**PLDC**

**Polynomial Laplace Dominion Converter**

**Introduction**

Polynomial Laplace Dominion Converter (PLD), es un lenguaje de programación nuevo creado para encontrar la transformada de Laplace de polinomios de cualquier grado. La creación de este lenguaje es motivada debido a que la transformada de Laplace es utilizada en muchas aplicaciones, como es el caso de Circuitos; un tema bastante usado en el área de Ingeniería Eléctrica. Una característica importante de PLD es que es muy lógico en cuanto a la forma en que acepta las expresiones polinomiales, ya que el usuario puede ingresar su expresión tal como se escribe el polinomio, sin preocuparse del grado, o si el polinomio está con términos completos, la única restricción es que el polinomio debe estar en forma descendente.

El objetivo principal de PLD es que el usuario pueda utilizarlo en diferentes problemas de aplicación que requieren el uso de la transformada de Laplace de un Polinomio, sin preocuparse en encontrar la transformada de Laplace manualmente, además que puede servir como herramienta de comprobación para aquellos que están aprendiendo a calcular la Transformada de Laplace de un polinomio. Por este motivo el presente documento es desarrollado con el propósito de proveer un tutorial y un manual de referencia sobre el uso de PLD. Además de mostrar información sobre cómo fue desarrollado el lenguaje. Finalmente presentamos las conclusiones en cuanto a la implementación de PLD.

**Language tutorial**

1. **Modo interactivo**

Usamos el interprete en modo interactivo, esto es; cuando los comandos son leídos desde la consola. Una vez estamos en modo interactivo en la consola aparecerá el comando "**(+\_+)>**" indicando que podemos ingresar nuestras instrucciones.

1. **Operación de Asignación para un polinomio:**

El singo "=" es usado para asignar un polinomio a una variable. Por ejemplo si deseamos que PLD reciba un polinomio, este debe ser asignado a una variable (con cualquier nombre) como se muestra en los siguientes ejemplos.

* Asignando

**(+\_+)>** var1= 3t^4-2t^2+5

* Asignado

**(+\_+)>** poli = 3t^7+5

1. **Función laplace:**

La Transformada de Laplace puede ser calculada de dos maneras.

Ejemplo: Deseamos calcular la Transformada de Laplace del polinomio

* Para esto usamos la palabra *laplace* seguido del *polinomio entre paréntesis*; esto es:

**(+\_+) >** laplace(3t^4-2t^2+5)

5.0/s-4.0/s^3+72.0/s^5

* Esta misma operación se puede hacer usando *laplace* *espacio* y el *polinomio*; esto es:

**(+\_+) >** laplace 3t^4-2t^2+5

5.0/s-4.0/s^3+72.0/s^5

1. **Función *laplace* de un polinomio asignado:**

Si deseamos calcular la transformada de *laplace* al polinomio asignado en una variable.

* Primero asignamos el polinomio a una variable.

(+\_+) > var=3t^2-4

* Ahora calculamos la transformada de *laplace* de *var*

**(+\_+) >** laplace(var)

-4.0/s+6.0/s^3

1. **Almacenar el resultado de la Transformada de Laplace en una variable.**

Ejemplos:

**(+\_+) >** result=laplace(var)

**(+\_+) >** result2=laplace(3t^4-1)

1. **Asignar una variable a otra variable:**

Si deseamos asignar el polinomio almacenado en *var* a una nueva variable.

**(+\_+) >** nuevaAsig=var

1. **La función show.**

Es usada para mostrar el contenido de una variable que fue asignada anteriormente.

Ejemplos:

* Deseamos mostrar lo que hay en result2 (que fue asignado previamente):

**(+\_+) >** show result2

-1.0/s+72.0/s^5

* Deseamos el contenido de una variable que guardo el contenido de otra variable; es decir:

**(+\_+) >** var=3t^2-4 #almacenamos un polinomio en var

**(+\_+) >** nuevaAsig=var #nuevaAsig guarda el contenido de var

#Ahora mostramos con *show* lo que contiene *nuevaasig*

**(+\_+) >** show nuevaAsig

3t^2-4

* También podemos imprimir un *string* con *show*

**(+\_+) >** show "This is a String "

This is a String

**Language manual reference:**

|  |  |
| --- | --- |
| var=polinomio | Asigna el polinomio a *var* |
| Laplace(polinomio) | Calcula la Transformada de Laplace de un polinomio |
| Laplace polinomio | Calcula la Transformada de Laplace de un polinomio |
| resultado=Laplace(polinomio) | Guarda la Transformada de Laplace del polinomio en resultado |
| show var | Muestra (imprime) el contenido de *var.* |

**Language development**

1. Translator architecture

La implementación de PLD fue hecha en *Python*. El primer paso fue establecer los *tokens*. Una vez definido los *tokens*, se procedió a construir el *lexer*. Para el desarrollo del *lexer* y *parser* se utilizó la librería PLY de *Python*, al importar *ply.lex* se construye el *lexer*. algo q se pueda aumentar del lex y parser?

