$$

Elements of Σ^* are called **words**. The unique sequence with length 0 in Σ^* is called the **empty word**, denoted by λ.

Remark 1.1. Lots of peculiarities already. Let $a, b \in \Sigma^*$ be words. Then ab is defined to be the **concatenation** of a with b. More precisely, if $a = (a_i)_{i=1}^n$ and $b = (b_i)_{i=1}^m$, then $ab := (c_i)_{i=1}^{n+m}$ such that

$$c_i := \begin{cases} a_i & \text{if } i \leq n \\ b_{i-n} & \text{otherwise} \end{cases}$$

We also identify letters $a \in \Sigma$ with the word $(a_i)_{i=1}^1$ such that $a_1 := a$.

Lemma 1.1. *Concatenation is associative.*

Definition 1.2. Given an alphabet Σ, we say that a **language** is a subset $L \subseteq \Sigma^*$.

Theorem 1.1. *Let Σ be an alphabet. Then*

- $\lambda \in \Sigma^*$
- *If $a \in \Sigma$ and $w \in \Sigma^*$, then $aw \in \Sigma^*$*

Moreover, if we axiomatically defined Σ^* this way, this would be equivalent to our original definition of Σ^*.

Proof. This is essentially equivalent to

$$\forall w : w \in \Sigma^* \iff w = \lambda \vee \exists a \in \Sigma, w' \in \Sigma^* : aw' = w$$

which kind of pops out trivially. □

Theorem 1.2. *Let $u, v \in \Sigma^*$, then their concatenation satisfies*

$$uv = \begin{cases} v & \text{if } u = \lambda \\ a(wv) & \text{if } u = aw \text{ for some } a \in \Sigma,\ w \in \Sigma^* \end{cases}$$

Moreover, if we recursively defined uv this way, this would be equivalent to our original definition of uv, and associativity therefore also holds.

Definition 1.3. Let $u \in \Sigma^*$ be a word. We define u^R, called the **reverse word** of u, to be

$$u^R := \begin{cases} \lambda & \text{if } u = \lambda \\ w^R a & \text{if } u = aw \text{ for some } a \in \Sigma,\ w \in \Sigma^* \end{cases}$$

Lemma 1.2. *Let $u, v \in \Sigma^*$ be words, then $(uv)^R = v^R u^R$.*

Proof. We do induction over u. If $u = \lambda$, then $(uv)^R = v^R = v^R \lambda^R$. Otherwise, if $u = aw$ for some $a \in \Sigma$, $w \in \Sigma^*$, we have

$$(uv)^R = (a(wv))^R = (wv)^R a = v^R w^R a = v^R (aw)^R = v^R u^R$$

□

Definition 1.4. Let Σ be an alphabet. We define the **language of palidromes** to be

$$L := \{w \in \Sigma^* \mid w = w^R\}$$

Definition 1.5. Let $u \in \Sigma^*$. Let $n \in \mathbb{N}$. Then we define

$$u^n := \begin{cases} \lambda & \text{if } n = 0 \\ uu^m & \text{if } n = \text{succ}(m) \end{cases}$$

Lemma 1.3. *We have $u^m u^n = u^{m+n}$ and $(u^m)^n = m^{mn}$ for all $u \in \Sigma^*$ and $m, n \in \mathbb{N}$.*

Proof. We only prove the first property. We proceed by induction on m. If $m = 0$, then $u^m = \lambda$, so

$$u^m u^n = \lambda u^n = u^n = u^{0+n} = u^{m+n}$$

Otherwise, if $m = \text{succ}(k)$ for some $k \in \mathbb{N}$, then

$$u^m u^n = uu^k u^n = uu^{k+n} = u^{\text{succ}(k+n)} = u^{m+n}$$

□

Definition 1.6. Let $w \in \Sigma^*$ be a word. We define the **length of** w to be the quantity $|w| \in \mathbb{N}$ given by

$$|w| := \begin{cases} 0 & \text{if } w = \lambda \\ \text{succ}(|v|) & \text{if } w = av \text{ for some } a \in \Sigma,\ v \in \Sigma^* \end{cases}$$

Definition 1.7. Let $L_1, L_2 \subseteq \Sigma^*$ be languages. We define the following sets (that are also languages)

$$L_1^R := \{w \in \Sigma^* \mid w^R \in L\}$$

$$L_1 L_2 := \{w \in \Sigma^* \mid \exists w_1 \in L_1, w_2 \in L_2 : w_1 w_2 = w\}$$

Theorem 1.3. *Let $L_1, L_2 \subseteq \Sigma^*$ be languages. Then*

- $|L_1 L_2| \leq |L_1||L_2|$
- $|L_1 \cup L_2| \leq |L_1| + |L_2|$

Definition 1.8. Let $L \subseteq \Sigma^*$ be a language. Let $n \in \mathbb{N}^*$. We define

$$L^n := \begin{cases} \{\lambda\} & \text{if } n = 0 \\ LL^m & \text{if } n = \text{succ}(m) \text{ for some } m \in \mathbb{N} \end{cases}$$

Definition 1.9 (Kleene Star)**.** Let $L \subseteq \Sigma^*$ be a language. We define the following set L^*, called the **Kleene Star of** L:

$$L^* := \bigcup \{L^n \mid n \in \mathbb{N}\}$$

and the **Kleene plus**, L^+, of L:

$$L^+ := \bigcup \{L^n \mid n \in \mathbb{N}^*\} = LL^*$$

Lemma 1.4. *Let L be a language. Then*

- $L^+ = LL^*$
- $L^* = L^+ \cup \{\lambda\}$
- *From the above two it follows; if $\lambda \in L$ then $L^+ = L^*$.*

Definition 1.10. A **regular expression** is a sentence comprised of parentheses (that match!), words, empty sets, unions, concatenations, and Kleene Stars. More precisely, $\varnothing$, λ and all $a \in \Sigma$ are regular expressions. If E_1, E_2 are regular expressions, then

- $E_1 E_2$ is a regular expression,
- $E_1 \cup E_2$ is a regular expression,

- E_1^* is a regular expression.

There exist no others. A bit of syntactic sugar:

$$E_1^+ := E_1 E_1^*$$

which is consistent in the sense that

$$\mathcal{L}(E_1)^+ = \mathcal{L}(E_1)\mathcal{L}(E_1)^* = \mathcal{L}(E_1^+)$$

Definition 1.11. Let E be a regular expression. The **language generated** by E, denoted by $\mathcal{L}(E)$, is given by

- If $E = \varnothing$, then $\mathcal{L}(E) := \varnothing$
- If $E = \lambda$, then $\mathcal{L}(E) := \{\lambda\}$
- If $E = a$ for $a \in \Sigma$ then $\mathcal{L}(E) := \{a\}$
- If $E = E_1 E_2$ then $\mathcal{L}(E) = \mathcal{L}(E_1)\mathcal{L}(E_2)$
- If $E = E_1 \cup E_2$, then $\mathcal{L}(E) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$
- If $E = \widetilde{E}^*$, then $\mathcal{L}(E) = \mathcal{L}(\widetilde{E})^*$

Definition 1.12. Let L be a language. L is called a **regular language** if there exists a regular expression E such that $L = \mathcal{L}(E)$.

Example 1.1. Some examples of regular languages. If $\Sigma := \{a, b, c\}$, we can define the language of all words beginning in ab and ending with ba by the set

$$L := \{w \in \Sigma^* \mid \exists \widetilde{w} \in \Sigma^* : w = abwba\} \cup \{aba\}$$

A regular expression for L is

$$E := \big(ab(a \cup b \cup c)^+ ba\big) \cup (aba)$$

such that $\mathcal{L}(E) = L$. Others are in the slides, I cba

2 DFA's

Definition 2.1. A **deterministic finite automaton**, or DFA for short, is a tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a set of states (any finite set)
- Σ is an alphabet
- $\delta : Q \times \Sigma \to Q$, called the **transition function**
- $q_0 \in Q$, called the **initial state**
- $F \subseteq Q$, called the **accepting states**. We call the set $Q \setminus F$ the set of **nonaccepting states**.

Definition 2.2. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We say that a word $w \in \Sigma^*$ is **accepted** by M, if either

- $w = \lambda$ and $q_0 \in F$, or
- $w = av$ for $a \in \Sigma$, $v \in \Sigma^*$ such that the word v is accepted by $(Q, \Sigma, \delta, \delta(q_0, a), F)$

We say that the **language generated by** M or the **accepting language** of M, denoted by $\mathcal{L}(M)$, is the set

$$\mathcal{L}(M) := \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$$

Definition 2.3. A **configuration** of a DFA $(Q, \Sigma, \delta, q_0, F)$ is an element of $Q \times \Sigma^*$.

Definition 2.4. Let $\sim \subseteq S \times S$ be a binary relation on some set S. We define the **reflexive transitive closure**, $\sim^*$, by

$$\sim^* := \bigcap \{\approx \subseteq S \times S \mid \sim \subseteq \approx \wedge \approx \text{ is reflexive and transitive}\}$$

i.e. the smallest reflexive-transitive relation that contains $\sim$.

