Formal Languages Final Study guide

Liam Dillingham

May 6, 2019

1 Definitions

1.1 Strings

- Σ: alphabet. An alphabet is a *finite* set of symbols (not including ϵ)
- Σ^k: strings from the alphabet Σ of length k
- Σ^*: The set of all strings over an alphabet (including Σ^0 i.e. ϵ).
- Σ^+: set of non-empty strings
- A language L is a set of strings from the alphabet Σ^* such that $L \subseteq \Sigma^*$

1.2 Finite Automata

1.2.1 Deterministic Finite Automata

A *DFA*, labeled as A, is defined as $A = (Q, \Sigma, \delta, q_0, F)$, such that:

1. Q: a finite set of *states*
2. Σ: a finite set of *input symbols*
3. $\delta(q, a)$: a transition function with arguments as q: the current state, and a: the current input symbol, where $\delta : Q \times \Sigma \to Q$
4. q_0, or the starting state in Q
5. F: The set of final or accepting states such that $F \subseteq Q$.

Extended Transition Function $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$
The *extended transition function* precisely describes what happens when we start in any state and follow any sequence of inputs i.e. defines δ for whole words instead of symbols

Language of DFA if A is a DFA, then $L(A) = \{w \mid w \in \Sigma^* \text{ and } \hat{\delta}(q_0, w) \in F\}$

1.2.2 Nondeterministic Finite Automata

The only difference between a *DFA* and *NFA* is that for an *NFA*, δ maps to a set of states. that is, $\delta : Q \times \Sigma \to 2^Q$ i.e. $\mathcal{P}(Q)$

Extended Transition Function

$$\text{basis: } \hat{\delta}(q,\epsilon) = q. \text{ induction: } \hat{\delta}(q,w) = \hat{\delta}(q,xa) = \bigcup_{p \in \hat{\delta}(q,x)} \delta(p,a)$$

1.2.3 ϵ-Nondeterministic Finite Automata

For ϵ-*NFA*, we explicitly define transitions for ϵ, i.e. $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$

Extended Transition Function

- **ECLOSE**(q): All the states that q can reach using only ϵ
- **ECLOSE**(S): $\bigcup_{r \in S}$ **ECLOSE**(r), where S is a set of states

For the precise definition, we have:

$$\text{basis: } \hat{\delta}(q,\epsilon) = \textbf{ECLOSE}(q). \quad \hat{\delta}(q,w) = \hat{\delta}(q,xa) = \textbf{ECLOSE}\left(\bigcup_{p \in \hat{\delta}(q,x)} \delta(p,a)\right)$$

The language described by an ϵ-*NFA*, A, is defined as: $A = \{w \mid \hat{\delta}(q_0,w) \cap F \neq \emptyset\}$.

1.2.4 Equivalence of States

We say two states p, q, are *equivalent*, if, for all input strings w, $\hat{\delta}(p,w)$ is an accepting state if and only if $\hat{\delta}(q,w)$ is an accepting state. That is:
$q \equiv p \Leftrightarrow \forall w \in \Sigma^*, \hat{\delta}(q,w), \hat{\delta}(p,w) \in F$ or $\hat{\delta}(p,w) \notin F$

1.3 Properties of Regular Languages

1.3.1 The Pumping Lemma

The pumping lemma for regular languages Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into three strings, $w = xyz$ such that:

1. $y \neq \epsilon$
2. $|xy| \leq n$
3. For all $k \geq 0$, the string xy^kz is also in L

That is, we can always find a nonempty strings y not too far from the beginning of w that can be "pumped"; that is, repeating y any number of times, or deleting it (case $k = 0$), keeps the string in the language L. If the string stays in the language L, then L is **not regular**.

1.4 Grammars

1.4.1 General Grammar Defintion

$G = (V, T, P, S)$

- V: finite set of *variables* or *nonterminals*
- T: finite set of terminals $V \cap T = \emptyset$
- P: finite set of *productions* or *rules* of the form $\alpha \to \beta$ are strings and what symbols a string may contain differentiates different types of grammars to be defined later
- S: the *start symbol*

1.4.2 Derivations

Relating two strings by a *production* or rule "$\Rightarrow$"

- Let $\alpha \to \beta$ be a rule, we have:
- $x\alpha y \Rightarrow x\beta y$, where $x, y \in (V \cup T)^*$

Left-most Derivations: At each step we replace the leftmost varaible by one of its production bodies

