Práctica 8. "IF"

Carmona Serrano Ian Carlo
Ingeniería en Inteligencia Artificial 5BM1
Teoría de la Computación , ESCOM- IPN

6 de Enero 2023

1 Introduction

El programa desarrollado es una herramienta para generar automáticamente gramáticas Backus-Naur (BNF) que definen la estructura de condicionales IF

El objetivo es permitir la generación controlada de estructuras IF utilizando la gramática BNF definida. El programa realiza derivaciones aleatorias hasta alcanzar un número especificado de condicionales IF, ya sea determinado por el usuario o de manera automática. Cada paso se registra en un archivo para revisión. Además, se genera pseudo-código basado en la cadena gramatical.

2 Marco Teórico

Las (BNF) son una notación formal utilizada para definir la estructura sintáctica de los lenguajes de programación. En el contexto de los condicionales IF, las gramáticas BNF proporcionan reglas formales que describen cómo se construyen estas sentencias.

La gramática BNF propuesta para el condicional IF se define como:

$$S \rightarrow iCtSA$$

$$A \rightarrow ; eS \mid \epsilon$$

Donde S representa la estructura base del condicional IF y A define posibles extensiones, permitiendo la finalización sin extensión (epsilon) o con una extensión adicional.

Estas gramáticas son esenciales para comprender y definir la sintaxis de los condicionales IF, proporcionando reglas formales que delimitan la composición y organización correcta de estas sentencias en el código de programación.

3 Desarrollo

La función principal del programa inicia con la solicitud de entrada al usuario para elegir entre un modo manual o automático. Este flujo de interacción con el usuario refleja la capacidad de definir el número de condicionales IF a generar, ajustándose a la esencia de las gramáticas BNF, donde se establecen reglas y posibles expansiones.

```
1) Manual
2) Automático
3) Salir
Ingrese su opción: 1

Número de ifs: 10
```

Figure 1: Consola

La función `procesar_if(n)` es el corazón del proceso de derivación. A partir de la gramática BNF definida, se realizan transformaciones sucesivas basadas en las reglas establecidas ($S \rightarrow iCtSA$, $A \rightarrow ; eS \mid \epsilon$). Estas transformaciones se aplican de manera controlada, reflejando la estructura recursiva y las posibles extensiones definidas en la gramática.

Además, el código genera dos archivos: uno para registrar las derivaciones de la gramática y otro para almacenar el pseudo-código generado a partir de la cadena gramatical.

```
1
2   if (CONDICIONAL):
3       if (CONDICIONAL):
4           if (CONDICIONAL):
5               if (CONDICIONAL):
6                   if (CONDICIONAL):
7                       if (CONDICIONAL):
8                           CODIGO
9       else
10          if (CONDICIONAL):
11              CODIGO
12      else
13          CODIGO
14      else
15          if (CONDICIONAL):
16              CODIGO
17          if (CONDICIONAL):
18          else
19              if (CONDICIONAL):
20                  CODIGO
21              else
22                  CODIGO
23              else
24                  CODIGO
```

Figure 2: pseudocodigo

```
S -> iCt(iCtSA)A
eS -> iCt(iCtSeS)A
eS -> iCt(iCtSeSeSA
S -> iCt(iCt(iCtSA)eSeSA
 -> iCt(iCt(iCtS)eSeSA
eS -> iCt(iCt(iCtS)eSeSAeS
S -> iCt(iCt(iCtS)e(iCtSA)eSAeS
 -> iCt(iCt(iCtS)e(iCtS)eSAeS
 -> iCt(iCt(iCtS)e(iCtS)eSAS
S -> iCt(iCt(iCtS)e(iCtS)eSA(iCtSA)
eS -> iCt(iCt(iCtS)e(iCtS)eSeS(iCtSA)
eS -> iCt(iCt(iCtS)e(iCtS)eSeS(iCteSA)
S -> iCt(iCt(iCt(iCtSA))e(iCtS)eSeS(iCteSA)
 -> iCt(iCt(iCt(iCtS))e(iCtS)eSeS(iCteSA)
 -> iCt(iCt(iCt(iCtS))e(iCtS)eSeS(iCteSA
S -> iCt(iCt(iCt(iCtS))e(iCtS)eSe(iCtSA)(iCteSA
 -> iCt(iCt(iCt(iCtS))e(iCtS)eSe(iCtS)(iCteSA
 -> iCt(iCt(iCt(iCtS))e(iCtS)eSe(iCtS)(iCteSA
S -> iCt(iCt(iCt(iCt(iCtSA)))e(iCtS)eSe(iCtS)(iCteSA
 -> iCt(iCt(iCt(iCt(iCtS)))e(iCtS)eSe(iCtS)(iCteSA
 -> iCt(iCt(iCt(iCt(iCtS)))e(iCtS)eSe(iCtS)(iCteSA
S -> iCt(iCt(iCt(iCt(iCtS)))e(iCtS)eSe(iCtS)(iCte(iCtSA)A
eS -> iCt(iCt(iCt(iCt(iCtS)))e(iCtS)eSe(iCtS)(iCte(iCtSeS)A
eS -> iCt(iCt(iCt(iCt(iCtS)))e(iCtS)eSe(iCtS)(iCte(iCtSeSeSA
S -> iCt(iCt(iCt(iCt(iCt(iCtSA))))e(iCtS)eSe(iCtS)(iCte(iCtSeSeSA
 -> iCt(iCt(iCt(iCt(iCt(iCtS))))e(iCtS)eSe(iCtS)(iCte(iCtSeSeSA
 -> iCt(iCt(iCt(iCt(iCt(iCtS))))e(iCtS)eSe(iCtS)(iCte(iCtSeSeSA
```

