Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Traductores e Interpretadores
Trimestre Septiembre-Diciembre 2019

Etapa II
Análisis Sintáctico

Esta segunda etapa del proyecto corresponde al módulo de análisis sintáctico de programas escritos en GuardedUSB. Esto incluye la definición de la gramática y la construcción del árbol sintáctico abstracto correspondiente.

Usted deberá escribir una gramática para el lenguaje que luego será pasada a una herramienta generadora de analizadores sintácticos, la cual generará un analizador que debe ser capaz de reconocer cualquier cadena válida en el lenguaje. Al ser suministrada tal cadena, su programa deberá imprimir una representación legible del árbol sintáctico abstracto creado.

Esta implementación incluye la programación de distintas clases (Java, Ruby o Python) o tipos de datos (Haskell) que representen el significado de las expresiones e instrucciones en GuardedUSB. Por ejemplo, las expresiones conformadas por dos subexpresiones y un operador binario, podría ser representado por la clase/tipo de datos BINEXPRESION, que pudiera tener como información las dos subexpresiones, el operador utilizado y el tipo resultante de la expresión.

También pudiera tenerse diferentes árboles de expresión binarios para los casos aritméticos, booleanos y relacionales; lo cual simplifica considerablemente el trabajo de verificación de tipos. Una posible estrategia (no en Haskell) es la de crear una clase abstracta que representa a todos los árboles de expresión y otra para los árboles de instrucción de forma de aprovechar la herencia y unificar las características que sean comunes (En el caso de Haskell, puede realizarse un tratamiento similar utilizando clases de tipos). De encontrar algún error sintáctico, deberá reportar dicho error y abortar la ejecución del analizador. ***Usted NO debe imprimir todos los errores sintácticos, solamente el primero encontrado.*** Sin embargo, aún deberá imprimir todos los errores léxicos.

En esta etapa, la gramática que utilice y los nombres que coloque a los símbolos terminales y no terminales presentes en la misma, son enteramente de su elección. Sin embargo, dicha gramática debe cumplir con algunas condiciones:

- Su gramática debe generar todas las posibles cadenas válidas en GuardedUSB y únicamente dichas cadenas (No puede generar cadenas que no pertenezcan al lenguaje).
- Los nombres escogidos para los símbolos terminales y no terminales deben ser apropiados y fácilmente reconocibles como parte del lenguaje.
- Se sugiere fuertemente que diseñe su gramática manteniéndola ambigua y recursiva por *la izquierda*. ***Los conflictos existentes deberán ser tratados entonces por los mecanismos de resolución de conflictos (precedencia y asociatividad) que incluye cada herramienta***.

Un ejemplo:
El siguiente programa:

```
|[
  declare
```

```
    a, b, i : int;
    x, y : array[2..5]

  // iterar entre a y b
  read a;
  read b;
  x := x(2:a)(3:b);
  for i in a to b -->
    |[
      println "Variable \"i\" es igual a: " || i;
    ]|
  rof;

  // iterar sobre los indices del segundo arreglo
  read y;
  for j in min(y) to max(y) -->
    |[
      print j || ":" || y[j] || " ";
    ]|
  rof;

  a := 3 + b;
  b := -4;

  if min(x) <= b /\ b <= max(x) --> println b
  [] min(y) < b /\ b < max(y) --> println a
  [] min(y) >= b \/ b >= max(y) -->
     |[
        declare z : array[2..3]
        println a || b;
        z := z(2:a)(3:b);
        println "array[" || min(z) || ".." || max(z) || "]"
     ]|
  fi;

  i := 3;
  do i < 10 -->
    |[
      read i;
      print "Still here!";
      i := i+1
    ]|
 od
]|
```