Right-most Derivations: At each step we replace the rightmost varaible by one of its production bodies

Relating two string by a sequence of productions or rules "$\Rightarrow^*$"

- Let $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, ..., w_{k-1} \Rightarrow w_k$
- $w_0 \Rightarrow^* w_k$ where $w_i \in (V \cup T)^*$

1.4.3 Sentential Form

- if $S \Rightarrow^* \alpha, \alpha \in (V \cup T)^*$, then α is called a *sentential form*
- Sentence: if $S \Rightarrow^* \alpha, \alpha \in T^*$, then α is called a *sentence*
- The language generated by grammar: $G = (V, T, P, S)$, where $L = \{w \mid S \Rightarrow^* w, w \in T^*\}$, or the set of all sentences.

1.4.4 Parse Trees

Given a grammar $G = (V, T, P, S)$. The *parse trees* for G are trees with the following conditions:

1. Each interior node is labeled by a variable in V.
2. Each leaf is labeled by either a variable, a terminal, or ϵ. However, if the leaf is labeled ϵ, then it must be the only child of its parent.

1.4.5 Definition: **Context-Free Grammar**

$G = (V, T, P, S)$

- V: finite set of *variables* or *nonterminals*
- T: finite set of terminals $V \cap T = \emptyset$
- P: a finite set of production rules of the form $\alpha \to \beta$, where $\alpha \in V$, and $\beta \in (T \cup V)^*$
- S: the *start symbol*

1.4.6 Definition: **Regular Grammar**

$G = (V, T, P, S)$

- V: finite set of *variables* or *nonterminals*
- T: finite set of terminals $V \cap T = \emptyset$
- P: a finite set of production rules of the form $\alpha \to \beta$, where $\alpha \in V$, and $\beta \in (T \cup TV \cup \{\epsilon\})$
- S: the *start symbol*

1.4.7 Definition: **Context-Sensitive Grammar**

$G = (V, T, P, S)$

- V: finite set of *variables* or *nonterminals*
- T: finite set of terminals $V \cap T = \emptyset$
- P: a finite set of production rules of the form $\alpha \to \beta$, where $\alpha \in (V \cup T)^+$, and $\beta \in (T \cup V)^*$, and $|\alpha| \leq |\beta|$.
- S: the *start symbol*

1.4.8 Definition: **Unrestricted Grammar**

$G = (V, T, P, S)$

- V: finite set of *variables* or *nonterminals*
- T: finite set of terminals $V \cap T = \emptyset$
- P: a finite set of production rules of the form $\alpha \to \beta$, where $\alpha \in (V \cup T)^+$, and $\beta \in (T \cup V)^+$.
- S: the *start symbol*

1.4.9 Ambiguous Grammars

For some CFG's, it is possible to find a terminal string with more than one parse tree, or equivalently, more than one most left-most derivation.

1.5 Pushdown Automata

1.5.1 Non-deterministic PDA

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

- Γ: Finite set of *stack symbols*
- Z_0: initial *Top of stack* (can be removed)
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$ or rather, $Q \times \Gamma^* = \{(q, \gamma) \mid q \in Q, \gamma \in \Gamma^*\}$

Instantaneous Description of PDA: (**current state**, **next input**, **top of stack**) $\vdash$ (**next state**, **next input**, **next top**)

Language Accepted by **Final State**: $L = \{w \mid (q, w, Z_0) \vdash^* (p\epsilon, \gamma)$, where $p \in F$, and $\gamma \in \Gamma^*\}$

Language Accepted by **Empty Stack**: $L = \{w \mid (q_0, wZ_0) \vdash^* (p, \epsilon, \epsilon), p \in Q\}$

1.5.2 Deterministic PDA (DPDA)

1. $\delta(q, a, z)$: contains single Entry $a \in \Sigma \cup \{\epsilon\}, z \in \Gamma$
2. if $delta(q, \epsilon, z)$ is defined, then $\delta(q, a, z)$ is empty, for $(a \in \Sigma, z \in \Gamma)$.

1.6 Context-Free Language Pumping Lemma

Let L be a CFL. Then there exitsts a constant n such that if z is any string in L such that $|z|$ is at least n, then we can write $z = uvwxy$, subject to the following conditions:

1. $|vwx| \leq n$. That is, the middle portion is not too long.
2. $vx \neq \epsilon$. Since v and x are the pieces to be "pumped", this condition says that at least one of the strings we pump must not be empty
3. For all $i \geq 0$, uv^iwx^iy is in L. That is, the two strings v and x may be "pumped" any number of times, including 0, and the resulting string will still be a member of L.

