# Programación Funcional

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

# The countdown problem

# Countdown show

- **Countdown** es un programa que se emite en la televisión británica desde 1982.


- Incluye un juego de números al que nos referiremos como el **countdown problem**.

# Countdown problem

*Dada una secuencia de números y un número objetivo tratar de construir una expresión aritmética que combine uno o más números de la secuencia y cuyo valor sea el número objetivo.*

# Restricciones del problema

- Todos los números son naturales positivos.
- La secuencia son siempre 6 números seleccionados de la lista:

$$[1\,..\,10, 1\,..\,10, 25, 50, 75, 100]$$

- Se permite suma, resta, multiplicación, división y paréntesis.
- Cada número puede ser usado a lo más una vez en la expresión.
- Los resultados intermedios también deben ser naturales positivos.

# Ejemplo

- Número objetivo: 813
- Secuencia de números: 1,10,25,50,75,100
- Posible solución: ((50 * (1 + (25 * (75 - 10)))) / 100
- Número de soluciones: 12
- Video de este exemplo:
  `http://www.youtube.com/watch?v=_JQYYz92-Uk`

# Countdown problem en Haskell

La solución que vamos a ver fue desarrollada por Graham Hutton, profesor de la Universidad de Nottingham, Inglaterra.

Está publicada en el Journal of Functional Programming:

The countdown problem
Graham Hutton
Journal of Functional Programming, 12 (6),
páginas 609-616, Noviembre 2002.

# Restricciones no consideradas

En la búsqueda de soluciones al problema nos vamos a olvidar de algunas restricciones que se tienen en el programa de TV.

- La secuencia debe ser de 6 números seleccionados de la lista:

$$[1\,..\,10, 1\,..\,10, 25, 50, 75, 100]$$

- El número objetivo está siempre en el rango 100..999.
- El tiempo límite es de 30 segundos.
- Es posible dar como solución una expresión que evalúe a un valor aproximado.

# Ejemplo

Otro ejemplo, ahora sin la restricción en la secuencia.

- Número objetivo: 765
- Secuencia de números: 1, 3, 10, 25, 75, 50.
- Posible solución: (25-10)*(1+50)
- Número de soluciones: 808

# Formalización del problema

- Operadores aritméticos:

$$\textbf{data}\ Op = Add \mid Sub \mid Mul \mid Div$$

- Aplicación de operadores:

$$\begin{aligned} &apply :: Op \to Int \to Int \to Int \\ &apply\ Add\ x\ y = x + y \\ &apply\ Sub\ x\ y = x - y \\ &apply\ Mul\ x\ y = x * y \\ &apply\ Div\ x\ y = x\ `div`\ y \end{aligned}$$

# Operaciones válidas

La siguiente función determina cuando es válido aplicar un operador.

$$
\begin{aligned}
&\mathit{valid} :: \mathit{Op} \to \mathit{Int} \to \mathit{Int} \to \mathit{Bool} \\
&\mathit{valid}\ \mathit{Add}\ \_\ \_ = \mathit{True} \\
&\mathit{valid}\ \mathit{Sub}\ x\ y = x > y \\
&\mathit{valid}\ \mathit{Mul}\ \_\ \_ = \mathit{True} \\
&\mathit{valid}\ \mathit{Div}\ \ x\ y = x\ \text{`}\mathit{mod}\text{`}\ y == 0
\end{aligned}
$$

# Expresiones aritméticas

$$\textbf{data}\ \mathit{Expr} = \mathit{Val}\ \mathit{Int} \mid \mathit{App}\ \mathit{Op}\ \mathit{Expr}\ \mathit{Expr}$$

Valores contenidos en una expresión:

$$\begin{aligned}
&\mathit{values} :: \mathit{Expr} \to [\mathit{Int}] \\
&\mathit{values}\ (\mathit{Val}\ n) \quad\ = [n] \\
&\mathit{values}\ (\mathit{App}\ _\ l\ r) = \mathit{values}\ l +\!\!+ \mathit{values}\ r
\end{aligned}$$

# Evaluación de expresiones

La evaluación devuelve una lista porque se está haciendo manejo de errores.

$$
\begin{aligned}
&\mathit{eval} :: \mathit{Expr} \to [\mathit{Int}] \\
&\mathit{eval}\ (\mathit{Val}\ n) \qquad\quad = [n \mid n > 0] \\
&\mathit{eval}\ (\mathit{App}\ \mathit{op}\ l\ r) = [\mathit{apply}\ \mathit{op}\ x\ y \mid x \leftarrow \mathit{eval}\ l \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad , y \leftarrow \mathit{eval}\ r \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad , \mathit{valid}\ \mathit{op}\ x\ y]
\end{aligned}
$$

La técnica de manejo de errores usando listas se conoce como list of successes en donde un error se marca con la lista vacía y el caso de éxito con una lista no vacía con el resultado.

