CS81 Notes Spring 2012 CONTEXT-FREE LANGUAGES[1]

For an alphabet Σ, let Σ_ϵ denote $\Sigma \cup \{\epsilon\}$.

1. A **non-deterministic pushdown automata** (PDA) M is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, s, F)$ where

   - Q is a finite set of states,
   - Σ is a finite input alphabet,
   - Γ is a finite stack alphabet,
   - $\delta \subseteq (Q \times \Sigma_\epsilon \times \Gamma^\star) \times (Q \times \Gamma^\star)$ is a transition relation.
   - $s \in Q$ is a start (initial) state,
   - $F \subseteq Q$ is a subset of final (accepting) states.

   A pair $((p, a, \alpha), (q, \beta)) \in \delta$ represents a *transition* of M where M being in state p with α on the top of the stack, may read a from the input tape, replace α with β on the top of the stack, and enter state q.

   A **configuration** of M is a tuple $(q, x, y) \in Q \times \Sigma^\star \times \Gamma^\star$, where q represents the current state of M, x represents the part of the input still to be read, and y represents the current contents of the stack (viewed from top to bottom). For each transition $((p, a, \alpha), (q, \beta)) \in \delta$, and for all $x \in \Sigma^\star$ and $\gamma \in \Gamma^\star$, we define a relation $\vdash_M$ (**yield-in-one-step**) on configurations of M:

   $$(p, ax, \alpha\gamma) \vdash_M (q, x, \beta\gamma).$$

   Let $\vdash_M^\star$ be the reflexive, transitive closure of $\vdash_M$.

   A **string** w **is accepted** by a PDA $M = (Q, \Sigma, \Gamma, \delta, s, F)$ iff $(s, w, \epsilon) \vdash_M^\star (q, \epsilon, \epsilon)$ for some $q \in F$. The **language** L **accepted** by a DFA M is the set of all strings accepted by M. We use the notation $L(M) = \{w : M \text{ accepts } w\}$. A language is called **context-free** iff it is accepted by a PDA.

   Examples

   (a) A PDA accepting $\{a^n b^n : n \geq 0\}$:

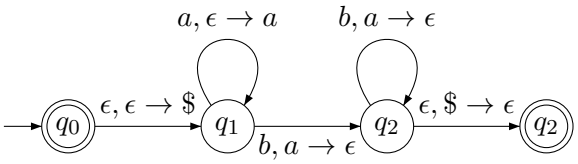
   

   Figure 1: PDA accepting $\{a^n b^n : n \geq 0\}$.

   (b) A PDA accepting $\{w \in \{a, b\}^\star : w \text{ is a palindrome}\}$:

[1] Some material are based on Sipser and Lewis-Papadimitriou. Last updated: April 4, 2012.

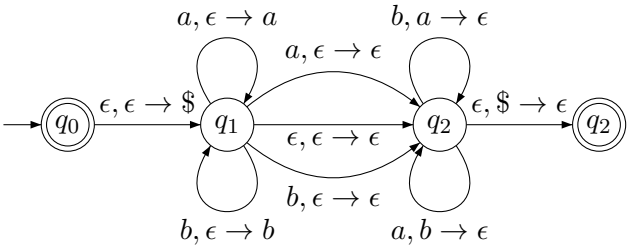


Figure 2: PDA accepting $\{w \in \{a, b\}^\star : w \text{ is a palindrome}\}$.

2. A **context-free grammar** (CFG) G is a 4-tuple $G = (V, \Sigma, R, S)$ where

   - V is a finite set of variables (also called non-terminals),
   - Σ is a finite alphabet (also called terminals),
   - $R \subseteq V \times (V \cup \Sigma)^\star$ is a finite set of rules,
   - $S \in V$ is the start symbol.

