Proyecto final - Fundamentos de Lenguajes Programación

Robinson Duque, Ph.D
robinson.duque@correounivalle.edu.co

Marzo de 2020

1. Introducción

El presente proyecto tiene por objeto enfrentar a los estudiantes del curso:

- a la comprensión de varios de los conceptos vistos en clase
- a la implementación de un lenguaje de programación
- al análisis de estructuras sintácticas, de datos y de control de un lenguaje de programación para la implementación de un interpretador

Importante: Tenga en cuenta que las descripciones presentadas en este documento son bastante generales y solo pretenden dar un contexto que le permita a cada estudiante entender el proyecto y las características del lenguaje que debe desarrollar. Así mismo, se proponen algunas construcciones sintácticas, es posible que se requiera modificar o adaptar la sintaxis para cumplir con los requerimientos.

2. (70 pts) Lenguaje: C-VID

C-VID es un lenguaje de programación (no tipado) con ciertas características de lenguaje de programación declarativo e imperativo creado en el curso de FLP durante la pandemia del COVID-19. Se propone para este proyecto que usted implemente un lenguaje de programación para lo cual se brindan algunas ideas de sintáxis básica pero usted es libre de proponer ajustes siempre y cuando cumpla con las funcionalidades especificadas. La semántica del lenguaje estará determinada por las especificaciones en este proyecto.

- La gramática **debe ser socializada con el profesor** durante las primeras dos semanas después de la asignación de este proyecto. Para esto, deberá utilizar el mecanismo definido por el profesor para enviar un archivo Racket con la gramática definida. Dentro de los siguientes 2 a 4 días hábiles después del envío recibirá una respuesta con observaciones y/o la aprobación de la gramática. Este proceso podrá tomar máximo 2 envíos con actualizaciones sobre las recomendaciones realizadas. Así pues, si deja su envío para la segunda semana, quizá sólo tenga 1 retroalimentación sobre su gramática.
- La gramática deberá contener ejemplos de cada producción utilizando llamados a `scan&parse`. Es decir, deberá contener ejemplos de cómo se crean las variables, procedimientos, invocación a procedimientos, etc.
- Todo interpretador cuya gramática esté sin aprobar por el profesor, tendrá una nota de sustentación de 0.

2.1. Valores:

- **Valores denotados:** enteros, flotantes, octales, caracteres, cadenas de caracteres, booleanos (true, false), procedimientos, listas, registros, vectores.
- **Valores expresados:** enteros, flotantes, octales, caracteres, cadenas de caracteres, booleanos (true, false), procedimientos, listas, registros, vectores.

Aclaración: Los números en una base distinta de 10, deberán representarse así: x8 (2 1 4 0 7), teniendo en cuenta que el primer elemento indica la base del número y la lista puede utilizar la representación BIGNUM vista en clase.

Sugerencia: trabaje los valores enteros, flotantes desde la especificación léxica. Implemente los números octales, cadenas de caracteres, booleanos (true, false) y procedimientos desde la especificación gramatical.

2.2. Sintaxis Gramatical

2.2.1. Expresiones

En C-VID, casi todas las estructuras sintáticas son expresiones, y todas las expresiones producen un valor. Las expresiones pueden ser clasificadas en:

- **Identificadores:** Son secuencias de caracteres alfanuméricos que comienzan con una letra.

```
<expresion>  := <identificador>
<identificador>::=<letter>| {<letter>|
    0,...,9| }*
<letter> ::= A..Z | a..z
```

- **Definiciones:** Este lenguaje permitirá crear distintas definiciones:

```
var x1 = a1, ...,xn = an ;
cons y1 = b1, ...,yn = bn ;
rec z1 = c1, ...,zn = cn ;
global w1 = d1, ...,wn = dn ;
unic q1=e1,...,qn=en;
```

