Universidad catolica “Nuestra señora de la Asunción”
Facultad de ciencias y tecnología
Departamento de electrónica e informática

Implementación de un esquema de traducción, que permite la traslación de un código en lenguaje PHP a un código equivalente en lenguaje Python.

Trabajo Práctico Final - Compiladores

Carrera: Ingeniería Informática.
Materia: Compiladores.
Profesores: Ernst Heinrich Goossen y Luis Martinez.
Alumnos: Alain Vega y Mathias Martínez.

Fecha: 07/07/2023

Índice

Introducción

El lenguaje PHP es ampliamente conocido y utilizado para desarrollar aplicaciones web y sistemas dinámicos. Sin embargo en los últimos años Python ha ganado popularidad debido a su sintaxis limpia, su amplia biblioteca estándar y su facilidad de uso.

El objetivo del proyecto consiste en construir un esquema de traducción, que permita la traducción de un código fuente L1 a un código equivalente en un lenguaje L2. Para lo cual el grupo seleccionó que PHP sea L1 y Python sea L2.

Para dicha traslación se hizo uso del lenguaje C, el generador de analizadores léxicos Flex y el generador de analizadores sintácticos Bison.

El archivo con extensión *.l* (*php2python.l*) implementa el analizador léxico, en el cual las expresiones regulares y acciones asociadas fueron descriptas.

El archivo con extensión *.y* (*php2python.y*) implementa el analizador sintáctico, en el cual la gramática y acciones asociadas sintácticas fueron descriptas.

Al utilizar Bison, procesamos una gramática LALR(1), una versión simplificada de la gramática canónica LR(1), donde la segunda L significa que la entrada se lee de izquierda a derecha, la R significa derivaciones por derecha.

Aspectos implementados

Tipos de datos y estructuras	Números, cadenas, booleanos, arreglos unidimensionales (tabla de símbolos)
Operadores	**Aritméticos**: Suma **+** , resta **-** , multiplicación *****, división **/**, módulo **%** , exponenciación ****** **De comparación:** Igualdad **==**, identidad **===**, no igualdad **!=**, no identidad **!==**, mayor que **>**, mayor o igual que **>=**, menor **<**, menor o igual que **<=**, **Lógicos**: Conjunción lógica **&&** y **and** , disyunción lógica **||** y **or** , negación lógica **!** , disyunción lógica exclusiva **xor** **Bit a bit:** Conjunción bit a bit **&** , disyunción bit a bit, **|**, negación bit a bit **~**, disyunción exclusiva bit a bit **^**, corrimiento de bits a la izquierda **<<**, corrimiento de bits a la derecha **>>** **De asignación:** **=**, **+=**, **-=**, ***=**, **/=**, ****=**, **%=**, **.=**, **&=**, **|=**, **^=**, **<<=**, **>>=** **De cadenas:** Concatenación **.** **Otros:** ternario **a?x:y** , casting (**type)expr**
Instrucciones condicionales	if, if-else, if-elif y switch case

Instrucciones de bucle	for, while y foreach
Funciones	Definición de funciones, llamado de funciones, funciones anónimas
Instrucciones de salida	echo, print
Otros	Comentarios de una línea, array_pop, array_push, array_sum,

Cómo utilizar el traductor

El proyecto se encuentra en un repositorio de la página github el cual se puede acceder a través de este enlace: https://github.com/AlainVega/Traductor-PHP-Python

Instalación de dependencias

Para el correcto funcionamiento del traductor es necesario Python versión 3.10 o superior y PHP versión 8.2 o superior.
La guía de instalación de dependencias, compilación y ejecución del traductor están dirigidas para el sistema operativo linux.
En distros basadas en Ubuntu:

Instalación de python

```
sudo apt install python
```

Instalación de php

```
sudo apt install php
```

Instalación de bison y flex

Existen distribuciones de linux que disponen de bison y flex, como Ubuntu, pero existen otras que no, como Manjaro.

```
sudo apt install flex
sudo apt install bison
```

Ahora podemos probar el traductor, no es necesario compilarlo para probarlo, pero también se muestra como se puede realizar dicha compilación.

