Proyecto final - Fundamentos de Lenguajes Programación

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

Abril de 2025

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. Mini-Py

Mini-Py es un lenguaje de programación (no tipado) con ciertas características de lenguaje de programación declarativo, imperativo y orientado a objetos inspirado en algunas características de Python. Adicionalmente, ofrece soporte para resolver operaciones con circuitos lógicos del lenguaje Clogic de los Talleres 2 y 3. 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.

2.1. Sintaxis Gramatical

- El desarrollo del proyecto y la participación de los integrantes debe ser rastreable y verificable en todo momento (no podrán aparecer entregas de la nada o cambios completos del interpretador sin evidenciar un desarrollo contínuo). Para esto, cada grupo deberá crear un **repositorio privado** en **Github** y en caso que se le solicite deberá invitar al usuario **robinsonduque**. Recuerde incluir la información de los integrantes en el archivo Readme al igual que información relevante del proyecto.

  La gramática **debe ser definida en el repositorio** durante las primeras tres semanas después de la asignación de este proyecto.

- **La gramática debe estar documentada con comentarios indicando el lenguaje en el cual se han inspirado para especificar cada regla de producción**.

- 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.

2.2. Valores:

- **Valores denotados:** Ref(valores expresados)
- **Valores expresados:** enteros, flotantes, hexadecimales, cadenas de caracteres, booleanos (true, false), procedimientos, listas, registros, **circuitos lógicos**, tuplas, objetos.

Aclaración: Los números en una base distinta de 10, podrán representarse así: x32 (0 23 12), x16 (4 1 0 7 14), 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 hexadecimales, cadenas de caracteres, booleanos (true, false), circuitos y procedimientos desde la especificación gramatical.

2.3. (50 pts) Sintaxis Gramatical

2.3.1. Expresiones

En Mini-Py, 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 in ...;
const y1 = b1, ...,yn = bn in ...;
rec z1 = c1, ...,zn = cn in ...;
```

Una definición *var* introduce una colección de variables actualizables y sus valores iniciales. Una definición *const* introduce una colección de constantes no actualizables y sus valores iniciales. *rec* introduce una colección de procedimientos recursivos. La gramática a utilizar se puede definir así:

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

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

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

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

- **Constructores de Datos Predefinidos:**

```
<expresion>
::= <lista>
::= <tupla>
::= <registro>
::= <expr-bool>
::= <circuit>

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

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

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

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

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

* Para los circuitos y sus primitivas utilice la gramática del Taller 3.

- **Estructuras de control**

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

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

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

<expresion>::= for <identificador> in
<expresion> do
<expresion> done
```

- **Primitivas aritméticas para enteros:** $+, -, *, \%, /, add1, sub1$
- **Primitivas aritméticas para hexadecimales:** $+, -, *, add1, sub1$
- **Primitivas sobre cadenas:** longitud, concatenar
- **Primitivas sobre listas:** las listas en Python son una estructura de datos mutable, es decir, sus valores pueden ser actualizados en algún momento durante la ejecucuón de un programa. Se deben crear primitivas que simulen el comportamiento de: `vacio?`, `vacio`, `crear-lista`, `lista?`, `cabeza`, `cola`, `append`, `ref-list`, `set-list`.
- **Primitivas sobre tuplas:** las tuplas en Python son una estructura de datos inmutable, es decir, una vez creadas no se pueden modificar. Se debe extender el lenguaje y agregar manejo de tuplas. Se deben crear primitivas que simulen el comportamiento de: `vacio?`, `vacio`, `crear-tupla`, `tupla?`, `cabeza`, `cola`, `ref-tuple`.
- **Primitivas sobre registros:** Los registros son estructuras mutables compuestas por conjuntos de claves y valores. En Python los registros reciben el nombre de diccionarios. 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`.
- **Primitivas sobre circuitos:** Los circuitos pueden ser construidos a través de compuertas/gates (and, or, xor, not). Se deben crear primitivas que permitan manipular circuitos: `eval-circuit`, `connect-circuits`, `merge-circuits`, para esto integre el taller 3 del curso en este interpretador.

- **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 (para valores numéricos, caracteres, cadenas, procedimientos, tuplas) y por referencia (para listas y registros) similar a como sucede en Python.
- **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 (para valores numéricos, caracteres, cadenas, procedimientos, tuplas) y por referencia (para listas y registros) similar a como sucede en Python.
- **Variables actualizables (mutables)**: introducen una colección de variables actualizables y sus valores iniciales. Una variable mutable puede ser modificada cuantas veces se desee. 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 (constante), deberá generar un error.**
- **Secuenciación:** el lenguaje deberá permitir expresiones para la creación de bloques de instrucciones
- **Iteración:** el lenguaje debe permitir la defición de estructuras de repetición tipo `while y for` sobre estructuras iterables (listas y tuplas). Por ejemplo: `for` x `in` $miLista$ `do` $(x * 3)$ `done`. Se sugiere agregar funcionalidad al lenguaje para que permita "imprimir" resultados por salida estándar tipo `print`.

2.4. (50pts) Programación Orientada a Objetos

Extienda el lenguaje Mini-Py y añada:

- **Declaración de clases y métodos inspirada en la sintáxis de Python (incluir el constructor init y el método str para imprimir el objeto).**

  Ejemplo de creación de clase y objeto:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

  Salida:
  John
  36

  Ejemplo de uso del método str al imprimir el objeto:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

  Salida:
  John
  36

- **Creación de objetos:** simples o planos, cualquier representación interna es válida
- **Invocación de métodos y selección de campos:** el lenguaje debe permitir invocar métodos asociados a objetos y obtener los valores asociados a los campos

  Ejemplo:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def getName(self):
    return self.firstname

x = Person("John", "Doe")
x.getName()
```

  Salida:
  John

- **Actualización de campos:** el lenguaje debe permitir actualizar los campos asociados a un objeto
- El lenguaje debe incluir conceptos como: herencia, llamados a métodos de la superclase, polimorfismo

3. Evaluación

El proyecto podrá ser realizado en los grupos ya definidos utilizando la librería SLLGEN de Dr Racket. Este debe ser sustentado 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 (la sustentación se llevará a cabo en el campus virtual).

El interpretador Mini-Py 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 Mini-Py, obtendrán una nota de de 0 en la sustentación asociada con dicho interpretador.