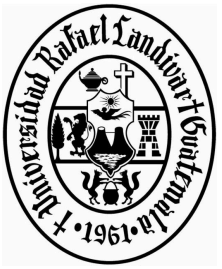
Universidad Rafael Landivar
Compiladores
Tarea 3
Tiempo límite: 18 de septiembre de 2024



Serie única - Gramáticas libres de contexto

Ejercicio 1:

1) Define una gramática que permita expresar condiciones con el operador ternario *condition* ? *value_if_true* : *value_if_false*

   Ejemplos de cadenas válidas:

   `x ? y : z`

   `(var1 ? 1 : 0) ? 2 : 3`

   `true ? 1 : (false ? 2 : 3)`

   Ejemplos de cadenas inválidas:

   `x ? y :` (falta la segunda parte del operador ternario)

   `? x : y` (falta la condición antes del `?`)

   `x ? : y` (falta la primera parte de la expresión ternaria)

2) Verifica si la gramática que has propuesto es **ambigua**. ¿Cómo podrías probarlo?
3) Evalúe con una cadena propuesta por usted utilizando **Recursive Descent Algorithm,** utilice backtracking si es necesario.
4) Examina si la gramática es **recursiva por la izquierda**. Si lo es, debe proponer las modificaciones necesarias para evitar la recursividad por la izquierda.

Ejercicio 2:

1) Define una gramática que permita construir expresiones lógicas usando operadores && (AND), || (OR), y ! (NOT).

   Ejemplos de cadenas válidas:

   `a && b`

   `!a || (b && c)`

   `!(a || b) && c`

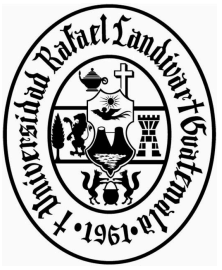
   Ejemplos de cadenas inválidas:

   `a && || b` (no se permite dos operadores consecutivos)

   `a && (b || )` (expresión incompleta dentro de los paréntesis)

   `! && a` (negación aplicada incorrectamente)

Universidad Rafael Landivar
Compiladores
Tarea 3
Tiempo límite: 18 de septiembre de 2024



2) Verifica si la gramática que has propuesto es **ambigua**. ¿Cómo podrías probarlo?
3) Evalúe con una cadena propuesta por usted utilizando **Recursive Descent Algorithm,** utilice backtracking si es necesario.
4) Examina si la gramática es **recursiva por la izquierda**. Si lo es, debe proponer las modificaciones necesarias para evitar la recursividad por la izquierda.

Ejercicio 3:

Define una gramática que permite construir expresiones que incluyan llamadas a funciones. Las funciones pueden tener argumentos, y las llamadas a funciones pueden estar anidadas dentro de otras expresiones.

```
func1(x, y)
func2(func1(x), y)
func3((x + y), func2(z))
```

Ejemplos de cadenas inválidas:

`func1(x, )` (falta un argumento después de la coma)

`func1(x, y` (falta el paréntesis de cierre)

`func1(x) y` (uso incorrecto fuera de la llamada a la función)

5) Verifica si la gramática que has propuesto es **ambigua**. ¿Cómo podrías probarlo?
6) Evalúe con una cadena propuesta por usted utilizando **Recursive Descent Algorithm,** utilice backtracking si es necesario.
7) Examina si la gramática es **recursiva por la izquierda**. Si lo es, debe proponer las modificaciones necesarias para evitar la recursividad por la izquierda.