- ¿Cómo compilar el traductor?

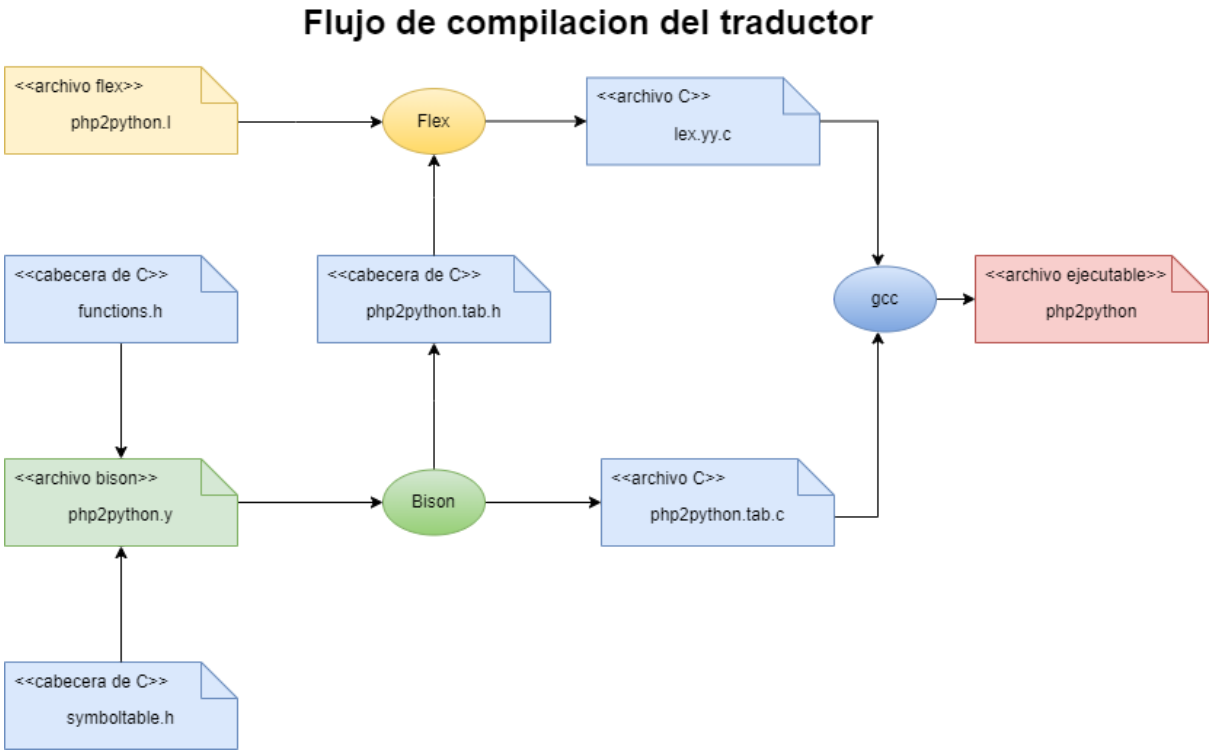
1. Ejecutar el comando `bison -d php2python.y`
2. Ejecutar el comando `flex php2python.l`
3. Ejecutar el comando `gcc php2python.tab.c lex.yy.c -lfl -o php2python`

- ¿Cómo ejecutar el traductor?

4. Ejecutar el comando `./php2python < <ruta_a_archivo.php>`
5. Abrir el archivo llamado output_file.py para ver el resultado en código Python

El proceso de construcción del traductor

A continuación se muestra gráficamente cual es el proceso o flujo actividades para la construcción del traductor.



También se realizó una investigación sobre las sintaxis de ambos lenguajes para poder identificar cómo traducir las diferentes estructuras que tienen. Durante el desarrollo del traductor se crearon los siguientes archivos adicionales:

- functions.h: Contiene las funciones que son utilizadas en las acciones semánticas en *php2python.y*.
- symboltable.h: Contiene la estructura de datos necesaria (en este caso, una lista enlazada) y las funciones que interactúan con ella.

