Context-free Grammars

Eklavya Sharma

Contents

1 Context-free grammar

1.1 Definition

Definition 1. *A context-free grammar (CFG) G is a quadruple (V, Σ, R, S) where*

- *V is an alphabet.*
- *$\Sigma \subset V$ is the set of terminals.*
- *$S \in V - \Sigma$ is the start symbol.*
- *$R \in (V - \Sigma) \times V^*$ is the set of rules.*

Definition 2. *$|R|$ is the number of rules in G. $|G|$ is the sum of length of all rules in G.*

Definition 3. $A \to_G u \iff (A, u) \in R$.

Definition 4. $u \Rightarrow_G v$
$\iff (\exists x, y, w \in V^*, \exists A \in V - \Sigma, (u = xAy \wedge v = xwy \wedge A \to_G w))$

Definition 5. $u \overset{L}{\Rightarrow}_G v$
$\iff (\exists x \in \Sigma^*, y, w \in V^*, \exists A \in V - \Sigma, (u = xAy \wedge v = xwy \wedge A \to_G w))$

Definition 6. $u \overset{R}{\Rightarrow}_G v$
$\iff (\exists x, w \in V^*, \exists y \in \Sigma^*, \exists A \in V - \Sigma, (u = xAy \wedge v = xwy \wedge A \to_G w))$

Definition 7 ($\Rightarrow_G^n$)**.**

$u \Rightarrow_G^0 v \iff u = v$

For $n \geq 1, u \Rightarrow_G^n v \iff (\exists w, u \Rightarrow_G w \wedge w \Rightarrow_G^{n-1} v)$

$u \Rightarrow_G^* v \iff (\exists n, u \Rightarrow_G^n v)$

Definition 8. *For $u \in V^*, L_G(u) = \{w \in \Sigma^* : u \Rightarrow_G^* w\}$. $L(G) = L_G(S)$. If $L = L(G)$, G generates L and L is a context-free language (CFL).*

Theorem 1. *For $w \in \Sigma^*$ and $x \in V^*$, $x \Rightarrow_G^* w \iff x \overset{L*}{\Rightarrow}_G w$.*

1.2 Right-linear grammar

Definition 9. *G is a right-linear grammar iff each rule is of the form $P \Rightarrow_G sQ$, where $P \in V - \Sigma, Q \in V - \Sigma \cup \{e\}$ and $s \in \Sigma^*$.*

Theorem 2. *A language is regular $\iff$ it is generated by a right-linear grammar.*

Proof of $\Rightarrow$. Let L be a regular language. Then it is accepted by the NFA $M = (Q, \Sigma, \Delta, s, F)$. Let $G = (Q \cup \Sigma, R, s, F)$, where $F = \{q \to_G e : q \in F\}$ and $R = \{q \to_G ap : (q, a, p) \in \Delta\}$. It can be proved that $L(G) = L(M)$. □

Proof of $\Leftarrow$. Let $G = (V, \Sigma, R, S)$ be a right-linear grammar. Let $M = ((V - \Sigma) \cup \{f\}, \Sigma, \Delta, S, \{f\})$ where

$$\Delta = \{(P, s, Q) : P \to_G sQ \wedge Q \not\to_G e\} \cup \{(P, s, f) : P \to_G sQ \wedge Q \to_G e\}$$

It can be proved that $L(G) = L(M)$. □

1.3 Examples

- $L(S \to e \mid aSb) = \{a^n b^n : n \geq 0\}$
- $L(S \to e \mid SS \mid (S))$ is the language of balanced parenthesis.
- $L(S \to e \mid a \mid b \mid aSa \mid bSb)$ is the language of palindromes.
- $L(S \to e \mid aSb \mid bSa \mid SS)$ is the language of strings with equal number of occurrences of a and b.

To prove that $L(G) \subseteq L'$, use induction on the length of derivations (this is usually easy). Proving that $L' \subseteq L(G)$ can be done by using induction on the length of input (this can be hard).

Lemma 3. *Let $f(w)$ be the number of occurrences of $\mathtt{a}$ minus the number of occurrences of $\mathtt{b}$. If $f(w) = 0$ and the first and last characters of w are same, then $w = xy$, where $x, y \in \{a, b\}^+$ and $f(x) = f(y) = 0$.*

Proof. It is trivial to prove that $f(u + v) = f(u) + f(v)$.
Let $w = czc$ and $|w| = n$, where $c \in \{a, b\}$.
Let $w[: k]$ be the k-length prefix of w.
$f(w[: 0]) = f(e) = 0, f(w[: 1]) = f(c)$.
$f(w) = 0 \Rightarrow f(cz) + f(c) = 0 \Rightarrow f(w[: n-1]) = -f(c)$.
Since $f(w[: i])$ changes sign for $i \in [1, n-1]$ and $f(w[: i])$ and $f(w[: i+1])$ can only differ by 1, $\exists i \in [1, n-1]$ such that $f(w[: i]) = 0$.
Let $w[: i] = x$. Let $w = xy$. Then $f(x) = 0$ and $f(y) = f(w) - f(x) = 0$. □

Theorem 4. *$G = (S \to e \mid aSb \mid bSa \mid SS)$ is the grammar of strings with equal number of occurrences of a and b.*

Proof. We will prove by induction that $L' \subseteq L(G)$

Base step: $e \in L(G)$

Inductive step: Assume that $\forall |x| < n, f(x) = 0 \Rightarrow x \in L(G)$ (induction hypothesis). Let $|w| = n$ and $f(w) = 0$.