   For strings $u, v, w \in (V \cup \Sigma)^\star$ and a variable $A \in V$, we say uAv **yields in one step** uwv, or $uAv \Rightarrow uwv$, if there is a rule $A \to w$. We say $u \overset{*}{\Rightarrow} v$ (**yields in** 0 **or more steps**) if $u = v$ or there is a sequence $u_1, \ldots, u_m \in (V \cup \Sigma)^\star$, where $m \geq 1$, so that $u \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_m \Rightarrow v$. A string w is generated by a grammar G iff $S \overset{*}{\Rightarrow} w$. The language generated by a grammar G is defined as $L(G) = \{w \in \Sigma^\star : S \overset{*}{\Rightarrow} w\}$. A language L is called **context-free** iff $L = L(G)$ for some context-free grammar G.

   Examples

   (a) A grammar generating $\{a^n b^n : n \geq 0\}$:
   $S \to aSb \mid \epsilon$

   (b) A grammar generating $\{w \in \{a, b\}^\star : w \text{ is a palindrome}\}$:
   $S \to aSa \mid bSb \mid a \mid b \mid \epsilon$

3. (**Closure**) Context-free languages are closed under union, concatenation, Kleene star, and an intersection with a regular language. Let $G_i = (V_i, \Sigma, R_i, S_i)$ be a context-free grammar for L_i, where $i = 1, 2, \ldots$. We assume (without loss of generality) that the variable sets V_i are pairwise disjoint.

   (a) *Union*: $L = L_1 \cup L_2$
   Construct a new context-free grammar $G = (V, \Sigma, R, S)$ where
   $V = V_1 \cup V_2 \cup \{S\}$
   $R = R_1 \cup R_2 \cup \{S \to S_1 | S_2\}$
   S is a new variable.

   (b) *Concatenation*: $L = L_1 L_2$
   Construct a new context-free grammar $G = (V, \Sigma, R, S)$ where
   $V = V_1 \cup V_2 \cup \{S\}$
   $R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$
   S is a new variable.

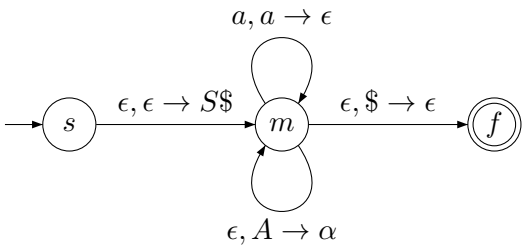
(c) *Kleene star*: $L = L_1^\star$
Construct a new context-free grammar $G = (V, \Sigma, R, S)$ where

$V = V_1 \cup \{S\}$
$R = R_1 \cup \{S \to S_1 S | \epsilon\}$
S is a new variable.

(d) *Regular intersection*: $L = L_1 \cap R$, where R is a regular language
(sketch) Construct a PDA that is a Cartesian product of the PDA for L_1 and the DFA for R.

4. (CFG2PDA) Suppose L is a language generated by a context-free grammar G. Then, there is a PDA M so that $L = L(M)$.

Proof. (sketch) The idea is to construct M that simulates the grammar $G = (V, \Sigma, R, S)$ by guessing a derivation. We create a PDA M with states $Q = \{s, m, f\}$ with $F = \{f\}$ (so f is the only accept state). The only transition from s to m is $(s, \epsilon, \epsilon) = (m, S\$)$ (which prepares the stack by pushing a special bottom-of-stack symbol \$). The only transition from m to f is $(m, \epsilon, \$) = (f, \epsilon)$ (which clears the empty stack). The remaining transitions are from m to itself:



(a) for each symbol $a \in \Sigma$, we let $(m, a, a) \to (m, \epsilon)$. This pops a off the stack if the same symbol appears on the input.

(b) for each grammar rule $A \to \alpha$, we let $(m, \epsilon, A) \to (m, \alpha)$. This replaces the top of stack A with a sequence of symbols derivable from it.

□

5. (PDA2CFG) Suppose L is a language accepted by a PDA M. Then, there is a context-free grammar G so that $L = L(G)$.

Proof. (sketch) Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \{q_1\})$ be a PDA that accepts a context-free language L. To simplify the proof, we assume (without loss of generality): (a) M has a single accept state q_1; (b) M empties the stack before accepting; (c) Each transition either pushes or pops a symbol on the stack but not both.

