This is a collection of definitions and simple proofs that were taught in the MFCS Module Part II: Formal Languages and Automata. I initially produced it for my own revision needs. This is by no means complete, may contain errors, and you should revise through all materials given in the module. **I could not be held responsible for any deviations of this collection from the original course content.** However, if you spot any problems in this collection, please do contact me through the email address on the top-right corner, so I can correct them for future readers.

This collection is based extensively on (public) lecture and practical materials produced by module leaders of MFCS Part II, most recently by Dr Steve King and Dr Mark Bartlett. You should consult copyright holders and the department before using it for any purposes other than personal academic revisions. Source: 2014-2015 materials from http://www-module.cs.york.ac.uk/mfcs/fla

DFA

A deterministic finite state automaton (DFA) is specified by 5-tuple, $M= <Q,\Sigma,\delta,i,F>$ where:

- Q, a finite set of states
- Σ, a finite set of possible input symbols, the alphabet
- δ, a transition function $Q\times\Sigma\rightarrow Q$
- i, an initial state ($i\in Q$)
- F, a set of final states ($F\subseteq Q$)

Definition: A non-empty string S is accepted by a DFA $M=<Q,\Sigma,\delta,i,F>$ iff $\delta(i, S) \in F$.
The empty string λ is accepted by M iff $i \in F$.

Definition: A language L is accepted by a **FSA** M
iff M accepts every string S in L and **nothing** else.

NDFA

A non-deterministic finite state automaton (NDFA) is specified by a tuple, $M = <Q,\Sigma,\rho,i,F>$ where:

- Q, a finite set of states
- Σ, a finite set of possible input symbols, the alphabet
- ρ, a transition **relation** $\subseteq 2^{Q \times (\Sigma\cup\{\lambda\}) \times Q}$
- i, an initial state ($i \in Q$)
- F, a set of final states ($F \subseteq Q$)

Definition: A string S is accepted by a NDFA $M = <Q,\Sigma,\rho,i,F>$ iff there exists a state q such that $(i,S,q) \in \rho$ and $q \in F$.

Definition: A language L is regular if there exists a DFA (or equivalently an NDFA) that accepts L.
Notation: If M is a DFA then we use L(M) to denote the language accepted by M.

Pumping Lemma

Let L be a regular language. Then there exists some positive integer n such that for every w in L with length at least n ($|w| \geq n$), we can decompose w into 3 strings, w=xyz, such that:
1. $y \mathrel{!}= \lambda$,
2. $|xy| \leq n$, and
3. for every $k \geq 0$, the string xy^kz is also in L.

Pumping Lemma Example

L_{eq} = {w∈ {0,1}* | w has an equal number of 0s and 1s}
Claim: L_{eq} is not regular.

Proof by contradiction:
Suppose that L_{eq} is regular. Let n be the positive integer that exists according to the Pumping Lemma. We pick a string w in L_{eq} , namely $w = 0^n1^n$.
By the Pumping Lemma, there are strings x,y,z such that $0^n1^n = xyz$, $y \mathrel{!}= \lambda$ and $|xy| \leq n$. Hence x and y consist of 0s only.

It follows that xy^2z contains more 0s than 1s (note that y is non-empty). Thus $xy^2z \notin L_{eq}$, contradicting the Pumping Lemma for the case k = 2. Hence our assumption that L_{eq} is regular must be wrong.

Regular Grammar

A grammar is specified by a quadruple G = <V,T,S,P> where:

- **T**, a finite set of terminal symbols, which make up the strings generated by the grammar.
- **V**, a finite set of nonterminal symbols or variables **disjoint** from T.
- **S** in V, a start symbol.
- **P**, a finite set of productions/rules of the form $\alpha \rightarrow \beta$, where α and β are in $(V \cup T)^*$and α contains at least one nonterminal.

For a context-free grammar, α should consist of a single nonterminal.

Definition: A grammar G generates a string α if: $S^* \Rightarrow \alpha$ where S is the start symbol of G.

Definition: A grammar G generates a language L(G) if L(G) is the set of all terminal strings that can be generated from G i.e.$L(G) = \{ \alpha \in T^* \mid S^* \Rightarrow \alpha\}$.

Definition: for a given CFG, a semiword is a string of terminals (maybe none) concatenated with exactly one nonterminal on the right:
(terminal)...(terminal)(nonterminal).

Theorem: Given any finite automaton there is a CFG that generates exactly the language accepted by the FSA. **In other words, all regular languages are context free languages**.

Theorem: If all the productions of a CFG are either of the form N → semiword or N → word (where the word may be λ), then the language generated by the CFG is regular.

Regular Grammar

A grammar G=<V,T,S,P> is a regular grammar if all rules in P are of the form:
$A \rightarrow xB$, $A \rightarrow x$
where A, B are nonterminals and $x \in T^*$ i.e. x is either λ or a sequence of terminal symbols.
Thus, all regular languages can be generated by regular grammars, and all regular grammars generate regular languages.

s-grammar

A context-free grammar G = (V,T,S,P) is said to be a simple or s-grammar if all its productions are of the form **A → aX**, where $A \in V$, $a \in T$ and $X \in V^*$, and any pair (A,a) occurs at most once in P.
For example, S → aS | bSS | aSS | c is not an s-grammar.

