Języki, Automaty i Obliczenia. Odcinek 3

# Gramatyki bezkontekstowe

Damian Niwiński

Październik 1996

## 1 Gramatyki i wyprowadzenia

Gramatyka bezkontekstowa może być przedstawiona jako następujący układ

$$G = (\Sigma, V, X_I, R)$$

gdzie

- $\Sigma$ jest skończonym zbiorem symboli, który nazywamy *alfabetem gramatyki*, elementy tego zbioru nazywamy też *symbolami końcowymi* gramatyki;
- $V$ jest skończonym zbiorem symboli rozłącznym z $\Sigma$; jego elementy nazywamy *symbolami niekońcowymi* albo *zmiennymi* gramatyki;
- $X_I \in V$ jest *symbolem startowym* gramatyki;
- $R$ jest skończonym zbiorem *reguł przepisywania* (krótko: reguł), $R \subseteq V \times (\Sigma \cup V)^*$.

Zgodnie z definicją, każda reguła jest parą postaci $(X, w)$, dla pewnej zmiennej $X \in V$ i słowa $w \in (\Sigma \cup V)^*$; mówimy że jest to reguła przepisywania zmiennej $X$. Fakt, że $(X, w) \in R$, będziemy także zapisywać $X \stackrel{G}{\to} w$. Jeżeli $\{(X, w_1), \ldots, (X, w_k)\}$, są wszystkimi regułami przepisywania zmiennej $X$, to będziemy je zapisywać w notacji skróconej, tzw. notacji Backusa-Naura,

$$X \to w_1 | w_2 | \ldots | w_k$$

Relację $R$ rozszerzymy do relacji $\stackrel{G}{\to} \subseteq (\Sigma \cup V)^* \times (\Sigma \cup V)^*$ określonej następująco

$$w \stackrel{G}{\to} v \Leftrightarrow (\exists (X, u) \in R)\, (\exists \alpha, \beta \in (\Sigma \cup V)^*)\, w = \alpha X \beta \wedge v = \alpha u \beta$$

Mówiąc intuicyjnie, relacja $w \stackrel{G}{\to} v$ zachodzi, kiedy słowo $v$ może być otrzymane przez jednokrotne zastosowanie reguły $(X, u)$ do pewnego wystąpienia zmiennej $X$ w słowie $w$.

Zauważmy, że gdy $X$ jest zmienną, to wprowadzona wyżej notacja $X \stackrel{G}{\to} w$ na oznaczenie reguły przepisywania zmiennej $X$ jest niesprzeczna z definicją relacji $\stackrel{G}{\to}$, gdyż $X \stackrel{G}{\to} w \Leftrightarrow (X, w) \in R$.

Każdy ciąg $w_0, w_1, \ldots, w_{k-1}, w_k$, gdzie $0 \leq k$, $w_0 = w$, $w_k = v$, a ponadto $w_i \stackrel{G}{\to} w_{i+1}$ dla $i = 0, \ldots, k-1$, nazywamy *wyprowadzeniem słowa* $v$ *ze słowa* $w$ w gramatyce $G$. Określamy relację $w \stackrel{G*}{\to} v$ przez

$$w \stackrel{G*}{\to} v \iff \text{istnieje wyprowadzenie } v \text{ z } w$$

Nietrudno jest wykazać, że relacja $\stackrel{G*}{\to}$ jest zwrotno–przechodnim domknięciem relacji $\stackrel{G}{\to}$, tj. najmniejszą relacją zwrotną i przechodnią zawierającą $\stackrel{G}{\to}$. Mówimy, że słowo $w \in \Sigma^*$ jest *wyprowadzalne* ze zmiennej $X$ w gramatyce $G$, jeśli $X \stackrel{G*}{\to} w$. Przyjmujemy oznaczenie

$$L(G, X) = \{w \in \Sigma^* : X \stackrel{G*}{\to} w\}$$

Jeśli słowo $w \in \Sigma^*$ jest wyprowadzalne z symbolu startowego gramatyki, to mówimy krótko, że słowo $w$ jest *wyprowadzalne w gramatyce* $G$, a zbiór wszystkich takich słów określamy jako *język wyprowadzalny* w gramatyce $G$ lub *generowany* przez gramatykę $G$, który oznaczamy $L(G)$. Mamy więc

$$L(G) = L(G, X_I) = \{w \in \Sigma^* : X_I \stackrel{G*}{\to} w\}$$

W dalszym ciągu, jeżeli gramatyka $G$ będzie znana z kontekstu, będziemy często opuszczali górny indeks $G$, pisząc po prostu $w \to v$ lub $w \stackrel{*}{\to} v$.

**Przykład 1** (a) Następująca gramatyka nad alfabetem $\{(, ), [, ]\}$ generuje poprawnie zbudowane ciągi nawiasów

$$X \to \epsilon \,|\, XX \,|\, (X) \,|\, [X]$$

(b) Następująca gramatyka nad alfabetem $\{(, ), +, *, 0, 1\}$ generuje poprawnie zbudowane wyrażenia arytmetyczne

$$X \to 0 \,|\, 1 \,|\, X + X \,|\, X * X \,|\, (X)$$

## 1.1 Wyprowadzenia lewostronne

Określamy relację $\stackrel{lG}{\to}$ stanowiącą podzbior relacji $\stackrel{G}{\to}$,

$$w \stackrel{lG}{\to} v \Leftrightarrow (\exists (X, u) \in R)\,(\exists \alpha \in \Sigma^*)(\exists \beta \in (\Sigma \cup V)^*) w = \alpha X \beta \wedge v = \alpha u \beta$$

Mówiąc intuicyjnie, słowo $v$ zostało otrzymane ze słowa $w$ podobnie jak w relacji $w \stackrel{G}{\to} v$, ale wystąpienie zmiennej $X$ do jakiego została zastosowana reguła przepisywania $X \stackrel{G}{\to} u$ było najbardziej na lewo wysuniętym (ang. *left-most*) wystąpieniem jakiegokolwiek symbolu niekońcowego w słowie $w$. Ciąg $w = w_0, w_1, \ldots, w_{k-1}, w_k = v$, $0 \leq k$, t. że $w_i \stackrel{lG}{\to} w_{i+1}$, dla $i = 0, \ldots, k-1$, nazywamy *wyprowadzeniem lewostronnym* (ang. *left-most derivation*) słowa $v$ ze słowa $w$. Zwrotno-przechodnie domknięcie relacji $\stackrel{lG}{\to}$ będziemy oznaczać $\stackrel{lG*}{\to}$.