Figure 3: derivaciones

4 Código

```
1 import random
2
```

```
3
4  def generar_codigo(texto_final):
5      def sangrias(espacios):
6          return "    " * espacios
7
8      espacios = 0
9
10     with open("pseudocodigo.txt", "a", encoding="utf-8") as
       archivo:
11         for caracter in texto_final:
12             if caracter == "(":
13                 espacios += 1
14
15             elif caracter == ")":
16                 espacios -= 1
17
18             if caracter == 'i':
19                 cadena_espacios = sangrias(espacios)
20                 archivo.write(f"\n{cadena_espacios}if ")
21             elif caracter == 'C':
22                 archivo.write("(CONDICIONAL)")
23             elif caracter == 't':
24                 archivo.write(":")
25             elif caracter == 'S':
26                 cadena_espacios = sangrias(espacios + 1)
27                 archivo.write(f"\n{cadena_espacios}CODIGO")
28             elif caracter == 'e':
29                 cadena_espacios = sangrias(espacios)
30                 archivo.write(f"\n{cadena_espacios}else")
31
32 def procesar_if(n):
33     def quitar_a(texto_con_a):
34         def elegir_cadena():
35             return "eS" if random.randint(0, 1) == 0 else ""
36
37         ubicaciones = [i for i, c in enumerate(texto_con_a)
       if c == "A"]
38         if not ubicaciones:
39             return texto_con_a
40
41         nueva_cadena = texto_con_a  # Copiamos la cadena
       base para modificarla
42
43         with open("derivaciones.txt", 'a', encoding='utf-8')
        as archivo:
44             archivo.write(f"S -> {texto_con_a}\n")
45             for i in ubicaciones:
46                 eleccion_a = elegir_cadena()
47                 nueva_cadena = nueva_cadena[:i] + eleccion_a
        + nueva_cadena[i + 1:]
```

```
                archivo.write(f"{eleccion_a} -> {nueva_cadena}\n")
        return nueva_cadena


    def concatenar_texto(texto):
        ubicaciones = [i for i, c in enumerate(texto) if c == "S"]
        if not ubicaciones:
            return texto

        opciones = random.randint(0, len(ubicaciones) - 1)
        posicion = ubicaciones[opciones]

        nuevo_texto = texto[:posicion] + "(iCtSA)" + texto[posicion + 1:]
        nuevo_texto = quitar_a(nuevo_texto)
        return nuevo_texto

    construyendo = "iCtSA"
    if n == 1:
        quitar_a(construyendo)
    else:
        for _ in range(n - 1):
            construyendo = concatenar_texto(construyendo)

    generar_codigo(construyendo)

while True:



    print("1) Manual")
    print("2) Autom tico")
    print("3) Salir")
    opcion = int(input("Ingrese su opci n: "))

    with open("derivaciones.txt", 'w', encoding='utf-8') as w:
        w.write("")

    with open("pseudocodigo.txt", "w", encoding="utf-8") as w:
        w.write("")

    if opcion == 1:
        num_ifs = int(input("\nN mero de ifs: "))
        if 0 < num_ifs < 1000:
            procesar_if(num_ifs)

```

```
    elif opcion == 2:
        num_ifs = random.randint(0, 1000)
        procesar_if(num_ifs)

    elif opcion == 3:
        break

    else:
        print("Opci n incorrecta.")
```

Listing 1: Practica 8

5 Conclusión

El programa desarrollado demuestra la implementación práctica de las gramáticas Backus-Naur (BNF) para la generación automatizada de estructuras condicionales IF. Esta herramienta refleja la capacidad de definir reglas sintácticas formales para la construcción de condicionales IF, siguiendo la notación BNF establecida.

Además, como se mencionó en clase, cada generación aleatoria representa un programa contenido dentro del vasto universo de programas posibles. Aunque con una probabilidad extremadamente baja, este enfoque de generación podría teóricamente dar lugar a la creación de un programa de gran complejidad, incluso sin darnos cuenta, como un potencial "Windows 15", ilustrando así la infinita diversidad y posibilidades de los programas que pueden emerger de esta metodología.

6 Bibliografía

Ullman, J.D. (2009-10). "CS154: Introduction to Automata and Complexity Theory". Sitio web: http://infolab.stanford.edu/ ullman/ialc/spr10/spr10.htmlLECTURE