Python Linter Synthetizer

Eduardo Eder Vázquez Salcedo
David Gabriel Palmerín Morales

26 de Noviembre del 2018

1 Introducción

En la disciplina de Ciencias de la Computación, el uso de lenguajes de programación(1) es elemental para la solución de un problema en particular dentro de una computadora. Los lenguajes de programación tienen distintas propiedades sintácticas que los diferencían. Mediante el compilador(2) de cada uno de estos lenguajes se pueden revisar los errores sintácticos(3) dentro de un programa. Eso es en cuestión de lo que una computadora entiende, sin embargo, que el programa esté libre de errores sintácticos no asegura que sea entendible para los demás desarrolladores. Existen buenas prácticas(4) de programación para que el código sea legible mediante una convensión acordada bajo muchos programadores. Un ejemplo es el PEP 8(5) para el lenguaje de programación Python(6). No obstante, cada lenguaje tiene una convención distinta por lo que cada programador tiene que memorizar estas reglas para que mantengan las buenas prácticas. Los *Linters*(7) solucionan este problema dentro de un espacio de trabajo (Emacs, Vim, Sublime Text, Atom, etc.) pero cada espacio de trabajo tiene que definir los *linters* adecuados a las convenciones establecidas. Esto es un problema debido a que a cada momento se crean nuevos lenguajes de programación y hay que definir cada uno de estos *linters* para ayudar al programador.

2 Propuesta de solución

Proponemos la realización de un sintetizador(8) de *linters* que siga las convenciones establecidas del lenguaje deseado. Para crear un nuevo *linter* se necesitarían ejemplos de programas con las convenciones adecuadas a su lenguaje de programación. La salida del sintétizador sería un programa en el lenguaje de programación Python que modifique un programa a su programa equivalente con "buenas prácticas".
Consideramos que la propuesta de este sintetizador es útil para todos aquellos desarrolladores que crean *linters* pues de esta forma, únicamente con ejemplos se puede crear el *linter* necesario para seguir la "buenas prácticas" para el lenguaje de programación en cuestión. De esta forma no sólo ayudamos a los

creadores de *linters* para nuevos lenguajes, sino también a los usuarios para que el *linter* necesario para su lenguaje de programación esté disponible, facilitando así la creación y entendimiento de software.

3 Desarrollo

Implementaremos la idea del proyecto en un sintetizador que origine un programa que cumpla con las reglas establecidas en ejemplos para un un linter de Python. En este proyecto se cubre el cambio de nombres de variables y de identación, pues consideramos que en Python son las principales prácticas de programación. Recordemos que alfabeto utilizado por los nombres de variables en Python es Σ = {a-z, A-Z, 0-9, _}

Usaremos enumeración exhaustiva por medio de la técnica *bottom-up*, por lo que definiremos una gramática la siguiente forma:

$$\text{F} := f_1 \mid f_2 \mid ... \mid f_n \mid F \circ F$$

Donde $f_i : \Sigma^+ \to \Sigma^+$ y son funciones terminales que son las que se encargarán de realizar modificaciones en las cadenas. Estas funciones se definen en las secciones posteriores.
Y además, $F \circ F$ es la composición de funciones, la cual está bien definida gracias a las definiciones de f_i.

El resultado del programa se obtiene en el archivo *new_[nombre_del_archivo].py*, donde *[nombre_del_archivo]* es el que proporciona el usuario a nuestro programa.

Se asume que los ejemplos dados por el usuario son ejemplos válidos de acuerdo a las reglas sintáticas del lenguaje Python.

3.1 Nombres de variables

Lo que trataremos en esta sección son los nombres de variables, pues son parte de los estándares comunes entre las buenas prácticas en python: CamelCase, undescore_case, etc. Para tratar con los nombres de variables primero definiremos su alfabeto y la expresión regular que trata con éstos.
Sabemos que los nombres de variables no pueden ser palabras reservadas y tienen que seguir la siguiente expresión regular: [_a-zA-Z][_a-zA-Z0-9]* ya que ningún nombre puede empezar con algún dígito.
Las funciones de cambio de nombre de variable que trataremos para sintetizar el linter son las siguientes:

- all_upper(name): Cambia todos lo caracteres a mayúsculas.
- all_lower(name): Cambia todos los caracteres a minúsculas.
- first_upper(name): Cambia el primer caracter a mayúscula.

- first_lower(name): Cambia el primer caracter a minúscula.
- to_upper_case(name): Cambia el nombre a CamelCase.
- to_underscore_case(name): Cambia el nombre a underscore_case.
- remove_underscores(name) Cambia el nombre quitando los underscores.
- insert_first_underscore(name): Cambia el nombre agregando un underscore al inicio del nombre.
- insert_last_underscore(name): Cambia el nombre agregando un underscore al final del nombre.

Escogimos sólo trabajar con estas funciones ya que si trabajaramos con cada caracter (ya sea agregando, quitando o cambiando caracteres) nos agregaría una complejidad tremenda en tiempo y como haremos una búsqueda exaustiva, nuestra complejidad será exponencial.
La entrada del sintetizador serán una serie de ejemplos que nos dirán qué reglas tiene que cumplir el linter que se quiere generar. El objetivo hasta el momento del linter es que para todos los nombres de variables se cambien de acuerdo a como se hacen en los ejemplos. Para esto, el linter que se generará deberá iterar por cada una de las líneas del código dado por el usuario y cambiar el nombre de todas las variables, si es que no cumplen con las prácticas en los ejemplos proporcionados.
El sintetizador primero verifica cada ejemplo y va creando programas de hasta k niveles usando la técnica *bottom-up* utilizando las funciones de cambio de variable que fueron definidas, o en su defecto, la composición de éstas. Por cada programa que se va generando se verifica si cumple con los ejemplos y si hay alguno que cumple con todos entonces termina y lo regresa. Una vex encontrado el programa que satisface los ejemplos dados, entonces procedemos a realizar los cambios en el archivo dado por el usuario; usando la expresión regular mencionada para identificadores de variables aplicaremos el programa sintetizada a estos nombres, y sustituimos el resultado de éste en el archivo generado.
Por lo que el programa encontrado es el que será nuestro linter para aquél conjunto de ejemplos proporcionados.

