Tarea 1
Programación avanzada

Emmanuel Peto Gutiérrez

23 de agosto de 2022

1. El paradigma funcional

La programación funcional puede ser vista como un estilo de programar en el cual el método básico de computación es la aplicación de funciones a argumentos. La simplicidad, la claridad y la elegancia son los objetivos clave. Promueve un estilo más abstracto de programar; en contraste con la programación imperativa que está ligada al hardware, en el sentido de que la programación está basada en la idea de cambiar los valores almacenados.

1.1. Características

- **Programas concisos.** Debido a la naturaleza de alto nivel del estilo funcional, los programas son más concisos que en otros paradigmas. Aunque es difícil hacer una comparación objetiva, los programas escritos en el paradigma funcional suelen ser más cortos en líneas de código que los escritos en programación imperativa.
- **Sistema de tipos.** El sistema de tipos permite que los errores de incompatibilidad sean detectados automáticamente, usando un proceso sofisticado llamado *inferencia de tipos*.
- **Funciones recursivas.** Los programas no triviales en el paradigma imperativo involucran cierta clase de repetición o ciclos. En la programación funcional no existen los ciclos, así que el mecanismo de repetición se logra usando funciones recursivas, las cuales están definidas en término de sí mismas.
- **Apareamiento de patrones.** Una forma de definir las funciones en el paradigma funcional es usando apareamiento de patrones (*pattern matching*), en la cual una secuencia de expresiones sintácticas llamadas patrones es usada para elegir entre una secuencia de resultados del mismo tipo. Si el argumento de la función hace *match*[1] con el primer patrón, se

[1] Con hacer *match* se refiere a: el argumento se "parece" al patrón.

elige el primer resultado; en otro caso, si el argummento hace *match* con el segundo patrón, se elige el segundo resultado, y así sucesivamente.

1.2. Lenguajes más conocidos

Los lenguajes más conocidos de programación funcional son:

- Elixir
- Erlang
- Common Lisp
- Haskell
- F#
- Clojure
- Elm
- Racket
- OCaml
- Idris
- PureScript

1.3. Ventajas y desventajas

Ventajas

- **Estructuras de datos.** La notación basada en expresiones permite que el estudio de las estructuras de datos sea presentada en una forma simple y directa, y uno puede ir más lejos describiendo y derivando algoritmos que manipulan estructuras de datos generales de lo que sería posible en un lenguaje de programación convencional.
- **Almacenamiento de funciones.** Se permite almacenar funciones en estructuras de datos.
- **Orden superior.** Los lenguajes funcionales tiene funciones de orden superior. Lo cual significa que las funciones pueden tomar funciones como argumentos y producir otras funciones como resultado.
- **Programas cortos.** Como se mencionó en las características, los programas suelen ser más cortos que en el estilo imperativo.
- **Evaluación perezosa.** Para evaluar expresiones se utiliza una técnica conocida como *evaluación perezosa*, la cual se basa en la idea de que no se debe realizar un cómputo hasta que sea requerido. Esto evita cálculos innecesarios y asegura que el programa termine siempre que sea posible.

Desventajas

- **Entrada y salida de datos (I/O).** La entrada y salida de datos requiere efectos secundarios, así que es inherentemente no funcional. El lenguaje de programación Haskell (por poner un ejemplo) permite leer entrada desde el teclado e imprimir salida en pantalla mediante el uso de *mónadas*, pero es de la parte no funcional del lenguaje.
- **Problemas de terminología.** La programación funcional no es fácil de explicar. La programación funcional tiene su raíz en las matemáticas, así que contiene términos difíciles de entender como *funciones puras* y *transparencia referencial*.
- **Bases de datos.** No es recomendable usarlo para conexiones de bases de datos o servidores.
- **Estado.** Representar un estado en programación funcional no es tan intuitivo como otras operaciones funcionales. Existen formas de representar un estado en la programación funcional, pero no es tan directo como en la programación imperativa. En Haskell hay formas de representar estados usando mónadas.

2. El paradigma lógico

Un programa en el paradigma lógico define datos y luego define las relaciones entre esos datos. El acto de *computar* una respuesta se reemplaza por el acto de *buscar* una respuesta con esos datos y sus relaciones. El lenguaje canónico con este poder es Prolog.

2.1. Características

- **Búsqueda exhaustiva.** El programador sólo especifica los datos y Prolog busca de forma exhaustiva.
- **Hechos y reglas.** Un programa sólo consiste en una serie de hechos y una colección de reglas.
- **Consultas.** Dado un programa, un usuario puede preguntarle a un evaluador de Prolog si un hecho en particular es verdadero o falso. Si la consulta está en el conjunto de hechos entonces es verdadero de forma automática; si no, el evaluador necesita aplicar las reglas para determinar la verdad.
- **Constantes.** En un programa lógico se definen constantes y los hechos relacionan esas constantes.

2.2. Lenguajes más conocidos

- Prolog

2.3. Ventajas y desventajas

Ventajas

- **Escalable.** Base de conocimiento fácilmente escalable.
- **Aplicaciones.** Tiene aplicaciones en la inteligencia artificial, sistemas expertos, reconocimiento del lenguaje natural, entre otras.
- **Verificador.** Puede usarse para codificar un verificador de tipos.

Desventajas

- **Eficiencia.** La búsqueda exhaustiva tiene la desventaja evidente de que es poco eficiente en términos de tiempo de ejecución.
- **Aplicaciones.** Las áreas de aplicación son pocas y muy específicas.
- **Depuración.** Existen muy pocas herramientas de depuración, en su mayoría poco efectivas.
- **Inferencia.** Inferencia limitada por su base de conocimiento.

3. Conclusiones

La diferencia principal entre la programación declarativa y la imperativa está en la forma de escribir los programas. En el paradigma imperativo importa saber cómo se realizan las operaciones paso a paso y cómo se maneja la memoria, mientras que en el declarativo lo más importante es conocer y describir cuál es la entrada y cuál es la salida de las funciones.

En el paradigma imperativo las estructuras se construyen paso a paso, siendo cuidadosos con el manejo de la memoria, mientras que en el declarativo se construyen de forma recursiva. Por ejemplo, una lista ligada en el paradigma imperativo se construye con un nodo que apunta al nodo siguiente, que a su vez apunta a otro nodo siguiente y así sucesivamente; mientras que en el declarativo basta decir: una lista puede ser vacía o puede ser un elemento pegado a otra lista.

Aunque la recursión se puede usar en el paradigma imperativo ésta es opcional, pues cualquier programa recursivo podría escribirse en su versión iterativa (con suficiente ingenio); sin embargo, en el paradigma declarativo, la recursión es la clave para resolver los problemas. Otra diferencia importante es que en el paradigma imperativo la recursión suele programarse con condicionales (if, switch-case), mientras que en el declarativo suele usarse *patern matching*.

Referencias

[1] Bird, R. and Wadler, P., *Introduction to functional programming*, Prentice Hall, 1988.

[2] Hutton, G., *Programming in Haskell*, Cambridge University Press, 2007.

[3] Krishnamurthi, S., *Programming Languages: Application and Interpretation*, Brown University, 2003.

[4] *https://www.spec-india.com/blog/functional-programming-languages*

[5] *https://spin.atomicobject.com/2019/08/29/functional-prog-pros-cons/*

[6] *https://www.incentro.com/es-ES/blog/que-programacion-funcional*

[7] *https://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/logica_teoria/proglogica.html*