Definition 2.5. For a DFA M, we define a relation $\vdash \subseteq (Q \times \Sigma^*)^2$ on all configurations of M by

$$(q, w) \vdash (\widetilde{q}, \widetilde{w}) :\iff \begin{cases} \widetilde{q} = q \wedge \widetilde{w} = w & \text{if } w = \lambda \\ \widetilde{q} = \delta(q, a) \wedge \widetilde{w} = v & \text{if } w = av \text{ for } a \in \Sigma,\ w \in \Sigma^* \end{cases}$$

Informally,

$$(q, aw) \vdash (\delta(q, a), w)$$

We denote its reflexive-transitive closure by $\vdash^*$.

Definition 2.6. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We define the **extended transition function** of M, $\widehat{\delta} : Q \times \Sigma^* \to Q$ by $(q, w) \mapsto \widehat{\delta}(q, w)$ where

$$\widehat{\delta}(q, w) := \begin{cases} q & \text{if } w = \lambda \\ \widehat{\delta}(\delta(q, a), v) & \text{if } w = av \text{ for } a \in \Sigma,\ v \in \Sigma^* \end{cases}$$

Theorem 2.1. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Let $w \in \Sigma^*$ be a word. The following are equivalent:*

- *w is accepted by M*
- *There exists a $q \in F$ such that $(q_0, w) \vdash^* (q, \lambda)$*
- *$\widehat{\delta}(q_0, w) \in F$*

Definition 2.7. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We define the **complement** of M, denoted by M', to be the DFA

$$M' := (Q, \Sigma, \delta, q_0, Q \setminus F)$$

Theorem 2.2. *Let M be a DFA, then $\mathcal{L}(M') = \Sigma^* \setminus \mathcal{L}(M) =: \overline{\mathcal{L}(M)}$.*

Proof. Let w be a word.

$$\begin{aligned} w \in \mathcal{L}(M') &\iff \widehat{\delta}(q_0, w) \in Q \setminus F \\ &\iff \widehat{\delta}(q_0, w) \notin F \\ &\iff w \notin \mathcal{L}(M) \\ &\iff w \in \Sigma^* \setminus \mathcal{L}(M) \end{aligned}$$

□

3 NFA's

Definition 3.1. A **nondeterministic finite automaton**, or NFA for short, is a tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a set of states (any finite set)
- Σ is an alphabet
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$, called the **transition function**
- $q_0 \in Q$, called the **initial state**
- $F \subseteq Q$, called the **accepting states**. We call the set $Q \setminus F$ the set of **nonaccepting states**.

Remark 3.1. Note the definition of $\delta : Q \times \Sigma \to \mathcal{P}(Q)$. Intuitively, this means that for any particular configuration, there may be 0 or more legal next configurations. Note also that every DFA is an NFA (up to injection) but the reverse is not true. The injection (embedding that we use is): $(Q, \Sigma, \delta, q_0, F) \mapsto (Q, \Sigma, \delta', q_0, F)$ where $\delta' : Q \times \Sigma \to \mathcal{P}(Q)$ is given by $(q, w) \mapsto \{\delta(q, w)\}$. We need to redefine acceptance.

Definition 3.2. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We say that a word $w \in \Sigma^*$ is **accepted** by M, if either

- $w = \lambda$ and $q_0 \in F$, or
- $w = av$ for $a \in \Sigma$, $v \in \Sigma^*$, and there exists a $q \in \delta(q_0, a)$ such that the word v is accepted by $(Q, \Sigma, \delta, q, F)$.

Definition 3.3. For an NFA M, we define a relation $\vdash \subseteq (Q \times \Sigma^*)^2$ on all configurations of M by

$$(q, aw) \vdash (\widetilde{q}, w)$$

if $\widetilde{q} \in \delta(q, a)$.

Definition 3.4. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We define the **extended transition function** of M, $\widehat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ by

$$\widehat{\delta}(q, w) := \begin{cases} \{q\} & \text{if } w = \lambda \\ \bigcup\limits_{\widetilde{q} \in \delta(q,a)} \widehat{\delta}(\widetilde{q}, v) & \text{if } w = av \text{ for } a \in \Sigma,\ v \in \Sigma^* \end{cases}$$

Theorem 3.1. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Let $w \in \Sigma^*$ be a word. The following are equivalent:*

- *w is accepted by M*
- *There exists a $q \in F$ such that $(q_0, w) \vdash^* (q, \lambda)$*
- $\widehat{\delta}(q_0, w) \cap F \neq \varnothing$

3.1 NFA with λ-steps

Definition 3.5. A **nondeterministic finite automaton** with λ-steps, or NFA-λ for short, is a tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a set of states (any finite set)
- Σ is an alphabet
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$, called the **transition function**
- $q_0 \in Q$, called the **initial state**
- $F \subseteq Q$, called the **accepting states**. We call the set $Q \setminus F$ the set of **nonaccepting states**.

Definition 3.6. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA-λ. We say that a word $w \in \Sigma^*$ is **accepted** by M, if either

- $w = \lambda$ and $q_0 \in F$, or
- w is accepted by $(Q, \Sigma, \delta, \delta(q_0, \lambda), F)$, or
- $w = av$ for $a \in \Sigma$, $v \in \Sigma^*$, and there exists a $q \in \delta(q_0, a)$ such that the word v is accepted by $(Q, \Sigma, \delta, q, F)$.

Definition 3.7. For an NFA-λ M, we define a relation $\vdash \subseteq (Q \times \Sigma^*)^2$ on all configurations of M by

$$(q, aw) \vdash (\widetilde{q}, w)$$

if $\widetilde{q} \in \delta(q, a)$, or $(q, w) \vdash (\widetilde{q}, w)$ if $\widetilde{q} \in \delta(q, \lambda)$.

Remark 3.2. We skip the definition of the extended transition function, we first need another concept.

3.1.1 λ-closure

Definition 3.8. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA-λ. Let $P \subseteq Q$ be a set of states. We say that P is **λ-closed** if for all $q_1 \in P$, if $q_2 \in \delta(q_1, \lambda)$ then also $q_2 \in P$..

Definition 3.9. Let M be an NFA-λ as before. Let $q \in Q$. We define the **λ-closure of** q to be the set $\Lambda(q)$ given by

$$\Lambda(q) := \bigcap \{P \subseteq Q \mid q \in P \wedge P \text{ is } \lambda\text{-closed}\}$$

In other words, $\Lambda(q)$ is the smallest λ-closed subset of Q that contains q. Intuitively, this is the set of states reachable from q by taking only λ-steps.

Definition 3.10. Let $P \subseteq Q$. We define the λ-closure of P to be the union of the λ-closures of all elements of P, that is

$$\Lambda(P) := \bigcup \{\Lambda(q) \mid q \in P\}$$

Intuitively, this is the set of states reachable by starting at any state in P and then taking only λ-steps.

3.1.2 Extended transition function

Definition 3.11. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We define the **extended transition function** of M, $\widehat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ by

$$\widehat{\delta}(q, w) := \begin{cases} \Lambda(q) & \text{if } w = \lambda \\ \bigcup\limits_{\widetilde{q} \in \delta(q,\lambda)} \Lambda(\widehat{\delta}(\widetilde{q}, av)) \cup \bigcup\limits_{\widetilde{q} \in \delta(q,a)} \Lambda(\widehat{\delta}(\widetilde{q}, v)) & \text{if } w = av \text{ for } a \in \Sigma,\ v \in \Sigma^* \end{cases}$$

Much simpler is:

$$\widehat{\delta}(q, w) := \begin{cases} \Lambda(q) & \text{if } w = \lambda \\ \bigcup\limits_{\widetilde{q} \in \widehat{\delta}(q,v)} \Lambda(\delta(\widetilde{q}, a)) & \text{if } w = va \text{ for } a \in \Sigma,\ v \in \Sigma^* \end{cases}$$

Theorem 3.2. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA-λ. Let $w \in \Sigma^*$ be a word. The following are equivalent:*

- *w is accepted by M*
- *There exists a $q \in F$ such that $(q_0, w) \vdash^* (q, \lambda)$*
- *$\widehat{\delta}(q_0, w) \cap F \neq \varnothing$*

3.1.3 NFA to DFA

Definition 3.12. Let $P \subseteq Q$, let $a \in \Sigma$. We define the **λ-transition** of P when **taking** a, to be the set $\Delta(P, a)$, defined by

$$\Delta(P, a) := \bigcup \{\Lambda(\delta(q, a)) \mid q \in \Lambda(P)\}$$

which makes $\Delta : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$. Using this, we can eliminate the λ-steps by defining the transition function $t : Q \times \Sigma \to \mathcal{P}(Q)$ by

$$(q, a) \mapsto \Delta(\{q\}, a)$$

which yields an NFA (**without** λ-steps) (Q, Σ, t, q_0, F).

