Taller Colecciones y Expresiones For

Estudiante: Juan Sebastian Getial Getial

Código: 202124644

Asignatura: Fundamentos de Programación Funcional y Concurrente

Docente: Juan Francisco Diaz Frias

Universidad del Valle

Santiago de Cali

2022

Informe de Uso de Colecciones y Expresiones For

Funcion	Uso de Colecciones y Expresiones For
canicasPosiblesFrasco	Si
canicasPorFrasco	Si
mezclarLCanicas	Si
distribucion	Si
agrupaciones	Si

Todas las funciones utilizaron colecciones y expresiones for, ya sea en funciones auxiliares y/o en su propio cuerpo.
El uso de colecciones y expresiones for es una técnica de programación muy poderosa debido a su gran versatilidad a la hora de operar con colecciones, ya que abstrae varias operaciones sobre listas(map, reduce, …) de una manera muy expresiva.

Informe de Corrección

Argumentación Sobre la Corrección

Definiciones

Frasco(f)

$\forall\, f \in Frasco:\ f\ == (a,\ b)\ \wedge\ a, b \in Int$

Distr

$\forall\, d \in Distr:\ d\ == List(f_0,\ ...,\ f_i)\ \wedge f_i \in Frasco$

Función canicasPosiblesFrasco

A continuación se demostrará que:

$\forall\, f, c \in Int: canicasPosiblesFrasco(f,\ c)\ ==\ List((f,\ 0),\ ...,\ (f,\ c)).$

Demostración

$canicasPosiblesFrasco(f,\ c)$

$\rightarrow\quad for\ (e\ <-\ (0\ to\ c).toList)\ yield\ (f, e)$

$\rightarrow\quad for\ (\ e\ <-\ List(0,\ ...,\ c)\ )\ yield\ (f, e)$

$\rightarrow\quad List((f,\ 0),\ ...,\ (f,\ c))\qquad \blacklozenge$

Función canicasPorFrasco

A continuación se demostrará que:

$\forall\, n, c \in Int: canicasPorFrasco(n,\ c)\ ==$

$List(\ List((1,\ 0),\ ...,\ (1,\ c)),\ ...,\ List((n,\ 0),\ ...,\ (n,\ c))\ )$

Demostración

$canicasPorFrasco(n,\ c)$

$\rightarrow\quad for\ (f\ <-\ (1\ to\ n).toList)\ yield\ canicasPosiblesFrasco(f,\ c)$

$\rightarrow\quad for\ (f\ <-\ List(1,\ ...,\ n))\ yield\ List((f,\ 0),\ ...,\ (f,\ c))$

$\rightarrow\quad List(\ List((1,\ 0),\ ...,\ (1,\ c)),\ ...,\ List((n,\ 0),\ ...,\ (n,\ c))\ )\qquad \blacklozenge$

Función mezclarLCanicas

A continuación se demostrará que:

$\forall\ lc \in List[Distr]: mezclarLCanicas(lc)\ ==$

$List(\ List((1,\ 0),\ ...,\ (n,\ 0)),\ ...,\ List((1,\ c),\ ...,\ (n,\ 0))\ )$, donde lc es un resultado de la función $canicasPorFrasco(n,\ c)$, y la lista que se da como resultado, contiene todas las listas que representan el conjunto de permutaciones $cP^{\star}n$.

Para ello, de manera equivalente, se demostrara que:

$\forall\ l1,\ l2 \in List[Distr]: mezclarLC(l1,\ l2)\ ==$

$List(\ List((1,\ 0),\ ...,\ (n,\ 0)),\ ...,\ List((1,\ c),\ ...,\ (n,\ 0))$.

Estado(S_i): $(l1,\ l2)$

Estado Inicial(S_0): $(distrHead,\ lc.tail)$,

$distrHead\ =\ for(f\ <-\ lc.head)\ yield\ f :: Nil\ =\ for(f\ <-\ List(f_1,\ ...,\ f_i))\ yield\ f :: Nil$

$=\ List(List(f_1),\ ...,\ List(f_c))$, donde $lc.head$ es la primera $Distr$.