Posteriormente se prosigue con la implementación de la función *coef*, que se encarga de extraer los coeficientes del polinomio, para luego pasárselo a la clase *Polynomial*, quien será la encargada de calcular la *Transformada de Laplace* del polinomio.

1. Describe the interfaces between the modules.

**Module:** lex.py

Este módulo se basa en reglas de expresiones regulares. Para utilizarlo, se debe escribir las expresiones regulares para cada *tokens*, como las mostradas a continuación.

import ply.lex as lex

tokens = ('LAPLACE',

'SHOW',

'NUMBER',

'PLUS',

'TIMES')

# reserved words

reserved = {

'show':'SHOW',

'laplace':'LAPLACE',

}

# Regular expression rules for simple tokens

t\_PLUS = r'\+'

t\_MINUS = r'-'

t\_TIMES = r'\\*'

t\_LPAREN = r'\('

# A regular expression rule with some action code

def t\_NUMBER(t):

#r'-?\d+'

r'\d+'

t.value = int(t.value)

return t

**Module** **Yacc**

Este módulo permite crear un programa que tome los "tokens" como entrada y que los reconozca como un lenguaje. Para ello usa la gramática BNF que describe el ensamblaje de los tokens.

Por cada regla se define una función que recibe como parámetro un objeto, como en una especie de lista, pero no se comporta como tal; este objeto contiene los valores de cada símbolo de la regla. Algunos de estos son:

def p\_langfunctions(p):

'langfunction : polynomial'

p[0] = str(p[1])

def p\_langfunctions1(p):

'langfunction : function'

p[0] = str(p[1])

iport ply.yacc as yacc

yacc.yacc()

1. Describe the software development environment used to create the Translator.

**????**

1. Describe the test methodology used during development, and Show programs used to test your translator.

Para probar PLD durante su desarrollo, se usaron dos métodos:

1. **Usando el modo interactivo:** Significa probar cada instrucción desde consola. Por ejemplo si deseábamos probar la función *laplace* desde consola se escribíamos:

**(+\_+) >** laplace(3t^4-2t^2+5)

5.0/s-4.0/s^3+72.0/s^5

1. **Leyendo un file:** Si deseamos probar todas las funciones, para que no sea tedioso ingresar uno a uno cada instrucción, podemos leer un file que contenga todas las posibles instrucciones, y los resultados se muestran por consola.

Ejemplo al leer "prueba.txt"

Aumentar si hay alguna instruccion q no consideré

varnam = 3t^2+t+6

show varnam

res1=laplace(3t^2+t+6)

show res1

res2=laplace 3t^2+t+6

show res2

other=varnam

laplace(3t^2+t+6)

show ("This is a polynomial" varnam) # este no me corrio

#Para leer las instrucciones desde un file

file = open('prueba.txt', 'r')

## Mostramos por pantalla lo que leemos desde el fichero

for line in file:

yacc.parse(line)

## Cerramos el fichero.

infile.close()

#El resultado es:

(+\_+) > varnam = 3t^2+t+6

(+\_+) > show varnam

3t^2+t+6

(+\_+) > res1=laplace(3t^2+t+6)

6.0/s+1.0/s^2+6.0/s^3

(+\_+) > show res1

6.0/s+1.0/s^2+6.0/s^3

(+\_+) > res2=laplace 3t^2+t+6

6.0/s+1.0/s^2+6.0/s^3

(+\_+) > show res2

6.0/s+1.0/s^2+6.0/s^3

(+\_+) > other=varnam

(+\_+) > laplace(3t^2+t+6)

6.0/s+1.0/s^2+6.0/s^3

**Conclusions:**

El desarrollo de este proyecto fue muy importante, porque nos permitió comprender el proceso que implica la creación de un lenguaje de programación. Y a la misma vez logramos entender de mejor manera los pasos que se deben seguir a la hora de crear un lenguaje, y las características que este debe cumplir.....