Case 1: First and last characters of w are same. If the first and last characters of w are same, $\exists x, y$ such that $w = xy$ and $1 \leq |x|, |y| \leq n-1$ and $f(x) = f(y) = 0$ (by lemma 3). By induction hypothesis, $x, y \in L(G)$. Therefore, $S \Rightarrow_G SS \Rightarrow_G^* xS \Rightarrow_G^* xy = w$. Therefore, $w \in L(G)$.

Case 2: $w = axb$. $w = axb \Rightarrow f(x) = 0 \Rightarrow x \in L(G)$. $S \Rightarrow_G aSb \Rightarrow_G^* axb$. Therefore, $w \in L(G)$.

Case 3: $w = bxa$. This is similar to case 2.

In all 3 cases, $w \in L(G)$. Therefore by mathematical induction, $L' \subseteq L(G)$. □

2 Parse-trees

Definition 10. *A derivation in which we always replace the leftmost non-terminal is a leftmost derivation.*

Theorem 5. *Every derivation has a unique parse tree associated with it. Every parse tree has a unique leftmost derivation.*

Proof. A derivation can be used to build the parse tree step-by-step. The string obtained after k steps in the derivation will match the yield of the parse tree after k steps.

In the beginning, the tree has a single node containing the start symbol.

In the k^{th} step, when a rule application happens, a certain non-terminal in the string gets replaced. Find the corresponding symbol in leaf nodes of the parse tree and expand that node.

Therefore, corresponding to a derivation, we get a persistent-parse-tree (look up the definition of 'persistent data structure'). The last time-slice is the parse tree we want. Since the construction is deterministic, the parse tree is unique.

We can also do the reverse: Given a parse tree, we can create a persistent parse tree out of it. The first time-slice will be a single node containing the start symbol.

To get the k^{th} time-slice from the $(k+1)^{\text{th}}$ time-slice, expand a leaf node containing a non-terminal. The rule to be used for expansion has to be looked up from the parse tree.

This way, we can get a sequence of parse-trees where each one is obtained from the next by expanding a single non-terminal leaf node. The yields of the parse trees will give the derivation of the string.

If we always expand the leftmost non-terminal in the parse-tree slice, the algorithm will become deterministic, so we will obtain a unique derivation. This derivation is a leftmost derivation. □

Definition 11. *If multiple distinct parse trees exist for a string in a grammar, that string is said to be 'ambiguous' for that grammar. A grammar which has an ambiguous string is an ambiguous grammar. A language for which all grammars are ambiguous is 'inherently ambiguous'.*

3 Pushdown Automata

Definition 12. *A pushdown automaton (PDA) M is a tuple $(K, \Sigma, \Gamma, \Delta, s, F)$ where*

- *K is the set of states.*
- *Σ is the input alphabet.*
- *Γ is the stack alphabet.*
- *$s \in K$ is the start state.*
- *$F \subseteq K$ are the final states.*

- $\Delta \subseteq (K \times (\Sigma ?) \times \Gamma^*) \times (K \times \Gamma^*)$ *(finite subset) is the transition function.* $((p, a, \beta), (q, \gamma)) \in \Delta$ *means 'when in state* p*, input symbol is* a *and top of the stack is* β*, go to state* q*, pop* β *and push* γ*'. Top of the stack being* β *means that* β *is a substring of the stack when the stack is read top-to-bottom. Pushing* γ *means first push its last symbol, then its second-last symbol, and so on.*

Definition 13. *A configuration of a PDA* M *is an element* $(q, w, x) \in (K \times \Sigma^* \times \Gamma^*)$ *which means that* M *is in state* q*, input* w *is remaining to be read and the stack content is* x *(left-to-right in* x *means top-to-bottom in stack).*

Definition 14. $(p, x, u) \vdash_M (q, y, v) \iff$ *there is a transition which can take* M *from* (p, x, u) *to* (q, y, v)*.* $\vdash_M^*$ *is the reflexive-transitive closure of* $\vdash_M$*.*

Definition 15. $w \in L(M) \iff (\exists f \in F, (s, w, e) \vdash_M (f, e, e))$*.*

