Análisis de las Características de un Conjunto de Lenguajes de Programación a partir de un Caso Práctico

Grado en Ingeniería Informática
(Mención en Computación)
Diseño de Lenguajes de Programación
2016/2017

Práctica 1:

Simulador de Máquinas de Turing

C

Python

OCaml

Autores:

Miguel Mosquera Pérez

Miguel.mosquera.perez@udc.es

Miguel.mosquera.perez

Adrián Blanco Costas

1. Lenaguaje: C

1.1. **Descripción y características:**

C es un lenguaje de programación imperativo y estructurado, es decir, se define como una programación basada en un estado y sentencias que cambian dicho estado, y dicha programación ha de ser clara y fácil de seguir basándose en la secuenciación, selección e iteración, en otras palabras, que siga un secuencia y no presente saltos incondicionales.

Se sitúa como un lenguaje de medio-alto nivel con un núcleo sencillo pero extensible en funcionalidad por medio de bibliotecas, aunque también permite operaciones de bajo nivel como puede ser el acceso a memoria o inclusión de código ensamblador.

Cuenta de una gran flexibilidad a la hora de programar, dando la opción de no limitarse al paradigma de programación estructurada, que es el estilo más utilizado.

Presenta un tipado débil y estático de los datos, aunque si mantiene cierta seguridad a la hora de operar con ellos e impedir operaciones sin sentido.

También permite la definición de tipos propios y/o tipos de datos agregados.

1.2. Elección:

A razón de la naturaleza intrínseca de las máquinas de Turing, la cual es una máquina que presenta un estado interno que cambia según la entrada que esta reciba, hemos decidido escoger un lenguaje con paradigma imperativo dada la similitud entre dicho paradigma y el funcionamiento de las máquinas de Turing. La flexibilidad que ofrece C ha sido el motivo de la elección del lenguaje.

1.3. Ventajas:

-El propio paradigma imperativo, como se ha comentado, dada la similitud hace que la implementación sea más sencilla y consuma menos tiempo, así como hace más entendible el código.
-El uso de bajo nivel de la memoria, nos ha permitido la implementación de la cinta de la que hace uso la máquina de Turing con punteros, lo que la hace más eficiente resultando en un tiempo de ejecución menor respecto al mismo ejercicio en otro lenguaje que no permita el uso de la memoria a bajo nivel.
-Uso de funcionalidades añadidas al lenguaje por medio de bibliotecas que nos han facilitado la implementación como ha sido la entrada por teclado, lectura de ficheros o salida por pantalla.
-El preprocesador de C (lenguaje preprocesado) nos ha permitido separar en varios ficheros diferentes partes del programa, haciéndolo más modular.
-Al ser un lenguaje de medio nivel, ya que las instrucciones en C resultan en unas pocas instrucciones en código máquina, obtenemos una implementación eficiente.
-El uso de tipos de datos agregados (struct) ha facilitado la implementación de la cola y el uso de esta.

1.4. Desventajas:

-Aunque el uso de punteros ha resultado en un código más eficiente, tiene la contrapartida de que hace más difícil su implementación así como su depuración, demandando más tiempo del programador y una mayor experiencia. También puede ocasionar problemas en tiempo de ejecución no observados en su codificación.

-En el mismo apartado de memoria, la ausencia de un recolector de basura hace que el programador sea el encargado de liberar la memoria y no dilapidarla.

-El tipado estático nos ha limitado al uso de un único tipo en la implementación de la cola en los elementos que esta contiene.

1.5. Implementación:

Se ha realizado una implementación que refleje el funcionamiento real de una máquina de Turing. Dicha implementación se ha realizado sin ningún tipo de librería (a excepción de I/O). Para esto hemos hecho uso de punteros para crear una lista doblemente enlazada para simular una cinta física, y únicamente se han implementado las operaciones que permitiría esta cinta, de tal forma que las realice como lo haría una cinta real. De esta forma conseguimos una simbiosis en la implementación entre realismo y eficiencia.