For any pair of states $p, q \in Q$, we let $A_{(p,q)}$ be a grammar variable that generates strings which cause M starting at state p (on an empty stack) move to state q (on an empty stack). We consider cases based on whether:

(i) $p = q$:

$$A_{(p,p)} \to \epsilon$$

(ii) the stack was emptied, say in state r, prior to reaching state q:

$$A_{(p,q)} \to A_{(p,r)}A_{(r,q)}$$

(iii) the stack was never emptied:

$$A_{(p,q)} \to aA_{(r,s)}b,$$

if $\delta(p,a,\epsilon) = (r,\tau)$ and $\delta(s,b,\tau) = (q,\epsilon)$, for some $\tau \in \Gamma_\epsilon$.

So, we define $G = (V, \Sigma, R, S)$, where $V = \{A_{(p,q)} : p, q \in Q\}$, $S = A_{(q_0,q_1)}$, and

$$\begin{aligned} R \;=\; & \{A_{(q,q)} \to \epsilon : q \in Q\} \cup \\ & \{A_{(p,q)} \to A_{(p,r)}A_{(r,q)} : p,q,r \in Q\} \cup \\ & \{A_{(p,q)} \to aA_{(r,s)}b : p,q,r,s \in Q, a,b \in \Sigma, \delta(p,a,\epsilon) = (r,\tau), \delta(s,b,\tau) = (q,\epsilon), \text{ for some } \tau \in \Gamma_\epsilon\}. \end{aligned}$$

It remains to show that $L(G) = L(M)$. □

6. (**Pumping Lemma**) If L is a context-free language, then there is a constant N so that for any string $w \in L$, with $|w| \geq N$, there are five strings u, v, x, y, z so that $w = uvxyz$ with:

   (a) $|vxy| \leq N$,

   (b) $vy \neq \epsilon$,

   (c) $uv^ixy^iz \in L$, for all $i \geq 0$.

   Proof. Let G be a context-free grammar for L. Suppose that no rule of G generates more than B symbols. So, the maximum length of a string derivable in m steps is at most B^m. Thus, for any string w of length $B^m + 1$, then there must be a path of length at least $m + 1$ steps in the parse tree for w.

   Let $N = B^{|V|} + 1$. So the parse tree of any string $w \in L$ with $|w| \geq N$ admits a path P whose length is at least $|V| + 1$. By the pigeonhole principle, in a leaf to root traversal upwards along P, there is a variable, say R, that is repeated. Let $R^{(2)}$ be the last occurrence of R on P (closest to the leaf level) and let $R^{(1)}$ be the occurrence of R before that. We consider the parse tree or derivation $S \stackrel{*}{\Rightarrow} w$:

$$\begin{aligned} S \;&\stackrel{*}{\Rightarrow}\; uR^{(1)}z \\ &\stackrel{*}{\Rightarrow}\; uvR^{(2)}yz, && \text{where } R^{(1)} \stackrel{*}{\Rightarrow} vR^{(2)}y \\ &\stackrel{*}{\Rightarrow}\; uvxyz, && \text{where } R^{(2)} \stackrel{*}{\Rightarrow} x \end{aligned}$$

   By replacing $R \stackrel{*}{\Rightarrow} x$ with $R \stackrel{*}{\Rightarrow} vRy$ for i times and then followed by $R \stackrel{*}{\Rightarrow} x$, we see that $S \stackrel{*}{\Rightarrow} uv^ixy^iz$. Also, we have $|vxy| \leq N$ since any substring of length at least N is sufficient to obtain a repeated variable in a derivation. Also, $vy \neq \epsilon$ since otherwise it allows $R \stackrel{*}{\Rightarrow} R$. □

Examples

(a) $L = \{a^nb^nc^n : n \geq 0\}$ is not context-free.