¿Por qué PHP y Python?

Como se explicó en la introducción, la popularidad del lenguaje Python lleva a muchos desarrolladores a migrar proyectos hechos con el lenguaje PHP a Python, aparte de esto, realizar el proyecto sirvió como “excusa” de aprender un poco más la sintaxis de ambos lenguajes que son de interés para ambos integrantes. Además del cierto desafío intelectual que supone dicha traslación.

Además, ambos lenguajes tienen ciertas cualidades en común lo cual facilita el esquema de traducción como tipado dinámico y arrays que tienen tamaños variables. Sin embargo, tienen diferencias en cuanto a su sintaxis. Por ejemplo, la indentación es necesaria en Python para definir un bloque mientras que en PHP se utilizan los caracteres {} para representar el inicio y el final de un bloque como en lenguaje C.

Limitaciones encontradas

La principal limitación que posee el compilador es que reconoce estructuras anidadas pero no las traduce correctamente. Esto es debido a que las líneas dentro de un bloque deben ser indentadas para ser traducidas correctamente pero la estructura de datos que almacena las sentencias de un bloque no almacena la “profundidad” (es decir, la cantidad de tabulaciones necesarias).

Por motivos similares tampoco se implementó la traducción de arrays multidimensionales sin embargo este igual puede ser simulado por medio de la traducción de las funciones *array_push()* y *array_pop()*.

Además, la comprobación de tipos sólo está implementada parcialmente. Para expresiones que son operaciones aritméticas como la suma o la concatenación no se realiza un control de los tipos lo cual permitirá la traducción de PHP que no es válido y no será detectado por el compilador. Sin embargo, la comprobación si está implementada para expresiones que acceden a elementos dentro de un array, ya que en la tabla de símbolos si se almacena si un identificador corresponde a un array. Además, las reglas de ámbito tampoco son almacenadas dentro de la tabla de símbolos, por lo que estas no se comprueban a la hora de analizar el programa.

También existieron ciertas limitaciones en cuanto al manejo de la memoria debido a que se utiliza un stack de *chars* y por ende existe una longitud máxima para las líneas que se almacenan en este. Durante el desarrollo se encontraron casos donde este límite fue alcanzado y por ende tuvo que ser alzado múltiples veces.

Los ciclos *for* fueron traducidos a ciclos *while* equivalentes debido a que los ciclos *for* en Python no tienen una sintaxis que sea fácil de traducir debido a que las 3 partes del *for* de PHP aceptan expresiones pero los *for* de Python sirven para iterar estructuras secuenciales como cadenas, listas y diccionarios.

Por último, la traducción de estructuras condicionales están implementadas pero para las expresiones *else if* solo se realiza la traducción si se utiliza la palabra reservada *elseif* de PHP. Además, no se está implementado el reconocimiento de estructuras condicionales con múltiples *elseif*.

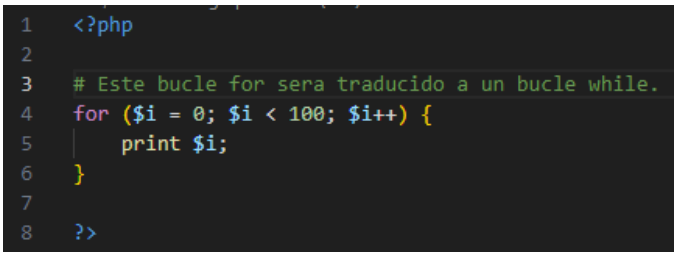
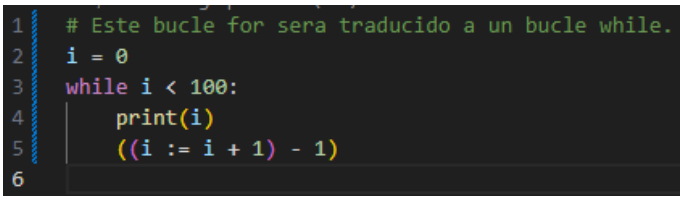
Técnicas de detección de errores implementadas

Para la detección de errores se utiliza la tabla de símbolos que se encuentra en el archivo *symboltable.h* el cual tiene la estructura de datos para la tabla y funciones para realizar ciertas comprobaciones la utilizan, existen 2 principales comprobaciones implementadas.

