Contents

Paradigma funcional

Expresiones imperativas vs. expresiones funcionales

En muchos lenguajes las construcciones básicas son imperativas, Por ejemplo, esta instrucción de asignación:

x: = 5

es una orden que el programador le da a la computadora para guardar el valor 5 en un lugar determinado. Los lenguajes de programación también contienen construcciones declarativas, como la declaración de la función

function f(int x) { return x+1; }

La distinción entre construcciones imperativas y declarativas se basa en que las imperativas cambian un valor y las declarativas declaran un nuevo valor.

La forma más sencilla de entender la diferencia entre declarar una nueva variable y cambiar el valor de una variable ya existente es cambiando el nombre de la variable. **Las variables ligadas,** pueden cambiar de nombre sin cambiar el significado de la expresión.

La asignación imperativa puede introducir **efectos secundarios** porque puede destruir el valor anterior de una variable, **En programación funcional** la asignación imperativa se conoce como **actualización destructiva.** una operación computacional es **declarativa** si, cada vez que se invoca con los mismos argumentos, devuelve los mismos resultados independientemente de cualquier otro estado de computación. Una operación declarativa es:

independiente

sin estado y

determinística

Una consecuencia de estas propiedades es la **transparencia referencial.** Una expresión es transparente referencialmente si se puede sustituir por su valor sin cambiar la semántica del programa. Esto hace que todas las componentes declarativas, se puedan usar como valores en un programa,

Veamos un ejemplo de dos funciones, **una referencialmente transparente** y **otra referencialmente opaca:**

globalValue = 0;

integer function rq(integer x) begin

globalValue = globalValue + 1; return x + globalValue;

end

integer function rt(integer x) begin

return x + 1;

end

La función **rt** es referencialmente transparente, significa que **rt(x) = rt(y)** si x = y. Sin embargo, no podemos decir lo mismo de rq porque usa y modifica una variable global.

la transparencia referencial – nos permite razonar sobre nuestro código de forma que podemos construir programas más robustos,

Propiedades valiosas de los lenguajes funcionales

Los lenguajes funcionales, realizan la mayor parte del cómputo mediante expresiones con declaraciones de funciones.

En algunos casos se usa el término "lenguaje funcional" para referirse a lenguajes que no tienen expresiones con efectos secundarios Para evitar ambig¨uedades entre estos y lenguajes como **Lisp o ML,** llamaremos a estos últimos **"lenguajes funcionales puros".** Los lenguajes funcionales puros pueden pasar el siguiente test:

Dentro del alcance de las declaraciones x1, ..., xn, todas las ocurrencias de una expresión e que contenga sólo las variables x1, ..., xn tendrán el mismo valor.

Como consecuencia los lenguajes funcionales puros tienen una propiedad muy útil: **si la expresión e ocurre en varios lugares dentro de un alcance específico, entonces la expresión sólo necesita evaluarse una vez.** Otra propiedad interesante de los programas declarativos es que son **composicionales.** Un programa declarativo consiste de componentes que pueden ser escritos, comprobados, y probados correctos independientemente de otros componentes Otra propiedad interesante es que **razonar** sobre programas declarativos es sencillo. Es más fácil razonar sobre programas escritos en el modelo declarativo que sobre programas escritos en modelos más expresivos. Estas dos propiedades son importantes tanto para programar en grande como en peque˜no.

Problemas naturalmente no declarativos

La forma más elegante y natural de representar algunos problemas es mediante **estado explícito.** En general, todo programa que realice algún tipo de Input-Output lo hará de forma más natural mediante estado explícito.

También hay tipos de problemas que se pueden programar más eficiente si se hace de forma imperativa. En esos casos podemos llegar a convertir un problema **intratable** con programación declarativa en un problema **tratable.** Este es el caso de un programa que realiza modificaciones incrementales de estructuras de datos grandes, Por esta razón muchos lenguajes funcionales proveen algún tipo de construcción para poder expresar instrucciones imperativas. estas construcciones, se conocen como **mónadas.** Las mónadas son una construcción de un lenguaje que permite crear un alcance aislado del resto del programa. Dentro de ese alcance, se permiten ciertas operaciones con efectos secundarios, Está garantizado que estos efectos secundarios no van a afectar a la parte del programa que queda fuera del alcance de la mónada.

Concurrencia declarativa

Una consecuencia muy valiosa de los programas escritos de forma funcional pura, **sin expresiones con efectos secundarios,** es que se pueden ejecutar de forma concurrente con otros programas con la garantía de que su semántica permanecerá inalterada. paralelizar programas funcionales puros es trivial. En cambio, para paralelizar programas imperativos, es necesario detectar primero las posibles regiones del programa donde puede haber condiciones de carrera, aunque paralelizar programas funcionales sea trivial, es difícil aprovechar este potencial en algunos puntos de un programa tiene sentido introducir paralelismo y en otros no. El paralelismo tiene un peque˜no coste en la creación de nuevos procesos, si la ejecución de un proceso depende del resultado de otro, no se obtiene ningún beneficio al paralelizarlos.