## 1.2 Drzewa wyprowadzeń

Struktura drzewa jest spotykana w informatyce we wielu kontekstach, przy czym używanych jest kilka nierównoważnych modeli matematycznych tego pojęcia. Na przyklad, jeśli zdefiniujemy drzewo jako spójny graf, w którym liczba wierzchołków jest o 1 większa od liczby krawędzi, to pominiemy taki aspekt drzewa jak wyróżnienie pewnego wierzchołka jako korzenia oraz uporządkowanie następników danego wierzchołka "od lewej do prawej". Dogodnym modelem uwzględniającym oba te aspekty jest przedstawienie drzewa jako pewnego zbioru słów, lub, w przypadku drzewa etykietowanego, funkcji ze zbioru słów w zbiór etykiet. Idea jest tu bardzo prosta: w drzewie, powiedzmy, binarnym, można utożsamić wierzchołek ze ścieżką jaką można do niego dojść od korzenia, tzn. ciągiem wyboru kierunków, np. *lewo, prawo, prawo* . W ogólniejszym przypadku, za zbiór kierunków możemy przyjąć dowolny segment początkowy zbioru liczb naturalnych.

Zbiór liczb naturalnych oznaczamy $\omega$. Zgodnie z naszą konwencją, $\omega^*$ oznacza zbiór słów nad $\omega$, a więc skończonych ciągów o wyrazach naturalnych. Jeśli $v \in \omega^*$, to każdy ciąg $vi$, $i \in \omega$, nazywamy *następnikiem* ciągu $v$. Jeśli $I \subseteq \omega^*$ i $v \in I$, to przez $\mathrm{succ}(I, v)$ oznaczamy zbiór tych następników ciągu $v$, które należą do $I$. Niepusty skończony zbiór $T \subseteq \omega^*$ nazywamy *drzewem* (nieetykietowanym), jeśli spełnione są następujące warunki

- $T$ jest zamknięty na prefiksy (tzn. $(w \in T \wedge v \leq w) \Rightarrow v \in T$)
- dla każdego $v \in T$, istnieje $k \in \omega$ (zależne od $v$), takie, że $\mathrm{succ}\,(I, v) = \{v1, \ldots, vk\}$ (jeśli $k = 0$, to $\mathrm{succ}(I, v) = \emptyset$).

Elementy drzewa $T$ nazywamy wierzchołkami, słowo $\epsilon$ (które oczywiście znajduje się w każdym drzewie) nazywamy *korzeniem*, a wierzchołki $v$, dla których $\mathrm{succ}(I, v) = \emptyset$, nazywamy *liśćmi* drzewa $T$. Zauważmy, że relacja "być prefiksem"częściowo porządkuje zbiór wierzchołków drzewa[1]. Liczbę

$$\mathrm{depth}(T) = \max\{|w| \,:\, w \in \mathrm{dom}(T)\}$$

nazywamy *głębokością* drzewa $T$.

Jeśli $E$ jest dowolnym zbiorem, a $T$ jest drzewem, to każdą funkcję $t : T \to E$ nazywamy *drzewem etykietowanym* zbiorem etykiet $E$. Zbiór $T$ nazywamy wówczas *dziedziną* drzewa $t$ i oznaczamy $\mathrm{dom}(t)$ (z angielskiego *domain* = dziedzina). Pojęcia "korzeń", "liść" itd. używane w odniesieniu do drzewa etykietowanego, odnoszą się oczywiście do $\mathrm{dom}(t)$.

Jeśli $t : \mathrm{dom}(t) \to E$ jest drzewem i $w \in \mathrm{dom}(t)$, to *poddrzewem drzewa $t$ wyznaczonym przez wierzchołek $w$*, nazywamy drzewo $t.w$ określone następująco:

- $\mathrm{dom}(t.w) = \{w\}^{-1}\,\mathrm{dom}(t) = \{u : wu \in \mathrm{dom}(t)\}$,
- $t.w(u) = t(wu)$, dla $u \in \mathrm{dom}(t.w)$.

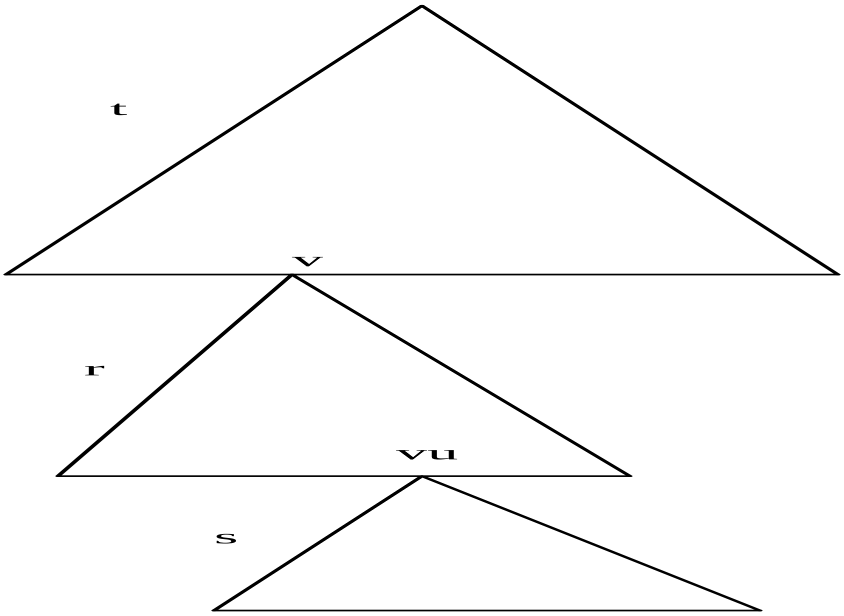
[1] Porządek ten można go rozszerzyć do porządku liniowego, np. porządku leksykograficznego na $\omega^*$ indukowanego przez standardowy porządek na $\omega$.

Zdefiniujemy teraz operację *podstawienia*, która, mówiąc intuicyjnie, pozwoli zastępować liście drzewa innymi drzewami. Niech $t : \mathrm{dom}(t) \to E$ będzie drzewem etykietowanym i niech $v_1, \ldots, v_m$ będą różnymi liśćmi drzewa $t$ (niekoniecznie wszystkimi). Niech $t_i : \mathrm{dom}(t_i) \to E$, $i = 1, \ldots, m$, będą etykietowanymi drzewami. Określimy drzewo $t[v_1 \leftarrow t_1, \ldots, v_m \leftarrow t_m]$, w skróconej notacji $t[\vec{v} \leftarrow \vec{t}]$, które stanowi rezultat jednoczesnego podstawienia w drzewie $t$ drzewa $t_i$ na wierzchołek $v_i$ odpowiednio, dla $i = 1, \ldots, m$.