Debería arrojar lo siguiente:

```
Block
 Declare
  Ident: a
  Ident: b
  Ident: i
  Sequencing
   Ident: x
   Ident: y
 Read
  Ident: a
 Sequencing
  Read
   Ident: b
  Sequencing
   Asig
    Ident: x
    Exp
     ArrayAsig
      ArrayAsig
       Ident: x
       Literal: 2
       Ident: a
      Literal: 3
      Ident: b
   Sequencing
    For
     In
      Ident: i
      Exp
       Ident: a
      Exp
       Ident: b
     Block
      Println
       Concat
        "Variable \"i\" es igual a: "
        Ident: i
     Sequencing
      Read
       Ident: y
      Sequencing
       For
        In
         Ident: j
         Exp
          Min
           Ident: y
         Exp
```

```
  Max
   Ident: y
 Block
  Print
   Concat
    Concat
     Concat
      Concat
       Ident: j
       ":"
      EvalArray
       Ident: y
       Exp
        Ident: j
    " "
Sequencing
 Asig
  Ident: a
  Exp
   Plus
    Literal: 3
    Ident: b
 Sequencing
  Asig
   Ident: b
   Exp
    Literal: -4
  Sequencing
   If
    Guard
     Exp
      And
       Leq
        Min
         Ident: x
        Ident: b
       Leq
        Ident: b
        Max
         Ident: x
     Println
      Ident: b
    Guard
     Exp
      And
       Less
        Min
         Ident: y
        Ident: b
       Less
```

```
      Ident: b
      Max
       Ident: y
    Println
     Ident: a
   Guard
    Exp
     Or
      Geq
       Min
        Ident: y
       Ident: b
      Geq
       Ident: b
       Max
        Ident: y
    Block
     Declare
      Ident z
     Println
      Concat
       Ident: a
       Ident: b
     Sequencing
      Asig
       Ident: z
       Exp
        ArrayAsig
         ArrayAsig
          Ident: z
          Literal: 2
          Ident: a
         Literal: 3
         Ident: b
      Sequencing
       Println
        Concat
         Concat
          Concat
           Concat
            "array["
            Min
             Ident: z
           ".."
          Max
           Ident: z
         "]"
  Sequencing
   Asig
```

```
 Ident: i
 Exp
  Literal: 3
Sequencing
 Do
  Exp
   Less
    Ident: i
    Literal: 10
  Block
   Read
    Ident: i
   Sequencing
    Print
     "Still here!"
    Sequencing
     Asig
      Ident: i
      Exp
       Plus
        Ident: i
        Literal: 1
```

Note que la declaración y tipos de las variables ***no se están*** en el árbol sintáctico abstracto.

Implementación

En esta etapa Ud. deberá desarrollar el analizador sintáctico de nuestro lenguaje como se indica a continuación:

1. Se recomienda que diseñe una gramática cuyo lenguaje generado y escriba la misma en el lenguaje de especificación de la herramienta ofrecida para el lenguaje de su elección.
2. Escriba las reglas para construir e imprimir el árbol sintáctico abstracto correspondiente a una frase reconocida.
3. Escriba un programa que lea un programa (potencialmente incorrecto) y genere el árbol sintáctico correspondiente, de ser correcto. Si el programa no es correcto por razones puramente léxicas, debe imprimir únicamente todos los errores encontrados. Si el programa tiene al menos un error sintáctico, debe imprimir únicamente el primero de tales errores.

Entrega

Ud. debe entregar un correo electrónico que contenga:

• El código fuente en el lenguaje y la herramienta de su elección, entre los permitidos, de su analizador lexicográfico. Todo el código debe estar debidamente documentado. Es posible que tenga que incorporar un script a su entrega que permita que la llamada a su programa se realice de esta forma. Este archivo tendrá extensión ‘‘.gusb“.

• Un archivo de texto (.txt) en el que se evidencie su gramática. Por ejemplo, un .txt que contenga:

S -> A B C
| A B

A -> C
...

• Un breve informe explicando la formulación/implementación de su analizador sintáctico y justificando todo aquello que Ud. considere necesario.

• Se requiere invocar al programa, ejecutando un programa llamado guardedusb.

Fecha de Entrega

Domingo de Semana 7, hasta las 11:59 pm. Valor: 7%