Chapter 11 - The Countdown Problem
What Is Countdown?

- A popular quiz programme on British television that has been running since 1982.

- Based upon an original French version called "Des Chiffres et Des Lettres".

- Includes a numbers game that we shall refer to as the countdown problem.
Example

Using the numbers

1  3  7  10  25  50

and the arithmetic operators

+  -  *  ÷

construct an expression whose value is  765
Rules

- All the numbers, including intermediate results, must be positive naturals (1,2,3,...).

- Each of the source numbers can be used at most once when constructing the expression.

- We abstract from other rules that are adopted on television for pragmatic reasons.
For our example, one possible solution is

\[(25-10) \times (50+1) = 765\]

Notes:

- There are 780 solutions for this example.
- Changing the target number to 831 gives an example that has no solutions.
Evaluating Expressions

Operators:

\[
\text{data Op} = \text{Add} \mid \text{Sub} \mid \text{Mul} \mid \text{Div}
\]

Apply an operator:

\[
\begin{align*}
\text{apply } \text{Add} \ x \ y &= x + y \\
\text{apply } \text{Sub} \ x \ y &= x - y \\
\text{apply } \text{Mul} \ x \ y &= x \times y \\
\text{apply } \text{Div} \ x \ y &= x \ `\text{div}` \ y
\end{align*}
\]
Decide if the result of applying an operator to two positive natural numbers is another such:

\[
\begin{align*}
\text{valid} & : \text{Op} \to \text{Int} \to \text{Int} \to \text{Bool} \\
\text{valid Add} \_ \_ & = \text{True} \\
\text{valid Sub} \; x \; y & = x > y \\
\text{valid Mul} \_ \_ & = \text{True} \\
\text{valid Div} \; x \; y & = x \; \text{`mod`} \; y == 0
\end{align*}
\]

Expressions:

\[
\text{data Expr = Val Int | App Op Expr Expr}
\]
Return the overall value of an expression, provided that it is a positive natural number:

\[
\text{eval} :: \text{Expr} \rightarrow \text{[Int]}
\]
\[
\text{eval} \ (\text{Val} \ n) = [n \mid n > 0]
\]
\[
\text{eval} \ (\text{App} \ o \ l \ r) = [\text{apply} \ o \ x \ y \mid x \leftarrow \text{eval} \ l
\]
\[
, \ y \leftarrow \text{eval} \ r
\]
\[
, \ \text{valid} \ o \ x \ y]
\]

Either succeeds and returns a singleton list, or fails and returns the empty list.
Formalising The Problem

Return a list of all possible ways of choosing zero or more elements from a list:

\[
\text{choices} :: [a] \rightarrow [[a]]
\]

For example:

\[
> \text{choices} [1,2]
\]

\[
[[],[1],[2],[1,2],[2,1]]
\]
Return a list of all the values in an expression:

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

Decide if an expression is a solution for a given list of source numbers and a target number:

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

Return a list of all possible ways of splitting a list into two non-empty parts:

\[
\text{split} :: [a] \rightarrow \left([([a],[a])]\right)
\]

For example:

\[
> \text{split [1,2,3,4]}
\]
\[
\left([([1],[2,3,4]),([1,2],[3,4]),([1,2,3],[4])]\right)
\]
Return a list of all possible expressions whose values are precisely a given list of numbers:

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]

The key function in this lecture.
Combine two expressions using each operator:

\[
\text{combine} :: \text{Expr} \rightarrow \text{Expr} \rightarrow [\text{Expr}]
\]

\[
\text{combine } l \text{ } r = [\text{App } o \text{ } l \text{ } r \text{ } l \text{ } o \leftarrow [\text{Add,Sub,Mul,Div}]]
\]

Return a list of all possible expressions that solve an instance of the countdown problem:

\[
\text{solutions} :: [\text{Int}] \rightarrow \text{Int} \rightarrow [\text{Expr}]
\]

\[
\text{solutions } n s \text{ } n = [e \text{ } l \text{ } n s' \leftarrow \text{choices } n s \text{ , } e \leftarrow \text{exprs } n s' \text{ , eval } e == [n]]
\]
How Fast Is It?

