Teorías de los lenguajes de Programación

Práctica curso 2016

Alumno: Diego Díaz Alvarellos (ddiaz136)

DNI: 14310048-T

Contenido

Introducción.

Este documento forma parte de la práctica de Teoría de los lenguajes de programación de la UNED del curso 2015-2016.

No se presenta en este documento el código de la práctica, ya que se encuentra en los archivos Skyline.hs y Skyline.pl, los cuales contienen, respectivamente, el código resultante en Haskell y en Prolog con las funciones descritas en el enunciado propuesto por el equipo docente en el documento "practicaTLP-2015-2016.pdf".

Se divide el documento en dos partes. En la primera parte se presenta una descripción de las funciones/predicados de los lenguajes Haskell y Prolog. En la segunda parte de este documento, se responden a las cuatro preguntas que comparan la práctica programada en **Haskell** con la de **Prolog** y la del Equipo Docente en **Java.**

1. Descripción de las funciones y predicados.

En la **Tabla 1** se describen las funciones de Haskell y se comparan con los predicados equivalentes en Prolog. Las descripciones son aproximadas, es decir, no se pretende detallar tanto las entradas como las salidas obtenidas, más bien una aproximación que intente aclarar el comportamiento general del programa.

Haskell (funciones)	Prolog (predicados)	Descripción
PRIMERA PARTE		
resuelveSkyline	**resuelveSkyline**	Se trata de la función principal que implementa el algoritmo del Skyline mediante un esquema Divide y Vencerás. Recibe una lista de Edificios y devuelve una lista de coordenadas (Skyline).
edificioAskyline	**edificioAskyline**	Es el caso trivial de la recursividad del algoritmo del Divide y vencerás. Es decir, cuando no se realizan nuevas subdivisiones. Recibe un edificio y devuelve un Skyline con un elemento base.
divide	**divide**	Divide una lista en una tupla de dos listas (de igual tamaño si la lista inicial contiene un número par de elementos).
combina	**combina**	Inicializa valores de las alturas de los últimos puntos consumidos a 0 y realiza la combinación.
merge	**merge**	Se especifican los diferentes casos de combinación posibles entre coordenadas Skyline según las especificaciones del enunciado.
SEGUNDA PARTE		
dibujaSkyline	**dibujaSkyline**	Recibe un Skyline y devuelve una cadena de caracteres que representa el dibujo del skyline. En Prolog existe otra versión que imprime por pantalla en vez de agrupar todo en una lista.
dibuja	**dibuja**	Recibe una altura y llama a dibujar línea. Se decrementa 1 a 1 y cuando el valor es 0 llama suelo. En Haskell suelo es llamado desde dibujaSkyline.
alturasSkyline	**alturasSkyline**	Recibe una Skyline y devuelve una lista de alturas.
concatena	**concatena + juntarSolar**	Concatena las subsoluciones de cada bloque de edificios y devuelve el skyline invertido.
quitaUltimo + last2	**last2_and_rest**	Recibe un Skyline y devuelve el mismo Skyline sin el último elemento. En Prolog el mismo predicado devuelve además los dos últimos elementos (ventaja de poder obtener varias soluciones y operar con ellas en un mismo predicado).
solarInicio	**solarInicio**	Ya que recorro la lista a la inversa, el primer elemento hay que compararlo con la posición (0,0). Si la posición en la abcisa del edificio es mayor que 0, habrá un solar antes de los bloques.
bloque	**bloque**	Recibe dos coordenadas y devuelve una lista con la altura del primer elemento repetida tantas veces como la distancia entre los dos puntos.
altura	**altura**	Devuelve la altura máxima de una lista. En Haskell la altura calculada es en la lista de coordenadas (Skyline) pero en Prolog utilizo la propia lista de alturas aprovechando el orden secuencial de las cláusulas (ventaja de poder obtener varias soluciones y operar con ellas en un mismo predicado).
dibujaLinea	**dibujaLinea**	Para generar cada línea del dibujo del skyline según las especificaciones del enunciado. En prolog hay dos versiones, una con write y otra concatenando caracteres.
suelo	**suelo**	Simula el suelo que corresponde con la longitud de la lista de alturas.

Tabla1. Descripción de funciones y predicados.

2. Preguntas optativas del enunciado.

1. Imprimir Skyline con asteriscos y espacios.

El objetivo principal ha sido que el método de resolución sea el mismo para poder así poder comparar con objetividad el resultado entre el código[1] de Haskell y de Prolog. La idea para la obtención de la lista de alturas es algo diferente a la utilizada por el equipo docente en la práctica de Java.

Para explicar el procedimiento se utiliza el ejemplo de coordenadas del enunciado [(3,5), (6,3), (9,2), (10,4), (12,0)]. Se ha construido la lista de alturas de la siguiente manera:

1. Obtención de sub-soluciones.
Se cogen los últimos dos elementos y se calcula su distancia. Esta distancia marca un bloque de edificios por lo que se obtiene una sub-solución de alturas.
En el ejemplo: los dos últimos elementos de la lista son (10,4) y (12,0) por lo que su distancia es 2, la sub-solución es [4,4].

