**1**

Para resolver este problema, podemos seguir los siguientes pasos:

1. Definir una función que tome una gramática incontextual y devuelva una lista de todas las producciones que generan cadenas de símbolos terminales (incluyendo la cadena vacía). Esto se puede lograr buscando todas las producciones que tengan solamente símbolos terminales en su lado derecho, o que tengan un lado derecho que sea una combinación de símbolos no terminales y símbolos generativos.
2. A continuación, podemos crear una lista de todos los símbolos auxiliares en la gramática.
3. Luego, podemos iterar sobre cada símbolo auxiliar en la lista y verificar si alguna de sus producciones se encuentra en la lista de producciones generativas obtenida en el paso 1. Si es así, agregamos el símbolo auxiliar a una lista de símbolos auxiliares directamente generativos.
4. Finalmente, devolvemos la lista de símbolos auxiliares directamente generativos.

A continuación, se muestra una posible implementación en Mathematica de esta solución:

mathematica

directlyGenerativeSymbols[grammar\_] := Module[{generativeProductions, allAuxiliarySymbols, directlyGenerativeSymbols},

(\* Paso 1: obtener la lista de producciones generativas \*)

generativeProductions = Select[Flatten[ReplaceList[grammar,

(lhs\_ -> rhs\_) :> If[StringMatchQ[rhs, (Except[NonWordCharacter]..)], lhs -> rhs, Nothing], Infinity]],

StringMatchQ[#, (Except[NonWordCharacter]..)] &];

(\* Paso 2: obtener la lista de todos los símbolos auxiliares \*)

allAuxiliarySymbols = Select[Union[Flatten[ReplaceList[grammar,

(lhs\_ -> rhs\_) :> If[StringMatchQ[lhs, CapitalLetterCharacter..], lhs, Nothing], Infinity]]],

StringMatchQ[#, CapitalLetterCharacter] &];

(\* Paso 3: encontrar los símbolos auxiliares directamente generativos \*)

directlyGenerativeSymbols = Select[allAuxiliarySymbols, MemberQ[generativeProductions, # -> \_] &];

(\* Paso 4: devolver la lista de símbolos auxiliares directamente generativos \*)

directlyGenerativeSymbols

]

Esta función toma como entrada una gramática incontextual en forma de lista de producciones, donde cada producción está representada como una regla de la forma lhs -> rhs, donde lhs es el símbolo no terminal del lado izquierdo y rhs es la cadena de símbolos del lado derecho. Por ejemplo:

rust

grammar = {"S" -> "aS", "S" -> "B", "B" -> "bC", "C" -> "", "C" -> "d"};

Luego, podemos llamar a la función directlyGenerativeSymbols con esta gramática como argumento:

css

directlyGenerativeSymbols[grammar]

Esto devolverá una lista con los símbolos auxiliares directamente generativos de la gramática. Por ejemplo, para la gramática de ejemplo, la salida sería:

arduino

{"B"}

Esto indica que el símbolo auxiliar B es directamente generativo, ya que una de sus producciones (B -> bC) genera una cadena de símbolos terminales (en este

**2**

Para resolver este problema, podemos seguir los siguientes pasos:

1. Definir una función que tome una gramática incontextual y devuelva una lista de todos los símbolos no terminales que aparecen en el lado derecho de alguna producción.
2. A continuación, podemos crear una lista de todos los símbolos auxiliares en la gramática.
3. Luego, podemos iterar sobre cada símbolo auxiliar en la lista y verificar si aparece en algún consecuente de alguna producción donde también aparece como antecedente. Si es así, agregamos el símbolo auxiliar a una lista de símbolos auxiliares directamente no generativos.
4. Finalmente, devolvemos la lista de símbolos auxiliares directamente no generativos.

A continuación, se muestra una posible implementación en Mathematica de esta solución:

mathematica

directlyNonGenerativeSymbols[grammar\_] := Module[{nonTerminalSymbols, allAuxiliarySymbols, directlyNonGenerativeSymbols},

(\* Paso 1: obtener la lista de símbolos no terminales en el lado derecho de alguna producción \*)

nonTerminalSymbols = Select[Flatten[ReplaceList[grammar,

(lhs\_ -> rhs\_) :> If[StringMatchQ[rhs, (WordCharacter ~~ CapitalLetterCharacter..)], StringCases[rhs, CapitalLetterCharacter..], Nothing], Infinity]],

StringMatchQ[#, CapitalLetterCharacter] &];

(\* Paso 2: obtener la lista de todos los símbolos auxiliares \*)

allAuxiliarySymbols = Select[Union[Flatten[ReplaceList[grammar,

(lhs\_ -> rhs\_) :> If[StringMatchQ[lhs, CapitalLetterCharacter..], lhs, Nothing], Infinity]]],

StringMatchQ[#, CapitalLetterCharacter] &];

(\* Paso 3: encontrar los símbolos auxiliares directamente no generativos \*)

directlyNonGenerativeSymbols = Select[allAuxiliarySymbols,

AnyTrue[ReplaceList[grammar, (lhs\_ -> rhs\_) :> If[lhs == #, rhs, Nothing]],

MemberQ[#, #] &] &];

(\* Paso 4: devolver la lista de símbolos auxiliares directamente no generativos \*)

directlyNonGenerativeSymbols

]

Esta función toma como entrada una gramática incontextual en forma de lista de producciones, donde cada producción está representada como una regla de la forma lhs -> rhs, donde lhs es el símbolo no terminal del lado izquierdo y rhs es la cadena de símbolos del lado derecho. Por ejemplo:

rust

grammar = {"S" -> "AB", "A" -> "aA", "A" -> "", "B" -> "bBc", "B" -> "D", "D" -> "dDd", "D" -> ""};

Luego, podemos llamar a la función directlyNonGenerativeSymbols con esta gramática como argumento:

css

directlyNonGenerativeSymbols[grammar]

Esto devolverá una lista con los símbolos auxiliares directamente no generativos de la gramática. Por ejemplo, para la gramática de ejemplo, la salida sería:

arduino

{"A", "D"}

Esto indica que los símbolos auxiliares A y D son directamente no generativos, ya que aparecen en algunas producciones donde también aparecen en el consecuente.