System: 1.5GHz Pentium 4 laptop

Compiler: GHC version 5.04.1

Example: solutions [1,3,7,10,25,50] 765

One solution: 0.62 seconds

All solutions: 74.08 seconds
Can We Do Better?

- Many of the expressions that are considered will typically be invalid - fail to evaluate.

- For our example, only around 5 million of the 33 million possible expressions are valid.

- Combining generation with evaluation would allow earlier rejection of invalid expressions.
Fusing Two Functions

Valid expressions and their values:

\[
\text{type Result} = (\text{Expr}, \text{Int})
\]

We seek to define a function that fuses together the generation and evaluation of expressions:

\[
\text{results} :: [\text{Int}] \rightarrow [\text{Result}]
\]
\[
\text{results ns} = [(e,n) \mid e \leftarrow \text{exprs ns}, n \leftarrow \text{eval e}]
\]
This behaviour is achieved by defining

\[
\begin{align*}
\text{results} & \, [\,] = [\,] \\
\text{results} & \, [n] = [(\text{Val} \, n,n) \mid n > 0] \\
\text{results} & \, \text{ns} = \\
& \, [\text{res} \mid (\text{ls},\text{rs}) \leftarrow \text{split} \, \text{ns} \\
& \, , \, \text{lx} \leftarrow \text{results} \, \text{ls} \\
& \, , \, \text{ry} \leftarrow \text{results} \, \text{rs} \\
& \, , \, \text{res} \leftarrow \text{combine}' \, \text{lx} \, \text{ry}]
\end{align*}
\]

where

\[
\text{combine}' :: \text{Result} \rightarrow \text{Result} \rightarrow [\text{Result}]
\]
Combining results:

\[
\text{combine'} \ (l,x) \ (r,y) = \\
\quad [(\text{App } o \ l \ r, \ \text{apply } o \ x \ y) \\
\quad \quad l \ o \leftarrow [\text{Add,Sub,Mul,Div}] \\
\quad \quad , \ \text{valid } o \ x \ y]
\]

New function that solves countdown problems:

\[
\text{solutions'} :: [\text{Int}] \rightarrow \text{Int} \rightarrow [\text{Expr}]
\]

\[
\text{solutions'} \ ns \ n = \\
\quad [e \ l \ ns' \leftarrow \text{choices } ns \\
\quad \quad , \ (e,m) \leftarrow \text{results } ns' \\
\quad \quad , \ m == n]
\]
How Fast Is It Now?

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

One solution: 0.06 seconds

All solutions: 7.52 seconds

Around 10 times faster in both cases.
Can We Do Better?

- Many expressions will be essentially the same using simple arithmetic properties, such as:
  \[ x \times y = y \times x \]
  \[ x \times 1 = x \]

- Exploiting such properties would considerably reduce the search and solution spaces.
Exploiting Properties

Strengthening the valid predicate to take account of commutativity and identity properties:

\[
\text{valid} \quad :: \text{Op} \rightarrow \text{Int} \rightarrow \text{Int} \rightarrow \text{Bool}
\]

\[
\begin{align*}
\text{valid Add } x \ y &= \text{True} \quad x \leq y \\
\text{valid Sub } x \ y &= x > y \\
\text{valid Mul } x \ y &= \text{True} \quad x \leq y \land x \neq 1 \land y \neq 1 \\
\text{valid Div } x \ y &= x \ `\text{mod}` \ y == 0 \quad \land y \neq 1 
\end{align*}
\]
How Fast Is It Now?

Example: solutions" [1,3,7,10,25,50] 765

Valid: 250,000 expressions

Solutions: 49 expressions

Around 20 times less.

Around 16 times less.
One solution: 0.03 seconds

Around 2 times faster.

All solutions: 0.80 seconds

Around 9 times faster.

More generally, our program usually produces a solution to problems from the television show in an instant, and all solutions in under a second.