Universidad del Valle

Proyecto final - Fundamentos de Lenguajes Programación

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

Mayo de 2023

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 instancias del problema SAT (Satisfacción booleana). 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** e 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, tuplas, **instancias SAT (se explicarán más adelante y sólo para estudiantes de la RES 048).**

Aclaración: Los números en una base distinta de 10, deberán 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 en base 32, hexadecimales, octales, cadenas de caracteres, booleanos (true, false) 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>

<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
```

- **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> = <
    expresion>
(to | downto) <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`.
- **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`. 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`.

2.4. (20 pts para la 048 y 50pts para la 047) Type Mini-Py

Extienda el lenguaje Mini-Py y añada los siguientes tipos a las definiciones de variables:

```
int
float
hex
bool
list
tuple
register
 FNC -> solo estudiantes de la 048
```

Por ejemplo, la creación de variables podría definirse de la siguiente manera:

```
var
   int x1 = 5,
   float xn = 4.6,
   list l1 = crear-lista(1,2,3,4,5)
in ...;
```

Los errores de tipos serán los mismos que se estudiaron en el curso y adicionalmente cualquier aplicación de primitivas a un tipo de dato incorrecto respecto a las listas, tuplas y registros.

2.5. (30 pts) Soporte para instancias SAT - Sólo Estudiantes de la RES 048

El Problema de Satisfactibilidad Booleana (SAT) consiste en un conjunto V de n variables booleanas $v_1, v_2, v_3 ... v_n$ y un conjunto C de m clausulas $c_1, c_2, c_3 ... c_m$ en **forma normal conjuntiva (FNC)**. Por ejemplo:

$$C = (x \vee \neg y \vee z \vee w) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (z \vee w) \wedge y$$

C en este caso es una instanica del problema y la idea es responder lo siguiente: ¿Existe una asignación de valores para (x, y, z, w), donde C sea verdadera?

En la actualidad existen competencias `http://www.satcompetition.org/` donde se desarrollan solvers especializados que intentan resolver este tipo de problemas. Sin embargo, usualmente estos solvers requieren de un manejo avanzado de técnicas basadas en inferencia lógica para su solución.

En nuestro caso, para el problema C realizaremos una exploración ingenua del las posbiles combinaciones de valores de verdad para las variables (x, y, z, w) hasta lograr encontrar una solución o hasta explorar todas las posibles combinaciones y demostrar que el problema no tiene solución.

Por ejemplo, podriamos intentar primero con $(\#t, \#t, \#t, \#t)$ y nos daremos cuenta que C no se puede satisfacer debido a la clausula $(\neg x \vee \neg y \vee \neg z)$ resulta ser falsa. Podriamos intentar luego con $(\#t, \#t, \#t, \#f)$, y luego con $(\#t, \#t, \#f, \#t)$, y luego con $(\#t, \#t, \#f, \#f)$ hasta llegar a una posible solución. Por ejemplo, una de las soluciones se logra con $(\#f, \#t, \#t, \#t)$ y se puede parar la búsqueda porque se ha encontrado una solución (sin embargo, es posible que existan otras). Así pues, se dice que la instancia C es SATISFACTIBLE para (x, y, z, w) con valores $(\#f, \#t, \#t, \#t)$

Veamos otro ejemplo:

$$C = (x \vee y) \wedge (\neg x) \wedge (\neg y)$$

Observe que después de explorar todas las posibles combinaciones para (x, y) con valores $(\#t, \#t)$, $(\#t, \#f)$, $(\#f, \#t)$, $(\#f, \#f)$ no se logra satisfacer C. Así pues, se dice que C es INSATISFACTIBLE.

- Implemente en la gramática del lenguaje Mini-Py producciones que permitan escribir expresiones en FNC. Una forma usual de escribir estas instancias es utilizando un formato numérico que represente el problema. Entonces la instanica:

  $$C = (x \vee \neg y \vee z \vee w) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (z \vee w) \wedge y$$

  puede ser representada así:

```
FNC 4 ( (1 or -2 or 3 or 4) and
        (-2 or 3) and
        (-1 or -2 or -3) and
        (3 or 4) and
        (2) )
```

Donde FNC indica que la expresión es una FNC, el 4 que le sigue indica que hay 4 variables y luego hay una construcción de clausulas con números de 1 a 4 donde: $x = 1, \neg x = -1, y = 2, \neg y = -2, z = 3, \neg z = -3, w = 4, \neg w = -4$. Esta construcción está dada por una gramática así:

```
<expresion> ::= FNC <numero> ( clausula-or
    )+ ("and")

<clausula-or> ::= ( <numero> )+ ("or")
```

- Implemente un mecanismo donde dada una instancia FNC, esta pueda ser evaluada así:

  Ejemplo 1:

```
var
x = FNC 4 ((1 or -2 or 3 or 4) and (-2 or
    3) and (-1 or -2 or -3) and (3 or 4)
    and (2)) ;
in
x.solve( )

% Respuesta:
(satisfactible (#f #t #t #t))
```

  Ejemplo 2:

```
var
x = FNC 2 ((1 or 2) and (-1) and (-2)) ;
in
x.solve( )

% Respuesta:
(insatisfactible '() )
```

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.

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.