4 PDAs and CFGs

4.1 Incremental left derivation

Definition 16. *Let* $G = (V, \Sigma, R, S)$ *be a grammar.*
For $u_1, u_2 \in \Sigma^*$*,* $A \in V$ *and* $v_1, v_2 \in V^*$*,* $(u_1, Av_1) \overset{L}{\Rightarrow}_G (u_2, v_2)$
$$\iff \begin{cases} u_2 = u_1 A \wedge v_2 = v_1 & \text{if } A \in \Sigma \\ u_2 = u_1 \wedge (\exists \beta \in V^*, A \to_G \beta \wedge v_2 = \beta v_1) & \text{if } A \in V - \Sigma \end{cases}$$

For $v \in V^*$ and $w \in \Sigma^*$, if $(e, v) \overset{L}{\Rightarrow}_G^* (w, e)$, then the sequence of $\overset{L}{\Rightarrow}_G$ operations is called an incremental left derivation of w from v.

Lemma 6. $((u_1, v_1) \overset{L}{\Rightarrow}_G (u_2, v_2)) \implies (u_1 v_1 = u_2 v_2 \vee u_1 v_1 \overset{L}{\Rightarrow}_G u_2 v_2)$

Lemma 7. *For* $x \in \Sigma^*$*,* $A \in V - \Sigma$ *and* $y, \beta \in V^*$*,*
$xAy \overset{L}{\Rightarrow}_G x\beta y \implies (x, Ay) \overset{L}{\Rightarrow}_G (x, \beta y)$*.*

Theorem 8. *For* $w \in \Sigma^*$*,* $(\gamma \Rightarrow_G^* w) \iff ((e, \gamma) \overset{L}{\Rightarrow}_G^* (w, e))$*.*

Therefore, $w \in L(G)$ iff an incremental left derivation of w exists from S, i.e. $(e, S) \overset{L}{\Rightarrow}_G^* (w, e)$.

4.2 Incremental right derivation

Definition 17. *Let* $G = (V, \Sigma, R, S)$ *be a grammar.*
For $u_1, u_2 \in V^*$*,* $A \in V$ *and* $v_1, v_2 \in \Sigma^*$*,* $(u_1 A, v_1) \overset{R}{\Rightarrow}_G (u_2, v_2)$
$$\iff \begin{cases} u_2 = u_1 \wedge v_2 = A v_1 & \text{if } A \in \Sigma \\ v_2 = v_1 \wedge (\exists \beta \in V^*, A \to_G \beta \wedge u_2 = u_1 \beta) & \text{if } A \in V - \Sigma \end{cases}$$

For $v \in V^*$ and $w \in \Sigma^*$, if $(v, e) \overset{R}{\Rightarrow}_G^* (e, w)$, then the sequence of $\overset{R}{\Rightarrow}_G$ operations is called an incremental right derivation of w from v.

Theorem 9. *For* $w \in \Sigma^*$*,* $(\gamma \Rightarrow_G^* w) \iff ((\gamma, e) \overset{R}{\Rightarrow}_G^* (e, w))$*.*

4.3 Standard non-deterministic top-down parser

Definition 18. *Let $G = (V, \Sigma, R, S)$ be a CFG. Let $M = (\{s, f\}, \Sigma, V, \Delta, s, \{f\})$ be a PDA containing the following transitions:*

- $((s, e, e), (f, S))$
- $((f, a, a), (f, e))$ *for all* $a \in \Sigma$.
- $((f, e, A), (f, \beta))$ *for all* $A \to_G \beta \in R$.

Then M is a standard non-deterministic top-down parser for G.

Denote by $(u, v, x)_w$ the configuration of M on input w when it is at state f, has read input u, has input v remaining to be read and contains x on the stack.

Theorem 10. $(u_1, v_1, x_1)_w \vdash_M (u_2, v_2, x_2)_w \iff (u_1, x_1) \overset{L}{\Rightarrow}_G (u_2, x_2)$.

Theorem 11. $L(M) = L(G)$.

Proof.

$$
\begin{aligned}
& w \in L(G) \\
\iff\ & (e, S) \overset{L}{\Rightarrow}{}^*_G (w, e) \\
\iff\ & (e, w, S)_w \vdash^*_M (w, e, e)_w \\
\iff\ & w \in L(M)
\end{aligned}
$$

□

Therefore, every CFG G has an equivalent PDA M, its standard non-deterministic top-down parser. Also, every step of M can be mapped to a step in an incremental left derivation of the input.

4.4 Standard non-deterministic bottom-up parser

Definition 19. *Let $G = (V, \Sigma, R, S)$ be a CFG. Let $M = (\{s, f\}, \Sigma, V, \Delta, s, \{f\})$ be a PDA containing the following transitions:*

- *'Shift' transitions:* $((s, a, e), (s, a))$ *for all* $a \in \Sigma$.
- *'Reduce' transitions:* $((s, e, \beta^R), (s, A))$ *for all* $A \to_G \beta \in R$.
- *'Final' transition:* $((s, e, S), (f, e))$.