# Funciones combinatorias

- Subsecuencias:

$$
\begin{aligned}
&subs :: [a] \to [[a]] \\
&subs\ [\,] \quad\ = [[\,]] \\
&subs\ (x : xs) = yss +\!\!+ map\ (x:)\ yss \\
&\quad \textbf{where}\ yss = subs\ xs
\end{aligned}
$$

- Intercalado:

$$
\begin{aligned}
&interleave :: a \to [a] \to [[a]] \\
&interleave\ x\ [\,] \quad\ = [[x]] \\
&interleave\ x\ (y : ys) = (x : y : ys) : map\ (y:)\ (interleave\ x\ ys)
\end{aligned}
$$

# Funciones combinatorias (2)

- Permutaciones:

$$
\begin{aligned}
&perms :: [a] \to [[a]] \\
&perms\ [] \quad\quad = [[]] \\
&perms\ (x : xs) = [zs \mid ys \leftarrow perms\ xs, zs \leftarrow interleave\ x\ ys]
\end{aligned}
$$

- Las permutaciones de las subsecuencias:

$$
\begin{aligned}
&choices :: [a] \to [[a]] \\
&choices\ xs = [zs \mid ys \leftarrow subs\ xs, zs \leftarrow perms\ ys]
\end{aligned}
$$

# Test de solución

Usando las definiciones anteriores podemos chequear si una cierta expresión es una solución de una instancia dada del problema.

$$solution :: Expr \to [Int] \to Int \to Bool$$
$$solution\ e\ ns\ n = elem\ (values\ e)\ (choices\ ns)\ \&\&\ eval\ e == [n]$$

- Primero vemos si los números contenidos en la expresión pertenecen a la secuencia de números.
- Luego evaluamos la expresión y vemos si da el número objetivo

# Solución por fuerza bruta

La solución por fuerza bruta genera todas las posibles expresiones que se pueden armar a partir de una secuencia dada de números.

Para ello precisamos la siguiente función que retorna todas las formas de separar una lista en dos listas no vacías.

$$
\begin{aligned}
&split :: [a] \to [([a],[a])] \\
&split\ [\,] \quad\ = [\,] \\
&split\ [\_] \quad\ = [\,] \\
&split\ (x : xs) = ([x], xs) : [(x : ls, rs) \mid (ls, rs) \leftarrow split\ xs]
\end{aligned}
$$

# Solución por fuerza bruta (2)

Generación de las posibles expresiones:

$$
\begin{aligned}
&exprs :: [Int] \to [Expr] \\
&exprs\ [\,] \ = [\,] \\
&exprs\ [n] = [Val\ n] \\
&exprs\ ns\ \ = [e \mid (ls, rs) \leftarrow split\ ns \\
&\qquad\qquad\quad , l \leftarrow exprs\ ls \\
&\qquad\qquad\quad , r \leftarrow exprs\ rs \\
&\qquad\qquad\quad , e \leftarrow combine\ l\ r]
\end{aligned}
$$

donde

$$
\begin{aligned}
&combine :: Expr \to Expr \to [Expr] \\
&combine\ l\ r = [App\ op\ l\ r \mid op \leftarrow [Add, Sub, Mul, Div]]
\end{aligned}
$$

# Solución por fuerza bruta (3)

$$
\begin{aligned}
&\mathit{solutions} :: [\mathit{Int}] \to \mathit{Int} \to [\mathit{Expr}] \\
&\mathit{solutions}\ \mathit{ns}\ n = [e \mid \mathit{ns}' \leftarrow \mathit{choices}\ \mathit{ns} \\
&\qquad\qquad\qquad\qquad , \ e \leftarrow \mathit{exprs}\ \mathit{ns}' \\
&\qquad\qquad\qquad\qquad , \ \mathit{eval}\ e == [n]]
\end{aligned}
$$

# Fusionando generación y evaluación

- La función *solutions* genera todas las posibles expresiones sobre los números dados a pesar que muchas de dichas expresiones van a fallar en la evaluación (por no ser válidas).
- Por ejemplo, hay 33.665.406 posibles expresiones sobre los números 1, 3, 7, 10, 25, 50.
- De estas, solo 4.672.540 (menos del 14 %) son válidas.
- Para mejorar la performance vamos a combinar la generación de expresiones con su evaluación para que se hagan en simultaneo.
- De esta forma, las expresiones que fallen en la evaluación van a ser descartadas en una etapa temprana.