Estado Final(S_f): $(l1,\ list())$

Respuesta(S_f): $l1$

Trans(S_i): $(\ aux(l1,\ l2.head),\ l2.tail\ )$, donde:

$\forall\ l \in List[Distr]\ d \in Distr : aux(l,\ d)\ ==$

$if(l1.isEmpty)\ for(f\ <-\ d2)\ yield\ f :: Nil$

$else\ for\ (d\ <-\ l1;\ f\ <-\ d2)\ yield\ d :+\ f$

Inv(S_i): $l1$ es la lista mezclada hasta el momento y $l2$ es lista que contiene los elementos restantes a mezclar con $l1$

Inv(S_0).

$Inv(\ (aux(Nil,\ lc.head),\ lc.tail)\ )$

$\rightarrow\quad aux(Nil,\ lc.head)$

$\rightarrow\quad for(f\ <-\ lc.head)\ yield\ f :: Nil$

$\rightarrow\quad List(\ List(f_1),\ ...,\ List(f_i)\ ),\ f_i \in lc.head$

Por lo que $l1\ ==\ List(\ List(f_1),\ ...,\ List(f_i)\ )$ es la lista mezclada en dicho momento y $l2\ ==\ lc.tail$ los elementos restantes a mezclar. ♦

$S_i \neq S_f \land Inv(S_i) \rightarrow Inv(Trans(S_i))$

$Hipótesis: l2 \neq Nil \land l1: Lista\ mezclada\ hasta\ el\ momento \land l2: elementos\ por\ mezclar$

$Hipótesis \Rightarrow Inv(Trans(S_i))$

$Inv(Trans(\ (l1,\ l2)\ ))$

$\rightarrow \quad Inv(\ (aux(l1,\ l2.head),\ l2.tail)\ )$

Donde $aux(l1,\ l2.head)$ da como resultado otra lista mezclada y $l2.tail$ son los elementos restantes por mezclar. ♦

$Inv(S_f) \rightarrow Respuesta(S_f) == List(\ List((1,\ 0),\ ...,\ (n,\ 0)),\ ...,\ List((1,\ c),\ ...,\ (n,\ 0))$

$Hipótesis: l1: Lista\ mezclada\ hasta\ el\ momento \land list(): elementos\ por\ mezclar$

$Hipótesis \Rightarrow Respuesta(S_f) == List(\ List((1,\ 0),\ ...,\ (n,\ 0)),\ ...,\ List((1,\ c),\ ...,\ (n,\ 0))$

$Respuesta(S_f)$

$\rightarrow \quad l1$

Como $list()$ contiene los elementos por mezclar y está vacía, $l1$ es la lista completamente mezclada. ♦

Por otro lado, en cada paso la lista $l2$ del estado reduce su tamaño. Por tanto, está cada vez más cerca de tener tamaño 0 . En consecuencia, después de n iteraciones llega a Nil. Esto implica que:

$mezclarLCanicas(lc) == mezclarLC(aux(Nil,\ lc.head),\ lc.tail)$

$== l1 == List(\ List((1,\ 0),\ ...,\ (n,\ 0)),\ ...,\ List((1,\ c),\ ...,\ (n,\ 0))$

Por lo que finalmente se tiene que:

$\forall\ lc \in List[Distr]: mezclarLCanicas(lc) ==$

$List(\ List((1,\ 0),\ ...,\ (n,\ 0)),\ ...,\ List((1,\ c),\ ...,\ (n,\ 0))\ ),$

donde lc es un resultado de la función $canicasPorFrasco(n,\ c)$, y la lista que se da como resultado, contiene todas las listas que representan el conjunto de permutaciones $cP^{\star}n$.

Función distribucion

A continuación se demostrará que:

$\forall\ m,\ n,\ c \in Int: distribucion(m,\ n,\ c) ==$

$List(\ List((1,\ a_{1_1}),\ ...,\ (n,\ a_{1_n})),\ ...,\ List((1,\ a_{p_1}),\ ...,\ (n,\ a_{p_n}))\ )$, donde para cada lista de tuplas p se cumple que $\sum_{i=1}^{n} a_{p_i} = m$, y $a_{p_i} \leq c$.