Proof. Assume L is context-free. Then there is a constant N so that for any string $w \in L$ with $|w| \geq N$, there are strings u, v, x, y, z so that $w = uvuxyz$ where

i. $|vxy| \leq N$.

ii. $vy \neq \epsilon$.

iii. $uv^ixy^iz \in L$, for each $i \geq 0$.

So, we may choose $w = a^N b^N c^N$. By the Pumping Lemma stated above, we have $w = uvxyz$ and $|vxy| \leq N$ with $vy \neq \epsilon$ and $uv^ixy^iz \in L$, for each $i \geq 0$. There are two cases to consider:

- vxy contains only a's, b's or c's only.
  Since $vy \neq \epsilon$, $uxz \notin L$ due to having fewer a's or b's or c's. But this contradicts the third property of the Pumping Lemma.
- vxy contains a's and b's or b's and c's.
  Then uv^2xy^2z will either contain fewer c's or a's; or it will have a not appearing before b's or b not appearing before c's. In either of these cases, this is a contradiction to the third property of the Pumping Lemma.

Thus, L is not context-free. □

7. (**Transformations**) There are several transformations useful to simplify a grammar. For example, we may use these to bring a grammar into a normal form (see below).

   (a) A variable A is *productive* if $A \stackrel{*}{\Rightarrow} w$ for some $w \in \Sigma^\star$.
   Remove unproductive variable from a grammar G:

   1: mark each variable in G as unproductive
   2: mark each terminal in G as productive
   3: **while** $\exists$ unproductive $A \in V$ and a rule $A \to \alpha$ where each symbol in α is productive **do**
   4: mark A as productive
   5: **end while**
   6: remove all unproductive variables
   7: remove all rules $A \to \alpha B \beta$ if A or B is unproductive

   (b) A variable A is *reachable* if $S \stackrel{*}{\Rightarrow} \alpha A \beta$ for some $\alpha, \beta \in (V \cup \Sigma)^\star$.
   Remove unreachable variables from a grammar G:

   1: mark the start symbol S as reachable
   2: mark all other variable as unreachable
   3: **while** $\exists$ unreachable $A \in V$ and a rule $A \to \alpha B \beta$ where B is reachable **do**
   4: mark A as reachable
   5: **end while**
   6: remove all unreachable variables
   7: remove all rules $A \to \alpha$ with A unreachable

   (c) A variable A is *nullable* if either $A \to \epsilon$ is a rule or $A \to A_1 \dots A_m$, where each A_i is nullable. So, A is nullable if $A \stackrel{*}{\Rightarrow} \epsilon$.
   Find the *nullable* variables:

   1: let $N = \{A : [A \to \epsilon] \in R\}$
   2: **while** $\exists A \notin N$ and a rule $A \to A_1 \dots A_m$ where $A_i \in N$ **do**
   3: $N \leftarrow N \cup \{A\}$
   4: **end while**

   (d) A rule $A \to \alpha B \beta$, where $\alpha, \beta \in (V \cup \Sigma)^\star$ is called *modifiable* if B is nullable.
   Remove ϵ rules from a grammar G:

1: let N be the *nullable*s of G
2: **while** $\exists$ unprocessed modifiable rules **do**
3: **for all** rule $A \to \alpha B \beta$, where $B \in N$ **do**
4: add $A \to \alpha\beta$ provided it has not been added before, $\alpha\beta \neq \epsilon$, and $A \neq \alpha\beta$
5: **end for**
6: **end while**
7: remove all rules of the form $A \to \epsilon$

(e) A rule is called *unit* if it is of the form $A \to B$, where $A, B \in V$.
Remove *unit* rules from a grammar G:

1: **while** there are unit rules in G **do**
2: choose a unit rule $A \to B$ and remove it from G
3: **for all** $B \to \alpha$ **do**
4: add the rule $A \to \alpha$ to G unless this rule had been removed once
5: **end for**
6: **end while**

8. (**Normal Forms**)

(a) A context-free grammar $G = (V, \Sigma, R, S)$ is in **Chomsky Normal Form** if all rules in R are of the form $A \to BC$ or $A \to a$, where $A, B, C \in V$ are variables and $a \in \Sigma$ is an alphabet symbol. The rule $S \to \epsilon$ is allowed only for the start symbol S.