Then M is a standard non-deterministic bottom-up parser for G.

Denote by $(u, v, x)_w$ the configuration of M on input w when it is at state s, has read input u, has input v remaining to be read and contains x on the stack.

Theorem 12. $(u_1, v_1, x_1)_w \vdash_M (u_2, v_2, x_2)_w \iff (x_2^R, v_2) \overset{R}{\Rightarrow}_G (x_1^R, v_1)$.

Theorem 13. $L(M) = L(G)$.

Proof.

$$
\begin{aligned}
& w \in L(G) \\
\iff & (S, e) \overset{R}{\Rightarrow}{}^*_G (e, w) \\
\iff & (e, w, e)_w \vdash^*_M (w, e, S)_w \\
\iff & w \in L(M)
\end{aligned}
$$

□

Therefore, every CFG G has an equivalent PDA M, its standard non-deterministic bottom-up parser. Also, every step of M can be mapped to a reverse-step in an incremental right derivation of the input.

4.5 Simple PDA

A PDA M is simple iff all of the following are true:

- There is no transition to the start state. There is a single transition from the start state and that transition does not read the input tape.
- There is a single final state. Transitions to the final state do not read the input tape.
- In every transition, M either pushes a single symbol or pops a single symbol or does nothing to the stack.
- M's stack is never empty except at the start state or in the final state where it should be empty.
- Every transition (except the one from the start state) consults the stack.

This implies that in a simple PDA, the start state always has to push a single symbol to the stack. We call it the 'bottom marker' and denote it by Z. Also, Z is always present in the stack except at the start state or the final state.

A PDA $M = (Q, \Sigma, \Gamma, \Delta, s, F)$ can always be transformed to be simple. This is how:

- Introduce a new stack symbol Z.
- Add a new start state s' and the transition $((s', e, e), (s, Z))$.
- Add a new final state f' and the transition $((f, e, Z), (f', e))$ for all $f \in F$.
- $\forall((p, a, \beta), (q, \gamma)) \in \Delta$, add new states and transitions which first iteratively pop symbols of β and then iteratively push symbols of γ. The stack will never get empty because no pop can remove Z.
- Replace $((p, a, e), (q, b))$ by the transitions $((p, a, c), (q, cb))$ for every $c \in \Sigma$.

4.6 PDA to CFG

Let $M = (Q, \Sigma, \Gamma \cup \{Z\}, \Delta, s, \{f\})$ be a simple PDA.
Let $((s, e, e), (s', Z)) \in \Delta$ and $((f', e, Z), (f, e)) \in \Delta$.

Let $G = (V, \Sigma, R, S)$ where $V = \Sigma \cup \{S\} \cup \{\langle q, A, p\rangle : p, q \in Q \wedge A \in \Gamma \cup \{e, Z\}\}$.

There are 4 kinds of rules in R:

- $S \to \langle s, Z, f\rangle$.
- For $q \in Q$: $\langle q, e, q\rangle \to e$.
- For every $p \in Q$ and every stack pop/no-op $((q, a, B), (r, C))$ $(C \in \{e, B\})$: $\langle q, B, p\rangle \to a\langle r, C, p\rangle$.
- For every $p, p' \in Q$ and every stack push $((q, a, B), (r, CB))$: $\langle q, B, p\rangle \to a\langle r, C, p'\rangle\langle p', B, p\rangle$.

Lemma 14. *(Can be proved using induction)*
$\langle p, A, q\rangle \Rightarrow_G^* x \iff (p, A, x) \vdash_M^* (q, e, e)$

Theorem 15. $L(G) = L(M)$

Proof.

$$
\begin{aligned}
& w \in L(G) \\
& \iff \langle s', Z, f\rangle \Rightarrow_G^* w \\
& \iff (s', w, Z) \vdash_M^* (f, e, e) \\
& \iff w \in L(M)
\end{aligned}
$$

□

5 Closure properties

Let $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ be CFGs. Since renaming non-terminals does not affect a grammar, we can assume that $V_1 - \Sigma_1$ and $V_2 - \Sigma_2$ are disjoint.

$L(G_1) \cup L(G_2) = L((V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \to S_1 | S_2\}, S))$

$L(G_1)L(G_2) = L((V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \to S_1 S_2\}, S))$

$L(G_1)^* = L((V_1 \cup \{S\}, \Sigma_1, R_1 \cup \{S \to e | S_1 S\}, S))$

Theorem 16. *Intersection of a regular language with a CFL is context-free.*

A PDA for the intersection can be obtained by taking the cross product of the states of the DFA of the regular language and the states of the PDA of the CFL.