# Combinando generación y evaluación (2)

Definimos un tipo *Result* de expresiones válidas junto con su valor.

$$\textbf{type } \mathit{Result} = (\mathit{Expr}, \mathit{Int})$$

$$\mathit{results} :: [\mathit{Int}] \to [\mathit{Result}]$$
$$\mathit{results}\ \mathit{ns} = [(e, n) \mid e \leftarrow \mathit{exprs}\ \mathit{ns}, n \leftarrow \mathit{eval}\ e]$$

# Combinando generación y evaluación (3)

A partir de *results* se puede derivar la siguiente definición:

$$
\begin{aligned}
&\mathit{results} :: [\mathit{Int}] \to [\mathit{Result}] \\
&\mathit{results}\ [\,] \ = [\,] \\
&\mathit{results}\ [n] = [(\mathit{Val}\ n, n) \mid n > 0] \\
&\mathit{results}\ \mathit{ns} \ = [\mathit{res} \mid (\mathit{ls}, \mathit{rs}) \leftarrow \mathit{split}\ \mathit{ns} \\
&\qquad\qquad\qquad\qquad , \mathit{lx} \leftarrow \mathit{results}\ \mathit{ls} \\
&\qquad\qquad\qquad\qquad , \mathit{ly} \leftarrow \mathit{results}\ \mathit{rs} \\
&\qquad\qquad\qquad\qquad , \mathit{res} \leftarrow \mathit{combine}'\ \mathit{lx}\ \mathit{ly}\,]
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{combine}' :: \mathit{Result} \to \mathit{Result} \to [\mathit{Result}] \\
&\mathit{combine}'\ (l, x)\ (r, y) \\
&\quad = [(\mathit{App}\ \mathit{op}\ l\ r, \mathit{apply}\ \mathit{op}\ x\ y) \mid \mathit{op} \leftarrow [\mathit{Add}, \mathit{Sub}, \mathit{Mul}, \mathit{Div}] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad , \mathit{valid}\ \mathit{op}\ x\ y\,]
\end{aligned}
$$

# Combinando generación y evaluación (4)

$$
\begin{aligned}
&\mathit{solutions}' :: [\mathit{Int}] \to \mathit{Int} \to [\mathit{Expr}] \\
&\mathit{solutions}'\ \mathit{ns}\ n = [e \mid \mathit{ns}' \leftarrow \mathit{choices}\ \mathit{ns} \\
&\qquad\qquad\qquad\qquad , (e, m) \leftarrow \mathit{results}\ \mathit{ns}' \\
&\qquad\qquad\qquad\qquad , m == n]
\end{aligned}
$$

# Explotando propiedades algebraicas

- Muchas de las expresiones generadas por *solutions'* son equivalentes.
- Esto se debe a propiedades que tienen los operadores aritméticos.
- Por ejemplo, $2 + 3$ y $3 + 2$. Lo mismo con $2$ y $2 \div 1$.
- Usaremos dichas propiedades para obtener una mejor solución.

# Propiedades a utilizar

$$
\begin{aligned}
x + y &= y + x \\
x * y &= y * x \\
x * 1 &= x \\
1 * y &= y \\
x \div 1 &= x
\end{aligned}
$$

# Nueva versión de *valid*

- Usando conmutatividad de la suma y el producto requeriremos que los argumentos estén en orden ascendente ($x \leqslant y$).
- La propiedad de identidad del producto y la división se reflejará requiriendo que el argumento apropiado sea no unitario ($/= 1$).

$$
\begin{aligned}
&\mathit{valid} :: \mathit{Op} \to \mathit{Int} \to \mathit{Int} \to \mathit{Bool} \\
&\mathit{valid}\ \mathit{Add}\ x\ y = x \leqslant y \\
&\mathit{valid}\ \mathit{Sub}\ x\ y = x > y \\
&\mathit{valid}\ \mathit{Mul}\ x\ y = x\ /= 1\ \&\&\ y\ /= 1\ \&\&\ x \leqslant y \\
&\mathit{valid}\ \mathit{Div}\ x\ y = y\ /= 1\ \&\&\ x\ `\mathit{mod}`\ y == 0
\end{aligned}
$$

# Solución final

Usando esta nueva versión de *valid*, el caso

$$[1, 3, 7, 10, 25, 50], 765$$

sólo genera 245.644 expresiones, de las cuales solamente 49 son solución (5% y 6% resp. de los números usados por *solutions'*).