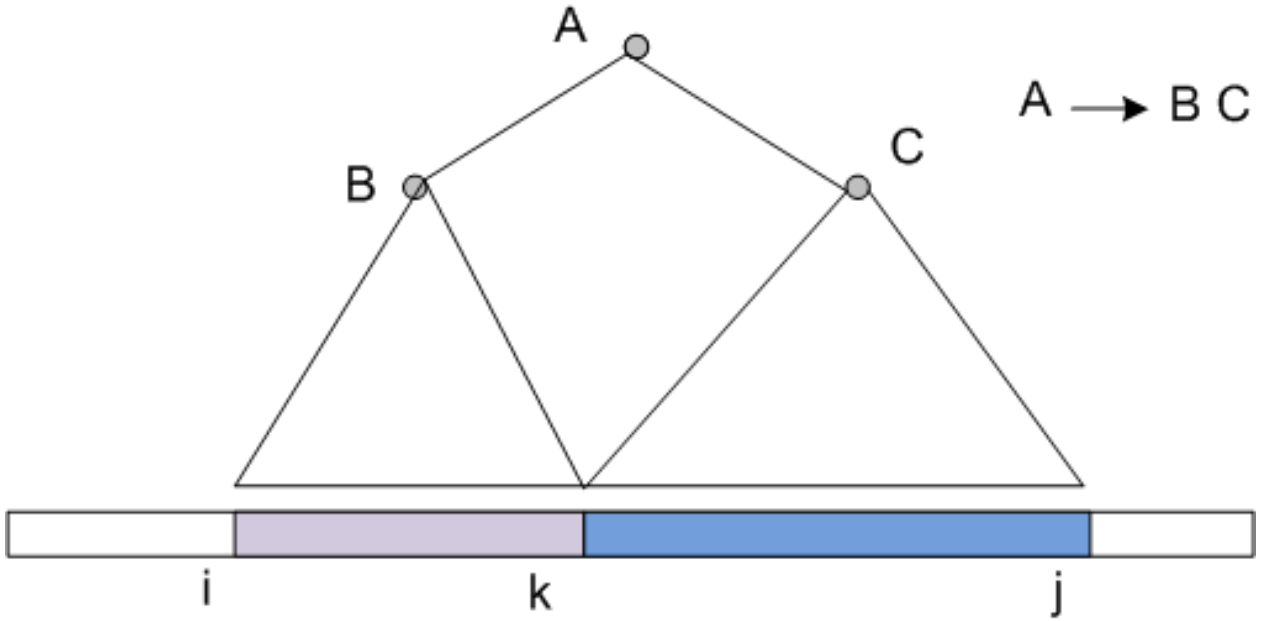
Algorytmy zwiazane z gramatykami bezkontekstowymi

Problem rozpoznawania języka L polega na sprawdzaniu przynależności słowa wejściowego x do L. Zakładamy, że język ma opis stałego rozmiaru, a rozmiarem wejścia jest $n = |x|$.
Niech G będzie gramatyką bezkontekstową w postaci normalnej Chomsky'ego oraz $x = a_1 a_2 a_n$ dane słowo wejściowe.
Niech $x(i..j]$ oznacza podsłowo $a_{i+1} \ldots a_j$. Zakładamy tutaj, że $i < j$. W szczególności $x(i..i+1]$ jest pojedyńczą literą.

Definicja

Trójka (A, i, j) jest poprawna, gdy istnieje wyprowadzenie $A \rightarrow^* x(i..j]$.

Niech $Valid(G, x)$ oznacza zbiór poprawnych trójek.



Następujący fakt wynika z dekompozycji drzewa wyprowadzenia podsłowa $a_{i+1}a_{i+2}\ldots a_j$ na dwa poddrzewa odpowiadające wyprowadzeniom krótszych podsłów.

Fakt

Dla $j - i > 1$ mamy $(A, i, j) \in Valid(G, x) \Leftrightarrow$
$\exists\, k, B, C\ (B, i, k) \in Valid(G, x)\ \&\ (C, k, j) \in Valid(G, x)\ \&\ A \to BC$

$$x \in L(G) \ \Leftrightarrow\ (S, 0, n) \in Valid(G, x)$$

Pierwszy algorytm spawdzania przynależności x oblicza $V = Val(G, x)$ inkrementalnie następująco:

Algorytm

Sprawdź czy $x \in L(G)$.

$V := \{(A, i, i+1) \ : \ A \to a_{i+1}\}$;

while V się nie stabilizuje **do**

dodaj do V wszystkie pary (A, i, j) takie, że

$(\exists\, i < k < j\,)\ A \to BC \ \&\ (B, i, k) \in V \ \&\ (C, k, j) \in V$;

if $(S, 0, n) \in V$ **then return** *true* **else return** *false*.

Policzmy tablicę T, gdzie $T[i,j]$ jest zbiorem A takich, że $(A,i,j) \in Val(G,x)$. Dla zbiorów nieterminali definiujemy operację:

$$X \otimes Y = \{A : A \to BC, \text{ dla pewnych } B \in X,\ C \in Y.$$

Algorytm

Youngera (sprawdza czy $x \in L(G)$)

for $i := 0$ **to** $n-1$ **do** $T[i,i+1] :=:= \{(\mathrm{A} : \mathrm{A} \to a_{i+1}\}$;

for $j := 0$ **to** n **do**

  for $i := j-1$ **downto** 0 **do**

    $T[i,j] := \bigcup_{i<k<j} T[i,k] \otimes T[k,j]$;

if $(S,0,n) \in V$ **then return** *true* **else return** *false*.