1.7 Chomsky Normal Form for CFG

1.7.1 **Chomsky Normal Form** Algorithm to transform into CNF:

- Eliminate <u>ϵ-productions</u>
- Eliminate <u>unit productions</u>
- Eliminate <u>useless</u> symbols, i.e., Variables which **do not** generate <u>terminals</u> or are **not** reachable from S

1.7.2 Nullable Computation

- $N_0 = \{A \mid A \to \epsilon\}$ (basis)
- $N_1 = \{A \mid A \to \alpha, \alpha \in N_0^*\} \cup N_0$ (induction)
- When $N_k = N_{k+1}.N_k$ is the set of <u>all</u> nullable

1. Eliminate $A \to \epsilon$
2. Introduce new rules for every combination of <u>nullable</u> to $\underline{\epsilon}$
3. repeat (1) and (2) for each rule

1.7.3 Unit Computation

When there is no ϵ-production:

- $unit(A) = \{B \mid A \Rightarrow^* B\}$
- $u_0(A) = \{A\}$
- $u_1(A) = \{B \mid A \Rightarrow B\} \cup u_0(A)$

When $u_k(A) = u_{k+1}(A), u_k(A)$ is the set of variables A can reach via production

1. Eliminate all unit production
2. Promote $B \to w$ to A for each $B \in u_k(A)$

1.7.4 Generating Computation

- $G_0 = \{A \mid A \to w, w \in T^*\}$
- $G_1 = \{A \mid A \to w, w \in (T \cup G_0)^*\} \cup G_0$

When $G_k = G_{k+1}, G_k$ is the set of all variables that is generating. Get rid of variables or rules that involve not generating Variables

1.7.5 Reachable from S Computation

- $S_0 = \{S\}$
- $S_1 = \{A \mid B \to \alpha AB, B \in S_0, \alpha, \beta \in (V \cup T)^*\} \cup S_0$

When $S_k = S_{k+1}, S_k$ is the set of variables reachable from S.

1.8 Church-Turing Thesis

"The unprovable assumption that any general way to compute will allow us to compute only the partial-recursive functions (or equivalently, what Turing machines or modern-day computers can compute) is known as *Church's hypothesis*
The *Church Turing Thesis* shows that if a TM always halts then it is a rec. lang. If it only halts on accept then it is rec. enum. (tentative)

1.9 Turing machines

1.9.1 Definition of Turing machine

- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, where $\delta(q, a) = (P, b, \overset{L}{R})$

1.9.2 Languages of Turing Machines

$L = \{q_0 w \vdash^* \alpha_1 q_f \alpha_2,\ \alpha_1, \alpha_2 \in \Gamma^*,\ q_f \in F\}$

Closure of Languages: regular $\subset$ CFL $\subset$ CSL $\subset$ recursive $\subset$ rec. enum. $\subset$ non-rec. enum

Recursive and Recursively Enumerable Languages A language which accepts on halting

1.10 Diagonalization Language (L_d)

The language L_d the *diagonalization language*, is the set of strings w_i such that w_i is not in $L(M_i)$. That is, $L_d = \{w_i \mid w_i \notin L(M_i) \text{ where } w_i = <M_i>\}$. Note that L_d is not recursively enumerable.

1.11 Universal Language (L_u)

$L_u = \{(M, w) \mid w \in L(M)\}$ Where M is a TM of the binary alphabet which accepts w. U is a *universal* TM such that: $L(U) = \{(M, w) \mid <M>111w$ is accepted by $U\}$, and $L(U) = L_u$.

1.12 L_e and L_{ne}

dont' know yet

1.13 Decidable or Undecidable Problems

A preview of **Undecidable CFL** problems:

1. is a given CFG G ambiguous?
2. is a given CFL inherently ambiguous?
3. is the intersection of two CFL's empty?
4. are two CFL's the same?
5. is a given CFL equal to Σ^*, where Σ is the alphabet of this language?

1.14 The Halting Problem

The halting problem is an issue of decidability

1.15 Halting Language (L_H)

L_H is the set of languages which halt. this is why the halting problem is recursively enumerable, because the language itselfs is TMs which halt on their own encoding. since a rec. enum. language always halts on accept, and the language is the set of TM's who halt on their own encoding, then L_H is rec. enum.