- $\mathrm{dom}(t[\vec{v} \leftarrow \vec{t}]) = \mathrm{dom}(t) \cup \bigcup_{i=1,\ldots,m} v_i \mathrm{dom}(t_i)$
- $t[\vec{v} \leftarrow \vec{t}](w) = t(w)$, dla $w \in \mathrm{dom}(t) - \{v_1, \ldots, v_m\}$
- $t[\vec{v} \leftarrow \vec{t}](v_i u) = t_i(u)$, dla każdego $u \in \mathrm{dom}(t_i)$, $i = 1, \ldots, m$.

Opierając się na definicji, nie jest trudno sprawdzić, że operacja podstawienia jest łączna w następującym sensie: niech $t, r, s$ będą drzewami i niech $v \in \mathrm{dom}(t)$, $u \in \mathrm{dom}(r)$. Wówczas:

$$(t[v \leftarrow r])[vu \leftarrow s] = t[v \leftarrow r[u \leftarrow s]]$$




Niech $G = (\Sigma, V, X_I, R)$ będzie gramatyką i niech $Y \in V$. Drzewo $D : \mathrm{dom}(D) \to \Sigma \cup V \cup \{\epsilon\}$ nazwiemy *drzewem wyprowadzenia* ze zmiennej $Y$, jeśli spełnione są następujące warunki.

1. $D(\epsilon) = Y$;
2. dla każdego $w \in \mathrm{dom}(D)$, jeśli $D(w) \in \Sigma \cup \{\epsilon\}$, to $w$ jest liściem;
3. dla każdego $w \in \mathrm{dom}(D)$, jeśli $D(w) \in V$, powiedzmy $D(w) = Z$, to zachodzi jedna z następujących możliwości:
   (a) $w$ jest liściem;
   (b) istnieje reguła $Z \overset{G}{\to} \epsilon$ i $w$ ma dokładnie jeden następnik $w1$, przy czym $D(w1) = \epsilon$;

(c) istnieje reguła $Z \stackrel{G}{\to} c_1 \dots c_k$, gdzie $c_1, \dots, c_k \in \Sigma \cup V$ i $w$ ma dokładnie $k$ następników: $w1, \dots, wk$, oraz $D(w\ell) = c_\ell$, dla $\ell = 1, \dots, k$.

Plonem drzewa wyprowadzenia $D$ nazwiemy słowo $\alpha \in (\Sigma \cup V)^*$, jakie można otrzymać z drzewa składając etykiety wszystkich liści w porządku "od lewej do prawej". Bardziej dokładnie, niech $v_1, \dots, v_m$ będą wszystkimi liśćmi drzewa $D$ w porządku leksykograficznym[2]. Określamy słowo $y(D) \in (\Sigma \cup V)^*$ (z angielskiego *yield*), nazywane *plonem drzewa* $D$,

$$y(D) = D(v_1) \dots D(v_m)$$

Drzewo wyprowadzenia nazwiemy maksymalnym, jeśli nie zachodzi przypadek 3a, tzn. żaden liść nie jest etykietowany zmienną. Nietrudno jest wykazać, że jeśli $D$ jest maksymalnym drzewem wyprowadzenia, to $y(D) \in \Sigma^*$.

Zbiór wszystkich drzew wyprowadzenia w gramatyce $G$ ze zmiennej $X$ będziemy oznaczali przez $\mathcal{D}_G(X)$, zbiór wszystkich drzew wyprowadzenia przez $\mathcal{D}_G$.

**Lemat 1** *Niech $G = (\Sigma, V, X_I, R)$ będzie gramatyką, $X \in V$ i $w \in (\Sigma \cup V)^*$. Następujące warunki są równoważne:*

**(i)** $X \stackrel{G*}{\to} w$,

**(ii)** *istnieje drzewo wyprowadzenia $D$ ze zmiennej $X$, takie że $y(D) = w$.*

*Dowód.*

$(i) \Rightarrow (ii)$

Niech $X = w_0 \stackrel{G}{\to} w_1 \stackrel{G}{\to} \dots w_{k-1} \stackrel{G}{\to} w_k = w$ będzie wyprowadzeniem słowa $w$ ze zmiennej $X$. Dla $\ell = 0, 1, \dots, k$, określamy kolejno drzewa wyprowadzenia $D_\ell$, takie, że $y(D_\ell) = w_\ell$.

Drzewo $D_0$ określamy następująco: $\mathrm{dom}(D_0) = \{\epsilon\}$, $D_0(\epsilon) = X$.

Przypuśćmy, że jest już określone drzewo $D_\ell$, $\ell < k$ i $y(D_\ell) = w_\ell$. Z definicji wyprowadzenia mamy $w_\ell = \alpha Y \beta$ i $w_{\ell+1} = \alpha u \beta$, gdzie $Y \stackrel{G}{\to} u$ jest regułą przepisywania gramatyki $G$, a $\alpha, \beta \in (\Sigma \cup V)^*$. Załóżmy, że $w_\ell = s_1 \dots s_m$, gdzie $s_1, \dots, s_m \in \Sigma \cup V$, $\alpha = s_1 \dots s_{i-1}$, $s_i = Y$ i $\beta = s_{i+1} \dots s_m$, dla pewnego $i$, $1 \le i \le m$ (gdy $i = 1$, $\alpha = \epsilon$, gdy $i = m$, $\beta = \epsilon$). Niech $v_1, \dots, v_{m'}$ będą tymi spośród liści drzewa $D_\ell$, dla których $D_\ell(v_i) \neq \epsilon$, w porządku leksykograficznym. Jest jasne, że $m' = m$ i $D_\ell(v_j) = s_j$, dla $j = 1, \dots, m$. W szczególności, $D_\ell(v_i) = Y$.

Rozróżnimy teraz dwa przypadki.

1. $u = \epsilon$. Określamy drzewo $D_{\ell+1}$ następująco: $\mathrm{dom}(D_{\ell+1}) = \mathrm{dom}(D_\ell) \cup \{v_i 1\}$, $D_{\ell+1}(v) = D_\ell(v)$ dla $v \in \mathrm{dom}(D_\ell)$ oraz $D_{\ell+1}(v_i 1) = \epsilon$.

2. $u = u_1 \dots u_r$, $r > 0$. Określamy drzewo $D_{\ell+1}$ następująco: $\mathrm{dom}(D_{\ell+1}) = \mathrm{dom}(D_\ell) \cup \{v_i 1, \dots, v_i r\}$, $D_{\ell+1}(v) = D_\ell(v)$ dla $v \in \mathrm{dom}(D_\ell)$ oraz $D_{\ell+1}(v_i 1) = u_1, \dots, D_{\ell+1}(v_i r) = u_r$.

[2] Liść $v$ leksykograficznie poprzedza liść $w$ jeśli istnieją $u \in \omega^*$, $i, j \in \omega$ takie, że $ui \le v$, $uj \le w$ i $i < j$.

