El algoritmo CYK (Cocke-Younger-Kasami) es un algoritmo utilizado para determinar si una cadena está en el lenguaje generado por una gramática libre de contexto en forma normal de Chomsky. El algoritmo utiliza una tabla bidimensional para realizar una búsqueda en la estructura de la gramática y determinar si la cadena de entrada es generada por la gramática.

belongsInLanguage[w\_, G\_] := Module[{n, T, N, P, S, table},

n = StringLength[w];

T = Union[StringCases[w, \_]];

N = Union[Flatten[G[[All, 1]]]];

P = G[[All, 2]];

S = Select[G, #[[1]] == N[[1]] &][[1, 1]];

table = ConstantArray[False, {n, n, Length[N]}];

(\* Paso 2 \*)

Do[

If[StringLength[p[[1]]] == 1 && StringMatchQ[p[[2]], Alternatives @@ T],

table[[All, All, Position[N, p[[1]]][[1, 1]]]] = True

],

{p, P}

];

(\* Paso 3 \*)

Do[

Do[

Do[

If[StringLength[p[[2]]] == 2,

table[[i, j, k]] =

Or @@ Table[

table[[i, s, Position[N, p[[2, 1]]][[1, 1]]]] &&

table[[s + 1, j, Position[N, p[[2, 2]]][[1, 1]]]],

{s, i, j - 1}

];

],

{i, n},

{j, i, n},

{p, P}

];

(\* Paso 4 \*)

Return[table[[1, n, Position[N, S][[1, 1]]]]];

]

Claro, aquí te explico línea por línea:

css

belongsInLanguage[w\_, G\_] := Module[{n, T, N, P, S, table},

Esta línea define la función belongsInLanguage con dos argumentos de entrada w y G. Además, se crea una variable local de tipo Module llamada table que será utilizada para almacenar la tabla que se utiliza en el algoritmo CYK.

lua

n = StringLength[w];

T = Union[StringCases[w, \_]];

N = Union[Flatten[G[[All, 1]]]];

P = G[[All, 2]];

S = Select[G, #[[1]] == N[[1]] &][[1, 1]];

En estas líneas se definen las variables necesarias para ejecutar el algoritmo CYK. La variable n almacena la longitud de la cadena w. La variable T almacena el conjunto de símbolos terminales de la gramática. La variable N almacena el conjunto de símbolos no terminales de la gramática. La variable P almacena el conjunto de producciones de la gramática. La variable S almacena el símbolo inicial de la gramática.

mathematica

table = ConstantArray[False, {n, n, Length[N]}];

Esta línea crea una matriz tridimensional llamada table con tamaño n x n x Length[N], donde n es la longitud de la cadena w y Length[N] es el número de símbolos no terminales de la gramática. Cada celda de la matriz se inicializa con el valor False.

lua

Do[

If[StringLength[p[[1]]] == 1 && StringMatchQ[p[[2]], Alternatives @@ T],

table[[All, All, Position[N, p[[1]]][[1, 1]]]] = True

],

{p, P}];

En este bucle Do, se recorren todas las producciones p de la gramática y se verifica si la producción tiene la forma A -> a, donde A es un símbolo no terminal y a es un símbolo terminal. En ese caso, se marca la celda correspondiente en la tabla table como True.

lua

Do[

Do[

Do[

If[StringLength[p[[2]]] == 2,

table[[i, j, k]] = Or @@ Table[ table[[i, s, Position[N, p[[2, 1]]][[1, 1]]]] && table[[s + 1, j, Position[N, p[[2, 2]]][[1, 1]]]], {s, i, j - 1}]; ], {i, n}, {j, i, n}, {p, P} ];

En este bucle Do, se recorren todas las posibles subcadenas de la cadena w de longitud creciente y se verifican todas las posibles producciones p de la gramática que generan cada subcadena. Si se encuentra una producción p que genera la subcadena, se marca la celda correspondiente en la tabla table como True.

lua

Return[table[[1, n, Position[N, S][[1, 1]]]]]

Finalmente, se devuelve el valor de la celda correspondiente a la primera fila, última