Lemma 3.1. *Let $P \subseteq Q$, $a \in \Sigma$,*

$$\Delta(P, a) = \bigcup \{\Delta(\{q\}, a) \mid q \in P\} = \bigcup \{t(q, a) \mid q \in P\}$$

Theorem 3.3. *NFA's, with or without λ-steps, are equivalent to DFA's.*

Proof. We saw that DFA's can be embedded into NFA's. NFA's also can be embedded into NFA-λ's, let $(Q, \Sigma, \delta, q_0, F)$ be an NFA, define the transition function $\delta' : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$ through

$$(q, a) \mapsto \begin{cases} \varnothing & \text{if } a = \lambda \\ \delta(q, a) & \text{otherwise} \end{cases}$$

Then $(Q, \Sigma, \delta', q_0, F)$ is an NFA-λ with the same accepted language.
Now let $(Q, \Sigma, \delta, q_0, F)$ be an NFA-λ. Then $(\mathcal{P}(Q), \Sigma, \Delta, \Lambda(q_0), F')$ with

$$F' := \{X \in \mathcal{P}(Q) \mid X \cap F \neq \varnothing\}$$

is a DFA that accepts the same language. □

Remark 3.3. Note that without λ-steps, it holds that for all $P \subseteq Q$, we have $\Lambda(P) = P$, so that

$$\Delta(P, a) = \bigcup \{\delta(q, a) \mid q \in P\}$$

which makes the equivalence even more apparent.

Remark 3.4. Note that most states in $\mathcal{P}(Q)$ might not be reachable: consider the case where we turned a DFA with states Q into an NFA-λ, and then back, which gets states $\mathcal{P}(Q)$. Note that $|\mathcal{P}(Q)| = 2^{|Q|}$, and of course if $Q \neq \varnothing$ then $|\mathcal{P}(Q)| > |Q|$, even though they accept the same language in the same kind of way.
So the following genius algorithm to construct a DFA from an NFA-λ exists that 'filters out' the unneccesary state: use a graph-search algorithm to visit every state in $\mathcal{P}(Q)$ by applying every possible symbol in Σ to any already visited state $P \in \mathcal{P}(Q)$, where you start with $\Lambda(q_0)$. Say this produces states $Q' \subseteq \mathcal{P}(Q)$, then the transition function is just Δ restricted to Q' in the domain and codomain.

3.1.4 RE to NFA

Definition 3.13. An NFA-λ $M = (Q, \Sigma, \delta, q_0, F)$ is called **composable** if

- There does not exist $q \in Q$ and $a \in \Sigma$ such that $q_0 \in \delta(q, a)$, and
- $F = \{q_f\}$ is a singleton set, and
- For all $a \in \Sigma$, $\delta(q_f, a) = \varnothing$.

Definition 3.14. Let $M = (Q_M, \Sigma, \delta_M, m_0, \{m_f\})$ and $N = (Q_N, \Sigma, \delta_N, n_0, \{n_f\})$ be composable NFA-λ's. We define the **composition** of M with N, denoted $M \circ N$, to be the NFA-λ

$$M \circ N := (Q_M \sqcup Q_N, \Sigma, \delta_{M\circ N}, m_0, \{n_f\})$$

where $\delta_{M\circ N} : (Q_M \sqcup Q_N) \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(()Q_M \sqcup Q_N)$ is given by

$$\delta_{M\circ N}(q, a) := \begin{cases} n_0 & \text{if } q = m_f \text{ and } a = \lambda \\ \varnothing & \text{if } q = m_f \text{ and } a \neq \lambda \\ \delta_M(q, a) & \text{if } q \in Q_M \\ \delta_N(q, a) & \text{if } q \in Q_N \end{cases}$$

Lemma 3.2. *The composition of composable NFA-λ's is composable.*

Proof. We go by each property.

- If $n_0 \in \delta(q, a)$ for some q, a, in perticular because $n_0 \in Q_N$ we have that $n_0 \in \delta_N(q, a)$, which is a contradiction.
- $\{m_f\}$ is clearly a singleton.
- Let $a \in \Sigma$. Then $\delta(m_f, a) = \delta_M(m_f, a) = \varnothing$. □

Theorem 3.4. *Let E be a regular expression over an alphabet Σ. Then there exists a (composable) NFA-λ that accepts the same language.*

Proof. We proceed by induction on E. The base cases are:

- If $E = \varnothing$, $\mathcal{L}(E) = \varnothing$, and for $M = (\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_1\})$ with $\delta(q, a) := \varnothing$ gives $\mathcal{L}(M) = \varnothing$.
- If $E = \lambda$, then $\mathcal{L}(E) = \{\lambda\}$, and for $M = (\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_1\})$ with

  $$\delta(q, a) := \begin{cases} \{q_1\} & \text{if } q = q_0 \text{ and } a = \lambda \\ \varnothing & \text{otherwise} \end{cases}$$

  gives $\mathcal{L}(M) = \{\lambda\}$, note that M is composable.
- If $E = b$ for some $b \in \Sigma$, then $\mathcal{L}(E) = \{b\}$, and for $M = (\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_1\})$ with

  $$\delta(q, a) := \begin{cases} \{q_1\} & \text{if } q = q_0 \text{ and } a = b \\ \varnothing & \text{otherwise} \end{cases}$$

  gives $\mathcal{L}(M) = \{b\}$, note that M is composable.

Suppose that we have composable M_1, M_2 for E_1, E_2 such that $\mathcal{L}(E_1) = \mathcal{L}(M_1)$ and $\mathcal{L}(E_2) = \mathcal{L}(M_2)$, we show that we can construct composable NFA-λ's for E_1E_2, $E_1 \cup E_2$ and E_1^*. Note that $\mathcal{L}(M_1 \circ M_2) = \mathcal{L}(M_1)\mathcal{L}(M_2) = \mathcal{L}(E_1)\mathcal{L}(E_2) = \mathcal{L}(E_1E_2)$. We can do very similar things for $E_1 \cup E_2$ and E_1^*, but this is a lot of effort and can most easily be summarized in a diagram. □

3.2 NFA to RE

Definition 3.15. An **expression graph** is an NFA-λ, where the labels are replaced with regular expressions. We get rid of the transition function δ and instead specify just the inferences. Given a current state $q \in Q$ and a word $w \in \Sigma^*$, we define $(q, w) \vdash (\widetilde{q}, \widetilde{w})$ if there exists a regular expression E on the edge from q to $\widetilde{q}$ such that there exists a word v such that $w = v\widetilde{w}$ and v is accepted by E.
More formally, an expression graph is a tuple $(Q, \Sigma, E, L, q_0, F)$ such that

- Q is any finite set of states
- Σ is any alphabet
- (Q, E) is a directed graph
- L is a function from E to regular expressions in Σ
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a set of accepting states

for which the relation $\vdash$ is given by: $(q, ab) \vdash (\widetilde{q}, b)$ if $L(q, \widetilde{q})$ accepts a.

Remark 3.5. Because the 'edges' of an NFA-λ are characters in Σ or λ, it is trivially an expression graph, by inclusion of characters and λ into the regular expressions. The only issue is when there are multiple 'edges' with different symbols on them, this can be resolved by merging them together and creating a '$\cup$' type regex.
The bigger picture idea is now to take this expression graph and simplify it until there are only various edges remaining and the resulting regex can be readily read off.

Theorem 3.5. *Let $(Q, \Sigma, E, L, q_0, F)$ be an expression graph. Let $q_1, q_2, q_3 \in Q$ be states such that there exists an edge (q_1, q_2) and an edge (q_2, q_3) in E, as well as $q_1 \neq q_2$ and $q_2 \neq q_3$, but possibly $q_1 = q_3$. Then by deleting q_2 and creating an edge (q_1, q_3) with the expression*

$$\begin{cases} L(q_1, q_2)L(q_2, q_3) & \text{if } (q_2, q_2) \notin E \\ L(q_1, q_2)L(q_2, q_2)^* L(q_2, q_3) & \text{if } (q_2, q_2) \in E \end{cases}$$

for every possible combination of q_1, q_2, q_3 like the above, yields an expression graph that accepts the same language.

3.3 Overarching conclusion

We have shown that NFA-λ's and DFA's are equivalent, as well as NFA-λ's being equivalent to regular expressions, so the class of languages definable through regexes, DFA's and NFA's are all the same: the class of regular languages.

4 Pumping lemma

Recall the Pigeonhole principle.

Theorem 4.1. *Let A and B be finite sets. If $|A| > |B|$, then there does not exist an injection $f : A \to B$. That is, for every $f : A \to B$, there exists $x, y \in A$ such that $f(x) = f(y)$ but $x \neq y$.*

Now suppose you have a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts a word w with $|w| \geq |Q|$. Note that then, the function $f : \{0, \ldots, |w|\} \to Q$ mapping a character index in w to the state the DFA is in after consuming the character at that index, with $f(0) := q_0$. Then $|\{0, \ldots, w\}| = |w| + 1 \geq |Q| + 1 > |Q|$, hence f cannot be an injection, hence there must exist a state $q \in Q$ that is visited twice. From this we obtain the pumping lemma.