Nietrudno sprawdzić, że $D_{\ell+1}$ jest drzewem wyprowadzenia i $y(D_{\ell+1}) = w_{\ell+1}$.

$(ii) \Rightarrow (i)$

Należy wykazać, że dla każdego drzewa wyprowadzenia $D \in \mathcal{D}_X$, $X \stackrel{G*}{\rightarrow} y(D)$. Dowód poprowadzimy przez indukcję ze względu na głębokość drzewa $D$. Gdy $D$ jest drzewem o głębokości 0, to $X = y(D)$ i oczywiście $X \stackrel{G*}{\rightarrow} y(D)$. Załóżmy, że $\mathrm{depth}(D) \geq 1$. Wówczas istnieje reguła $X \stackrel{G}{\rightarrow} u$, taka że zachodzi jeden z dwóch przypadków

1. $u = \epsilon$ i korzeń drzewa $D$ (również $\epsilon$) ma dokładnie jeden następnik, mianowicie 1, przy czym $D(1) = \epsilon$.

2. $u = u_1 \ldots u_r$, gdzie $r > 0$, $u_1, \ldots, u_r \in \Sigma \cup V$ i $\epsilon$ ma dokładnie $r$ następników w drzewie $D$, mianowicie $1, \ldots, r$, przy czym $D(i) = u_i$, dla $i = 1, \ldots, r$.

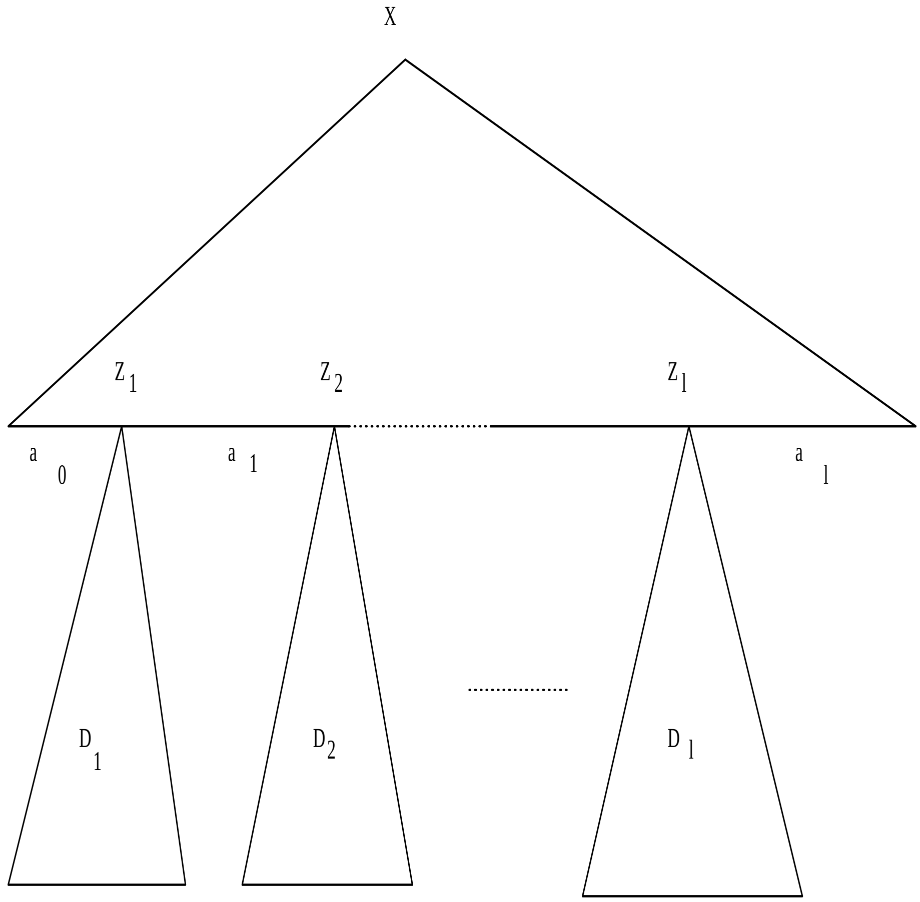
W pierwszym przypadku, oczywiście $X \stackrel{G*}{\rightarrow} y(D) = \epsilon$.

Załóżmy drugi przypadek. Zauważmy najpierw, że o ile $u_i = Y$, to poddrzewo $D.i$ wyznaczone przez wierzchołek $i$ jest drzewem wyprowadzenia ze zmiennej $Y$, przy czym $\mathrm{depth}(D.i) < \mathrm{depth}(D)$, a zatem, z założenia indukcyjnego, $Y \stackrel{G*}{\rightarrow} y(D.i)$. Z kolei $u$ możemy przedstawić $u = \alpha_0 Z_1 \alpha_1 Z_2 \alpha_2 \ldots Z_m \alpha_m$, gdzie, dla $i = 0, 1, \ldots, m$, $\alpha_i \in \Sigma^*$ (tzn. słowa $\alpha_i$ nie zawierają zmiennych). Z definicji plonu, łatwo wynika, że

$$y(D) = \alpha_0 y(D_1) \alpha_1 y(D_2) \alpha_2 \ldots y(D_\ell) \alpha_\ell$$

Pamiętając o tym iż, jak już zauważyliśmy, $Z_i \stackrel{G*}{\rightarrow} y(D_i)$, dla $i = 1, \ldots, \ell$, otrzymujemy

$$\begin{array}{rcl}
X & \stackrel{G}{\rightarrow} & \alpha_0 Z_1 \alpha_1 Z_2 \alpha_2 \ldots Z_\ell \alpha_\ell \\
 & \stackrel{G*}{\rightarrow} & \alpha_0 y(D_1) \alpha_1 Z_2 \alpha_2 \ldots Z_\ell \alpha_\ell \\
 & \stackrel{G*}{\rightarrow} & \alpha_0 y(D_1) \alpha_1 y(D_2) \alpha_2 \ldots Z_\ell \alpha_\ell \\
. & . & . \\
 & \stackrel{G*}{\rightarrow} & \alpha_0 y(D_1) \alpha_1 y(D_2) \alpha_2 \ldots y(D_\ell) \alpha_\ell \\
 & = & y(D)
\end{array}$$




□

Kiedy słowo $w$ nie zawiera symboli nieterminalnych, to udowodniony przed chwilą lemat możemy rozszerzyć o jeszcze jeden równoważny warunek: istnieje wyprowadzenie lewostronne słowa $w$.