Una verifica si una variable es un array la cual es utilizada para detectar si al acceder a un elemento de una variable por medio de los operadores de acceso. Por ejemplo, si se reconoce $x[0] pero $x no es un array, se lanzará un error.

La otra detección que se realiza es para la cantidad de argumentos que se utilizan cuando se llama a una función. Para esto se verifica si la cantidad de parámetros pasados coincide con los requeridos cuando se definió la función. Por ejemplo, si se define un procedimiento llamado *proc()* y se reconoce que fue llamado como *proc($x)*, se lanzará un error indicando que esa llamada es incorrecta. Para esto no se toman en cuenta si los argumentos de las funciones tienen valores por defecto. Es decir, si se definió la función *proc($x = 1)* y se reconoce la llamada *proc(),* no se lanzará un error.

Casos de ejemplo

Código de entrada en PHP	Codigo salida en python
	
	

```php
<?php

$var = 0;
while ($var < 10) {
    $var += 1;
}
echo "El valor de var despues del bucle es: ";
print $var;

?>
```

```python
var = 0
while (var < 10):
    var += 1
print("El valor de var despues del bucle es: ")
print(var)
```

```php
<?php

$a = 1 < 100 ? "1 es menor que 100" : "1 es mayor o igual a 100";

print $a;

?>
```

```python
a = "1 es menor que 100" if 1 < 100 else "1 es mayor o igual a 100"
print(a)
```

```php
<?php

function saludar($nombre="Alain", $apellido="Vega") {
    echo "Hola ".$nombre." ".$apellido." como estas?";
}

saludar();
saludar("Carlos");
saludar("Mathias", "Martinez");

?>
```

```python
def saludar(nombre = "Alain", apellido = "Vega"):
    print("Hola " + nombre + " " + apellido + " como estas?")
saludar()
saludar("Carlos")
saludar("Mathias", "Martinez")
```

```php
<?php
$i = 1;
switch ($i) {
    case 0:
        echo "cero";
        break;
    case 1:
        echo "uno";
        break;
    case 2:
        echo "dos";
        break;
    default:
        echo "valor por defecto";
}
?>
```

```python
i = 1
match i:
    case 0:
        print("cero")
    case 1:
        print("uno")
    case 2:
        print("dos")
    case _ :
        print("valor por defecto")
```

A continuación se muestran errores detectados

```php
<?php

$x = 1;
echo $x[0];

?>
```

```
Error: La variable no es un array en la linea 4
```

```php
<?php

function my_func($x, $y) {
    echo $x . $y;
}

my_func($x);

?>
```

```
Error: Cantidad de argumentos incorrectos para la llamada de una funcion en la linea 7
```

Conclusión

El proyecto ha brindado la oportunidad de profundizar los conocimientos en el campo de la compilación y los analizadores léxicos y sintácticos. La experiencia adquirida con Flex y Bison ha ampliado las habilidades del equipo, en el desarrollo de un esquema de traducción y en la comprensión de los procesos fundamentales que ocurren detrás de las escenas en la interpretación de lenguajes de programación.

A lo largo del desarrollo, hemos adquirido un mayor conocimiento de la sintaxis y las estructuras de control de ambos lenguajes, lo que ha permitido realizar conversiones precisas y consistentes.

Durante el desarrollo del proyecto, enfrentamos desafíos al comprender y manejar las diferencias sutiles entre PHP y Python. Sin embargo, a través de la investigación y el análisis exhaustivo, hemos logrado superar algunos de estos obstáculos y generar un código Python equivalente y funcional.

Referencias

- https://www.gnu.org/software/bison/manual/bison.html
- https://westes.github.io/flex/manual/
- https://www.php.net/docs.php
- https://docs.python.org/3/
- Compiladores: principios técnicas y herramientas (1 edición y 2 edición) Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman.
- https://onlinephp.io/
- https://regexr.com/