PROGRAMANDO EN HASKELL

Capítulo 9 - El Problema del Conteo Regresivo

¿Qué Es El Conteo Regresivo?

- Un programa popular de televisión Británica basado en preguntas que estuvo al aire desde 1982.

- Inspirado en la versión original en Francés llamado "Des Chiffres et Des Lettres".

- Incluye un juego sobre números al cual nos hemos de referir como el problema del conteo regresivo.

Ejemplo

Utilizando los números

1 3 7 10 25 50

y los operadores aritméticos

+ − * ÷

construye una expresión cuyo valor es 765

Reglas

- Todos los números, incluyendo los resultados intermedios, deben ser números naturales positivos (1,2,3,…).

- Cuando construyas la expression, cada uno de los números originales debe ser usado como máximo una sola vez.

- Evitamos otras reglas usadas en el programa de televisión por razones prácticas.

En nuestro ejemplo, una solución posible es

```
(25-10) * (50+1)   =   765
```

Notas:

- Hay 780 soluciones para este ejemplo.
- Si cambiamos el número objetivo a `831` nos da un ejemplo sin soluciones.

Evaluando Expresiones

Operadores:

```
data Op = Add | Sub | Mul | Div
```

Aplicar un operador:

```
apply :: Op → Int → Int → Int
apply Add x y = x + y
apply Sub x y = x - y
apply Mul x y = x * y
apply Div x y = x `div` y
```

Decide si el resultado de aplicar un operador a dos números positivos naturales es otro tal que:

```
valid :: Op → Int → Int → Bool
valid Add _ _ = True
valid Sub x y = x > y
valid Mul _ _ = True
valid Div x y = x `mod` y == 0
```

Las expresiones:

```
data Expr = Val Int | App Op Expr Expr
```

Regresa el valor del resultado de una expresión, provisto que es un número natural positivo:

```
eval :: Expr → [Int]
eval (Val n)      = [n | n > 0]
eval (App o l r) = [apply o x y | x ← eval l
                                , y ← eval r
                                , valid o x y]
```

O tiene éxito y regresa una lista con un solo elemento, o falla y regresa la lista vacía.

Formalización Del Problema

Regresa todas las formas posibles de elegir cero o mas elementos de una lista:

```
choices :: [a] → [[a]]
```

Por ejemplo:

```
> choices [1,2]

[[],[1],[2],[1,2],[2,1]]
```

Regresa una lista de todos los valores en una expresión:

```
values :: Expr → [Int]
values (Val n)      = [n]
values (App _ l r) = values l ++ values r
```

Decide si una expresión es una solución para una lista de números fuente y un número objetivo:

```
solution :: Expr → [Int] → Int → Bool
solution e ns n = elem (values e) (choices ns)
                  && eval e == [n]
```

Solución de Fuerza Bruta

Regresa una lista de todas las formas posibles de dividir una lista en dos partes no vacías:

```
split :: [a] → [([a],[a])]
```

Por ejemplo:

```
> split [1,2,3,4]

[([1],[2,3,4]),([1,2],[3,4]),([1,2,3],[4])]
```

Regresa una lista de todas las posibles expresiones cuyos valores son precisamente una lista de números proporcionada:

```
exprs :: [Int] → [Expr]
exprs []  = []
exprs [n] = [Val n]
exprs ns  = [e | (ls,rs) ← split ns
               , l       ← exprs ls
               , r       ← exprs rs
               , e       ← combine l r]
```

La función fundamental de este tema.

Combina dos expresiones utilizando cada operador:

```
combine :: Expr → Expr → [Expr]
combine l r =
   [App o l r | o ← [Add,Sub,Mul,Div]]
```

Regresa una lista de todas las posibles expresiones que resuelven una instancia del problema del conteo regresivo:

```
solutions :: [Int] → Int → [Expr]
solutions ns n = [e | ns' ← choices ns
                    , e   ← exprs ns'
                    , eval e == [n]]
```

¿Qué Tan Rápido Es?

Sistema:	2.8GHz Core 2 Duo, 4GB RAM
Compilador:	GHC version 7.10.2
Ejemplo:	`solutions [1,3,7,10,25,50] 765`
Una solución:	0.108 segundos
Todas las soluciones:	12.224 segundos

¿Podemos Mejorarlo?

- Muchas de las expresiones que se han considerado, típicamente serán inválidas – fallan en la evaluación.

- En nuestro ejemplo, solo alrededor de 5 millones de 33 millones de posibles expresiones son válidas.

- Combinar la generación con la evaluación nos permitirá el rechazo oportuno de expresiones inválidas.

Fusionando Dos Funciones

Expresiones válidas y sus respectivos valores:

```
type Result = (Expr,Int)
```

Buscamos definir una función que fusiona la generación y evaluación de expresiones:

```
results :: [Int] → [Result]
results ns = [(e,n) | e ← exprs ns
                    , n ← eval e]
```

Este comportamiento se puede lograr definiendo

```
results []  = []
results [n] = [(Val n,n) | n > 0]
results ns  =
   [res | (ls,rs) ← split ns
        , lx      ← results ls
        , ry      ← results rs
        , res     ← combine' lx ry]
```

donde

```
combine' :: Result → Result → [Result]
```

Combinando los resultados:

```
combine' (l,x) (r,y) =
   [(App o l r, apply o x y)
      | o ← [Add,Sub,Mul,Div]
      , valid o x y]
```

Nueva función que resuelve problemas de conteo regresivo:

```
solutions' :: [Int] → Int → [Expr]
solutions' ns n =
   [e | ns'   ← choices ns
      , (e,m) ← results ns'
      , m == n]
```

¿Ahora Qué Tan Rápido Es?

Ejemplo:

```
solutions' [1,3,7,10,25,50] 765
```

Una solución: 0.014 segundos

Todas las soluciones: 1.312 segundos

Alrededor de 10 veces más rápida en ambos casos.

¿Podemos Mejorarlo?

- Muchas expresiones serán lo mismo en esencia usando propiedades aritméticas, tales como:

```
x * y  =  y * x
```

```
x * 1  =  x
```

- Explotando tales propiedades podríamos reducir considerablemente la búsqueda y los espacios de solución.

Explotando Propiedades

Reforzando el predicado de válidez, tomando en cuenta las propiedades de conmutatividad e identidad:

```
valid :: Op → Int → Int → Bool
valid Add x y  = x ≤ y
valid Sub x y  = x > y
valid Mul x y  = x ≤ y && x ≠ 1 && y ≠ 1
valid Div x y  = x `mod` y == 0 && y ≠ 1
```

¿Ahora Qué Tan Rápido Es?

Ejemplo:

```
solutions'' [1,3,7,10,25,50] 765
```

Válidas: 250,000 expresiones

Aproximadamente 20 veces menos.

Soluciones: 49 expresiones

Aproximadamente 16 veces menos.

Una solución: 0.007 segundos

Alrededor de 2 veces más rápida.

Todas las soluciones: 0.119 segundos

Alrededor de 11 veces más rápida.

De manera general, nuestro programa regresa todas las soluciones en una fracción de segundo. Aproximadamente 100 veces más rápido que la versión original.

Fuente.

Profesor Graham Hutton.
http://www.cs.nott.ac.uk/~pszgmh/

Traducción al español.

Caleb Josue Ruiz Torres.
https://www.calebjosue.com