**Twierdzenie 1** *Niech $G = (\Sigma, V, X_I, R)$ będzie gramatyką, i niech $w \in \Sigma^*$. Następujące warunki są równoważne:*

**(i)** *istnieje wyprowadzenie słowa $w$ ze zmiennej $X$,*

**(ii)** *istnieje maksymalne drzewo wyprowadzenia ze zmiennej $X$, którego plonem jest $w$,*

**(iii)** *istnieje wyprowadzenie lewostronne słowa $w$ ze zmiennej $X$.*

*Dowód.* Implikacja $(i) \Rightarrow (ii)$ została udowodniona w Lemacie 1 (maksymalność drzewa jest oczywista, gdy plon nie zawiera symboli niekońcowych), a implikacja $(iii) \Rightarrow (i)$ jest oczywista. Dla dowodu implikacji $(ii) \Rightarrow (iii)$, wystarczy wykazać, że dla każdego drzewa wyprowadzenia $D \in \mathcal{D}_X$, takiego że $y(D) \in \Sigma^*$, $X \overset{lG*}{\to} y(D)$. Dowód jest podobny do dowodu analogicznego faktu w dowodzie Lematu 1. □

# 2 Postaci normalne gramatyk bezkontekstowych

W tym rozdziale przedstawimy efektywną metodę przekształcania gramatyki bezkontekstowej w równoważną gramatykę o stosunkowo prostej postaci, tzw. postaci normalnej Chomsky'ego. Jako produkt uboczny naszych rozważań uzyskamy także algorytm odpowiadający na pytanie, czy język generowany przez gramatykę jest niepusty.

## 2.1 Eliminacja symboli nieużytecznych

Mówiąc intuicyjnie, zmienną w gramatyce można uznać za nieużyteczną, kiedy nie da się jej osiągnąć w żadnym wyprowadzeniu z symbolu startowego, lub kiedy *z niej* nie można wyprowadzić żadnego słowa złożonego z symboli końcowych.

Niech $G = (\Sigma, V, X_I, R)$ będzie gramatyką. Powiemy, że zmienna $Y$ jest *osiągalna* ze zbioru zmiennych $E \subseteq V$, jeśli istnieją $X \in E$ oraz $\alpha, \beta \in (\Sigma \cup V)^*$, takie że $X \stackrel{G*}{\to} \alpha Y \beta$.

Podobnie, jak to było dla automatu, określimy funkcję, dla $E \subseteq V$

$$\mathit{Step}(E) = \{Y \,:\, (\exists X \in E)(\exists \alpha, \beta \in (\Sigma \cup V)^*) X \stackrel{G}{\to} \alpha Y \beta\}$$

Poniższy algorytm, podobny do Algorytmu 3 z Odcinka 1, oblicza zbiór zmiennych osiągalnych z $E$.

**Algorytm 5**
**Dane:** Gramatyka $G$, zbiór zmiennych $E \subseteq V$
1. $\mathcal{X} := \emptyset; \mathcal{X}_1 := I;$
2. **Dopóki** $(\mathcal{X}_1 \neq \emptyset)$ **wykonuj**
3. $\quad\mathcal{X} := \mathcal{X} \cup \mathcal{X}_1;$
4. $\quad\mathcal{X}_1 := \mathit{Step}(\mathcal{X}_1) \setminus \mathcal{X};$

**Wynik:** zbiór $\mathcal{X}$

Zauważmy, że linia 3 może być wykonana co najwyżej $|V|$ razy (za każdym razem dokładamy coś do $\mathcal{X}$), zatem algorytm na pewno się zatrzymuje po wykonaniu nie więcej niż $|V|$ obrotów pętli 2. Ponadto, łatwo jest widzieć, że, na każdym etapie wykonywania algorytmu, wszystkie zmienne w zbiorze stanowiącym wartość zmiennej $\mathcal{X}$, są osiągalne z $E$. Pozostaje wykazać, że w końcowej wartości zmiennej $\mathcal{X}$ znajdują się już wszystkie zmienne osiągalne ze zbioru $E$. Istotnie, przypuśćmy, że zmienna $Z$ jest osiągalna z $Y \in E$ przy pomocy wyprowadzenia

$$Y = w_0 \stackrel{G}{\to} w_1 \stackrel{G}{\to} \ldots \stackrel{G}{\to} w_{k-1} \stackrel{G}{\to} w_k = \alpha Z \beta$$

Wówczas można zaobserwować, że jeśli po wykonaniu pętli (2.) $i$ razy wszystkie zmienne występujące w słowie $w_\ell$ znajdują się w wartości zmiennej $\mathcal{X}$, to każda ze zmiennych występujących w słowie $w_{\ell+1}$ znajdzie się w w wartości zmiennej $\mathcal{X}$ nie później niż po wykonaniu pętli $i + 1$ razy. W szczególności zmienna $Z$ znajdzie się w końcowej wartości zmiennej $\mathcal{X}$.

Z kolei, powiemy, że zmienna $X \in V$ jest *produktywna*, jeżeli $X \stackrel{G*}{\to} w$, dla pewnego $w \in \Sigma^*$. Niech, dla $u \in (\Sigma \cup V)^*$, $\mathrm{var}(u)$ oznacza zbiór wszystkich zmiennych, które występują w $u$. Określimy następującą funkcję na $P(V)$,

$$BackStep(E) = \{X \,:\, (\exists u \in (\Sigma \cup V)^*)(X \stackrel{G}{\to} u \wedge \mathrm{var}(u) \subseteq E)\}$$

Zauważmy, że jeżeli wszystkie zmienne w zbiorze $E$ są produktywne, to również wszystkie zmienne w zbiorze $BackStep(E)$ są produktywne. Zauważmy także, że $BackStep(\emptyset)$ jest zbiorem tych zmiennych, z których w jednym kroku można wygenerować słowo złożone z symboli końcowych.

Następujący algorytm oblicza zbiór wszystkich zmiennych produktywnych:

**Algorytm 6**
**Dane:** Gramatyka $G$
1. $\mathcal{X} := \emptyset$; $\mathcal{X}_1 := BackStep\ (\emptyset)$;
2. **Dopóki** $(\mathcal{X}_1 \neq \emptyset)$ **wykonuj**
3. $\quad\quad \mathcal{X} := \mathcal{X} \cup \mathcal{X}_1$;
4. $\quad\quad \mathcal{X}_1 := BackStep(\mathcal{X}) \setminus \mathcal{X}$;

**Wynik:** zbiór $\mathcal{X}$

Tu również, podobnie jak w algorytmie 5, linia 3 może być wykonana co najwyżej $|V|$ razy, zatem algorytm na pewno się zatrzymuje po wykonaniu nie więcej niż $|V|$ obrotów pętli 2. Wykażemy, że końcowa wartość zmiennej $\mathcal{X}$ jest zbiorem wszystkich zmiennych produktywnych.