Lemma 4.1. *Let L be a regular language. Then there exists a $k > 0$ such that for all words $z \in L$ with $|w| \geq k$, we can write $z = uvw$ with $|uv| \leq k$, $v \neq \lambda$, and for all $i \in \mathbb{N}$, $uv^i w \in L$.*

Proof. Let L be a regular language, let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that generates it, i.e. $\mathcal{L}(M) = L$, which exists by equivalence of DFA languages and regular languages. Let $k := |Q|$, let $z \in L$ be such that $|z| \geq k$. By the previous argument this gives us a state $q \in Q$ that is visited twice while accepting z. Let $z = z_1 \cdot \ldots \cdot z_n$, $z_1, \ldots, z_n \in \Sigma$. Suppose that state q occurs at indices $i, j \in \mathbb{N}$ of z, wlog suppose $i < j$. Let $u := w_1 \cdot \ldots \cdot w_i$, let $v := w_{i+1} \cdot \ldots \cdot w_j$, let $w := w_{j+1} \cdot \ldots \cdot w_n$, then clearly $z = uvw$. Also, because $i + 1 \leq j$, we have $v \neq \lambda$. Note that $|uv| \leq k$ would follow if $j \leq k$, which it wlog could by only considering those indices of z, the pigeonhole principle still holds. Note that by construction,

$$\widehat{\delta}(q_0, u) = q, \qquad \widehat{\delta}(q, v) = q, \qquad \widehat{\delta}(q, w) \in F$$

Now by doing a hidden induction, $\widehat{\delta}(q, v^i) = q$ pops right out, whence $\widehat{\delta}(q_0, uv^i w) \in F$ for all i, so $uv^i w \in F$. □

Corollary. *Any regular language L generated by a DFA that accepts a word with length at least the amount of states it has, is infinite.*

5 DFA minimization

Definition 5.1. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We say that two states $q_1, q_2 \in Q$ are **equivalent** if they accept the same language, that is,

$$\mathcal{L}(Q, \Sigma, \delta, q_1, F) = \mathcal{L}(Q, \Sigma, \delta, q_2, F)$$

Definition 5.2. A DFA $(Q, \Sigma, \delta, q_0, F)$ is called **minimal** if

- every state has a word that produces is, that is, for every $q \in Q$ there exists a $w \in \Sigma^*$ such that $\widehat{\delta}(q_0, w) = q$, and
- for any two states $q_1, q_2 \in Q$, if they are equivalent, then $q_1 = q_2$.

Definition 5.3. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. A binary relation D on Q is called **distinguishing** if for all $x, y \in Q$ and $a \in \Sigma$ we have

- If $x \in F \iff y \notin F$ then xDy,
- If $\delta(x, a) D \delta(y, a)$ then xDy

The **distinguishability relation** is the intersection (the smallest) of the distinguishing relations on M.

Lemma 5.1. *The negation of the distinguishability relation determines equivalence of states, which makes it an equivalence relation.*

Theorem 5.1. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA such that every state is reachable. Let $D \subseteq Q \times Q$ be the distinguishability relation on M. Let $\neg D$ be its negation, i.e. $x(\neg D)y :\iff \neg xDy$. Then the DFA*

$$M' := (Q/\neg D, \Sigma, \delta', [q_0]_{\neg D}, F')$$

where $\delta' : Q/\neg D \times \Sigma \to Q/\neg D$ is (well-)defined by

$$([q], a) \mapsto [\delta(q, a)]$$

and

$$F' := \{[q] \in Q/\neg D \mid q \in F\}$$

makes M' a minimal equivalent DFA of M.

Proof. There are some comments below here that attempt to prove this. □

Definition 5.4. Let $M = (Q_M, \Sigma, \delta_M, m_0, F_M)$ and $N = (Q_N, \Sigma, \delta_N, n_0, F_N)$ be DFA's. A function

$$f : Q_M \to Q_N$$

is called a **DFA isomorphism** from M to N if

- f is invertible,
- for all $q \in Q_M$ and $a \in \Sigma$, we have $f(\delta_M(q, a)) = \delta_N(f(q), a)$,
- $F_N = f(F_M)$,
- $n_0 = f(m_0)$.

If such an f exists, we say that M and N are **isomorphic** or **equal up to isomorphism**.

Theorem 5.2. *For every DFA M, a unique (up to isomorphism) DFA M' exists that accepts the same language, i.e. $\mathcal{L}(M) = \mathcal{L}(M')$.*

6 Composition of automata

Definition 6.1. Let M_1 and M_2 be DFA/NFA/NFA-λ's. We say that a DFA/NFA/NFA-λ $M_1 \times M_2$ is the **product** of M_1 and M_2 if

$$\mathcal{L}(M_1 \times M_2) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$$

Lemma 6.1. *Let $M = (Q_M, \Sigma, \delta_M, m_0, F_M)$ and $N = (Q_N, \Sigma, \delta_N, n_0, F_N)$ be DFA's. A unique (up to isomorphism) product of M and N is given by*

$$M \times N := (Q_M \times Q_N, \Sigma, \delta, (m_0, n_0), F_M \times F_N)$$

where $\Sigma := \Sigma_M \cap \Sigma_N$,

$$\delta : (Q_M \times Q_N) \times \Sigma \to Q_M \times Q_N$$

is given by

$$((q_m, q_n), a) \mapsto (\delta_M(q_m, a), \delta_N(q_n, a))$$

Lemma 6.2. *Let $M = (Q_M, \Sigma_M, \delta_M, m_0, F_M)$ and $N = (Q_N, \Sigma_N, \delta_N, n_0, F_N)$ be NFA-λ's. A unique (up to isomorphism) product of M and N is given by the same as before, where*

$$\delta : (Q_M \times Q_N) \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q_M \times Q_N)$$

is given by

$$((q_m, q_n), a) \mapsto \begin{cases} (\delta_M(q_m, \lambda) \times \{q_n\}) \cup (\{q_m\} \times \delta_N(q_n, \lambda)) & \text{if } a = \lambda \\ \delta_M(q_m, a) \times \delta_N(q_n, a) & \text{otherwise} \end{cases}$$

Definition 6.2. Let M_1 and M_2 be DFA/NFA/NFA-λ's. We say that a DFA/NFA/NFA-λ $M_1 \parallel M_2$ is the **parallel composition** of M_1 and M_2 if

$$\mathcal{L}(M_1 \parallel M_2) = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$$

Definition 6.3. Let $M = (Q_M, \Sigma_M, \delta_M, m_0, F_M)$ and $N = (Q_N, \Sigma_N, \delta_N, n_0, F_N)$ be NFA-λ's. We say that the **parallel composition** of M and N is given by

$$M \parallel N := (Q_M \times Q_N, \Sigma, \delta, (m_0, n_0), F_M \times F_N)$$

where $\Sigma := \Sigma_M \cup \Sigma_N$,

$$\delta : (Q_M \times Q_N) \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q_M \times Q_N)$$

is given by

$$((q_m, q_n), a) \mapsto \begin{cases} \delta_{M\times N}((q_m, q_n), a) & \text{if } a \in (\Sigma_M \cap \Sigma_N) \cup \{\lambda\}, \\ \delta_M(q_m, a) \times \{q_n\} & \text{if } a \in \Sigma_M \text{ but } a \notin \Sigma_N, \\ \{q_m\} \times \delta_N(q_n, a) & \text{if } a \in \Sigma_N \text{ but } a \notin \Sigma_M. \end{cases}$$

where $\delta_{M\times N}$ refers to the previous definition of δ for $M \times N$.

