Programming Language: ParserTongue

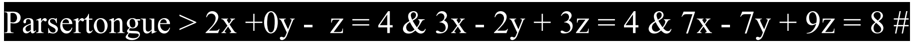
Alejandro Proskauer Valerio
Carlos I.Lopez Sosa
Aryam L. Yulfo Soto
Prof. Alcibiades Bustillo
Mayo 14, 2021

Objetivo y Motivación

Cuando tenemos un problema nuestras mentes buscan la manera de resolver el mismo de una manera rápida y eficaz. Este es el objetivo y la motivación detrás del lenguaje ParserTongue. Nuestro lenguaje será capaz de resolver sistemas de ecuaciones lineales de manera rápida y eficaz.

Tutorial de ParserTongue

Para obtener los resultado deseados el usuario debe ofrecer el sistema de ecuaciones de manera correcta. Después de cada ecuación se debe colocar el símbolo ¨&¨ y al final de la última ecuación el símbolo ¨#¨. Ejemplo:



```
Parsertongue > 2x +0y -  z = 4 & 3x - 2y + 3z = 4 & 7x - 7y + 9z = 8 #
```

Ejemplos de situaciones ¨Unsolvable¨

```
Parsertongue > 4x+0y=4 & 5x+0y=3 #

Equation 1: (4)x + (0)y = 4

Equation 2: (5)x + (0)y = 3

ERROR OCCURRED, THIS SYSTEM OF EQUATIONS IS NOT SOLVABLE
```

```
Parsertongue > 5x-3y+0z = 1 & -x + 1.2y -0z = -2.4 & 8.6x - 69y +0z = 32#

Equation 1: (5.0)x + (-3.0)y + (0.0)z = 1.0

Equation 2: (-1.0)x + (1.2)y + (0.0)z = -2.4

Equation 3: (8.6)x + (-69.0)y + (0.0)z = 32.0

ERROR OCCURRED, THIS SYSTEM OF EQUATIONS IS NOT SOLVABLE
```

Limitaciones

ParserTongue tiene ciertas limitaciones al momento de ejecutar. Estas limitaciones pueden ser trabajadas en un futuro.

Limitaciones:

1) No reconoce valores vacios.
2) El input del usuario debe estar en orden. X -> Z.
3) Cada término lider X debe tener un coeficiente o un +, -.

Desarrollo del Lenguaje

Arquitectura de Traductor: Lexer Tokens

```
TOKEN_INT = r'\d+’
```

(This is used to identify integers)

```
TOKEN_PERIOD = r'\.'
```

(This is used to identify a floating point to later interpret decimal numbers)

```
TOKEN_SUM = r'\+'
```

(This is used to identify when the linear equation involves a positive coefficient or constant)

```
TOKEN_MINUS = r'-'
```

(This is used to identify when the linear equation involves a negative coefficient or constant)

```
TOKEN_IGUAL = r'='
```

(This is used to mark when the linear combination of variables is equal to the constant)

```
TOKEN_X = r'[xX]'
```

(This is used to identify the 1st variable of the linear equation)

```
TOKEN_Y = r'[yY]'
```

(This is used to identify the 2nd variable of the linear equation)

```
TOKEN_Z = r'[zZ]'
```

(This is used to identify the 3rd variable of the linear equation)

```
NEXT_EQUATION = r'&'
```

(This is used to separate between equations in the system)

```
END_SYSTEM = r'#'
```

(This is used at the end of the system of equations)

Arquitectura de Traductor: Grammar Rules

`S' -> number`

`number -> TOKEN_INT`

Returns the value of the integer

`number -> TOKEN_INT TOKEN_PERIOD TOKEN_INT`

Returns value of the interpreted floating point number

`number -> TOKEN_MINUS number`

Returns value of the number times -1

`number -> TOKEN_SUM number`

Returns value of the number

`number -> x_term`

Returns value of the coefficient of x

`number -> y_term`

Returns value of the coefficient of y

`number -> z_term`

Returns value of the coefficient of z

`number -> system`

Uses numpy.linalg.solve and the coefficient/constant arrays that have been filled to return a tuple of the coefficients, constants, and solutions.

`x_term -> TOKEN_MINUS TOKEN_X`

Assign the coefficient -1 to x

`x_term -> TOKEN_SUM TOKEN_X`

Assign the coefficient 1 to x

`x_term -> number TOKEN_X`

Assign the coefficient *number* to x

`y_term -> TOKEN_MINUS TOKEN_Y`

Assign the coefficient -1 to y

`y_term -> TOKEN_SUM TOKEN_Y`

Assign the coefficient 1 to y

`y_term -> number TOKEN_Y`

Assign the coefficient *number* to x

`z_term -> TOKEN_MINUS TOKEN_Z`

Assign the coefficient -1 to z

`z_term -> TOKEN_SUM TOKEN_Z`

Assign the coefficient 1 to z

`z_term -> number TOKEN_Z`

Assign the coefficient *number* to z

Arquitectura de Traductor: Grammar Rules

```
system -> x_term y_term TOKEN_IGUAL number NEXT_EQUATION x_term y_term TOKEN_IGUAL number

           END_SYSTEM
```

Identify a system of 2 equations and assign the corresponding coefficients and constants to the arrays stored by the language.

```
system -> x_term y_term z_term TOKEN_IGUAL number NEXT_EQUATION x_term y_term z_term

          TOKEN_IGUAL number NEXT_EQUATION x_term y_term z_term TOKEN_IGUAL number

          END_SYSTEM
```

Identify a system of 3 equations and assign the corresponding coefficients and constants to the arrays stored by the language.

Arquitectura de Traductor: Token Precedence

Level 1 (left): TOKEN_X, TOKEN_Y, TOKEN_Z

Level 2 (left): TOKEN_SUM, TOKEN_MINUS

Level 3 (left): TOKEN_INT, TOKEN_PERIOD

Arquitectura de Traductor: Shell

- Dentro de “__main__”
- Solicita texto del usuario
- Lo tokeniza y parsa
- Chequea si es tuple
- Si es tuple, chequea número de variables y si se pudo resolver
- Imprime resultado o mensaje que no se pudo resolver

Ambiente de Desarrollo de Software

- Usando distribución Anaconda de Python 3.8
- Desarrollado en Spyder IDE
- Usando módulos de librería SLY
- Usando módulos de librería NumPy

Metodología de Pruebas

- Verificar que Lexer identifica los tokens correctamente
- Verificar los grammar rules que interpretan los números
- Verificar los grammar rules que interpretan los x/y/z terms
- Verificar los grammar rules que se usan para resolver los sistemas de ecuaciones
- Verificar que el shell imprime el resultado formalmente

Librerías Usadas

- https://docs.anaconda.com/anaconda/
- https://numpy.org/doc/
- https://sly.readthedocs.io/en/latest/sly.html