Istotnie, z własności funkcji *BackStep* wynika, że wartości jakie zmienna $\mathcal{X}$ przyjmuje w trakcie wykonywania algorytmu są zbiorami zmiennych produktywnych. Pozostaje wykazać, że końcowa wartość zmiennej $\mathcal{X}$ obejmuje już wszystkie zmienne produktywne. W tym celu przypuśćmy, że $Y$ jest zmienną produktywną i

$$Y = w_0 \stackrel{G}{\to} w_1 \ldots w_{k-1} \stackrel{G}{\to} w_k = w$$

jest wyprowadzeniem pewnego słowa $w \in \Sigma^*$. Wówczas można zaobserwować, że jeśli po wykonaniu pętli (2.) $i$ razy wszystkie zmienne występujące w słowie $w_\ell$, $\ell > 0$, znajdują się w wartości zmiennej $\mathcal{X}$, to każda ze zmiennych występujących w słowie $w_{\ell-1}$ znajdzie się w w wartości zmiennej $\mathcal{X}$ nie później niż po wykonaniu pętli $i+1$ razy. W szczególności zmienna $Y$ znajdzie się w końcowej wartości zmiennej $\mathcal{X}$.

Zauważmy, że $L(G) \neq \emptyset$ wtedy i tylko wtedy, gdy symbol startowy $X_I$ jest zmienną produktywną. Mamy więc

**Fakt 1** *Istnieje algorytm, który, dla danej gramatyki bezkontekstowej $G$ rozstrzyga, czy $L(G) \neq \emptyset$.* $\Box$

Przedstawione wyżej algorytmy zastosujemy teraz dla sprowadzenia gramatyki do takiej postaci, w której nie ma symboli nieużytecznych.

Mówimy, że gramatyki $G$ i $H$ są równoważne, kiedy $L(G) = L(H)$.

**Fakt 2** *Dla każdej gramatyki $G$ takiej, że $L(G) \neq \emptyset$ można skonstruować równoważną gramatykę $G'$, w której każda zmienna jest produktywna i osiągalna z symbolu początkowego.*

*Dowód.* Jeśli $G = (\Sigma, V, X_I, R)$ jest gramatyką i $E \subseteq V$, $X_I \in E$, to przez *obcięcie* gramatyki $G$ do zbioru $E$ rozumiemy gramatykę $G|E =_{df} (\Sigma, E, X_I, R')$, gdzie $R' = R \cap E \times (\Sigma \cup E)^*$. (Innymi słowy, reguła $Y \to w$ gramatyki $G$ pozostaje regułą gramatyki $G|E$, o ile $Y \in E$ i $\mathrm{var}(w) \subseteq E$.) Niech $E$ będzie zbiorem wszystkich zmiennych produktywnych gramatyki $G$. Z założenia, $X_I \in E$. Oczywiście, gramatyka $G|E$ jest równoważna gramatyce $G$. (Wszystkie zmienne występujące w wyprowadzeniu słowa $w \in \Sigma^*$ w gramatyce $G$ muszą być produktywne.) Z kolei, niech $E' \subseteq E$ będzie zbiorem tych zmiennych, które są osiągalne z $X_I$ w gramatyce $G|E$. Rozważmy gramatykę $(G|E)|E'$. Łatwo widzieć, że gramatyka ta jest równoważna gramatyce $G|E$ (a więc i $G$) i, oczywiście, wszystkie zmienne tej gramatyki są w niej osiągalne z $X_I$. Z drugiej strony, nietrudno wykazać, że każda zmienna z $E'$, która była produktywna w $G$, jest nadal produktywna w $(G|E)|E'$. Zatem gramatyka $(G|E)|E'$ spełnia żądane warunki. □

**Przykład 2** Niech

$$G = (\{a\}, \{X, Y, Z\}, X, \{(X, \epsilon), (X, YZ), (Z, \epsilon)\})$$

Wszystkie zmienne są osiągalne z $X$, ale zmienna $Y$ jest nieproduktywna. Stosując konstrukcję z powyższego dowodu, otrzymujemy

$$G|E = (\{a\}, \{X, Z\}, X, \{(X, \epsilon), (Z, \epsilon)\})$$

i

$$(G|E)|E' = (\{a\}, \{X\}, X, \{(X, \epsilon)\})$$

Zauważmy, że gdybyśmy odwrócili kolejność wykonywanych operacji i *najpierw* ograniczyli gramatykę $G$ do zmiennych osiągalnych z $X$, a *następnie* do zmiennych produktywnych, to otrzymalibyśmy kolejno gramatyki $G$ i $G|E$, a zatem taka konstrukcja nie byłaby poprawna.

## Eliminacja trywialnych reguł

Regułę postaci $X \stackrel{G}{\to} \epsilon$ nazywamy krótko $\epsilon$-*regułą*.

**Fakt 3 (eliminacja $\epsilon$-reguł)** *Dla każdej gramatyki $G$, można skonstruować gramatykę $G'$ bez $\epsilon$-reguł, taką, że $L(G') = L(G) - \{\epsilon\}$.*

*Dowód.* Niech $G = (\Sigma, V, X_I, R)$ będzie gramatyką. Zmienną $X \in V$ nazwiemy *zerowalną*, jeśli $X \stackrel{G*}{\to} \epsilon$. Zbiór zmiennych zerowalnych można wyznaczyć efektywnie przy pomocy procedury podobnej do algorytmu obliczania zbioru zmiennych produktywnych (Algorytm 6); szczegóły pozostawiamy Czytelnikowi.

Zauważmy, że nie wystarczy po prostu usunąć wszystkie $\epsilon$-reguły, bo w ten sposób moglibyśmy przy okazji uniemożliwić niektóre wyprowadzenia słów różnych od $\epsilon$ (np. gdy produkcje są $X \to YZ$, $Y \to \epsilon$, $Z \to a$, to, usuwając regułę $Y \to \epsilon$, uniemożliwimy wyprowadzenie $X \stackrel{*}{\to} a$).

Przedstawimy dwa sposoby, z których pierwszy jest bardziej bezpośredni i łatwy do zrozumienia, ale drugi bardziej ekonomiczny.

**Sposób 1**

Usuwamy ze zbioru $R$ wszystkie $\epsilon$-reguły, a następnie, dla każdej reguły $X \to \alpha$ tworzymy wszystkie reguły jakie można utworzyć zastępując w $\alpha$ dowolne zmienne zerowalne przez $\epsilon$ z wyjątkiem (jeśli taka reguła mogłaby powstać) reguły $X \to \epsilon$.

Dowód, że, dla każdego $w \neq \epsilon$,

$$X \stackrel{G*}{\to} w \Leftrightarrow X \stackrel{G'*}{\to} w$$

można przeprowadzić przez indukcję ze względu na głębokość drzewa wyprowadzenia.