Remove Ambiguity

To remove the ambiguity, we use:

- a factor (F) is an expression that cannot be broken apart by any adjacent operators. Thus identifiers or bracketed expressions are the only possibilities.
- a term (T) cannot be broken by a **+** operator. A term therefore is a product of two or more factors.
- an expression (E) is any possible expression and can include adjacent **+** symbols.

Unambiguous Expressions

E → T
  | E **+** T
T → F
  | T* F
F → *I*
  | (E)
I → a | b | c

Inherently Ambiguous Languages

Context-free languages for which no unambiguous context-free grammar can exist.

Non-deterministic Pushdown Stack Automaton

A non-deterministic pushdown stack automaton is defined by 7 components: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

- Here Q, Σ, q_0 and F are defined as before: there are states, input symbols, an initial state and accept states.
- δ is a transition function, but defined differently from how it is defined in a FSA.
- Γ is a finite set of symbols called the *stack alphabet*.
- $z_0 \in \Gamma$ is the *start stack symbol*. The NPDA's stack consists of one instance of this symbol at the outset.

Transition Function For NPDA

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow 2^{(Q \times \Gamma^*)}$$

Suppose

- $\delta(q_0, a, 0) = \{(q_1, 10), (q_3, \lambda)\}$

This gives two transition arrows in the diagram:

- $q_0 \xrightarrow{a,0/10} q_1$
- $q_0 \xrightarrow{a,0/\lambda} q_3$

NPDA ID

(q, aw, xβ) ⊢ (p, w, αβ)
By consuming input a and replacing x by α on (the top of) the stack, we move from state q to state p.

NPDA, CFG Equivalence

Languages Accepted by Final State in NPDA = Languages Accepted by empty stack in NPDA.
Context Free Languages is a subset of Languages Accepted by Final State in NPDA.
Languages Accepted by empty stack in NPDA is a subset of Context Free Languages.

NPDA, CFG Conversion Constructions

CFG -> NPDA by empty stack:

Let G=<V,T,S,R> be a CFG. Construct the NPDA P accepting by empty stack, with N(P) = L(G), as follows: P=<{q},T,V∪T,δ,q,S>.

Because P is accepting by empty stack, there are no final states specified. The transition function δ is defined by:

1. For each non-terminal symbol A, δ(q,λ,A) = { (q,β) | A→β is a production rule in R }
2. For each terminal symbola,δ(q,a,a) = {(q,λ)}

NPDA by empty stack -> NPDA by final state:

More formally: $P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{x_0\}, \delta_F, p_0, x_0, \{p_f\})$,

where δ_F is defined by:

1. $\delta_F(p_0, \lambda, x_0) = \{(q_0, z_0 x_0)\}$. P_F makes a spontaneous transition to the start state of P_N, pushing its start symbol z_0 onto the stack.
2. $\forall q \in Q, a \in \Sigma \cup \{\lambda\}, y \in \Gamma, \delta_F(q, a, y)$ contains at least all the pairs in $\delta_N(q, a, y)$.
3. In addition, $(p_f, \lambda) \in \delta_F(q, \lambda, x_0), \forall q \in Q$.
4. No other pairs are found in $\delta_F(q, a, y)$.

NPDA by final state -> NPDA empty stack:

The construction is: $P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{x_0\}, \delta_N, p_0, x_0)$.

δ_N is defined by:

1. $\delta_N(p_0, \lambda, x_0) = \{(q_0, z_0 x_0)\}$. Start symbol of P_F is pushed onto the stack at the outset and then go to start state of P_F.
2. $\forall q \in Q, a \in \Sigma \cup \{\lambda\}, y \in \Gamma, \delta_N(q, a, y) \supseteq \delta_F(q, a, y)$. P_N simulates P_F.
3. $(p, \lambda) \in \delta_N(q, \lambda, y), \forall q \in F, y \in \Gamma \cup \{x_0\}$. Whenever P_F accepts, P_N can start emptying its stack without consuming any more input.
4. $\forall y \in \Gamma \cup \{x_0\}, \delta_N(p, \lambda, y) = \{(p, \lambda)\}$. Once in p when P_F has accepted, P_N pops every symbol on its stack, until the stack is empty. No further input is consumed.

NPDA -> CFG:

Let $P = \langle Q, \Sigma, \Gamma, \delta, q_0, z_0 \rangle$ be an NPDA accepting by empty stack. Then there is a CFG G such that $L(G) = N(P)$.

Construct $G = \langle V, \Sigma, S, R \rangle$ as follows:

- The non-terminals V include the start symbol S, together with non-terminals of the form $[pXq]$ where $p, q \in Q$ and $X \in \Gamma$.
- The terminal symbols are Σ.
- We define the productions R in turn for the different cases.
  - Starting Productions
  - Popping Transitions
  - Pushing Transitions

Deterministic Pushdown Automata

We define a DPDA P in the same way as we would an NPDA, but with a restriction on the transition function δ.

$$P = \langle Q, \Sigma, \Gamma, \delta, q_0, z_0, F \rangle$$

$$\text{with } \delta\colon Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to 2^{(Q \times \Gamma^*)}$$

P is deterministic if and only if the following conditions are met:

1. $\delta(q, a, X)$ has at most one member for any $q \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $X \in \Gamma$; and

2. If $\delta(q, a, X)$ is nonempty for some $a \in \Sigma$, then $\delta(q, \lambda, X)$ must be empty.