1.6. Entorno:

Sistema Operativo:

SO: Linux

Distribución: Linux Mint 18 Sarah

Kernel: 4.4.0-21-generic

Arquitectura: x86_64 (64bits)

Compilador & Lenguaje:

GCC:

thread model: posix

gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.2)

Target: x86_64-linux-gnu

Instrucciones:

make

./turing <archivo reglas> [<archivo salida cinta final>]

1.7. Pseudocódigo

```
regla = [estado_actual <Estado>, estado_destino <Estado>, simbolo_leido <Simbolo>, simbolo_escrito <Simbolo>, direccion <Direccion>]
reglas = Lista <Reglas>
entrada = Lista <Simbolo>
cinta = [izquierda Lista <Simbolo>, cabeza <Simbolo>, derecha Lista <Simbolo>]

INICIO(archivo_entrada, archivo_salida)
        SI (archivo_entrada != vacio) ENTONCES
                reglas <- archivo_entrada
                entrada <- leer_teclado()

                SI (entrada = vacio) ENTONCES
                        cinta <- BLANCO
                SINO ENTONCES
                        cinta <- entrada

                estado_actual <- reglas[1].estado_actual
                entrada_actual <- cinta.cabeza
                regla_actual <- reglas[1]
                pasos <- 0

                MIENTRAS (estado_actual != META) Y (regla_actual <= numero_reglas) ENTONCES

                        SI (estado_actual = relga_actual.estado_actual) Y (entrada_actual = regla_actual.simbolo_leido) ENTONCES

                                estado_actual <- regla_actual.estado_destino
                                cinta.cabeza <- relga_actual.simbolo_escrito

                                SI (relga_actual.direccion = IZQUIEDA) ENTONCES
                                        SI (cinta.izquierda = vacio) ENTONCES
                                                cinta.izquierda <- BLANCO
                                        cinta.derecha <- cinta.cabeza + cinta.derecha
                                        cinta.cabeza <- cinta.izquierda[ultimo]

                                SINO Y SI (relga_actual.direccion = DERECHA) ENTONCES
                                        SI (cinta.derecha = vacio) ENTONCES
                                                cinta.derecha <- BLANCO
                                        cinta.izquierda <- cinta.derecha + cinta.cabeza
                                        cinta.cabeza <- cinta.derecha[primero]

                                entrada_actual <- cinta.cabeza
                                regla_actual <- reglas[1]
                                pasos <- pasos + 1

                        SINO ENTONCES

                                regla_actual <- regla_actual + 1


                SI (estado_actual = META) ENTONCES
                        escribir("Accept: yes")
                SINO ENTONCES
                        escribir("Accept: no")

                escribir("Steps: " + pasos)

                SI (archivo_salida != vacio) ENTONCES
                        archivo_salida.linea[0] <- cinta.izquierda
                        archivo_salida.linea[1] <- cinta.cabeza + cinta.derecha
FIN
```

2. Lenaguaje: Python

2.1. **Descripción y características:**

Python es un lenguaje de programación de alto nivel multiparadigma, permitiendo orientación a objetos, imperativa o funcional. Una de sus principales características es que es un lenguaje interpretado, lo que no compila el código en el sentido estricto de la palabra (se generan unos “bytecodes” y luego este es interpretado por una máquina virtual que es la que interactúa con el procesador). Esto permite tanto que Python sea multiplataforma como tipado dinámico o enlace dinámico de métodos, lo que hace que sea más flexible pero también más susceptible a errores en su ejecución. Esta forma de ejecutar el código también hace que no sea tan rápido como sería un lenguaje compilado, aunque de ser necesario puede usarse código en C dentro de Python, como este último permite usarse en C para ofrecer las posibilidades de scripting.
Su forma sencilla e indentada hace que sea un lenguaje fácil tanto de entender como de codificar.

2.2. Elección:

El querer tratar la cinta de la Máquina de Turing o esta misma como algo externo, con su propia funcionalidad, nos ha hecho decantarnos por una programación orientada a objeto como nos permite Python, y así definir dicha cola como un objeto. La facilidad de uso y pequeña curva de aprendizaje que tiene este lenguaje nos ha permitido implementar este ejercicio en un corto período de tiempo, así que no necesita de un complejo entorno para ejecutarse, ya que al ser interpretado y multiplataforma, no necesita de instrucciones de compilación ni prepararlo para una máquina específica.