**Sposób 2**

Objaśnimy go na przykładzie. Przypuśćmy, że mamy regułe $X \to Z_1 \ldots Z_m$ i wszystkie zmienne $Z_1, \ldots, Z_m$ są zerowalne. Wprowadzamy nowe zmienne $[Z_1 Z_2 \ldots Z_m]$, $[Z_2 Z_3 \ldots Z_m]$, $\ldots$, $[Z_{m-1} Z_m]$, $[Z_m]$ i dodajemy następujące reguły:

$$\begin{array}{rcll}
X & \to & Z_i & i = 1, \ldots, m \\
X & \to & Z_i [Z_{i+1} Z_{i+2} \ldots Z_m] & i = 1, \ldots, m \\
[Z_i Z_{i+1} \ldots Z_m] & \to & Z_j & i = 1, \ldots, m-1,\ j = i, i+1, \ldots, m \\
[Z_i Z_{i+1} \ldots Z_m] & \to & Z_j [Z_{j+1} Z_{j+2} \ldots Z_m] & i = 1, \ldots, m-1,\ j = i, i+1, \ldots, m
\end{array}$$

Zauważmy, że stosując pierwszy sposób, zastąpilibyśmy regułe $X \to Z_1 \ldots Z_m$ przez $2^m - 1$ reguł, a przy drugim sposobie przez $O(m^2)$ reguł. □

Regułę postaci $X \stackrel{G}{\to} Y$ nazywamy *regułą jednostkową.*

**Lemat 2** *Dla każdej gramatyki $G$, takiej, że $L(G) \neq \emptyset$ i $\epsilon \notin L(G)$, można skonstruować równoważną gramatykę bez reguł jednostkowych, bez $\epsilon$-reguł i taką, że każda zmienna jest produktywna i osiągalna z symbolu początkowego.*

*Dowód.* Niech $G = (\Sigma, V, X_I, R)$ będzie gramatyką. Na mocy Faktu 3, możemy założyć, że $G$ jest już bez $\epsilon$-reguł.

Usuwamy z $R$ wszystkie reguły jednostkowe, natomiast dla każdych $X, Y$, jeśli $X \stackrel{G*}{\to} Y$ i $Y \to \alpha$, gdzie $\alpha$ jest różne od zmiennej, dokładamy regułę $X \to \alpha$. Zauważmy, że możemy łatwo rozstrzygnąć czy $X \stackrel{G*}{\to} Y$ w $|V|$ krokach, ponieważ nie ma $\epsilon$-reguł.

Niech $R'$ będzie otrzymanym zbiorem reguł.

Oczywiście, nie dodaliśmy żadnej $\epsilon$–reguły.

Dla zapewnienia ostatniego z warunków Lematu, stosujemy procedurę z dowodu Faktu 2, która eliminuje ewentualne symbole nieużyteczne. □

## 2.2 Twierdzenia o postaci normalnej gramatyk

**Twierdzenie 2 (o postaci normalnej Chomsky'ego)** *Dla każdej gramatyki $G$, takiej, że $L(G) \neq \emptyset$ i $\epsilon \notin L(G)$, można skonstruować równoważną gramatykę, w której każda reguła jest postaci $X \to \sigma$, gdzie $\sigma \in \Sigma$, lub $X \to YZ$, gdzie $Y, Z \in V$ ($X, Y, Z$ nie muszą być różne).*

*Dowód.* Dla każdego $\sigma \in \Sigma$, tworzymy nową zmienną $Z_\sigma$ i do zbioru reguł gramatyki $G$ dokładamy regułę $Z_\sigma \to \sigma$. Na mocy Lematu 2, możemy założyć, że każda reguła gramatyki $G$, która nie jest postaci $X \to \sigma$, jest postaci $X \to \alpha$, $|\alpha| \geq 2$. Modyfikujemy tę regułę zastępując w $\alpha$ każde $\sigma \in \Sigma$ przez $Z_\sigma$. W ten sposób otrzymujemy gramatykę, w której każda reguła jest postaci $X \to \sigma, \sigma \in \Sigma$, lub $X \to Z_1 \ldots Z_m$, gdzie $Z_1, \ldots, Z_m \in V$, $m \geq 2$. Jeżeli $m > 2$, to wprowadzamy nowe zmienne $[Z_2 Z_3 \ldots Z_m]$, ..., $[Z_{m-1} Z_m]$ i reguły

$$\begin{array}{rcll} X & \to & Z_1[Z_2 Z_3 \ldots Z_m] & \\ [Z_i \ldots Z_m] & \to & Z_i[Z_{i+1} \ldots Z_m] & \text{dla } i < m-1 \\ [Z_{m-1} Z_m] & \to & Z_{m-1} Z_m & \end{array}$$

□

**Pytanie.** Jeśli gramatyka $G$ jest w postaci Chomsky'ego, to jak możemy oszacować z góry długość wyprowadzenia słowa $w \in L(G)$, w zależności od długości $w$ ?

W uzupełnieniu powyższego twierdzenia, podamy dla informacji, bez dowodu, inne twierdzenie o postaci normalnej, którego autorem jest pani S.Greibach.

**Twierdzenie 3 (o postaci normalnej Greibach)** *Dla każdej gramatyki $G$, takiej, że $L(G) \neq \emptyset$ i $\epsilon \notin L(G)$, można skonstruować równoważną gramatykę, w której każda reguła jest postaci $X \to \sigma\alpha$, gdzie $\sigma \in \Sigma$ i $\alpha \in V^*$.*

## 2.3 Algorytm rozpoznawania języków bezkontekstowych

Rozważmy następujący problem:

> **Dane** Gramatyka $G$, słowo $w \in \Sigma^*$
> **Pytanie** Czy $w \in L(G)$?

Jeśli $w = \epsilon$ to problem możemy rozwiązać obliczając zbiór zmiennych $\epsilon$-produktywnych, tzn. takich, że $X \stackrel{G*}{\to} \epsilon$ podobnie jak obliczaliśmy zbiór zmiennych produktywnych w Algorytmie 6 (str. 9), z tym, że w funkcji *Back-Step* bierzemy pod uwagę tylko te reguły w których nie występują w ogóle symbole terminalne.

Ciekawszy jest przypadek, gdy $|w| > 0$.

Moglibyśmy wówczas znaleźć gramatykę Chomsky'ego $G'$ generującą $L(G) - \{\epsilon\}$ i przeszukiwać wszystkie wyprowadzenia o ograniczonej długości (por. "Pytanie" powyżej). Metoda ta byłaby jednak zupełnie nieefektywna.

Poniżej naszkicujemy algorytm zaproponowany przez Cocke'a, Youngera i Kasamiego, i od nazwisk autorów zwany algorytmem CYK. Algorytm ten, dla gramatyk Chomsky'ego rozwiązuje zadanie w wielomianowym czasie.