6 Pumping Theorem

Definition 20. *The fanout of a grammar G, denoted as $\phi(G)$, is the maximum RHS length in its rules.*

Lemma 17. *Let $G = (V, \Sigma, R, S)$ be a CFG. For $w \in L(G)$, $|w| > \phi(G)^h$ implies that the parse tree of w has height greater than h.*

Theorem 18 (Pumping theorem)**.** *Let $G = (V, \Sigma, R, S)$ be a CFG. Then $\forall w \in L(G), |w| > \phi(G)^{|V-\Sigma|} \implies$*
$(\exists u, v, x, y, z \in \Sigma^*, w = uvxyz \wedge vy \neq e \wedge$
$(\forall i \geq 0, uv^ixy^iz \in L(G)))$.

Corollary 18.1. *$L = \{a^nb^nc^n : n \geq 0\}$ is not context-free.*

Proof. Let $w = a^nb^nc^n$ where $n > \phi(G)^{|V-\Sigma|}/3$. Let $w = uvxyz$ such that $vy \neq 0$. If vy has all occurrences of all of $\{a, b, c\}$, then v or y has occurrences of 2 of $\{a, b, c\}$. Therefore, uv^kxy^kz will have characters in the wrong order for $k \geq 2$. If vy does not have a certain character, then uv^kxy^kz will have unequal number of characters for $k \neq 1$. This contradicts the pumping theorem, so L is not context-free. □

Theorem 19 (Corollary of Parikh's Theorem)**.** *Let L be a language over a single character. Then L is context-free iff L is regular.*

Theorem 20. *CFLs are not closed under intersection or complementation.*

Proof. $L_1 = \{a^mb^nc^n : m, n \geq 0\}$ and $L_2 = \{a^mb^mc^n : m, n \geq 0\}$ are context-free. But $L_1 \cap L_2 = \{a^nb^nc^n : n \geq 0\}$ is not context-free. Complementation cannot be closed under CFLs, because $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. □

7 Algorithms for CFGs

7.1 Finding erasable non-terminals

Definition 21. *A non-terminal A is erasable iff $A \Rightarrow_G^* e$.*

To find the set of erasable non-terminals, follow this algorithm:

Algorithm 1 Finding all erasable non-terminals in $G = (V, \Sigma, R, S)$

```
ℰ = {}                                   ▷ Implement as a bitset on V − Σ.
do
    ℰ′ = ℰ
    for A →_G β ∈ R where A ∉ ℰ do
        if β ∈ ℰ* then
            ℰ.add(A)
        end if
    end for
while ℰ′ ≠ ℰ
```

Running time: $O(|V - \Sigma||G|)$. Auxiliary space: $\Theta(|V - \Sigma|)$.

7.2 Finding unit derivability

Definition 22. *$B \in V$ is unit-derivable from A iff $A \Rightarrow_G^* B$.* $\mathrm{D}(A) = \{B \in V : A \Rightarrow_G^* B\}$.

Algorithm to find $D(A)$ for all $A \in V$, if the grammar $G = (V, \Sigma, R, S)$ has no erasable non-terminals:

- Make a directed graph where V is the set of vertices. Make an edge from A to B iff $A \to_G B \in R$.
- Apply depth-first search from every $A \in V$. Add every vertex encountered during depth-first search to $\mathrm{D}(A)$.

Running time: $O(|V - \Sigma| \min(|R|, |V||V - \Sigma|))$.
Auxiliary space: $O(\min(|R|, |V||V - \Sigma|) + |V - \Sigma|)$.
Output size: $O(|V - \Sigma||V|)$.

7.3 Chomsky normal form

Definition 23. *A grammar $G = (V, \Sigma, R, S)$ is in Chomsky normal form iff every rule is of one of these forms:*

- $A \to_G BC$ *where* $A \in V - \Sigma$ *and* $B, C \in V - \{S\}$.
- $A \to_G a$ *where* $A \in V - \Sigma$ *and* $a \in \Sigma$.
- $S \to_G e$

The following algorithm transforms a grammar G into Chomsky normal form grammar G'. The algorithm proceeds in stages, and in each state the language of the grammar remains unchanged:

1. Eliminate S from RHS: Change the start symbol to S' and add the rule $S' \to_G S$.
2. Eliminate long rules, i.e. rules of the form $A \to_G B_1 B_2 \dots B_n$ where $B_i \in V$ and $n \geq 3$.
3. Eliminate $e-$rules, i.e. rules of the form $A \to_G e$.
4. Eliminate unit rules, i.e. rules of the form $A \to_G B$, where $B \in V$.

7.3.1 Eliminating long rules

Replace each rule of the form $A \to_G B_1 B_2 \dots B_n$ by the rules $A \to_G B_1 A_1$, $A_1 \to_G B_2 A_2$, $\dots$, $A_{n-3} \to_G B_{n-2} A_{n-2}$, $A_{n-2} \to_G B_{n-1} B_n$.

- Running time: $\Theta(|G|)$.

