Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Guadalajara

Documentación Proyecto Final Compiladores

Frases a Código

Integrantes:

Edgar Alfredo Briceño - A01221672
Edgar Javier Diaz - A01229692
Luis Arturo Mendoza - A01362800
Raúl Flores Miramontes - A01224188
Rodrigo Reyes - A01064215

Introducción

Como parte del curso de compiladores se nos ha pedido un proyecto que utilice los principios vistos a fin de aplicar en temáticas reales y actuales. Es por esto que trabajamos con lenguaje natural y su procesamiento a código utilizando la librería NLTK (Natural Language Toolkit).

El lenguaje natural es aquel que usamos de manera cotidiana para escribir o comunicarnos. Esto es lenguaje natural. El propósito de este proyecto es el de convertir oraciones estructuradas que describen una clase con sus métodos y atributos, como si se estuviera haciendo un código, a código funcional en el lenguaje Python.

Desarrollo

Primero se diseñó el compilador junto con los inputs que recibiría y los outputs que pasaría a las siguientes fases de este. El proceso fue algo como lo siguiente:

Léxico:
- Input: Lenguaje natural
- Output: Matriz de sentencias divididas en tokens

Sintáctico:
- Input: Archivo txt de sentencias tokenizadas
- Output: Valor booleano que valida o no el return para continuar

Semántico:
- Input: Matriz de sentencias tokenizadas
- Output: Valor booleano que valida o no el return para continuar

Creador de código:
- Input: Archivo txt de sentencias tokenizadas
- Output: Archivo txt con el código esperado en python

Para asegurar su funcionamiento se creó una gramática, que asegurara que siguiendo un formato particular en el lenguaje natural de entrada, se lograra transmitir de la mejor manera el contenido.

Existe la primera versión, y después la versión final corregida:

1era versión:

Class_Definition -> Noun Verb-to-Be CLASS
Attribute_Definition -> PersonalPronoun AttribVerb("Include", "Own", "Possess", "Have") Attribute
Attribute -> Noun ["=" | "is" (String | Number | Boolean | Pair | Array]
String -> STRSTART ANY STRSTOP
Boolean -> TRUE | FALSE
Pair -> "(" NUMBER "," NUMBER ")"
Array -> "[" [ANYBASIC] "]"
Function_Declaration -> PersonalPronoun MethodVerb("Can", "May") ProperNounCycle
ProperNounCycle -> Noun ["," ProperNounCycle]
Function_Definition -> MethodStart ProperNoun [Function_Parameters] (Function_Body | Function_Print)
Function_Parameters -> PersonalPronoun ParameterVerb("Need", "Require", "Demand") ProperNounCycle
Function_Body -> PersonalPronoun MethodBodyVerb("Use", "Utilize", "Exert", "Employ", "Wield", "Apply") FBody_Cycle
FBody_Cycle: (FunctionCall | Action) ["," FBody_Cycle] [RETURN (BASIC | ProperNoun | Pair)]
FunctionCall -> ProperNoun ["(" ANYBASIC ")"]
ANYBASIC -> BASIC ["," ANYBASIC]
Action -> (ADD | SUBSTRACT | ASSIGN) (BASIC | ProperNoun)
BASIC -> String | Number | Boolean
FUNCTION -> Function_Declaration Function_Definition_Cycle
Function_Definition_Cycle -> Function_Definition ["," Function_Definition_Cycle]
Function_Print -> PRINT AnyTokenCycle
AnyTokenCycle -> AnyToken [AnyTokenCycle]
AnyToken -> (All tokens except "," and End of Sentence)

Versión final:

Class -> Class_Definition Attribute_Definition Functions
Class_Definition -> Noun "ASSIGN" "CLASS"
Attribute_Definition -> "PRP" "OWN" Attribute_Cycle
Attribute_Cycle -> Attribute ["," Attribute_Cycle]
Attribute -> Noun ["ASSIGN" (Number | String | Boolean | Pair | Array)]
Functions -> Function_Declarations Function_Definition_Cycle
Function_Declarations -> "PRP" NounCycle

Function_Definition_Cycle -> Function_Definition ["," Function_Definition_Cycle]
Function_Definition -> "TO" Noun [Function_Parameters] (Function_Body | Function_Print)
Function_Parameters -> "PRP" "PARAMVERB" NounCycle
Function_Print -> "PRP" "PRINT" AnyTokenCycle
Function_Body -> FBody_Cycle ["RETURN" Basic_Type]
FBody_Cycle -> (Function_Call | Action) ["," FBody_Cycle]
Function_Call -> "PRP" "BODYVERB" Noun ["PARL" [Argument_Cycle] "PARR"]
AnyTokenCycle -> AnyToken [AnyTokenCycle]
AnyToken -> (All tokens until "," or "PTRSTOP")
NounCycle -> Noun ["," NounCycle]
Argument_Cycle -> Basic_Type ["," Argument_Cycle]
Action -> "PRP$" Noun ("ADD" | "SUBS" | "ASSIGN") Basic_Type
Basic_Type -> Number | String | Noun | Boolean
Noun -> "NN" | "NNP"
Number -> "CD"
String -> "STRSTART [Content] STRSTOP"
Content -> (anything found inside is valid until STRSTOP is found)
Boolean -> "TRUE" | "FALSE"
Pair -> "PARL" "CD" "PARR"
Array -> "BRACKL" "CD" "BRACKR"

A partir de esto, se trabajó de manera simultánea en los 3 componentes del compilador, más el parser final a código.

