Języki formalne i automaty
Ćwiczenia 2

Autor: Marcin Orchel

Spis treści

Wstęp teoretyczny

Jednym z pytań w teorii języków formalnych jest pytanie, jak sprawdzić czy dane słowo należy do danego języka. Na tych zajęciach będziemy rozważali to pytanie dla języków formalnych wyrażonych w postaci gramatyk. Sprawdzanie czy dane słowo należy do języka nazywamy *parsowaniem*.

Pierwszą zaprezentowaną metodą będzie metoda brute force.

Metoda brute force

Metoda brute force polega na sprawdzeniu wszystkich możliwych słów, jeśli dane słowo nie należy do języka, lub aż do momentu napotkania tego słowa, jeśli należy do tego języka.

Algorytm parsowania brute force:
Lista aktualnych łańcuchów na początku ustawiona jest na (S).

1. Wybierz produkcje, które można zastosować do poszczególnych łańcuchów z aktualnej listy łańcuchów.
2. Zastosuj odpowiednie produkcje do wszystkich elementów z listy aktualnych łańcuchów.
3. Uaktualnij listę łańcuchów.
4. Sprawdź czy szukane słowo znajduje się na liście aktualnych łańcuchów. Jeśli tak to przerwij działanie programu, jeśli nie to przejdź do kroku 1.

Przykład:

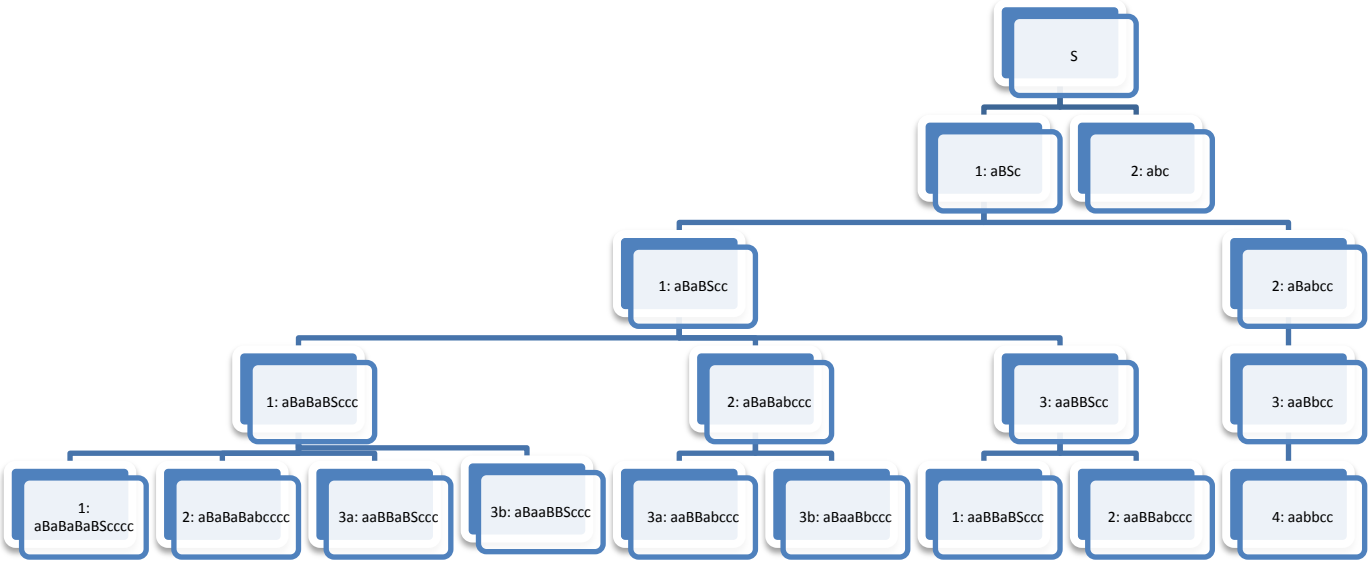
Gramatyka:

1. $S \rightarrow aBSc$
2. $S \rightarrow abc$
3. $Ba \rightarrow aB$
4. $Bb \rightarrow bb$

Pytamy czy słowo aabbcc należy do tego języka generowanego przez powyższą gramatykę. Kolejne listy łańcuchów dla algorytmu brute force będą wyglądały następująco:

(S),
(Prod 1 od S : aBSc, Prod 2 od S : abc),
(Prod 1 od aBSc : aBaBScc, Prod 2 od aBSc : aBabcc),
(Prod 1 od aBaBScc : aBaBaBSccc, Prod 2 od aBaBScc : aBaBabccc, Prod 3 od aBaBScc : aaBBScc, Prod 3 od aBabcc: aaBbcc),
(Prod 1 od aBaBaBSccc : aBaBaBaBScccc, Prod 2 od aBaBaBSccc : aBaBaBabcccc, Prod 3a od aBaBaBSccc : aaBBaBSccc, Prod 3b od aBaBaBSccc : aBaaBBSccc, Prod 3a od aBaBabccc: aaBBabccc, Prod 3b od aBaBabccc: aBaaBbccc, Prod 1 od aaBBScc: aaBBaBSccc, Prod 2 od aaBBScc: aaBBabccc, Prod 4 od aaBbcc: aabbcc). W aktualnej liście łańcuchów znajduje się szukane słowo aabbcc, a więc słowo to należy do języka.

Lista generowanych łańcuchów dla podanego przykładu może być zobrazowana za pomocą drzewa:



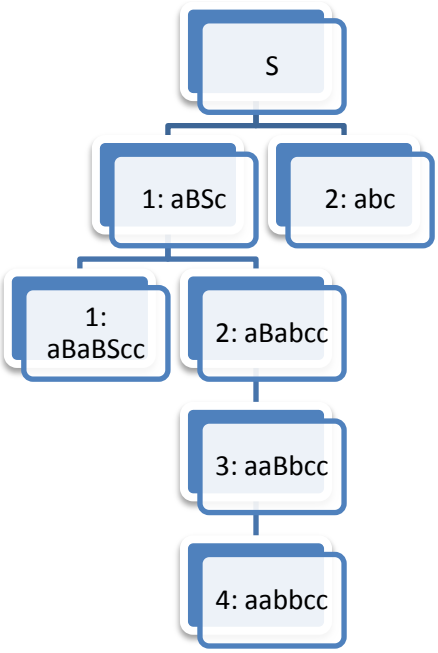
Powyższe drzewo w algorytmie brute force jest konstruowane począwszy od najwyższego poziomu i jednocześnie sprawdzane są nowo powstające poziomy, a więc przechodzenie przez drzewo jest typu *level-order*.

Wyróżniamy dwa rodzaje algorytmów parsowania: top-down parsing, oraz bottom-up parsing. W *top-down parsing* zaczynamy poszukiwania od symbolu startowego S, a w *bottom-up parsing* zaczynamy od poszukiwanego słowa i dążymy do symbolu startowego S.
Powyższy algorytm parsowania brute force jest algorytmem top-down parsing.

ambigous grammar – gramatyka, w której istnieje słowo, które może być wyprowadzone na kilka sposobów, to znaczy, że istnieją dwa różne ciągi produkcji prowadzące do tego słowa.

Problemem parsowania brute force jest to, że należy ominąć konieczność sprawdzenia nieskończonej liczby słów w przypadku, gdy słowo nie należy do języka. Najprostszym rozwiązaniem, ale nie dającym pewności, że słowo nie należy do języka jest możliwość zatrzymania programu przez użytkowania po pewnym czasie. Kolejnym rozwiązaniem jest podanie liczby poziomów, które należy sprawdzić. Innym rozwiązaniem dającym pewność czy dane słowo należy do języka po skończonej liczbie kroków jest przestanie rozwijania łańcuchów, które są dłuższe niż szukane słowo, algorytm ten będzie działał poprawnie dla gramatyk, które mają właściwość taką, że z każdej produkcji może powstać łańcuch tylko o tej samej liczbie symboli lub większej oraz taką, że nie istnieje nieskończenie wiele produkcji, które nie zmieniają długości łańcucha. Takimi gramatykami są np. gramatyki kontekstowe bez produkcji łańcuchowych generujące język bez słowa pustego, gramatyki bezkontekstowe bez ε-produkcji i bez produkcji łańcuchowych, gramatyki regularne.

Podana wyżej gramatyka przykładowa jest gramatyką kontekstową, ponadto posiada własności opisane wyżej. Dlatego można zastosować udoskonalenie dla podanego algorytmu takie, że gałąź dla której aktualny łańcuch przekracza długość szukanego słowa zostaje porzucona. Po tej modyfikacji lista generowanych łańcuchów dla podanego przykładu będzie wyglądała następująco:

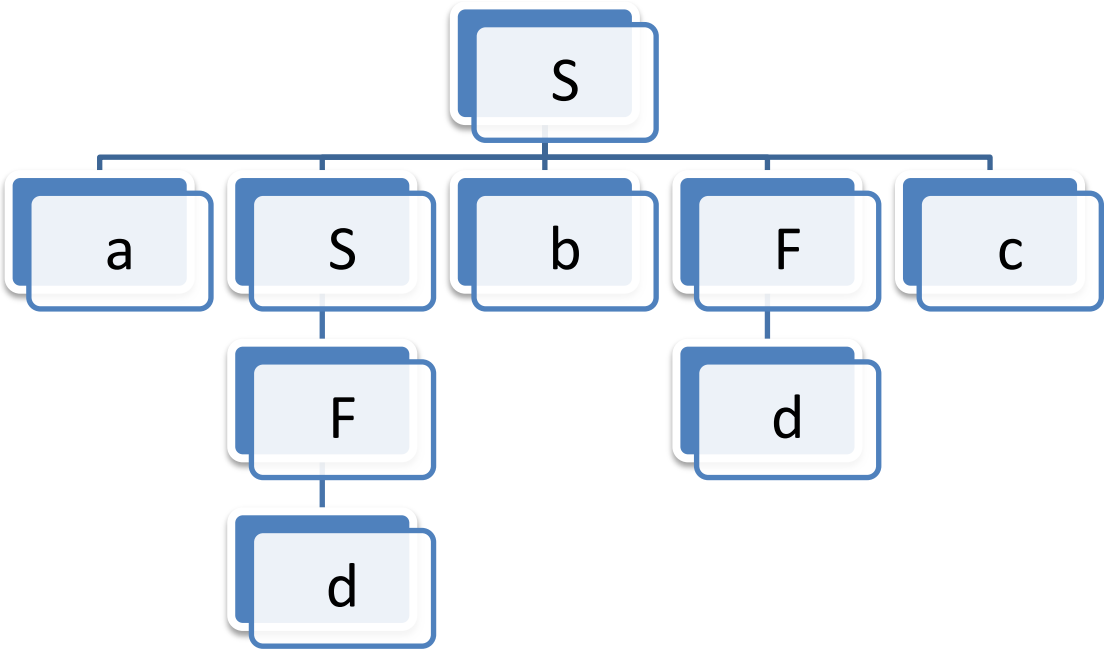


Wynikiem algorytmu parsowania jest nie tylko informacja o tym czy dane słowo należy do danego języka, ale również informacja o *wyprowadzeniu* tego słowa, czyli o tym, jakie należy zastosować produkcje i w jaki sposób, aby otrzymać dane słowo. W rozważanym przykładzie są to produkcje (1,2,3,4), zastosowane w następujący sposób:
$S \Rightarrow_1 aB\mathbf{S}c \Rightarrow_2 a\mathbf{Ba}bcc \Rightarrow_3 aa\mathbf{Bb}cc \Rightarrow_4 aabbcc$.

Każde wyprowadzenie dla gramatyki bezkontekstowej można przedstawić za pomocą *drzewa rozbioru*, w którym wewnętrznymi węzłami są nieterminale, a zewnętrznymi terminale. Przykładowo dla gramatyki bezkontekstowej

1. $S \rightarrow F$
2. $S \rightarrow aSbFc$
3. $F \rightarrow d$

wyprowadzenie dla słowa adbdc wygląda następująco:
$\mathbf{S} \Rightarrow_2 a\mathbf{S}bFc \Rightarrow_1 a\mathbf{F}bFc \Rightarrow_3 adb\mathbf{F}c \Rightarrow_3 adbdc$. Drzewo rozbioru wygląda następująco:



Konwersja do postaci normalnej Chomskiego

Gramatyka jest w postaci normalnej Chomskiego, kiedy produkcje są postaci:

$A \rightarrow BC$
$A \rightarrow \alpha$
$S \rightarrow \varepsilon$
gdzie A, B, C to symbole nieterminalne, α jest terminalem.

Można zauważyć, że każda gramatyka w postaci normalnej Chomskiego jest gramatyką bezkontekstową.

Można udowodnić również, że każda gramatyka bezkontekstowa może być przekształcona do postaci normalnej Chomskiego.

Przy wyprowadzaniu jakiegoś słowa każdy kolejny łańcuch jest takiej samej długości, lub o jeden symbol dłuższy, przy pominięciu ostatniej produkcji.

Każde wyprowadzenie słowa o długości *n* składa się z $2n$-1 produkcji.
Dowód: Jeśli słowo ma długość n to potrzebujemy zastosować produkcje typu 1 dokładnie n – 1 razy, a następnie produkcje typu 2 dokładnie n razy.

Drzewo rozbioru jest drzewem binarnym, o wysokości co najwyżej *n*.

Algorytm konwersji gramatyki bezkontekstowej do postaci normalnej Chomskiego.