Para ello tendremos en cuenta la siguiente definición:

$condicion(List(\ (1,\ a_1),\ ...,\ (n,\ a_n)\ ))\ ==$

$(for\ (f\ <-\ List(\ (1,\ a_1),\ ...,\ (n,\ a_n)\ ))\ yield\ a_i).sum\ ==\ m$

$==\quad List(a_1,\ ...,\ a_n).sum\ ==\ m$

$==\quad a_1 + ... + a_n == m$

Demostración.

$distribucion(m,\ n,\ c)$

$\rightarrow\quad for\ (d\ <-\ mezclarLCanicas(canicasPorFrasco(n,\ c))\ if\ condicion(d))\ yield\ d$

$\rightarrow\quad for\ (d\ <-\ List(\ List((1,\ 0),\ ...,\ (n,\ 0)),\ ...,\ List((1,\ c),\ ...,\ (n,\ 0))\ )$

$if\ condicion(\ List((1,\ a_{p_1}),\ ...,\ (n,\ a_{p_n}))\ )\ yield\ List((1,\ a_{p_1}),\ ...,\ (n,\ a_{p_n}))$

$\rightarrow\quad List(Distr_1,\ ...,\ Distr_w)$, tal que cada $Distr$ cumple la condición definida anteriormente

♦

Función agrupaciones

A continuación se demostrará que:

$\forall\ m \in Int$: $agrupaciones(m)\ ==\ List(L_1,\ ...,\ L_i)$, donde $L_i \in List(Int)$ y representa una manera diferente de agrupar el número m.

Para ello tendremos en cuenta las siguientes definiciones:

$noTieneReps(li: List[Int])$: Da como resultado True si la lista no tiene valores repetidos, de lo contrario da False.

$nMax(m: Int)$: Da como resultado el número máximo de posibles subgrupos en los que puede dividirse m.

Demostración.

$agrupaciones(m)$

→

$agr = for\ (d\ <-\ distribucion(m,\ nMax(m), m))\ yield\ for\ (f\ <-\ d;\ if\ f.\_2\ != \ 0)\ yield\ f.\_2$

Se transforma cada distribución de $distribución(m,\ nMax(m), m))$ en una lista que solo contiene los segundos valores de sus tuplas diferentes de 0 y se lista cada una.

→ $agr2\ =\ for\ (li\ <-\ agr;\ if\ noTieneReps(li))\ yield\ li.toSet$

Se listan solo las listas sin números repetidos convertidas en conjuntos.

→ $for\ (li\ <-\ agr2.distinct)\ yield\ li.toList$

Al hacer $agr2.distinct$ eliminamos los conjuntos repetidos y luego se listan los conjuntos convertidos a lista. ♦

Por lo que finalmente, teniendo en cuenta que la función distribución da un resultado correcto, se tiene que:

$\forall\ m \in Int$: $agrupaciones(m)\ ==\ List(L_1,\ ...,\ L_i)$, donde $L_i \in List(Int)$ y representa una manera diferente de agrupar el número m.

Casos de Prueba

Nota: Para no extender demasiado el documento, a continuación no se pondrán los casos de prueba. Por favor revisarlos en el archivo Pruebas.sc.

En general, todos los casos de prueba demostraron un buen comportamiento para los argumentos recibidos, reflejando un comportamiento homogéneo para cada función. Aunque, cabe recalcar que, en el caso de la función **agrupaciones**, para valores mayores que 12 la función presenta un tiempo de ejecución demasiado alto e incluso llega a saturar la memoria y por consiguiente, generar un error. Esto ocurre seguramente por la ineficacia del método utilizado, sin embargo, no es un tema que le compete a este curso.

Finalmente, teniendo en cuenta lo anteriormente expuesto, podemos concluir que los casos de prueba si son suficientes para confiar en la corrección de su respectivo programa, ya que todos reflejan un comportamiento homogéneo como se demuestra en su argumentación.

Conclusiones

El uso de colecciones y expresiones for es una técnica de programación muy poderosa que abstrae varias operaciones con colecciones de una manera muy expresiva y por lo tanto, muy sencilla. Sin embargo, su uso inadecuado puede llegar a obtener resultados que, aunque lógicamente son correctos, físicamente sobrecargan la memoria de la máquina.