- Una definición *var* introduce una colección de variables actualizables (mutables) y sus valores iniciales (e.g., enteros, listas, procedimientos, etc).
- Una definición *cons* introduce una colección de constantes no actualizables y sus valores iniciales. Una constante es de asignación unica y debe ser declarada con un valor inicial, por ejemplo: `cons y = 9;`. Para este caso `y` no podrá ser modificada durante la ejecución de un programa.
- *rec* introduce una colección de procedimientos recursivos.
- Una definición *global* introduce una colección de variables globales; deberán estar definidas siempre al inicio del programa y deberán tener su propio ambiente; no podrán haber variables globales repetidas y podrán ser accedidas por cualquier expresión en cualquier punto del programa en caso que una variable de no pueda ser encontrada en los respectivos ambientes de ejecución; por lo demás, las variables globales se comportan igual que una variable actualizable *var*.
- Una definición *unic* introduce una colección de variables actualizables una única vez. Una variable de asignación unica puede ser declarada así: `unic z = 9, x = C-VID-VAL;`. Para este caso, `z` no podrá ser modificada durante la ejecución de un programa puesto que ya ha sido asignada. Sin embargo, `x` podrá ser asignada una única vez a algún valor del lenguaje `C-VID`. Por ejemplo: `x ->29;`. En caso que se utilice la variable `x` sin estar ligada, es decir, que contenga el valor especial C-VID-VAL el interpretador deberá lanzar un error.

La gramática a utilizar se puede definir aproximadamente así:

```
<programa>::= <globales> <expresion>

<globales> ::= {<identificador> = <
    expresion> }*(,)

<expresion>
::= var {<identificador> = <expresion>
    }*(,) in <expresion>
::= cons {<identificador> = <expresion>
    }*(,) in <expresion>
::= rec {<identificador>
    ( { <identificador> } *(,) ) = <expresion>
     }* in <expresion>
::= unic {<identificador> = <expresion>
    }*(,) in <expresion>
```

- **Datos:** Definen las constantes del lenguaje y permiten crear nuevos datos.

```
<expresion> ::= <numero>
            ::= <caracter>
            ::= <cadena>
            ::= <bool>

Ejemplos :
0, 1, -1, 9.5    numeros
'a'              caracteres
" abc "          cadena
true, false      bool
```

- **Constructores de Datos Predefinidos:**

```
<expresion>
::= <lista>
::= <vector>
::= <registro>
::= <expr-bool>

<lista> ::= [{<expresion>} *(;)]

<vector> ::= vector[{<expresion>} *(;)]

<registro> ::= { {<identificador> =
<expresion> }+(;) }

<expr-bool>
  ::= compare(<expresion> <pred-prim> <
      expresion>)
  ::= <oper-bin-bool>(<expr-bool>, <expr-
      bool>)
  ::= <bool>
  ::= <oper-un-bool>(<expr-bool>)

<pred-prim> ::= < | > | <= | >= | == | <>
<oper-bin-bool> ::= and|or|xor
<oper-un-bool> ::= not
```

- **Estructuras de control**

```
<expresion>::= sequence {<expresion>}+(;)
     end

<expresion>::= if <expr-bool> then <
    expresion>
[ else < expresion > ] end

<expresion> ::= cond { "[" <expresion> <
    expresion> "]" }* else <expresion>

<expresion>::= while <expr-bool> do
<expresion> done

<expresion>::= for <identificador> = <
    expresion>
(to | downto) <expresion> do
<expresion> done
```