Definition 6.4. Let Σ be an alphabet. Let $w \in \Sigma^*$. Let $\widetilde{\Sigma} \subseteq \Sigma$. We say that the **projection** of w **onto** $\widetilde{\Sigma}$ (in Σ) is

$$w \upharpoonright \widetilde{\Sigma} := \begin{cases} (a \upharpoonright \widetilde{\Sigma})(v \upharpoonright \widetilde{\Sigma}) & \text{if } w = av \text{ for some } a \in \Sigma \text{ and } v \in \Sigma^*, \\ \lambda & \text{otherwise.} \end{cases}$$

where for $a \in \Sigma$ we define

$$a \upharpoonright \widetilde{\Sigma} := \begin{cases} a & \text{if } a \in \widetilde{\Sigma} \\ \lambda & \text{otherwise} \end{cases}$$

Definition 6.5. Let Σ be an alphabet. Let $L \subseteq \Sigma^*$ be a language. Let $\widetilde{\Sigma} \subseteq \Sigma$. We say that the **projection** of L **onto** $\widetilde{\Sigma}$ (in Σ) is

$$L \upharpoonright \widetilde{\Sigma} := \{w \upharpoonright \widetilde{\Sigma} \mid w \in L\}$$

Definition 6.6. Let Σ be an alphabet. Let $\widetilde{\Sigma} \subseteq \Sigma$. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA-λ. We say that the **projection** of M **onto** $\widetilde{\Sigma}$ is the NFA-λ $\widetilde{M} := (Q, \widetilde{\Sigma}, \widetilde{\delta}, q_0, F)$ where

$$\widetilde{\delta} : Q \times (\widetilde{\Sigma} \cup \{\lambda\}) \to \mathcal{P}(Q)$$

is given by

$$(q, a) \mapsto \delta(q, a) \cup \begin{cases} \bigcup_{\widetilde{a} \in \Sigma \setminus \widetilde{\Sigma}} \delta(q, \widetilde{a}) & \text{if } a = \lambda, \\ \varnothing & \text{otherwise.} \end{cases}$$

Theorem 6.1. *Let $M = (Q_M, \Sigma_M, \delta_M, m_0, F_M)$ and $N = (Q_N, \Sigma_N, \delta_N, n_0, F_N)$ be NFA-λ's. Then, for all $w \in \Sigma^*$ where $\Sigma := \Sigma_M \cup \Sigma_N$, it holds that*

$$w \in \mathcal{L}(M \parallel N) \iff w \restriction \Sigma_M \in \mathcal{L}(M) \wedge w \restriction \Sigma_N \in \mathcal{L}(N)$$

7 Grammars

Definition 7.1. Let

- V be any finite set, called the **variables**,
- Σ be an alphabet, such that $V \cap \Sigma = \varnothing$,
- $P \subseteq V \times (V \cup \Sigma)^*$, called the **production rules**,
- $S \in V$, called the **start symbol**.

The tuple (V, Σ, P, S) is then called a **context free grammar**. If $(A, w) \in P$, we write $A \to w$, read 'A produces w'. If a sequence $(A, w_1), \ldots, (A, w_n) \in P$, we also write

$$A \to w_1 \mid \ldots \mid w_n$$

Definition 7.2. Let (V, Σ, P, S) be a context-free grammar. Let $A \in V$ and $w \in (V \cup \Sigma)^*$ such that $A \to w$. If $u, v \in \Sigma^*$, we say that the uwv is **derived from** the word $uAv \in (V \cup \Sigma)^*$. We sometimes write

$$u\mathbf{A}v \implies u\mathbf{w}v$$

Definition 7.3. Let (V, Σ, P, S) be a context-free grammar. Let $v \in (V \cup \Sigma)^*$ be a string. We say that a set $D \subseteq (V \cup \Sigma)^*$ is **closed under derivation** (from v) if

- $v \in D$, and
- for all $x, y \in (V \cup \Sigma)^*$ such that xAy and $(A \to w) \in P$, then $xwy \in D$.

We define the set $\mathrm{Der}(v)$ to be the smallest such set that contains v, i.e.

$$\mathrm{Der}(v) := \bigcap \{D \subseteq (V \cup \Sigma)^* \mid v \in D \wedge D \text{ is closed under derivation}\}$$

If $w \in \mathrm{Der}(v)$ we also write

$$v \overset{*}{\implies} w$$

read: 'v **derives** w' or 'w is **derivable from** v'.

Definition 7.4. Let (V, Σ, P, S) be a CFG. Let $w \in \Sigma^*$ be derivable from S, i.e. $S \overset{*}{\implies} w$. The **derivation tree** from S to w given a derivation is defined to be the tree with root S, such that when at a certain leaf with variable A, a certain rule $A \to x_1 \cdot \ldots \cdot x_n$ is applied, we add $x_1, \ldots, x_n$ as children to A (where every $x_i \in V \cup \Sigma$).

Definition 7.5. A CFG (V, Σ, P, S) is called **ambiguous** if multiple possible leftmost derivations/ derivation trees exist for a certain w such that $S \overset{*}{\implies} w$.

Definition 7.6. Let $G = (V, \Sigma, P, S)$ be a CFG. Let $w \in (V \cup \Sigma)^*$.

- If w is derivable from S, we call w **sentential** in G
- If additionally $w \in \Sigma^*$ (it contains no variables), w is called a **sentence** in G
- The set of all sentences in G is called the **language** of G

$$\mathcal{L}(G) := \{w \in \Sigma^* \mid S \overset{*}{\implies} w\}$$

Definition 7.7. Let Σ be an alphabet. Let $L \subseteq \Sigma^*$ be a language. We say that L is **context-free** if a CFG G with alphabet Σ exists such that $\mathcal{L}(G) = L$.

Definition 7.8. Let G_1, G_2 be CFG's. We say that G_1 and G_2 are **equivalent** if their languages coincide, that is

$$\mathcal{L}(G_1) = \mathcal{L}(G_2)$$

7.1 Regular grammars

Definition 7.9. A grammar G is called **regular** if every rule is of the form (for some $A, B \in V$ and $a \in \Sigma$)

- $A \to a$, or
- $A \to aB$, or
- $A \to \lambda$.

Remark 7.1. There exist equivalent CFG's where one is regular and the other is not (quite obviously, I suppose?).

Theorem 7.1. *NFA languages (and thus regular languages) are context-free. Moreover (and we will not show this) for every regular grammar, there exists a DFA that accepts the same language, ergo, regular grammars and regular languages coincide.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Let $G = (Q, \Sigma, P, q_0)$ be a CFG where P we will now construct. Remember that $\delta : Q \times \Sigma \to \mathcal{P}(Q)$. We define

$$P := \{q \to a\widetilde{q} \mid \widetilde{q} \in \delta(q, a) \wedge q \in Q \wedge a \in \Sigma\} \cup \{(q, \lambda) \mid q \in F\}$$

Clearly, $P \subseteq (Q \cup \Sigma)^*$, $q_0 \in Q$, so yes, G is a context free grammar. We show that $\mathcal{L}(M) = \mathcal{L}(G)$. Notice that every inference $[q, aw] \vdash [\delta(q, a), w]$ corresponds one-to-one with a derivation step $q \to a\delta(q, a)$, and from this essentially the rest of the proof follows.
Because every NFA has a corresponding CFG that is regular, NFA languages are also context-free. □

Remark 7.2. **Parsing** is the act of taking a word $w \in \mathcal{L}(G)$ for some unambiguous grammar G and finding the derivation from the starting symbol.
Unambiguous grammars exist for regular languages.

7.2 Normal forms

Lemma 7.1. *Let $G = (V, \Sigma, P, S)$ be a grammar. Let $\widetilde{S} \notin V \cup \Sigma$ be any set. Then*

$$\widetilde{G} := (V \cup \{\widetilde{S}\}, \Sigma, P \cup \{\widetilde{S} \to S\}, \widetilde{S})$$

satisfies $\mathcal{L}(\widetilde{G}) = \mathcal{L}(G)$.

Proof. We show inclusion both ways. Let $w \in \mathcal{L}(G)$, suppose we have a derivation $S \overset{*}{\Longrightarrow} w$, then clearly $\widetilde{S} \implies S \overset{*}{\Longrightarrow} w$ is a derivation for w in $\widetilde{G}$, whence $w \in \mathcal{L}(\widetilde{G})$.
Now suppose $w \in \mathcal{L}(\widetilde{G})$. Its derivation starts with $\widetilde{S} \to S$, because $\widetilde{S}$ does not have any other production rules in P, since $\widetilde{S}$ is not in V or Σ, whence $S \overset{*}{\Longrightarrow} w$ is a valid derivation of w in G, whence $w \in \mathcal{L}(G)$. □

Lemma 7.2. *Let $G = (V, \Sigma, P, S)$ be a grammar. Let $u, v \in (V \cup \Sigma)^*$ and $A, B \in V$ such that there exists a set $\widetilde{P}$ such that*

$$P = \widetilde{P} \cup \{A \to uBv\}$$

Let

$$\widetilde{G} := (V, \Sigma, P')$$

with

$$P' := \widetilde{P} \cup \{(A \to uwv) \mid (B \to w) \in P, w \in (V \cup \Sigma)^*\}$$

Then $\widetilde{G}$ and G are equivalent in the sense that $\mathcal{L}(\widetilde{G}) = \mathcal{L}(G)$.

Definition 7.10. Let $G = (V, \Sigma, P, S)$ be a CFG. We say that G is in **Chomsky normal form** if for all rules $(A \to w) \in P$ for some $A \in V$ and $w \in (\Sigma \cup V)^*$ we have that either

- there exist $B, C \in V \setminus \{S\}$ such that $w = BC$, or
- there exists $a \in \Sigma$ such that $w = a$, or
- $A = S$ and $w = \lambda$.