- Auxiliary space: $\Theta(1)$.
- Change in grammar size: $|G'| \in \Theta(|G|)$ and $|R'| \in \Theta(|G|)$ and $|V' - \Sigma| \in O(|G|)$.

Since $|R'| = \Theta(|G'|)$, we will use $|R'|$ as a measure of grammar size in subsequent steps.

7.3.2 Eliminating e-rules

1. Find $\mathcal{E}$, the set of erasable non-terminals.
2. Remove all rules of the form $A \Rightarrow_G e$.
3. For every rule $A \rightarrow_G BC$, if $B \in \mathcal{E}$ then add the rule $A \rightarrow_G C$ and if $C \in \mathcal{E}$ then add the rule $A \rightarrow_G B$.
4. If $S \in \mathcal{E}$, add the rule $S \rightarrow_G e$.

- Running time: $O(|V - \Sigma||G|)$.
- Auxiliary space: $\Theta(|V - \Sigma|)$.
- Side-effects on grammar: Some non-terminals can become non-instantiable.
- Change in grammar's size: $|R| \leq |R'| \leq 3|R| - 1$.

7.3.3 Eliminating unit rules

1. Find $\mathrm{D}(A) = \{B \in V : A \Rightarrow_G^* B\}$ for each $A \in V - \Sigma$. This can be done easily since there are no e-rules.
2. Remove all rules of the form $A \rightarrow_G B$.
3. For all $A \in V - \Sigma$, add rule $A \rightarrow_G a$ for $a \in \Sigma \cap \mathrm{D}(A)$ and add rule $A \Rightarrow CD$ for all $B \in \mathrm{D}(A)$ and $B \rightarrow_G CD$.

- Running time: $O(|V - \Sigma|(|\Sigma| + |R|)$.
- Auxiliary space: $O(|V||V - \Sigma|)$.
- Side-effects on grammar: Some non-terminals can become non-instantiable.
- Change in grammar's size: $|R'| \leq |V - \Sigma| \min(|R|, |V|^2)$ because each new rule's LHS is a non-terminal and each new rule's RHS is the RHS of some old rule.

7.3.4 Eliminating unit rules (alternate algorithm)

This algorithm has a worse time-complexity than the previous algorithm, but it is mentioned here for completeness.

1. Find $\mathrm{D}(A) = \{B \in V : A \Rightarrow_G^* B\}$ for each $A \in V - \Sigma$. This can be done easily since there are no e-rules.
2. Remove all rules of the form $A \rightarrow_G B$.
3. For each rule $A \rightarrow_G BC$, add the rule $A \rightarrow_G B_iC_i$ for each $B_i \in \mathrm{D}(B)$ and $C_i \in \mathrm{D}(C)$.
4. For each rule of the form $A \rightarrow_G BC$, if $A \in \mathrm{D}(S)$, add $S \rightarrow_G BC$.
5. Add the rule $S \rightarrow_G a$ for all $a \in \Sigma \cap \mathrm{D}(S)$.

- Running time: $O(|V|(|V - \Sigma|^2 + |R||V|))$.
- Auxiliary space: $O(|V||V - \Sigma|)$.
- Side-effects on grammar: Some non-terminals can become non-instantiable.
- Change in grammar's size: $|R'| \leq |V - \Sigma||V|^2$.

7.3.5 Converting to Chomsky normal form

- Running time: $O(|G|^2)$.
- Auxiliary space: $O(|G|^2)$.
- Change in grammar size: $|V' - \Sigma| \in O(|G|)$, $|R'| \in O(|G|^2)$ and $|G'| \in O(|G|^2)$.

7.4 CYK Algorithm

The CYK algorithm takes (w, G) as input and uses dynamic programming to find out whether $w \in L(G)$. Here $G = (V, \Sigma, R, S)$ is a grammar in Chomsky-normal form.

Let $R_A = \{\beta \in V^* : (A \rightarrow_G \beta) \in R\}$. Therefore, $|R| = \sum_{A \in V - \Sigma} |R_A|$.

Let $f(A, i, j) = (A \Rightarrow_G^* w[i : j])$. Then

$$f(A,i,j) = \begin{cases} \text{false} & i \geq j \\ w[i] \in R_A & j = i+1 \\ \bigvee_{BC \in R_A} \bigvee_{k=i+1}^{j-1} (f(B,i,k) \wedge f(C,k,j)) & j > i+1 \end{cases}$$

$w \in L(G) \iff f(S, 0, |w|)$. The CYK algorithm computes $f(A, i, j)$ for all $A \in V - \Sigma$ and all $0 \leq i < j \leq n$ using dynamic programming.

Time to compute $f(A, i, j)$ is $\Theta(|R_A|(j-i))$ if sub-problem solutions are already available. Time to compute f for all (A, i, j) is therefore $\Theta(|R||w|^3)$.