2.3. Ventajas:

-El paradigma orientado a objetos permite codificar tanto la cinta como la máquina como una clase con las ventajas que POO nos da, esto es, nos permite reutilizar la clase cinta, abstrae también dicha clase del problema con lo que la hace más fácil de mantener o mejorar su rendimiento, y la facilidad a la hora de implementarla al dividir la Máquina de Turing en problemas más sencillos, cada uno con sus funciones y atributos (sentencias o estados).
-Como se ha comentado, el no necesitar de un entorno complicado, solo ha hecho falta instalar Python y ejecutar el código sin necesidad de compilarlo.
-Aunque el uso de punteros ha hecho más eficiente el código en C, en Python hemos hecho el uso de una simple cola para implementar la cinta, lo que ha facilitado enormemente la implementación de la Máquina de Turing.

-La I/O salida del programa, ya sea ficheros, teclado o pantalla es sencilla y no necesita de librerías para ello.
-La ausencia de manejo a bajo nivel de la memoria, e incorporar un recolector de basura permite al programador olvidarte de esta tarea, ya que el uso de la memoria le es transparente a este.
-El tipado dinámico permite no limitar el tipo de dato en la definición de una variable, por ejemplo, lo que nos permite contener en la cinta elementos de cualquier tipo.
-También permite la sobrecarga en las funciones, lo que nos permite por ejemplo definir la función "end" en la máquina de Turing con diferentes argumentos, algo que C no permite.

2.4. Desventajas:

-Al ser un lenguaje interpretado, como se ha comentado, en comparación con C la versión en Python demanda mucho más tiempo respecto al mismo ejercicio que su versión en el lenguaje compilado.
-También pierde eficiencia a la hora de no poder usar punteros, ya que se ha implementado la cinta con una simple cola, que a la hora de editarla (añadir elementos), dicha operación demanda más tiempo que en su versión a más bajo nivel.
-Aunque el tipado dinámico nos dé más libertad a la hora de programar, a la hora de ejecución puede presentar varios problemas debido a la inconsistencia entre tipos.
-En consonancia con esto último, y al no ser un lenguaje compilado, sino que se va "compilando" y ejecutando línea por línea, explicado burdamente, también

pueden aparecer diferentes errores durante su ejecución y más difíciles de depurar debido a la naturaleza de un lenguaje interpretado.

2.5. Implementación:

Una vez decidido usar un paradigma orientado a objetos en relación a lo explicado de la cinta o la máquina de Turing , se ha seleccionado Python por las facilidades que este presenta a la hora de empezar a programar algo "sencillo" y desde 0, y aunque se contaba con más conocimiento de, por ejemplo, Java, la curva de aprendizaje de Python es relativamente pequeña, y el código resultante de esta implementación, la cual se ha llevado de la forma más sencilla posible, ha quedado legible y claro, desde nuestra opinión, más que en su versión en C, la cual también demandó más tiempo para programar.

2.6. Entorno:

Sistema Operativo:

SO: Linux

Distribución: Linux Mint 18 Sarah

Kernel: 4.4.0-21-generic

Arquitectura: x86_64 (64bits)

Compilador & Lenguaje:

Python:

3.5.2 (default, Sep 10 2016, 08:21:44)

[GCC 5.4.0 20160609]

Instrucciones:

./turing.py <archivo reglas> [<archivo salida cinta final>]

[alternativa] python3 turing.py <archivo reglas> [<archivo salida cinta final>]