Theorem 7.2. *For every CFG, there exists an equivalent CFG in Chomsky normal form (CNF). That is, for all CFG's G, there exists $\widetilde{G}$ in CNF such that $\mathcal{L}(G) = \mathcal{L}(\widetilde{G})$.*

Proof. We do this by explicit construction, that is, providing an algorithm to produce from any grammar $G = (V, \Sigma, P, S)$ an equivalent grammar in CNF. We do this in several steps. First of all, we use the lemma from before where we introduce a nonrecursive start symbol S by introducing $\widetilde{S} \to S$, so without loss of generality we can suppose the start symbol does not contain recursive production rules.
Let $N \subseteq V$ be the smallest set such that for all $A \in N$, there exists $w \in N^*$ such that $A \to w$ is a rule in P, as well as that N that for all $A \in V$ such that $A \to \lambda$ is a rule in P, $A \in N$.
In a sense, the set N is the set of all variables for which a λ derivation exists. We now define P_1, the set of production rules after the first step, λ-elimination. We first define $\widetilde{P} \supseteq P$ by including all of P and then for all $(A \to w) \in P$, if there exist $A_1, \ldots, A_k \in N$ and words $w_1, \ldots, w_{k+1} \in (\Sigma \cup V)^*$ such that

$$w = w_1 A_1 w_2 A_2 \cdot \ldots \cdot w_k A_k w_{k+1}$$

we add $A \to w_1 w_2 \cdot \ldots \cdot w_{k+1}$ in $\widetilde{P}$. Then we define

$$P_1 := \{(A \to w) \in \widetilde{P} \mid w = \lambda \implies A = S\}$$

This is the set of rules after the first step. It is somewhat annoying but trivial to show that then $G_1 := (V, \Sigma, P_1, S)$ is an equivalent grammar that has no λ-productions except possibly $S \implies \lambda$.
Let $A \in V$, then define the set

$$\text{Chain}(A) := \{B \in V \mid (A \overset{*}{\Longrightarrow} B) \in P_1\}$$

Then we define P_2 to be the set of production rules where for all $A \in V$, $B \in \text{Chain}(A)$, and $w \in (\Sigma \cup V)^*$ with $w \notin V$ and $B \to w$ in P_1, we add $A \to w$ in P_2, there exist no other rules. One may again show

that $G_2 := (V, \Sigma, P_2, S)$ yields $\mathcal{L}(G_2) = \mathcal{L}(G_1) = \mathcal{L}(G)$.
Finally, we bring G_2 to normal form. There exist no λ productions and no chain rules, that is, transitive dependencies, in a sense. Therefore, every rule $A \to w$ with $|w| < 2$ already satisfies the CNF requirement. Now if $|w| \geq 2$ then we write

$$w = av$$

for $a \in V \cup \Sigma$, $v \in (V \cup \Sigma)^*$. If $a \in \Sigma$, let B be any set disjoint with V and Σ, and update $V := V \cup \{B\}$, so in a sense, without loss of generality, suppose $a \in V$. Then we define a new rule $w = aC$ for some new variable C with update $V := V \cup \{C\}$ where C has a single production rule, namely $C \to v$. We repeat this procedure, if necessary ($|v| \geq 2|$) for C. After this, we are left with an updated set of variables $\widetilde{V}$ and an updated set of production rules P_3 such that $G_3 := (\widetilde{V}, \Sigma, P_3, S)$ satisfies $\mathcal{L}(G_3) = \mathcal{L}(G_2) = \ldots = \mathcal{L}(G)$, and G_3 is in CNF is then left to show, which is true by construction (one may check this). □

7.3 Optimization of CNF

One may detect rules that do not terminate by backwardsly substituting all rules that DO terminate, which gives a set $T \subseteq V$ for a grammar $G = (V, \Sigma, P, S)$ in CNF for which all inferences of T do not result in loops. We define

$$\widetilde{P} := \{(A \to w) \in P \mid A \in T\}$$

and then $\widetilde{G} := (T, \Sigma, \widetilde{P}, S)$ is an equivalent grammar, still in CNF (because we just deleted rules).
Likewise, we may find the set of unreachable variables (ones that may never be produced) by first finding the set of all variables that do get produced by forwardsly substituting, starting with just the set containing the starting variable. Let $R \subseteq V$ be the set of all reachable variables, that is, all variables for which there exists a derivation from S to a word that contains that variable. Then $\widetilde{G} := (R, \Sigma, \widetilde{P}, S)$ with

$$\widetilde{P} := \{(A \to w) \in P \mid A \in R\}$$

yields an equivalent grammar, that is potentially smaller.
Summary: to optimize the CNF (or any grammar, really), we first delete all nonterminating variables, and then we delete all unreachable variables.

7.4 CYK parsing

Let $G = (V, \Sigma, P, S)$ be a grammar in CNF. Let $w \in (\Sigma \cup V)^*$ be a word to be parsed (remember, that means finding a derivation for it, finding a parse tree). Let $w = a_1 \cdot \ldots \cdot a_n$ for $n \in \mathbb{N}$, $a_i \in V \cup \Sigma$ for all $i = 1, \ldots, n$. We define the subword $w_{i,j}$ for all $1 \leq i \leq j \leq n$ and $i, j \in \mathbb{N}$ by

$$w_{i,j} := a_i \cdot a_{i+1} \cdot \ldots \cdot a_j$$

Then we define the set $X_{i,j} \subseteq V$ to be the set of variables that have a derivation to $w_{i,j}$. This set can be found by **dynamic programming**. Start first with the set $X_{i,j}$ with $i = j$. This is easy, (the base case)

$$X_{i,i} := \{A \in V \mid (A \to a_{i,i}) \in P\}$$

Then for all $L = 2, \ldots, n$, in that order, and all $i = 1, \ldots, n - L + 1$, take $j := i + L$ (L is thus the length of the subword we try to parse), then a variable A may parse $w_{i,j}$ only if there exists a rule $A \to BC$ such that B can parse a part of it, and C can parse the other part, i.e.

$$X_{i,j} := \{A \in V \mid \exists k = i, \ldots, j-1 : \exists B \in X_{i,k}, C \in X_{k+1,j} : (A \to BC) \in P\}$$

and indeed if we iterate in the appropriate order, this will work out. Now clearly $w \in \mathcal{L}(G)$ if and only if $S \in X_{1,n}$, but this is not yet the same as parsing, of course. Though, the derivation tree may be readily read off 'tracing the algorithm backwards'.

8 Pushdown automata

Definition 8.1. Let X be any set that does not contain the empty word λ. We define (for convenience) the set $X^\lambda := X \cup \{\lambda\}$.

Definition 8.2. Let

- Q be any set, the **set of states**,
- Σ and Γ be alphabets, called the **input** and **stack** alphabets, respectively,
- $\delta : Q \times \Sigma^\lambda \times \Gamma^\lambda \to \mathcal{P}\big(\Gamma^\lambda \times Q\big)$, the **transition function**,
- $q_0 \in Q$, the **initial state**,
- $F \subseteq Q$, the **accepting states**.

Then the tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ is called a **pushdown automaton**. Note that the book models δ as a relation

$$\delta \subseteq Q \times \Sigma^\lambda \times \Gamma^\lambda \times \Gamma^\lambda \times Q$$

Definition 8.3. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a pushdown automaton. We say that a tuple $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ is a **configuration** of the automaton. We define the relation $\vdash$ on configurations by: If $(q, a, o, i, \widetilde{q}) \in \delta$ and there exists $v \in \Sigma^*$ such that $w = av$ and there is some $\beta \in \Gamma^*$ such that $\alpha = o\beta$ ($a, o = \lambda$ is also possible) then

$$(q, av, o\beta) \vdash (\widetilde{q}, v, i\beta)$$

We define $\vdash^*$ to be the reflexive-transitive closure of $\vdash$. Then we define the **language** of the automaton through

$$\mathcal{L}(M) := \{w \in \Sigma^* \mid \exists q \in F : (q_0, w, \lambda) \vdash^* (q, \lambda, \lambda)\}$$

Remark 8.1. Clearly, every NFA-λ and thus every other automaton discussed so far can be transformed into a pushdown automaton by having an empty / arbitrary stack alphabet, and never pushing to the stack in δ. But, we can show that for example, the **non-regular** language

$$\{a^i b^i \mid i \in \mathbb{N}\}$$

is definable through a pushdown automaton: let $Q := \{q_0, q_1\}$, $\Sigma := \{a, b\}$, $\Gamma := \{a\}$, $F := \{q_1\}$, then

$$\delta := \{(q_0, a, \lambda, a, q_0), (q_0, b, a, \lambda, q_1), (q_1, b, a, \lambda, q_1)\}$$

Definition 8.4. We say that the transitions $(q, x_1, A_1, B_1, \widetilde{q_1}), (q, x_2, A_2, B_2, \widetilde{q_2}) \in \delta$ overlap if

- $x_1 = x_2$, or at least one of x_1, x_2 is λ, and
- $A_1 = A_2$, or at least one of A_1, A_2 is λ

Intuitively, this means that you 'cannot tell' which transition will happen.

Definition 8.5. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA. We say that M is **deterministic** if for any two transitions $\delta_1, \delta_2 \in \delta$, if δ_1 and δ_2 overlap, then $\delta_1 = \delta_2$, i.e. no distinct overlapping transitions exist.