Claim: For any context-free grammar G, there is a context-free grammar G' in Chomsky Normal Form so $L(G) = L(G')$.
Here is a sketch of an algorithm that transforms any grammar into Chomsky Normal Form:

1: G_1 is G with ϵ rules removed
2: G_2 is G_1 with all unit rules removed
3: G_3 is G_2 with all rules whose RHS have length greater than 1 and include a terminal are removed.
4: G_4 is G_3 with all rules whose RHS have length greater than 2 variables removed.

(b) A context-free grammar $G = (V, \Sigma, R, S)$ is in **Greibach Normal Form** if all rules in R are of the form $A \to a\alpha$, where $A \in V$, $a \in \Sigma$, and $\alpha \in (V \cup \Sigma)^\star$. The rule $S \to \epsilon$ is allowed only for the start symbol S.

Let $G = (V, \Sigma, R, S)$ be a grammar. A rule of the form $A \to \alpha$ is called an A-rule.

i. *Step A*: Suppose $A \to \alpha B\beta$ and $B \to \beta_1 | \ldots | \beta_r$ are rules in R.
If we replace $A \to \alpha B\beta$ by $A \to \alpha\beta_i\beta$, for each $i = 1, \ldots, r$, to obtain a new grammar G', then $L(G') = L(G)$.

ii. *Step B*: Suppose $A \to A\alpha_1 | \ldots | A\alpha_r$ and $A \to \beta_1 | \ldots | \beta_s$ are rules in R.
If we replace the above rules with $A \to \beta_i | \beta_i B$, for $1 \leq i \leq s$, and $B \to \alpha_j | \alpha_j B$, for $1 \leq j \leq r$, where B is a new variable, to obtain a new grammar G', then $L(G') = L(G)$.

Claim: For any context-free grammar G, there is another context-free grammar G' in Greibach Normal Form so that $L(G) = L(G')$.

Proof. (sketch) Let $V = \{A_1, \ldots, A_n\}$ be the variables. The first goal is to have all rules be of the form $A_i \to A_j\gamma$ where $j > i$. We proceed inductively on k: assume that $A_i \to A_j\gamma$ only if $j > i$, for $1 \leq i < k$. Next, we work on A_k-rules:

- If $A_k \to A_j\gamma$, $j < k$, is a rule, we apply Step A to obtain a rule of the form $A_k \to A_\ell\gamma$, where $\ell \geq k$.
- If $A_k \to A_\ell\gamma$, $\ell \geq k$, is a rule, we apply Step B to obtain a rule of the form $A_k \to A_\ell\gamma$, where $\ell > k$. This transformation introduces a new variable B_k.

This is repeated until all rules are of the form:

- $A_i \to A_j\gamma$, where $j > i$.
- $A_i \to a\gamma$, where $a \in \Sigma$.
- $B_i \to \gamma$, where $\gamma \in (V \cup \{B_1, \ldots, B_{i-1}\})^\star$.

Now, note all rules A_m-rules must start with a terminal. All A_{m-1}-rules must start with A_m or a terminal. We may apply Step A to replace A_m so that all A_{m-1}-rules start with a terminal. We proceed until all A_i-rules start with a terminal. We need the property that no B_j-rules start with another B_k variable (only with a terminal or a variable A_ℓ). □

9. (**Ogden's Lemma**) Let L be context-free. Then, there is a constant N so that for any string $w \in L$ with at least N marked positions, there are strings u, v, x, y, z so that $w = uvxyz$ where:

   (a) vxy contains at most N marked positions.

   (b) v and y together contain at least one marked position.

   (c) $uv^ixy^iz \in L$, for each $i \geq 0$.