Zakładamy, że $G = (\Sigma, V, X_I, R)$ jest w postaci Chomsky'ego. Niech $w = w_1 \ldots w_n$, gdzie $w_1, \ldots, w_n \in \Sigma$. Główną pracą algorytmu jest indukcyjne obliczanie tablicy zbiorów $V_{i,j}$, $i = 1, \ldots, n$, $j = 1, \ldots, n+1-i$, gdzie

$$V_{i,j} = \{X : X \stackrel{G*}{\to} w_i w_{i+1} \ldots w_{i+j-1}\} \quad (*)$$

Następująca obserwacja jest kluczowa dla algorytmu:

jeśli $Y \in V_{i,j}$, $Z \in V_{i+j,k}$ i $X \stackrel{G}{\to} YZ$, to $X \in V_{i,k}$.

Ostateczna odpowiedź na pytanie zależy oczywiście od tego, czy $X_I \in V_{1,n}$. Przyjmijmy oznaczenie

$$\mathit{Fusion}(V_{i,j}, V_{i+j,k}) = \{X \, : \, (\exists Y, Z) \, X \stackrel{G}{\to} YZ \, \wedge \, Y \in V_{i,j} \, \wedge \, Z \in V_{i+j,k}\}$$

Wówczas algorytm CYK można przedstawić następująco.

**Algorytm CYK**

**Dane:** Gramatyka $G$ w postaci Chomsky'ego, słowo $w = w_1 \ldots w_n$

1. Dla $i = 1, \ldots, n$, wykonuj : $V_{i,1} := \{X \, : \, X \stackrel{G}{\to} w_i\}$;
2. Dla $j = 2, \ldots, n$, wykonuj :
3. Dla $i = 1, \ldots, n + 1 - j$, wykonuj :
4. $V_{i,j} := \emptyset$;
5. Dla $k = 1, \ldots, j - 1$, wykonuj :
6. $V_{i,j} := V_{i,j} \cup \mathit{Fusion}(V_{i,k}, V_{i+k,j-k})$.

Nietrudno wykazać, że dla ustalonej gramatyki $G$, algorytm działa w czasie $O(n^3)$, $n = |w|$. Dalej, można zaobserwować, że po wykonaniu dla danego $j$ instrukcji 3–6, zawartość zmiennej $V_{i,j}$ (dla każdego $i$, takiego że $j \leq n + 1 - i$) spełnia warunek (*). Zatem $w \in L(G)$ wtedy i tylko wtedy, gdy po zakończeniu pracy algorytmu $X_I \in V_{1,n}$.

## ćwiczenia

1. Podać gramatyki bezkontekstowe generujące następujące języki:

   (a) zbiór słów nad alfabetem $\{a, b\}$, które zawierają tyle samo $a$ co $b$;

   (b) zbiór słów nad alfabetem $\{a, b\}$, które zawierają dwa razy więcej $a$ niż $b$;

   (c) zbiór wyrażeń arytmetycznych nad alfabetem $\{0, 1, (, ), +, \cdot\}$, które, przy zwykłej interpretacji działań dla liczb naturalnych, mają wartość 3;

   (d) zbiór wyrażeń arytmetycznych w notacji polskiej (nad alfabetem $\{0, 1, +, \cdot\}$) o wartości 4;

   (e) zbiór poprawnie zbudowanych formuł rachunku zdań ze zmiennymi zdaniowymi $p, q$ i stałymi logicznymi $\mathbf{0}$, $\mathbf{1}$ (alfabet: $\{p, q, \mathbf{0}, \mathbf{1}, \wedge, \vee, \Rightarrow, \neg\}$);

   (f) zbiór formuł rachunku zdań bez zmiennych, o wartości logicznej "prawda";

   (g) $\{a^i b^j c^k \, : \, i \neq j \, \vee \, j \neq k\}$;

   (h) $\{a^i b^j a^k \, : \, i + k = j\}$;

   (i) $\{a, b\}^* - \{ww \, : \, w \in \{a, b\}^*\}$.

2. Dla danych gramatyk bezkontekstowych $G, H$, skonstruować gramatyki generujące języki $L(G) \cup L(H)$, $L(G)L(H)$, $(L(G))^*$, $(L(G))^R$ (=lustrzane odbicie).

3. Dowieść, że następujące warunki są równoważne dla języka $L \subseteq \Sigma^*$:

(a) $L$ jest regularny,

(b) $L$ jest generowany przez gramatykę bezkontekstową, w której każda reguła jest postaci $X \to \epsilon$, $X \to Y$, lub $X \to \sigma Y$, $\sigma \in \Sigma$,

(c) $L$ jest generowany przez gramatykę bezkontekstową, w której każda reguła jest postaci $X \to \epsilon$, $X \to Y$, lub $X \to Y\sigma$, $\sigma \in \Sigma$,

(d) $L$ jest generowany przez gramatykę bezkontekstową, w której każda reguła jest postaci $X \to \alpha$ lub $X \to \beta Y$, $\alpha, \beta \in \Sigma^*$.

4. Podać przykład gramatyki bezkontekstowej w której każda reguła jest postaci $X \to \epsilon$, $X \to Y$, $X \to \sigma Y$ lub $X \to Y\sigma$, $\sigma \in \Sigma$, ale język generowany przez gramatykę nie jest regularny.

5. (*) Czy każdy język bezkontekstowy jest generowany przez gramatykę w postaci z poprzedniego zadania?

6. Dla dowolnej gramatyki $G$ oszacować z góry liczbę reguł gramatyki Chomsky'ego generującej język $L(G) - \{\epsilon\}$, jaką otrzymamy przez konstrukcje opisane w punkcie 2.2.

7. Niech $G$ będzie gramatyką bezkontekstową, z $m$ zmiennymi i niech, dla każdej reguły $Y \overset{G}{\to} w$, $|w| \leq \ell$. Dowieść, że jeśli $X_I \overset{G*}{\to} \epsilon$, to istnieje wyprowadzenie o długości $1 + \ell + \ell^2 + \ldots + \ell^{m-1}$. Czy to oszacowanie jest optymalne?

8. Dowieść, że dla każdej gramatyki $G$ istnieje stała $C$, taka, że dla dowolnego $w \neq \epsilon$, jeśli $X_I \overset{G*}{\to} w$, to istnieje wyprowadzenie o długości $\leq C \cdot |w|$.

9. Oszacować czas działania powyższego Algorytmu 6 (str. 9) oraz wynikającego zeń algorytmu rozstrzygającego, dla danej gramatyki $G$, czy $L(G) \neq \emptyset$.

10. Oszacować długość najkrótszego słowa generowanego przez gramatykę $G$

    (a) jeśli $G$ jest w postaci Chomsky'ego,

    (b) w dowolnym przypadku.

11. Zaprojektować algorytm, który, dla danej gramatyki $G$, odpowiada na pytanie, czy język $L(G)$ jest nieskończony.