Example 8.1. Let Σ be any alphabet. Let

$$L := \{ww^R \mid w \in \Sigma^*\}$$

be the language of palindromes. Recall that L is non-regular. But we can construct the PDA $M := (Q, \Sigma, \Sigma, \delta, q_0, F)$ where $Q := \{q_0, q_1\}$, $F := \{q_1\}$, and

$$\delta := \{(q_0, a, \lambda, a, q_0) \mid a \in \Sigma\} \cup \{(q_1, a, a, \lambda, q_1) \mid a \in \Sigma\} \cup \{(q_0, \lambda, \lambda, \lambda, q_1)\}$$

Then $\mathcal{L}(M) = L$. Note that M is **not** deterministic (check!).

Remark 8.2. In the remainder of the lecture this section belongs to, we research whether CFG languages and PDA languages are equivalent. But we can already say something about unambiguity not being equivalent to determinism on PDA's: note that the language of palindromes has the unambiguous grammar (V, Σ, P, S) where $V = \{S\}$, and

$$P := \{S \to aSa \mid a \in \Sigma\} \cup \{S \to \lambda\}$$

is unambiguous, but no deterministic PDA exists.

Definition 8.6 (Alternative notions of the language of a PDA)**.** Let $M := (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA. We define the **final state language** $\mathcal{L}_F(M)$ of M through

$$\mathcal{L}_F(M) := \{w \in \Sigma^* \mid \exists q \in F : \exists \alpha \in \Gamma^* : (q_0, w, \lambda) \vdash^* (q, \lambda, \alpha)\}$$

We define the **empty stack language** $\mathcal{L}_E(M)$ of M through

$$\mathcal{L}_E(M) := \{w \in \Sigma^* \mid \exists q \in Q : (q_0, w, \lambda) \vdash^* (q, \lambda, \lambda)\}$$

Lemma 8.1. $\mathcal{L}(M) \subseteq \mathcal{L}_F(M)$

Lemma 8.2. *Let $M := (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA. Then there exists a PDA $\widetilde{M}$ such that*

$$\mathcal{L}(M) = \mathcal{L}_F(\widetilde{M})$$

Proof. Let $q_s, q_f \notin Q$ be any sets, let $Z \notin \Gamma$ also be any set. Define $\widetilde{Q} := Q \cup \{q_s, q_f\}$, $\widetilde{\Gamma} := \Gamma \cup \{Z\}$, and

$$\widetilde{\delta} := \delta \cup \{(q_s, \lambda, \lambda, Z, q_0)\} \cup \{(q, \lambda, Z, \lambda, q_f) \mid q \in F\}$$

Then the PDA $\widetilde{M} := (\widetilde{Q}, \Sigma, \widetilde{\Gamma}, \widetilde{\delta}, q_s, \varnothing)$ by construction satisfies $\mathcal{L}_F(\widetilde{M}) = \mathcal{L}(M)$. □

Lemma 8.3. *Let $M := (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA. Then there exists a PDA $\widetilde{M}$ such that*

$$\mathcal{L}_F(M) = \mathcal{L}(\widetilde{M})$$

Proof. Let $q_f \notin Q$ be any set. Define $\widetilde{Q} := Q \cup \{q_f\}$, $\widetilde{F} := \{q_f\}$ and

$$\widetilde{\delta} := \delta \cup \{(q, \lambda, \lambda, \lambda, q_f) \mid q \in F\} \cup \{(q_f, \lambda, a, \lambda, q_f) \mid a \in \Gamma\}$$

Then the PDA $\widetilde{M} := (\widetilde{Q}, \Sigma, \Gamma, \widetilde{\delta}, q_0, \widetilde{F})$ by construction satisfies $\mathcal{L}(\widetilde{M}) = \mathcal{L}_F(M)$. □

Corollary. *We cannot define more languages through the final language of a PDA than we already could with PDA's.*

Remark 8.3. In the above proofs we pick arbitrary sets disjoint of Q for example, like the q_s, q_f. In order to avoid this, we can use the disjoint union instead, if desired.

Remark 8.4. Pretty much the same proofs and constructions (in spirit) we can do for $\mathcal{L}_E$, but it bores me.

Definition 8.7. An **extended pushdown automaton** is a PDA with instead a transition relation

$$\delta \subseteq Q \times \Sigma^\lambda \times \Gamma^\lambda \times \Gamma^* \times Q$$

The appropriate languages associated with such an automaton are the same as before, where we define the inference relation of the configuration: if $(q, a, A, W, \widetilde{q}) \in \delta$, then

$$(q, aw, A\alpha) \vdash (\widetilde{q}, w, W\alpha)$$

Note that $A \in \Gamma$ whereas $W \in \Gamma^*$. Perhaps we use W^R if that is intuitively more clear.

Theorem 8.1. *The class of extended PDA's and regular ones are equivalent, that is, for every extended PDA there exists an equivalent regular PDA. The reverse is of course trivial through the embedding $\Gamma \to \Gamma^*$ where $w \mapsto w$.*

8.1 Relation to CFGs

Definition 8.8. Let $G = (V, \Sigma, P, S)$ be a CFG. G is said to be in **Greibach normal form** if for all $(A \to w) \in P$ it holds that either

- $A = S$ and $w = \lambda$, or
- there exists $a \in \Sigma^\lambda$ and $B_1, \ldots, B_n \in (V \cup \Sigma)^* \setminus \{S\}$ such that

$$w = aB_1 \cdot \ldots \cdot B_n$$

Theorem 8.2. *For every CFG G in GNF (Greibach normal form) there exists an extended PDA M such that $\mathcal{L}(M) = \mathcal{L}(G)$ and the PDA M has two states.*

Proof. Let $G = (V, \Sigma, P, S)$ be in GNF. Let q_0, q_1 be any sets. We define

$$M := (\{q_0, q_1\}, \Sigma, V \setminus \{S\}, \delta, \{q_1\})$$

where δ is given by:

- If $(S \to \lambda) \in P$, then $(q_0, \lambda, \lambda, \lambda, q_1) \in \delta$,
- Then any other rule is of the form $(A \to aW) \in P$ with $a \in \Sigma^\lambda$ and $W \in V^*$. Define

$$B := \begin{cases} A & \text{if } A \neq S, \\ \lambda & \text{otherwise.} \end{cases}, \qquad q := \begin{cases} q_0 & \text{if } A = S \\ q_1 & \text{otherwise} \end{cases}$$

  Then $(q, a, B, W, q_1) \in \delta$

Through this construction, $\mathcal{L}(M) = \mathcal{L}(G)$. □

Theorem 8.3. *For every CFG G there exists a CFG $\widetilde{G}$ in GNF such that $\mathcal{L}(G) = \mathcal{L}(\widetilde{G})$.*

Theorem 8.4. *For every PDA M there exists a CFG G such that $\mathcal{L}(G) = \mathcal{L}(M)$.*

Remark 8.5. The result of all of this: the class of PDA's and CFG's are also the same.

8.2 Pumping lemma for CFLs

Lemma 8.4. *Let a grammar G be in CNF. Let $w \in \mathcal{L}(G)$. If a parse tree for w has a longest path of length $n \in \mathbb{N}^*$, then $|w| \leq 2^{n-1}$.*

Lemma 8.5. *Let L be a context-free language. Then there exists a $k > 0$ such that for all words $z \in L$ with $|w| \geq k$, we can write $z = uvwxy$ for some $u, v, w, x, y \in \Sigma^*$ with $|vwx| \leq k$, $vx \neq \lambda$, and for all $i \in \mathbb{N}$, $uv^iwx^iy \in L$.*

8.3 Properties of CFLs

Theorem 8.5. *The class of context-free languages is closed under union, concatenation and Kleene Star. It is, however, not closed under complement or intersection.*

Proof. Let L, M be context-free languages, let $G_L = (V_L, \Sigma_L, P_L, S_L)$ and $G_M = (V_M, \Sigma_M, P_M, S_M)$ be CFGs that generate L and M, respectively. Then

- The union $L \cup M$ is generated by taking any set $S \notin V_L \cup V_M$ and defining

$$\widetilde{G} := (V_L \cup V_M \cup \{S\}, \Sigma_L \cup \Sigma_M, \widetilde{P}, S)$$

  where

$$\widetilde{P} := P_L \cup P_S \cup \{S \to S_L, S \to S_M\}$$

  Then $\mathcal{L}(\widetilde{G}) = L \cup M$.

- The concatenation LM is generated by defining

$$\widetilde{G} := (V_L \cup V_M, \Sigma_L \cup \Sigma_M, \widetilde{P}, S_L)$$

where

$$\widetilde{P} := \{(A \to w) \in P_L \mid A \neq S_L\} \cup P_S \cup \{(S_L \to wS_M) \mid (S_L \to w) \in P\}$$

Then $\mathcal{L}(\widetilde{G}) = LM$.