7.5 Removing non-productive symbols

Definition 24. *For the CFG $G = (V, \Sigma, R, S)$, $A \in V$ is productive iff $\exists w \in \Sigma^*, A \Rightarrow_G^* w$.*

Algorithm 2 Finding all productive non-terminals in $G = (V, \Sigma, R, S)$

```
P = {}                                   ▷ Implement as a bitset on V − Σ.
do
    P' = P
    for A →_G β ∈ R where A ∉ P do
        if β ∈ (P ∪ Σ)* then
            P.add(A)
        end if
    end for
while P' ≠ P
```

Running time: $O(|V - \Sigma||G|)$. Auxiliary space: $\Theta(|V - \Sigma|)$.

To remove non-productive symbols from G, remove all rules which have a non-productive symbol on the LHS or in the RHS. This will not change the language of G.

8 Deterministic PDA

Definition 25. *Two strings are* consistent *iff one is a prefix of the other.*

Definition 26. *For a PDA, two transitions $((p, a_1, \beta_1), (q_1, \gamma_1))$ and $((p, a_2, \beta_2), (q_2, \gamma_2))$ are* compatible *iff a_1 and a_2 are consistent and β_1 and β_2 are consistent.*

Definition 27. *A PDA is* deterministic *iff no two of its transitions are compatible.*

Definition 28. *A language L is deterministic context-free iff there is a DPDA M such that $L\$ = L(M)$. Here $\$$ is a symbol which is not in the alphabet of L.*

Theorem 21. *For every DCFL L, there is a DPDA with these properties:*

- *It halts on an input iff it reads the whole input.*
- *It has an empty stack and is not in the starting state iff it has halted.*

From now on we can (and we will) assume (WLOG) that all DPDAs have the above properties.

8.1 Closure under complementation

Theorem 22. *Let $\overline{M}$ be the DPDA obtained by switching the finalness of states of DPDA M. Then w halts $\overline{M}$ iff it halts M. Also, if w halts M, $w \in L(\overline{M}) \iff w \in \overline{L(M)}$.*

Definition 29. *The configuration (p, w, x) is a dead-end iff $(p, w, x) \vdash_M^* (q, w', y) \implies (w = w' \wedge |x| \leq |y|)$.*

Theorem 23. *DPDA M does not halt on input w iff it has a dead-end configuration.*

Theorem 24. *Every DPDA has an equivalent simple DPDA.*

If (q, w, x) is a dead end in a simple DPDA, then by theorem 21, $w \neq e$ and $x \neq e$.

Theorem 25. *(q, aw, bx) is a dead end iff (q, a, b) is also a dead end.*

Let $D \subseteq Q \times \Sigma \times \Gamma$ be the set of all dead-end triples. This is a well-defined and finite set.

Steps to convert a simple DPDA M into an equivalent simple DPDA M' without dead-ends:

- For all $(q, a, b) \in D$, remove from Δ all transitions compatible with (q, a, b).
- Add 2 non-final states r and r'.
- For all $(q, a, b) \in D$, add the transition $((q, a, b), (r, e))$.
- Add the transitions $((r, a, e), (r, e))$ for all $a \in \Sigma$.
- Add the transition $((r, \$, e), (r', e))$.
- Add the transitions $((r', e, b), (r', e))$ for all $b \in \Gamma$ (by the way, Γ includes the bottom-marker).

If M has no dead-end configurations, $L(\overline{M}) = \overline{L(M)}$. Therefore, deterministic CFLs are closed under complementation.

Theorem 26. *Let $L = \{a^m b^n c^p : m \neq n \vee n \neq p\}$. Then L is context-free but not deterministic context-free.*

Proof. $L' = \overline{L} \cap a^* b^* c^* = \{a^n b^n c^n : n \geq 0\}$ is known to not be context-free. If L was DCFL, $\overline{L}$ would be a DCFL and so L' would be a CFL. □

9 Deterministic Parsing

Definition 30. *For a grammar G, a deterministic parser M is a DPDA which can identify whether $w \in L(G)$ and construct a parse-tree for w in G. It does this by sometimes outputting a rule when following a transition. These rules, taken in order, form the rules applied in a leftmost derivation of w.*

9.1 LL(1) top-down parser

Definition 31. *For grammar $G = (V, \Sigma, R, S)$, an LL(1) top-down parser is a DPDA $M = (\{p, q\} \cup \{q_a : a \in \Sigma \cup \{\$\}\}, \Sigma, V, \Delta, p, \{q_\$\})$ such that Δ has the following transitions:*

- $((p, e, e), (q, S))$

- $((q, a, e), (q_a, e))$ *for all* $a \in \Sigma \cup \{\$\}$
- $((q_a, e, a), (q, e))$ *for all* $a \in \Sigma \cup \{\$\}$
- $((q_a, e, A), (q_a, \beta))$ *for all* $A \to_G B \in R$ *for some* $a \in \Sigma \cup \{\$\}$. *Output* $A \to_G B$ *when following this transition.*

For M *to be deterministic, for transitions* $((q_a, e, A), (q_a, \beta_1))$ *and* $((q_b, e, A), (q_b, \beta_2))$, $a \neq b$.