   Proof. (sketch) Let $G = (V, \Sigma, R, S)$ be a context-free grammar in Chomsky Normal Form for L. Suppose $N = 2^{|V|} + 1$. Consider a string $w \in L$ which has at least N marked positions and look at the parse tree T of w. For a node X in T, the marked positions of X is the set of marked positions appearing as leaves of the subtree rooted at X. We construct a root-to-leaf path P on the parse tree for w as follows. If X is in P, then we include in P the child of X which contains at least half of the marked positions of X. We proceed until we reach a leaf node (which is a terminal). The last $|V| + 1$ nodes along this path P must contain a variable which appears at least twice. The proof now proceeds in the same manner as the Pumping Lemma. □

   Examples

   (a) $L = \{a^ib^jc^kd^\ell : i = 0 \text{ or } j = k = \ell\}$ is not context-free.

   (b) $L = \{a^ib^jc^k : i \neq j, j \neq k, \text{ and } i \neq k\}$ is not context-free.

10. (**Parikh's Theorem**) A set $M \in \mathbb{N}^d$ of d-dimensional vectors over $\mathbb{N}$ is **linear** if for some finite sets $B, V \subseteq \mathbb{N}^d$ we have

$$M = \mathcal{L}(B, V) = \{\vec{b} + \sum_{v \in V} a_v\vec{v} : \vec{b} \in B, \vec{v} \in V, a_v \in \mathbb{N}\}.$$

    A set $M \subseteq \mathbb{N}^d$ is called **semilinear** if it is a finite union of linear sets.

    For a string $w \in \Sigma^\star$, let $\psi(w) = (\#_a(w) : a \in \Sigma)$ where $\#_a(w)$ is the number of times a appears in w. Note $\psi(w) \in \mathbb{N}^{|\Sigma|}$. For a language $L \subseteq \Sigma^\star$, let $\Psi[L] = \{\psi(w) : w \in L\}$. Two languages L_1 and L_2 are called *letter-equivalent* if $\Psi[L_1] = \Psi[L_2]$.

(a) If L be a context-free language, $\Psi[L]$ is semilinear.

Proof. For a subset $Q \subseteq V$, let L_Q be the set of strings $w \in L$ for which there is a derivation $S \stackrel{*}{\Rightarrow} w$ where all the variables of Q appear at least once. Let $t = |Q|$. It suffices to show $\Psi[L_Q]$ is semilinear for each Q.

Consider two types of parse trees. A *type I* parse tree satisfies: (1) the root is S; (2) the yield is in $\Sigma^\star$; (3) all variables in Q appear as labels; (4) the height is $\leq t^2$. A *type II* parse tree satisfies: (1) the root is $A \in Q$; (2) the yield is in $\Sigma^\star A \Sigma^\star$; (3) the labels form a subset of Q; (4) the height is $\leq t^2 + 1$. The yield of a parse tree of type II excludes the label A of its root. Let $C = \{w : w \text{ is a yield of some parse tree of type I}\}$ and $D = \{w : w \text{ is a yield of some parse tree of type II}\}$.

Claim: $\Psi[L_Q] = \mathcal{L}(\Psi[C], \Psi[D])$.

($\subseteq$) Let $w \in L_Q$. So, w has a parse tree T of type I but without condition (4). We show that $\psi(w) \in \mathcal{L}(\Psi[C], \Psi[D])$ by induction on the number of nodes n in T.

- *Base case*: $n \leq t^2$.
  Then T is of type I and $w \in C$. This implies $\psi(w) \in \mathcal{L}(\Psi[C], \Psi[D])$.
- *Inductive case*:
  Assume the claim holds for $n \leq K$. Suppose T is a parse tree for w of type I but without condition (4). Without loss of generality, assume the height of T is $\geq t^2 + 1$. So, there is a path P of length $\geq t^2 + 1$ where there is a variable $A \in Q$ which appears $\geq t + 1$ times. Let $T_1, \ldots, T_{t+1}$ be the subtrees along the path P labeled with the variable A. Let A_i denote the set of labels of all nodes in T_i. Then, we have $A_1 \supseteq \ldots \supseteq A_t \supseteq \{A\}$. So, there is an i for which $A_i = A_{i+1}$. Let T' be obtained from T by replacing the subtree T_i by T_{i+1} (both roots are labeled with A). Let T'' be obtained from T_i by replacing its subtree T_{i+1} by a node labeled A (so only its root). Note that the yield w' of T' is in C whereas the yield w'' of T'' is in D. This shows that $\psi(w) = \psi(w') + \psi(w'') \in \mathcal{L}(\Psi[C], \Psi[D])$.