- Take any set $S \notin V_L$ and define

$$\widetilde{G} := (V_L \cup \{S\}, \Sigma_L, \widetilde{P}, S)$$

where

$$\widetilde{P} := P_L \cup \{S \to S_L S, S \to \lambda\}$$

Then $\mathcal{L}(\widetilde{G}) = L^*$.

The intersection property can be proven by counterexample. Then, of course, it would be a contradiction if the class were closed under union and complement but not intersection. □

Theorem 8.6. *The intersection of a regular and a context-free language is context-free.*

Proof. Let $M = (Q_M, \Sigma_M, \delta_M, q_{M,0}, F_M)$ be a DFA that generates a regular language, and let $N = (Q_N, \Sigma_N, \Gamma_N, \delta_N, q_{N,0}, F_N)$ be a PDA that generates a context-free language. Define the PDA

$$M \times N := (Q_M \times Q_N, \Sigma_M \sqcup \Sigma_N, \Gamma_N, \delta, (q_{M,0}, q_{N,0}), F_M \times F_N)$$

where δ is defined to be the smallest set that satisfies

- If $(q_N, a, B, C, \widetilde{Q}_N) \in \delta_N$ and $q_M \in Q_M$ where $a \neq \lambda$, then define $\widetilde{q}_M := \delta_M(q_M, a)$ and

$$((q_M, q_N), a, B, C, (\widetilde{Q}_M, \widetilde{Q}_N)) \in \delta$$

- If $(q_N, \lambda, B, C, \widetilde{Q}_N) \in \delta_N$ and $q_M \in Q_M$, then

$$((q_M, q_N), \lambda, B, C, (\widetilde{Q}_M, Q_N)) \in \delta$$

By construction, $\mathcal{L}(M \times N) = \mathcal{L}(M) \cap \mathcal{L}(N)$. □

9 Turing machines

Definition 9.1. Let

- Q be any finite set of states,
- Γ be any alphabet called the **tape alphabet**,
- $B \in \Gamma$, called the **blank element** on the tape,
- $\Sigma \subseteq \Gamma \setminus \{B\}$, called the **input alphabet**,
- $\delta : Q \times \Gamma \rightharpoonup \Gamma \times \{L, R\} \times Q$, the **transition relation**, where $L := -1$ and $R := 1$,
- $q_0 \in Q$ the **start state**,
- $F \subseteq Q$ a set of **final states**.

Then we call the tuple $M := (Q, \Sigma, \Gamma, \delta, q_0, F)$ a **Turing machine**.
A **configuration** of a Turing machine M is a tuple $(q, S, i, w) \in Q \times \Gamma^{\mathbb{Z}} \times \mathbb{Z}$, where

- q is the **current state**,
- $S : \mathbb{Z} \to \Gamma$ is the **current tape**, with the property that there exists $N \in \mathbb{N}$ such that for all $|j| \geq N$, $S(j) = B$,
- i is the **current index** in the tape.

The relation $\vdash$ on these configurations is defined by: if $\delta(q, S(i)) = (\gamma, M, \widetilde{q})$, then

$$(q, S, i) \vdash (\widetilde{q}, \widetilde{S}, i + M)$$

where $\widetilde{S} : \mathbb{Z} \to \Gamma$ is defined by

$$\widetilde{S}(j) := \begin{cases} S(j) & \text{if } j \neq i \\ \gamma & \text{otherwise.} \end{cases}$$

We denote by $\vdash^*$ the reflexive-transitive closure of $\vdash$. We say that a Turing machine M **accepts a word** $w \in \Sigma^*$ if there exists an inference $(q_0, S_0, 1) \vdash^* (q, \widetilde{S}, \widetilde{i})$ for some $q \in F$, $\widetilde{S}$ any tape, $\widetilde{i} \in \mathbb{Z}$, where $S_0 : \mathbb{Z} \to \Sigma$ is defined by writing $w = a_1 \cdot \ldots \cdot a_n$ for some $n \in \mathbb{N}$ and $a_i \in \Sigma$ for all $i = 1, \ldots, n$, and

$$S_0(j) := \begin{cases} a_j & \text{if } 1 \leq j \leq n, \\ B & \text{otherwise.} \end{cases}$$

such that also, $\delta(q, \widetilde{S}(\widetilde{i}))$ is not defined (we say the Turing machine **halts**).
The **language** of M is defined by

$$\mathcal{L}(M) := \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$$

Definition 9.2. Let M be a Turing machine. We say that M decides the language L if $\mathcal{L}(M) = L$ and for all words $w \in \Sigma^*$, there exists a finite sequence of inference relations like in the above, $(q_0, S_0, 1) \vdash^* (q, \widetilde{S}, \widetilde{i})$, where $q \in F$ may or may not be true. In other words, the Turing machine **halts for all inputs** (in the given alphabet).

Definition 9.3. Let L be any language over an alphabet Σ. We say that L is

- **recursively enumerable** if there exists a TM M such that $\mathcal{L}(M) = L$,
- **recursive** if there exists a TM M such that $\mathcal{L}(M) = L$ and M decides L.

Theorem 9.1. *The class of recursive languages is bigger than the class of context-free languages, that is, all context-free languages can be defined by a TM but the reverse is false.*

Proof. Note that every PDA has an embedding into the class of TM. But there exists a Turing machine M with

$$\mathcal{L}(M) = \{a^i b^i c^i \mid i \in \mathbb{N}\}$$

which is **not** context-free. □

Remark 9.1. Many variations of TM exist. All are equivalent. Very boring, but cool to see that practically every extension / stronger model of computation is actually the same. The following variations do nothing

- More tapes
- Multiple tracks per tape
- Non-determinism
- Two-way tapes

9.1 Katharsis

Definition 9.4. An **(unrestricted) grammar** is a tuple (V, Σ, P, S) where

- V is a finite set of variables,
- Σ is an alphabet, such that $V \cap \Sigma = \varnothing$,
- P is a binary relation from $(V \cup \Sigma)^+$ to $(V \cup \Sigma)^*$, the set of production rules,
- $S \in V$ the **start symbol**.

The grammar is called **context-free** if every $(A \to w) \in P$ has $A \in V$ (this restores the original CFG definition).

Definition 9.5. Let $G = (V, \Sigma, P, S)$ be a grammar. Let $A \in (V \cup \Sigma)^+$ and $w \in (V \cup \Sigma)^*$ such that $(A \to w) \in P$. If $u, v \in (\Sigma \cup V)^*$, we say that the uwv is **derived from** the word uAv. We sometimes write

$$uAv \implies uwv$$

Like before, we define the relation $\overset{*}{\implies}$ by chains of derivations with $\implies$. The **language** of G like before is

$$\mathcal{L}(G) := \{w \in \Sigma^* \mid S \overset{*}{\implies} w\}$$

Theorem 9.2. *Let L be any language. L is recursively enumerable if and only if there exists an unrestricted grammar G such that $\mathcal{L}(G) = L$.*

Remark 9.2. Turing machines can

- Simulate all grammars, including context-free, including regular
- Simulate nondeterministic Turing machines

And they are equivalent to unrestricted grammars as well. This gives rise to the Chomsky hierarchy

Type	Languages	Grammars	Machines
0	Recur. enum.	Unrestricted	TM
(1)	(context-sensitive)		(LBA)
2	CFL	CFG	PDA
3	Regular	Regular	(Regexp) DFA, NFA(-λ)

Horizontally, entries are equivalent. Vertically down, the top row includes the bottom row as class of languages.

9.2 Universality

Let M be a Turing machine. Suppose we can embed M into binary, that is, we have $R(M) \in \{0,1\}^*$. Can we make a Turing machine U that can simulate M? This should have the property that for all TM M and words w in the alphabet of M, we have

- If input w terminates in M, then $R(M)w$ must terminate in U,
- If input w does not terminate in M, then $R(M)w$ must not terminate in U

Note that this is already impossible because the alphabet of U is then the set of all sets. Shucks. We can either

- Make an embedding of words w into binary, or
- suppose that words $w \in \{0,1\}^*$ are already binary.

Is this possible? Yes, this is a personal computer. The hypothesis is that any algorithm can be encoded in a Turing machine.

Definition 9.6. Let $A \subseteq B$. A **decision problem** is the problem of finding an algorithm that can decide whether for any $b \in B$, also if $b \in A$. A Turing Machine M **solves the decision problem** if

- there exists an encoding $R : B \to \{0,1\}^*$,
- the tape alphabet of M includes $\{0,1\}^*$,
- M always halts for any input,
- for all $b \in B$, let $q \in Q$ be the final state of M with input $R(b)$, then $q \in F \iff b \in A$.

In other words, M always 'makes up its mind' on whether $b \in A$ by ending in an accepting state, or not. We say that a decision problem is **decidable** if there exists a Turing machine that decides it.

Remark 9.3. Note that from this definition it immediately follows that the language of M is $R(A)$, which then must be recursive.