3.2 Espacios en blanco y tabs

Python es un lenguaje de programación en el que es muy importante tener consistencia en el número de espacios o tabs usados, debido a que la ejecución del programa se basa en sus reglas de identación para las estrcturas de control. Debido a esto, consideramos primordial incluir la funcionalidad de cambio de identación para la generación de un *linter*.
El resolver el cambio de identación se realizó de forma muy similar al cambio de variables mencionado anteriormente. Se realiza la composición de funciones temrinales para la modificación de espacios, estas funciones son las siguientes:

- add_space_begin_line_if_no_word(line) Agrega un espacio al inicio si es espacio o tab.
- add_tab_begin_line_if_no_word(line) Agrega un tab al inicio si es espacio o tab.
- change_beggining_to_tab(line) Cambia por un tab el inicio de la línea.
- change_beggining_to_space(line) Cambia por un espacio el inicio de la línea.

Nuevamente, se usa el algoritmo bottom-up de síntesis, por lo que se hace una enumeración exhaustiva de las composiciones de funciones que satisfacen el ejemplo dado por el usuario. De esta forma obtenemos la representación adecuada para el programa recibido. La implementación de esta funcionalidad se obtiene en nuestra función llamada *transformation_generator* el cual nos obtiene una función generada por síntesis para el cambio de variables y otra función para el cambio de identación.Análogamente, leemos el archivo dado por el usuario y buscamos el inicio de cada linea para sustituir por la evaluación de nuestra función generada.

4 Implementación

Se puede encontrar la implementación de este proyecto en la siguiente liga:

https://github.com/EderVs/Python-Linter-Synthetizer

Tenemos los siguientes archivos en el proyecto:

- *default_functions.py* Donde se definen las funciones terminales mencionadas en las secciones anteriores.
- *vars_corrector.py* Contiene la implementación concreta del proyecto.

Ejecución: python vars_corrector.py
El nombre del archivo ejemplo se encuentra hard-coded en el archivo *vars_corrector.py* por lo que se espera un archivo con el nombre *example.py* como input para el programa. Así mismo, los ejemplos de nombres de variables y espacios se encuentran hard-coded en *vars_corrector.py*

5 Conclusiones

Debido a que se realizó la síntetización de un *linter* para un lenguaje en particular nos permitió facilitar la búsqueda de variables usando expresiones regulares, por lo que fue sencillo sustituir los identificadores una vez encontrada la función correcta.

Vimos que es posible realizar fácilmente un linter para un lenguaje de programación, en particular para Python, sin embargo, nos gustaría extender esta idea para realizar un sintetizador de *linters* para cualquier lenguaje de programación dado por el usuario. Como mencionamos anteriormente, el uso de expresiones regulares nos facilitó la sustitución de identficadores de variables, por lo que pensamos que el primer paso que haríamos para generalizar la idea de este proyecto sería realizar un sintetizador de *expresiones regulares*(9), de esta forma una vez que obtenemos la expresión regular podemos resolver el problema de sustituir un nuevo nombre tal y como se hizo en el proyecto presentado.
Además, el problema de sintetizar las funciones que modifiquen cadenas será más difícil, pues nosotros definimos funciones terminales debido a que sabemos el lenguaje de programación usado, sin embargo, esto lo desconocemos en el caso general, por lo que habrá que sintetizar las funciones que cambien los identificadores con una técnica de síntesis distinta a la que se presenta en este documento.

Dicho esto, pensamos que es posible la realización de *linters* para cualquier lenguaje de programación por medio de síntesis, sin embargo, es claro que tendremos que usar técnicas más avanzadas a las presentadas en este proyecto.

Consideramos que este proyecto es de relevancia para la comunidad de de desarrolladores de software, por lo que sería muy interesante ver este proyecto realizado en el caso general que planteamos en esta sección.

References

[1] *What are Computer Programming Languages?*
https://www.computerscience.org/resources/computer-programming-languages
Consultado: 2018-09-27

[2] *Compiler Design — Introduction of Compiler design*
https://www.geeksforgeeks.org/introduction-compiler-design
Consultado: 2018-09-27

[3] *Syntax error*
https://en.wikipedia.org/wiki/Syntax-error
Consultado: 2018-09-27

[4] *Best Coding Practices*
https://en.wikipedia.org/wiki/Best-coding-practices
Consultado: 2018-09-27

[5] *PEP 8*
https://www.python.org/dev/peps/pep-0008/
Consultado: 2018-09-27

[6] *Welcome to Python.org*
https://www.python.org/
Consultado: 2018-09-27

[7] *Linter*
https://en.wikipedia.org/wiki/Lint-(software)
Consultado: 2018-09-27

[8] *Program Synthesis by Sketching*
http://people.csail.mit.edu/asolar/papers/thesis.pdf
Consultado: 2018-09-27

[9] *Expresiones Regulares*
https://en.wikipedia.org/wiki/Regular_expression
Consultado: 2018-11-26