($\supseteq$) Suppose $w \in C$ and $v_1, \ldots, v_k \in D$. Let T be a parse tree of type I with yield w and let T_i be parse trees of type II with yield v_i for $i = 1, \ldots, k$. Let $A_1, \ldots, A_k$ be the root labels of $T_1, \ldots, T_k$, respectively. Since each A_i appears as labels of subtrees T' in T, we may replace such a subtree T' with T_i repeatedly. This shows $\mathcal{L}(\Psi[C], \Psi[D]) \subseteq \Psi[L_Q]$. □

(b) If L has a semilinear $\Psi[L]$, then L is letter-equivalent to a regular language.

Proof. Let L be so that $\Psi[L]$ is semilinear with sets B and V. Suppose Σ is of size d. Let ψ^{-1} be a mapping from $\mathbb{N}^d$ to $\Sigma^\star$ so $\psi^{-1}(\vec{n})$ is the lexicographically smallest string w with $\psi(w) = \vec{n}$. For each $\vec{b} \in B$, let $R(\vec{b})$ be a regular expression for $\psi^{-1}(\vec{b})$. We define $R(\vec{v})$ similarly for each $\vec{v} \in V$. Consider a language L' defined by the regular expression $R = \cup_{\vec{b} \in B} R(\vec{b}) \cup [\cup_{\vec{v} \in V} R(\vec{v})]^\star$. Then, $\Psi[L] = \Psi[L']$. □

(c) Any context-free language over a unary alphabet is regular.

Proof. This is a direct corollary of the previous statement. □

11. (**Decision problems**) We consider several decision problems related to context-free languages.

(a) *Membership*:
Given a context-free grammar $G = (V, \Sigma, R, S)$ (in Chomsky Normal Form) and a string $w \in \Sigma^\star$, decide if $w \in L(G)$.

The idea behind the CYK algorithm is as follows. Let $w_{i,j}$ be the substring of w that starts at position i and has length j. For each variable A, we determine inductively if $A \stackrel{*}{\Rightarrow} w_{i,j}$. The base case is when $j = 1$: $A \stackrel{*}{\Rightarrow} w_{i,j} = w_i$ iff $A \to w_i$ is a rule. For the induction, we note that $A \stackrel{*}{\Rightarrow} w_{i,j}$ iff $A \to BC$ and $B \stackrel{*}{\Rightarrow} w_{i,k}$ and $C \stackrel{*}{\Rightarrow} w_{i+k,j-k}$ where $1 \leq k < j$.

CYK algorithm:

for $i = 1$ to n **do**
  $V_{i,1} = \{A : [A \to a] \in R \text{ and } w_i = a\}$
end for
for $j = 2$ to n **do**
  for $i = 1$ to $n - j + 1$ **do**
    $V_{i,j} = \emptyset$
    for $k = 1$ to $j - 1$ **do**
      $V_{i,j} = V_{i,j} \cup \{A : [A \to BC] \in R, B \in V_{i,k}, C \in V_{i+k,j-k}\}$
    end for
  end for
end for

(b) *Emptiness*:
Given a context-free grammar $G = (V, \Sigma, R, S)$ and a string $w \in \Sigma^\star$, decide if $L(G) = \emptyset$.
The idea is to remove unproductive rules: $L(G) = \emptyset$ iff S is removed.

(c) *Finiteness*:
Given a context-free grammar $G = (V, \Sigma, R, S)$ and a string $w \in \Sigma^\star$, decide if $L(G)$ is finite.