- **Primitivas aritméticas para enteros:** $+, -, *, \%, /, ++, --$; las primitivas binarias deberán ser escritas en notación infija.
- **Primitivas aritméticas para octales:** $+, -, *, ++, --$; las primitivas binarias deberán ser escritas en notación infija.
- **Primitivas sobre cadenas:** longitud, concatenar
- **Primitivas sobre listas:** se deben crear primitivas que simulen el comportamiento de: `vacio?`, `vacio`, `crear-lista`, `lista?`, `cabeza`, `cola`, `append`. Recuerde que las listas son estructuras secuenciales inmutables.
- **Primitivas sobre vectores:** se debe extender el lenguaje y agregar manejo de vectores. Se deben crear primitivas que simulen el comportamiento de: `vector?`, `crear-vector`, `ref-vector`, `set-vector`. Recuerde que los vectores son estructuras mutables de acceso aleatorio con un índice predefinido desde 0 hasta n-1.
- **Primitivas sobre registros:** se debe extender el lenguaje y agregar manejo de registros. Se deben crear primitivas que simulen el comportamiento de: `registros?`, `crear-registro`, `ref-registro`, `set-registro`. Recuerde que los registros son estructuras mutables de acceso aleatorio con estructura clave-valor.
- **Definición/invocación de procedimientos:** el lenguaje debe permitir la creación/invocación de procedimientos que retornan un valor al ser invocados. El paso de parámetros será por valor y por referencia, el lenguaje deberá permitir identificar de alguna manera la forma como se enviarán los argumentos.
  Algunos lenguajes como C++ utilizan el símbolo `&` para indicar que el paso de valor es por referencia. Por ejemplo, invocar la función `ajuste(x,y)` envía los valores de `x` e `y` respectivamente (i.e., invocación por valor). Sin embargo, invocar la función `ajuste(x,&y)` enviará el valor de `x` y una referencia de `y`, por lo cual si se modifica el valor de `y` dentro de la función `ajuste`, el cambio se verá reflejado externamente.
- **Definición/invocación de procedimientos recursivos:** el lenguaje debe permitir la creación/invocación de procedimientos que pueden invocarse recursivamente. . El paso de parámetros será por valor y por referencia, el lenguaje deberá permitir identificar de alguna manera la forma como se enviarán los argumentos.
- **Variables actualizables (mutables) - Aclaraciones**: Una variable mutable puede ser modificada cuantas veces se desee (e.g., var, rec, global). Una variable mutable puede ser declarada así: `var a = 5, b = 6;`. En ambos casos, ambas variables podrán ser modificadas durante la ejecución de un programa. Por ejemplo: `a ->9;` o `b->true;`. **El intento de modificar una variable inmutable (cons) o de asignación única (unic) que ya esté ligada, deberá generar un error.** Las variables *globales* también son mutables y sólo podrán ser mutadas en caso que una variable var no pueda ser modificada, por ejemplo:

```
global(x = 5, y = 3, w = 4, p = 3)
var (x = 9, z = 4, p = 7) in
  sequence
      x -> 8;    %modifica x del ambiente
          de variables var
      y -> 1;    %y no esta en var, se
          debe modificar la y del ambiente
           global
      z -> 2;    %modifica z del ambiente
          de variables var
      ((((x + y) + z) + p) + w);
  end

El programa anterior evalua: ((((8 + 1) +
     2) + 7) + 4)
Los ambientes quedan en el siguiente
    estado:

global(x = 5, y = 1, w = 4, p = 3)
var (x = 8, z = 2, p = 7)
```

- **Secuenciación:** el lenguaje deberá permitir expresiones para la creación de bloques de instrucciones `sequence` y deberá retornar el valor de la última instrucción.
- **Estructuras condicionales:** el lenguaje debe permitir la defición de estructuras de control tipo `if y cond`. La semántica del `cond` consiste en evaluar una serie de expresiones de prueba en el orden en que aparecen y ejecutar la expresión consecuente de la primera que tenga éxito. Si ninguna expresión de prueba es verdadera, entonces deberá ejecutarse la expresión contenida en el `else`.
- **Iteración:** el lenguaje debe permitir la defición de estructuras de repetición tipo `while y for`. Por ejemplo: `for x = 1 to` a_2 `do` a_3 `done`. Se sugiere agregar funcionalidad al lenguaje para que permita "imprimir" resultados por salida estándar tipo `print`.

3. (30 pts) C-VID+ Con Tipos de Datos

3.1. Tipos de datos

el lenguaje C-VID+ es una extensión del lenguaje C-VID que le permite al programador el uso de tipos de datos. El lenguaje debe incluir solemanete chequeo de

tipos (no es necesario implementar inferencia de tipos) y se deben definir reglas para:

- todas las primitivas (aritméticas para enteros y octales, cadenas, listas, vectores, registros)
- condicionales (if, cond)
- invocación de procedimientos
- iteración

Los tipos de datos a incluir son:

- Int
- Float
- Bool
- Oct
- Char
- List
- Register
- Vector
- Function

De esta manera, deberá extender su lenguaje para escribir programas tipados por el estilo de:

```
global(
  Int x = 5,
  Float y = 3.4,
  Oct w = x8 (1,3,0),
  List p = [3;7;1;9],
  Register q={"nombre"="Robinson"; "cursos"=3}
)

var (
 Function f = proc(Int a, Int b) return (a*b),
 Int y = 3
) in
    sequence
      x -> (y+5);
      f(x, (y+3));
    end
```

4. Evaluación

El proyecto podrá ser realizado en grupos de hasta 3 personas utilizando la librería SLLGEN de Dr Racket. Este debe ser sustentado individualmente y cada persona del grupo obtendrá una nota entre 0 y 1 (por sustentación), la cual se multiplicará por la nota obtenida en el proyecto.

Sólo la especificación léxica y gramatical del lenguaje base C-VID, deberá ser aprobado por el profesor. El interpretador C-VID+ podrá contener algunas variaciones simples del lenguaje base. Dado el caso que un grupo se presente con un lenguaje distinto a la gramática definida para el interpretador C-VID, obtendrán una nota de de 0 en la sustentación asociada con dicho interpretador puesto que no hay forma de justificar dicho cambio.