**3**

Para verificar si una gramática incontextual está en Forma Normal de Greibach, debemos comprobar si todas sus producciones tienen la forma A → aβ, donde A es un símbolo no terminal, a es un símbolo terminal, y β es una cadena de símbolos no terminales.

Podemos implementar esta comprobación en Mathematica siguiendo los siguientes pasos:

1. Definir una función que tome una gramática incontextual y devuelva True si todas sus producciones tienen la forma A → aβ, y False en caso contrario.
2. Para cada producción de la gramática, comprobar si su lado izquierdo es un símbolo no terminal, si su lado derecho tiene la forma aβ, donde a es un símbolo terminal y β es una cadena de símbolos no terminales, y si no hay más símbolos terminales a la derecha de la primera aparición de un símbolo no terminal en el lado derecho.
3. Si todas las producciones cumplen estas condiciones, entonces la gramática está en Forma Normal de Greibach y la función devolverá True. De lo contrario, devolverá False.

A continuación, se muestra una posible implementación en Mathematica de esta solución:

isGreibachNormalForm[grammar\_] := AllTrue[grammar,

Function[{rule},

StringMatchQ[rule[[1]], CapitalLetterCharacter] &&

StringMatchQ[rule[[2]], (CharacterRange["a", "z"].. ~~ CapitalLetterCharacter..)] &&

And @@ (Not@\*StringMatchQ[#, CharacterRange["a", "z"]..] & /@ Rest[StringCases[rule[[2]], CapitalLetterCharacter..]])

]

]

Esta función toma como entrada una gramática incontextual en forma de lista de producciones, donde cada producción está representada como una regla de la forma lhs -> rhs, donde lhs es el símbolo no terminal del lado izquierdo y rhs es la cadena de símbolos del lado derecho. Por ejemplo:

grammar = {"S" -> "aA", "A" -> "bB", "B" -> "cC", "C" -> "dD", "D" -> ""};

Luego, podemos llamar a la función isGreibachNormalForm con esta gramática como argumento:

isGreibachNormalForm[grammar]

Esto devolverá True si la gramática está en Forma Normal de Greibach, o False en caso contrario. Por ejemplo, para la gramática de ejemplo, la salida sería True, ya que todas sus producciones tienen la forma A → aβ.

**4**

Para obtener una gramática equivalente lineal por la derecha a partir de una gramática en la forma indicada, podemos seguir los siguientes pasos:

1. Definir una función que tome una gramática en la forma indicada como entrada y devuelva una lista de reglas que representen la gramática equivalente lineal por la derecha.
2. Para cada producción de la gramática, comprobar si su lado derecho tiene la forma αB o α, donde α es una cadena de símbolos terminales y B es un símbolo no terminal. Si el lado derecho tiene la forma αB, crear una nueva producción que tenga el mismo símbolo no terminal en el lado izquierdo y la cadena α seguida del símbolo no terminal en el lado derecho. Si el lado derecho tiene la forma α, crear una nueva producción que tenga el mismo símbolo no terminal en el lado izquierdo y la cadena α como lado derecho.
3. Devolver la lista de reglas resultante como la gramática equivalente lineal por la derecha.

A continuación, se muestra una posible implementación en Mathematica de esta solución:

toRightLinear[grammar\_] :=

With[{rules = grammar /. (s\_String -> t\_String) :> ({s, t})},

Flatten[

Map[

If[StringMatchQ[#[[2]], (CharacterRange["a", "z"] .. ~~ CapitalLetterCharacter)],

{#[[1]], #[[2]]},

{#[[1]], StringTake[#[[2]], 1], StringDrop[#[[2]], 1]}

] &,

rules

],

1] /. (s\_String -> t\_String) :> s -> {t}

]

Esta función toma como entrada una gramática en la forma indicada como una lista de producciones, donde cada producción está representada como una regla de la forma lhs -> rhs. Por ejemplo:

grammar = {"S" -> "aB", "B" -> "bC", "C" -> "dD", "D" -> "eE", "E" -> ""};

Luego, podemos llamar a la función toRightLinear con esta gramática como argumento:

toRightLinear[grammar]

Esto devolverá la gramática equivalente lineal por la derecha como una lista de reglas. Por ejemplo, para la gramática de ejemplo, la salida sería:

{"S" -> {"a", "B"}, "B" -> {"b", "C"}, "C" -> {"d", "D"}, "D" -> {"e", "E"}, "E" -> {}, "S" -> {"a"}, "B" -> {"b"}, "C" -> {"d"}, "D" -> {"e"}}

En esta salida, las reglas adicionales corresponden a las producciones que tienen el lado derecho de la forma α, donde α es una cadena de símbolos terminales.