2.7. Pseudocódigo:

```
regla = [estado_actual <Estado>, estado_destino <Estado>, simbolo_leido <Simbolo>, simbolo_escrito <Simbolo>, direccion <Direccion>]
reglas = Lista <Reglas>
entrada = Lista <Simbolo>
cinta = [izquierda Lista <Simbolo>, cabeza <Simbolo>, derecha Lista <Simbolo>]

INICIO(archivo_entrada, archivo_salida)
        SI (archivo_entrada != vacio) ENTONCES
                reglas <- archivo_entrada
                entrada <- leer_teclado()

                maquina <- MaquinaTuring(entrada, reglas)
                maquina.Iniciar()

                maquina.Terminar(archivo_salida)

FIN
```

```
MaquinaTuring(entrada, reglas)

        MT.cinta = cinta
        MT.relgas = reglas
        MT.estado = Reglas[1].estado_actual
        MT.pasos = 0

    Transicion(estado_actual, entrada_actual)=
        POR CADA REGLA
            SI (estado_actual = relga_actual.estado_actual) Y (entrada_actual = regla_actual.simbolo_leido) ENTONCES

                estado_actual <- regla_actual.estado_destino
                cinta.cabeza <- relga_actual.simbolo_escrito

                SI (relga_actual.direccion = IZQUIEDA) ENTONCES
                    SI (cinta.izquierda = vacio) ENTONCES
                        cinta.izquierda <- BLANCO
                    cinta.derecha <- cinta.cabeza + cinta.derecha
                    cinta.cabeza <- cinta.izquierda[ultimo]

                SINO Y SI (relga_actual.direccion = DERECHA) ENTONCES
                    SI (cinta.derecha = vacio) ENTONCES
                        cinta.derecha <- BLANCO
                    cinta.izquierda <- cinta.derecha + cinta.cabeza
                    cinta.cabeza <- cinta.derecha[primero]

                entrada_actual <- cinta.cabeza
                regla_actual <- reglas[1]
                pasos <- pasos + 1

                Si (MT.estado = META) ENTONCES
                    devolver FALSO
                SINO ENTONCES
                    devolver CIERTO
        FIN REGLAS

        devolver Falso
```

```
MT.Iniciar
        REPETIR HASTA (Transicion(MT.estado, TM.cinta.cabeza) = falso)

MT.Final(archivo_salida)

        SI (TM.estado = META) ENTONCES
                escribir("Accept: yes")
        SINO ENTONCES
                escribir("Accept: no")

        escribir("Steps: " + pasos)

        SI (archivo_salido != vacio) ENTONCES
                archivo_salida.linea[0] <- cinta.izquierda
                archivo_salida.linea[1] <- cinta.cabeza + cinta.derecha
```

3.Lenaguaje: OCaml

3.1. **Descripción y características:**

Ocaml es un lenguaje de programación multiparadigma, es decir se puede utilizar para programar tanto en orientado a objetos como en funcional o imperativo, o una mezcla de estos tres. Es la principal implementación del lenguaje de programación Caml. Tiene una gran librería estándar que lo convierte en un lenguaje muy útil para realizar aplicaciones de diversa índole.
Tiene un sistema de tipos estático, inferencia de tipos, polimorfismo paramétrico, y un recolector de basura que facilita en gran medida la programación.
Permite también la definición de tipos propios.

3.2. Elección:

Dado que se ha de hacer un programa que implemente una máquina de Turing, hacerlo en programación funcional hace más llevadero el diseño de el programa, dividiendo este en funciones aisladas, y ayuda a realizarlo de una manera más modular. Además el pattern matching de Ocaml resulta muy útil a la hora de manejar máquinas que aceptan lenguajes como las de Turing.

3.3. Ventajas:

-El paradigma funcional ayuda a diseñar mentalmente el programa y a aislar funcionalidades.

-El tener un recolector de basura nos ahorra el hecho de tener que estar constantemente controlando la memoria.
-Una biblioteca estandar que ha evitado el hecho de tener que recurrir a bilbioteca externas.
-El pattern matching hace mas natural la parte de reconocer lenguajes.

3.4. Desventajas:

-El paradigma funcional es un paradigma complicado de empezar a utilizar, por eso el inicio es un poco más duro que con otros paradigmas.
-El manejo de los ficheros es complicado y el hecho de utilizar Entrada-Salida provoca que el programa no sea funcional 100%.
-El utilizar listas en lugar de arrays provoca que ciertas operaciones sean menos eficientes que en otros paradigmas.

3.5. Implementación:

La cinta se ha implementado como dos listas, estando el cabezal situado en el primer elemento de la segunda cola.
Se han realizado varias funciones para imitar el comportamiento real de la cinta.
Los tipos State y Simbol se han realizado sobre strings dado que se ha asumido que puede haber lenguajes cuyos elmentos puedan estar formados por combinaciones de letras.

3.6. Entorno:

Sistema Operativo: [SO: Linux \| SO: LINUX

DIST: Linux Ubunut 14.04

KERNEL: 3.13.0-96-generic

ARCH: x86_64 [64 bits]

Compilador & Lenguaje: The OCaml toplevel, version 4.01.0

Instrucciones:

Una vez dentro de la carpeta de Ocaml realizar “make”.

A continuacion:

./turing <Fichero con las reglas> [<Fichero opcional donde escribirá la salida>]

3.7. Pseudocódigo