Rozważmy gramatykę G: $S \to SS \mid AA \mid b$; $A \to AS \mid AA \mid a$ i słowo $x = a\,a\,b\,b$. Wtedy tablica T wygląda następująco:

$$\begin{bmatrix} \emptyset & \{A\} & \{S,A\} & \{S,A\} & \{A,S\} \\ \emptyset & \emptyset & \{A\} & \{A\} & \{A\} \\ \emptyset & \emptyset & \emptyset & \{S\} & \{S\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{S\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

W tym przypadku $n = 4$ oraz $S \in T[0,4]$, tak więc $aabb \in L(G)$.

Załóżmy, że gramatyka jest jednoznaczna i nie ma zbędnych nieterminali. Dla każdego nieterminal A oraz numeru kolumny j zdefiniujmy: $LISTA(A,j) = \{ i : A \in T[i,j] \}$.

Obserwacja. Jeśli znamy j-tą kolumnę tablicy T to $LISTA(A,j)$ dla wszystkich A (stała liczba) możemy policzyć w czasie liniowym.

W algorytmie Youngera najbardziej kosztowną jest operacja

$$T[i,j] := \bigcup_{i<k<j} T[i,k] \otimes T[k,j];$$

Teraz zamieniamy tę operację "na drobne", tak aby koszt każdej jednostkowej operacji wstawiania do $T[i,j]$ był proporcjonalny do tworzenia jednego węzła w drzewie generacji dla pewnego podsłowa. Ponieważ z powodu jednoznaczności takich węzłów jest tylko $O(n^2)$ zmodyfikowany algorytm działa w czasie $O(n^2)$ podczas gdy algorytm Youngera działa ogólnie w czasie $O(n^3)$.

Algorytm Youngera dla gramatyk jednoznacznych

for $i := 0$ **to** $n-1$ **do** $T[i, i+1] :=:= \{(\mathrm{A} : \ \mathrm{A} \rightarrow a_{i+1}\}$;

Oblicz $Lista(A, 1)$ dla każdego nieterminala A;

for $j := 0$ **to** n **do**

for $k := j-1$ **downto** 0 **do**

for each A, B, C takich, że $A \rightarrow BC$, $C \in T[k, j]$ **do**

for each $i \in LISTA(B, k)$ **do**

dodaj A do $T[i, j]$;

Oblicz $LISTA(A, j)$ dla każdego nieterminala A;

{kolumna j-ta tablicy T jest już policzona}

if $(S, 0, n) \in V$ **then return** *true* **else return** *false*.

Lemat

Jeśli znamy tablicę T to możemy skonstruować drzewo wyprowadzenia słowa (o ile istnieje) w czasie $O(n^2)$.

Cofamy się w tablicy T, znajdujemy takie k, że $S \in T[0, n]$ gdyż $S \to XY$ oraz $X \in T[i, k]$, $Y \in T[k, j]$. W ten sposób znajdujemy synów dla $(A, 0, n)$ w drzewie.

Podobnie znajdujemy synów dla węzła (A, i, j) rozbijając (A, i, j) na (B, i, k) i (C, k, j), gdzie $A \to BC$.

Dla każdego węła przeglądamy $O(n)$ elementów tablicy żeby znaleźć odpowiednie k, w sumie wykonujemy pracę $O(n^2)$ gdyż węzłów w drzewie wyprowadzenia jest $O(n)$.

Gramatyki atrybutywne

Gramatyka S-atrybutywne (litera S od *syntezowane atrybuty*) w czasie generacji drzewa wyprowadzenia liczy w tym drzewie pewne wartości (atrybuty) skojarzone z wierzchołkami.

Obliczenia są wykonywane w kolejności "bottom-up" (wartości ojca są liczone za pomocą wartości synów).

Przez $$ oznaczamy atrybut aktualnego węzła, a przez $$_i$ atrybut skojarzony z i-tym synem (i-tym elementem po prawej stronie produkcji).

Gramatyka musi być jednoznaczna, tak aby było tylko jedno drzewo wyprowadzenia i jednoznacznie zdefiniowane wartości atrybutów.

Przykład gramatyki jednoznaczej (wyrażenia arytmetyczne).

Załóżmy dla uproszczenia, że mamy tylko dwie operacje $+$, $*$, oraz stałe a, b.

Wtedy gramatyką jednoznaczną dla wyrażeń jest:

$$E \to E + T \mid T;$$

$$T \to T * F \mid F$$

$$F \to a \mid b \mid (E).$$

Gramatyka atrybutywna dla wyrażeń

W liściach drzewa mamy stałe a, b, które mają pewne określone wartości (z wejścia). Możemy skojarzyć z nieterminalami atrybut określający wartość wyrażenia.

$$E \to E + T \quad \$\$:= \$1 + \$_3$$

$$E \to T \quad \$\$:= \$1$$

$$T \to T * F \quad \$\$:= \$1 * \$_3$$

$$T \to F \quad \$\$:= \$1$$

$$F \to a \quad \$\$:= \$_1$$

$$F \to b \quad \$\$:= \$_1$$

$$F \to (E) \quad \$\$:= \$_2$$