2. Recursión.
Se elimina el último elemento y se llama recursivamente a la función.
Se elimina (12,0). Compara (9,2) y (10,4). Sub-solución [2].
Se elimina (10,4). Compara (6,3) con (9,2). Sub-solución [3,3,3].
Se elimina (9,2). Compara (3,5) con (6,3). Sub-solución [5,5,5].
Se elimina (6,3) y se llega al caso base (un solo término en lista) y se llama a la función para concatenar el solar del inicio [0,0,0] con la última coordenada (3,5).

3. Lista final de alturas.
Se concatenan las sub-soluciones y el resultado final es: [0,0,0,5,5,5,3,3,3,2,4,4].

2. Comparación de la programación en Haskell, Prolog y Java.

1. Lectura del código y comprensión inmediata del algoritmo.

En Haskell la lectura y comprensión de las funciones es inmediata. Como se trata de un lenguaje fuertemente tipado se pueden observar en cada función los valores de entrada y de salida. Al mismo tiempo esta lectura permite poder comprobar el funcionamiento independiente de cada función.

En Prolog la lectura no es tan evidente porque no es un lenguaje fuertemente tipado, así que se suele necesitar ver el cómputo general del programa para entender que es lo que se está haciendo y declarar las variables con nombres que recuerden su funcionalidad. Por otro lado, el programador tiene la ventaja de no tener que preocuparse por los tipos, ya que los predicados serán funcionales con objetos diferentes y es fácil poder rehusar el código. Otra ventaja que tiene Prolog es poder obtener varias salidas al mismo tiempo.

[1] El código no se incluye en este documento.

En Java el código está muy organizado pero la sintaxis en Java es mucho más compleja que en los dos lenguajes anteriores. La necesidad de un programa principal para el control de los datos de entrada, la definición de las clases con sus atributos, constructores y métodos de la programación orientada a objetos, aumentan considerablemente las líneas de código necesarias para llegar a la misma solución.

2. Recursividad.

La ventaja de la recursividad es que al pensar en primera instancia en el caso base y su solución, permite resolver un problema tanto de manera directa como de manera inversa (tal como se ha hecho en la segunda parte del ejercicio). La recursión que utiliza Haskell es fácil e intuitiva de manejar. Los casos se construyen de manera sencilla, incluso para principiantes.

La recursividad en el Prolog es muy parecida a la de Haskell, sin embargo, las variables son locales y necesitan del apoyo de otras variables de apoyo que actúan como almacenamiento; este sistema sin variables globales puede resultar un poco tedioso si no se está acostumbrado al lenguaje pero al mismo tiempo pueden ser de utilidad puesto que los hechos pueden recoger estos valores en un misma regla y utilizarse o no como parte de la solución de un predicado o como entrada a otro predicado dentro de la misma regla.

En la parte de Java solamente se utiliza la recursividad en la llamada principal por lo que comparar el resultado de la parte iterativa con la parte recursiva de Haskell y Prolog es un tanto complicado.

3. Conclusión.

Basándome en lo anteriormente descrito, considerando que no se tiene experiencia nueva en ninguno de los lenguajes anteriores, mi primera elección sería Haskell, en segundo lugar Prolog y en último lugar Java.

3. Predicado *corte (!)* de Prolog y sentencia *break* de Java.

Aunque las utilidades del ***corte (!)*** en Prolog son diversas (confirmación de regla elegida, ahorrar comprobaciones innecesarias, construir la negación conjuntamente con el predicado fail), quiero destacar su uso similar a la sentencia break de Java.

Cuando en una regla, alguno de los antecedentes no se cumple, el resultado lógico a la llamada de dicha regla será FALSE. El predicado no lógico ***corte (!)*** puede situarse tras un antecedente que se utilice como condición de salto para romper un proceso recursivo y seguir la ejecución del programa con el siguiente predicado en el orden de lectura del programa.[2]

[2] Se ha hecho uso de esta utilidad en el predicado merge.

Este mismo efecto se utiliza con la sentencia de ***break*** en el control de bucles de Java. Dentro de la iteración en un bucle, el uso de esta sentencia rompe la iteración de dicho bucle cuando se cumple una condición.

4. Relación de tipos de datos en Haskell, Prolog y Java.

Java	Haskell	Prolog
Clase Skyline: es la clase principal. Instancia un array de edificios.	**Tipo Skyline:** contiene una lista de tipo coordenada.	**No se define globalmente** pero se utiliza localmente una lista de tipo coordenada.
Sub-clase Edificio: contiene constructor con 3 instancias de tipo entero.	**Tipo Edificio:** contiene una tupla con 3 elementos de tipo entero.	**No se define globalmente**, pero localmente se utiliza una tupla con 3 elementos de tipo entero.
Sub-clase Coordenada: contiene constructor con 2 instancias de tipo entero.	**Tipo Coordenada:** contiene una tupla con 2 elementos de tipo entero.	**No se define globalmente**, pero se utiliza una tupla con 2 elementos de tipo entero.
Sub-clase Subproblemas: contiene un constructor que instancia dos array de edificios recibiendo como parámetro un solo array de edificios.	No se define	No se define
Clase Entrada: instancia un fichero con los datos de entrada.	No se define. Se hace una llamada a la función introduciendo los parámetros dinámicamente.	No se define. Se hace una llamada a la función introduciendo los parámetros dinámicamente.
Clase Principal: contiene el método principal del programa.	No se define	No se define

Tabla 2. Relación de las clases de Java con los tipos de datos en Haskell y Prolog.