Na wejściu mamy gramatykę bezkontekstową, bez ε-produkcji i bez produkcji łańcuchowych.
Do zbioru nowych produkcji dodajemy wszystkie poprawne produkcje.
Dla każdej pozostałej produkcji, która jest postaci $A \rightarrow X_1X_2...X_n$, gdzie $n \geq 2$, X_i jest terminalem lub nieterminalem:

1. Dla każdego *i* od 1 do *n*
2. Jeśli $X_i = x_i$ jest terminalem to wtedy:
   dołączamy do zbioru nieterminali nieterminal C_{xi},
   dołączamy do zbioru produkcji produkcje $C_{xi} \rightarrow x_i$,
   $B_i = C_{xi}$.
3. Jeśli X_i jest nieterminalem to wtedy $B_i = X_i$.
4. Koniec pętli *i*
5. Jeśli *n* = 2, to produkcje postaci $A \rightarrow B_1B_2$ dodajemy do zbioru nowych produkcji
6. Jeśli n > 2, to:
   do zbioru nieterminali dodajemy: D_1, D_2, …, D_{n-2},
   do zbioru produkcji dodajemy: $A \rightarrow B_1D_1$, $D_1 \rightarrow B_2D_2$, … $D_{n-3} \rightarrow B_{n-2}D_{n-2}$, $D_{n-2} \rightarrow B_{n-1}B_n$.

Przykład:

Dana jest gramatyka postaci:
$S \rightarrow bA \mid aB$
$A \rightarrow bAA \mid aS \mid a$
$B \rightarrow aBB \mid bS \mid b$

Dana gramatyka nie zawiera ε-produkcji ani produkcji łańcuchowych. Poprawne produkcje:
$A \rightarrow a$
$B \rightarrow b$
Wprowadzamy nieterminale: C_b oraz C_a i otrzymujemy:
$S \rightarrow C_bA \mid C_aB$
$A \rightarrow C_bAA \mid C_aS \mid a$
$B \rightarrow C_aBB \mid C_bS \mid b$
$C_a \rightarrow a$
$C_b \rightarrow b$

Produkcję $A \rightarrow C_bAA$ zastępujemy:

$A \rightarrow C_bD_1$
$D_1 \rightarrow AA$

Produkcję $B \rightarrow C_aBB$ zastępujemy:

$B \rightarrow C_aD_2$
$D_2 \rightarrow BB$

i otrzymujemy:
$S \rightarrow C_bA \mid C_aB$
$A \rightarrow C_bD_1 \mid C_aS \mid a$
$B \rightarrow C_aD_2 \mid C_bS \mid b$
$C_a \rightarrow a$
$C_b \rightarrow b$
$D_1 \rightarrow AA$
$D_2 \rightarrow BB$

Algorytm Cocke'a-Youngera-Kasamiego (CYK).

Algorytm sprawdza czy dane słowo należy do języka generowanego przez gramatykę podaną w postaci normalnej Chomskiego. Złożoność algorytmu CYK wynosi $O(n^3)$.

Sprawdzamy czy łańcuch x o długości $n > 0$ należy do języka. Algorytm polega na wyznaczeniu wszystkich wyprowadzeń postaci $A \Rightarrow^* x_{i,j}$, gdzie A to dowolny nieterminal, $x_{i,j}$ to dowolny podłańcuch o długości j, rozpoczynający się od i-tej pozycji łańcucha x. Wyprowadzenia wyznaczamy począwszy od wyprowadzeń podłańcuchów o długości 1, później 2 itd. Dla podłańcuchów o określonej długości wyznaczamy wyprowadzenia dla podłańcuchów rozpoczynających się od kolejnych symboli sprawdzanego słowa. Przykładowo dla j = 1 sprawdzamy czy istnieją wyprowadzenia postaci $A \Rightarrow^* x_{i,1}$. Wyprowadzenie takie istnieje, jeśli istnieje produkcja postaci $A \rightarrow x_{i,1}$. Dla j = 2 sprawdzamy czy istnieje wyprowadzenie postaci $A \Rightarrow^* x_{i,2}$. Wyprowadzenie to możemy zapisać w postaci:

$$A \Rightarrow^* x_{i,1}\, x_{i+1,1},$$

które można rozpisać na:

$$A \Rightarrow BC \Rightarrow x_{i,1}\, x_{i+1,1}.$$

Takie wyprowadzenie istnieje, jeśli istnieją produkcje $A \Rightarrow BC$ oraz $B \Rightarrow x_{i,1}$ oraz $C \Rightarrow x_{i+1,1}$. Dla j = 3 sprawdzamy czy istnieją wyprowadzenia postaci $A \Rightarrow^* x_{i,3}$. Wyprowadzenie takie istnieje, jeśli istnieją wyprowadzenia:

$$A \Rightarrow BC,\ B \Rightarrow^* x_{i,2},\ C \Rightarrow^* x_{i+2,1} \text{ lub}$$

$$A \Rightarrow BC,\ B \Rightarrow^* x_{i,1},\ C \Rightarrow^* x_{i+1,2}$$

Wyprowadzenia drugie i trzecie w obydwu alternatywach były już rozpatrowane w trakcie analizy przypadku gdy j = 2. Dla j = n jeśli w wyprowadzeniu znajdzie się produkcja postaci $S \Rightarrow BC$ to słowo należy do języka generowanego przez daną gramatykę.

Przykład:

Gramatyka postaci:

1. $S \rightarrow AB$
2. $S \rightarrow BC$
3. $A \rightarrow BA$
4. $A \rightarrow a$
5. $B \rightarrow CC$
6. $B \rightarrow b$
7. $C \rightarrow AB$
8. $C \rightarrow a$

Sprawdzamy łańcuch wejściowy: aabbab.
W celu zastosowania algorytmu CYK konstruujemy tabelę w której każdej komórce odpowiada podłańcuch o początku i i długości j. Wiersze odpowiadają długościom podłańcuchów, a kolumny

początkom podłańcuchów. W każdym polu wpisujemy nieterminale z których można wyprowadzić dany podłańcuch.

→ i / j	1	2	3	4	5	6
1	A,C	A,C	B	B	A,C	B
2	B	S, C	∅	A,S	S,C	x
3	B	∅	A	S,C	x	x
4	∅	∅	S,C	x	x	x
5	A	B	x	x	x	x
6	S,C	x	x	x	x	x

x oznacza, ze nie istnieje łańcuch o podanej długości i początku, np. nie istnieje łańcuch w tym przykładzie o długości 2 zaczynający się na ostatniej pozycji w rozpatrywanym słowie.

Na początku wypełniamy pole [1,1]. Polu temu odpowiada podłańcuch a. Istnieją dwie produkcje z których można wyprowadzić dany podłańcuch: produkcje 4 i 8. Po wypełnieniu pierwszego wiersza zaczynamy wypełniać wiersz 2.
Polu [2, 1] odpowiada podłańcuch aa. Sprawdzamy czy istnieje produkcja postaci ? → [1, 1][1, 2]. A zatem czy istnieje produkcja postaci ? → AA lub ? → AC lub ? → CA lub ? → CC. Istniejącą produkcją jest produkcja 5. A zatem do komórki [2, 1] wpisujemy nieterminal B. Po wypełnieniu drugiego wiersza zaczynamy wypełniać wiersz 3.
Polu [3, 1] odpowiada podłańcuch aab. Szukamy produkcji postaci ? → [1, 1][2, 2] lub ? → [2, 1][1, 3]. Istniejąca produkcja to produkcja 5. Itd.
W polu [6, 1] jest nieterminal S, a zatem dane słowo należy do języka.
Wyprowadzenie możemy otrzymać przechodząc przez tabelkę od końca.

Referencje

[1] http://kompilatory.agh.edu.pl/pages/ta-wyklady/3-6-wlasciwosci-jezykow-bezkontekstowych.htm

Zadania

Zadanie na 3.0

Zapoznać się z wpisywaniem gramatyk do programu JFLAP.

Zadanie 6.16 ze strony http://kompilatory.agh.edu.pl/pages/ta-zadania/%20Zadania06-przeksztalcenia-gramatyk.htm

Zadanie 7.15 ze strony http://kompilatory.agh.edu.pl/pages/ta-zadania/%20Zadania07-wlasnosci-jbezkontekstowych.htm
Wybrać kilka komórek tabeli CYK i opisać ich powstawanie.

Dodatek: Dla gramatyki i słowa z zadania 7.15 narysować drzewa rozbioru dla algorytmu parsowania CYK.

Zadanie na 4.0

Implementacja parsera brute force w Javie ze sprawdzaniem długości łańcucha.
Dodatek: Porównać wyniki tego algorytmu z wynikami algorytmu CYK dla gramatyki z zadania 7.15.

Zadanie na 5.0

Implementacja w Javie konwersji do postaci normalnej Chomskiego i algorytmu CYK.
Dodatek: Porównać wyniki z konwersją do postaci normalnej za pomocą programu JFLAP dla gramatyki z zadania 6.16. Porównać wyniki algorytmu CYK z wynikami z programu JFLAP dla gramatyki z zadania 7.15.

Dodatki są nieobowiązkowe.