Not all grammars have an LL(1) top-down parser. But it is sometimes possible to transform a grammar so that it has an LL(1) top-down parser. We now mention two heuristic rules for such transformations.

9.1.1 Heuristic transformation rules

Left factoring. If there are rules $A \to \alpha\beta_i$ for multiple i, transform them to rules $A \to \alpha B$ and $B \to \beta_i$ for all i.

Removing left recursion. If there are rules $A \to A\alpha_i$ and $A \to \beta_j$ for $i \geq 1$ and $j \geq 1$, transform them to rules $A \to \beta_j B$, $B \to e$ and $B \to \alpha_i B$ for $i \geq 1$ and $j \geq 1$.

9.2 First and Follow

Let $G = (V, \Sigma, R, S)$ be a CFG.

Definition 32. $\text{first} : V^* \mapsto \Sigma \cup \{e\}$ *is a function where*

$$\text{first}(\alpha) = \{a \in \Sigma : \exists u \in \Sigma^*, \alpha \Rightarrow_G^* au\} \cup \{e \text{ if } \exists u \in \Sigma^*, \alpha \Rightarrow_G^* e\}$$

Theorem 27.

$$\forall a \in \Sigma \cup \{e\}, \text{first}(a) = \{a\}$$

$$\text{first}(A\gamma) = (\text{first}(A) - \{e\}) \cup \begin{cases} \{\} & \text{if } e \notin \text{first}(A) \\ \text{first}(\gamma) & \text{if } e \in \text{first}(A) \end{cases}$$

$$\text{first}(A) = \bigcup_{A \to_G \beta} \text{first}(\beta)$$

The above theorem can be used to find first for any string in V^*. This isn't an algorithm, but an algorithm exists but I won't mention it here.

Definition 33. $\text{follow} : V - \Sigma \mapsto \Sigma \cup \{\$\}$ *is a function where*

$$\text{follow}(A) = \{a \in \Sigma : \exists x, y \in V^*, S \Rightarrow_G^* xAay\} \cup \{\$ \text{ if } \exists x \in V^*, S \Rightarrow_G^* xA\}$$

Theorem 28.

$$\$ \in \text{follow}(S)$$

$$(A \to_G xBy) \implies$$
$$\left((\text{first}(y) - \{e\}) \cup \begin{cases} \{\} & \text{if } e \notin \text{first}(y) \\ \text{follow}(A) & \text{if } e \in \text{first}(y) \end{cases}\right) \subseteq \text{follow}(B)$$

A closure-like algorithm can find follow for all non-terminals.

9.3 LL(1) grammar

A CFG is LL(1) iff it has an LL(1) top-down parser.

Theorem 29. *A CFG $G = (V, \Sigma, R, S)$ is LL(1) iff all of the following conditions hold for every pair of distinct transitions $A \to_G \alpha$ and $A \to_G \beta$:*

- $\text{first}(\alpha) \cap \text{first}(\beta) = \{\}$.
- $e \in \text{first}(\beta) \implies \text{first}(\alpha) \cap \text{follow}(A) = \{\}$.

Definition 34. $\text{P} : ((V - \Sigma) \times \Sigma) \times V^*$ *where* $\text{P}((A, a), \beta)$
$\iff (A \to_G \beta \in R \wedge (a \in \text{first}(\beta) \vee (e \in \text{first}(\beta) \wedge a \in \text{follow}(A))))$.

Theorem 30. *If G is an LL(1) grammar,*
$(\text{P}((A, a), \beta) \wedge \text{P}((A, a), \gamma)) \implies \beta = \gamma$.

Therefore, P can be expressed as function like $(V - \Sigma) \times \Sigma \mapsto V^* \cup \{\bot\}$ where $P(A, a) = \beta \iff P((A, a), \beta)$ and $P(A, a) = \bot \iff (\forall \beta \in V^*, \overline{P((A, a), \beta)})$.

P is called the parsing table for G. An LL(1) top-down DPDA $M = (\{p, q\} \cup \{q_a : a \in \Sigma \cup \{\$\}\}, \Sigma, V, \Delta, p, \{q_\$\})$ can be made by using the following transitions:

- $((p, e, e), (q, S))$
- $((q, a, e), (q_a, e))$ for all $a \in \Sigma \cup \{\$\}$
- $((q_a, e, a), (q, e))$ for all $a \in \Sigma \cup \{\$\}$
- $((q_a, e, A), (q_a, P(A, a)))$ for all $A \in V - \Sigma$ and $a \in \Sigma$ if $P(A, a) \neq \bot$. Output $A \to_G B$ when following this transition.