Es interesante mencionar que convertir lenguaje natural a código es bastante similar al proceso de un compilador, con la diferencia de que un compilador normalmente recibe código en algún lenguaje, y lo procesa a lenguaje máquina. Aquí hacemos “lo mismo” puesto que el lenguaje natural puede ser visto como un conjunto de reglas gramaticales que lo forman, al igual que un lenguaje de programación.

Pero debemos ser conscientes que el lenguaje natural es bastante ambiguo, y esta ambigüedad es aceptada, muy diferente que lo que se espera de un lenguaje funcional de programación. Es por ello que el grado de dificultad es mayor al momento de procesarlo por una máquina o algoritmo.

Funcionalidad

Para correr el código:

- Clonar el repositorio https://github.com/raulfm94/FrasesACodigoPY
- Acceder al folder “src”
- Ejecutar en terminal “python main.py”
    - El input puede ser modificado en el archivo “input.txt”

Troubleshooting
Si se presentara algún problema, asegurarse de tener NLTK instalado, junto con los paquetes que se necesitan, los más comunes siendo:

- stopwords
- maxent_treebank_pos_tagger
- punkt
- wordnet

Se instalan con el comando: “python -m nltk.downloader xxxx” dónde xxxx es el paquete.

Conclusiones

Rodrigo Reyes
Trabajar con lenguaje natural siempre ha sonado como una tarea monumental, y este proyecto fue una muestra de lo complicado que efectivamente es. Librerias como nltk en Python nos proveen de herramientas y utilidades que facilitan el manejo de texto en lenguaje natural, nos permiten utilizar conceptos y funciones probadas por muchas otras personas, para no tener que reinventar la rueda en un área tan grande, sin embargo esto termina siendo sólo el principio del trabajo. Para poder realizar el proyecto se tuvieron que hacer reglas gramaticales definidas que ofrezcan una cierta flexibilidad para la elección de palabras “reservadas” basadas en verbos, pero sabemos que esto no cubre la complejidad del lenguaje natural real. Ser capaz de manejar el texto en cualquier orden en el que el usuario lo introduzca y con todas las posibilidades de maneras de hablar se convierte en una tarea monumental que en lo personal no parece posible con los métodos “tradicionales” que vemos en otros compiladores, es necesario mucha adaptación, aprendizaje y flexibilidad para poder generar un buen resultado y aún así es aún más complicado convertir algo tan variable en un resultado determinista.

Edgar
Este proyecto final fue una buena oportunidad para practicar cada uno de los tópicos que vimos a lo largo del semestre. Nuestro proyecto en general, se dedica a compilar código de Python a partir del lenguaje natural. Como tal tenemos que utilizar analizadores léxicos, sintácticos y semánticos para poder generar dicho código. A pesar de que probablemente no sea completamente libre de errores, se siente bien saber que pudimos crear nuestro propio compilador.

Luis Mendoza
Fue un proyecto muy interesante porque aprendí cómo crear un compilador desde el principio, viendo el léxico, sintáctico y semántico. Fue muy interesante que cada parte se hizo separada pero con la gramática y todo lo que definimos, al unirlo todo funcionó como se esperaba. Faltan varios detalles por mejorar, pero creo que la versión que se logró fue bastante buena y completa.

Edgar Alfredo
Este proyecto me deja el conocimiento de saber que el análisis de compilador puede ser aplicado a diferentes objetivos y no solamente un compilador, también me deja un poco de sabor de boca de machine learning ya que la herramienta que usamos para el análisis sintáctico utiliza una librería que básicamente es el uso de machine learning. El análisis sintáctico por otro lado me pareció interesante porque el lenguaje español se presta para muchas formas de expresar la misma idea, creo que nuestro análisis sintáctico es básico pero es un buen ejemplo del idioma.

Raul Flores
El proyecto fue tanto un reto, como algo interesante de realizar. Ciertamente suena algo futurista el pensar seriamente en poder convertir lenguaje natural a código realmente funcional, pues eso necesitaría que se elimina la ambigüedad del lenguaje natural, lo que resultaría en reducir y limitar, algo opuesto a lo que sucede con el lenguaje natural, pues éste evoluciona y crece, no se reduce. Quizás en algún momento la humanidad pueda hablar una versión del lenguaje natural compatible con código.