Chapter 9 - 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

\[+ \ - \ * \ \div\]

construct an expression whose value is \(765\)
Rules

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

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

z 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:

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

Operators:

data Op = Add | Sub | Mul | Div

Apply an operator:

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 if the result of applying an operator to two positive natural numbers is another such:

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

\[
\text{valid Add } \_ \_ = \text{True}
\]

\[
\text{valid Sub } x \ y = x > y
\]

\[
\text{valid Mul } \_ \_ = \text{True}
\]

\[
\text{valid Div } x \ y = x \ `\text{mod}` \ y == 0
\]

Expressions:

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

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

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:

```
choices :: [a] → [[a]]
```

For example:

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

\[
\begin{align*}
\text{values :: Expr} & \rightarrow [\text{Int}] \\
\text{values (Val n)} & = [n] \\
\text{values (App _ l r)} & = \text{values l} \ ++ \ \text{values r}
\end{align*}
\]

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

\[
\begin{align*}
\text{solution :: Expr} & \rightarrow [\text{Int}] \rightarrow \text{Int} \rightarrow \text{Bool} \\
\text{solution e ns n} & = \text{elem (values e) (choices ns)} \ \\
& \quad \text{&& eval e == [n]}
\end{align*}
\]
**Brute Force Solution**

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

```haskell
split :: [a] → [(a, a)]
```

For example:

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

([(1), (2,3,4)], [(1,2), (3,4)], [(1,2,3), (4)])
```
Return a list of all possible expressions whose values are precisely a given list of numbers:

\[
\text{exprs} :: [\text{Int}] \rightarrow [\text{Expr}]
\]
\[
\text{exprs} \ [\ ] = [\ ]
\]
\[
\text{exprs} \ [n] = [\text{Val} \ n]
\]
\[
\text{exprs} \ ns = [e \mid (ls,rs) \leftarrow \text{split} \ ns,
\text{exprs} \ ls, \text{exprs} \ rs, \text{combine} \ l \ r]
\]

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

```haskell
combine :: Expr -> Expr -> [Expr]
combine l r = [App o l r | o <- [Add, Sub, Mul, Div]]
```

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

```haskell
solutions :: [Int] -> Int -> [Expr]
solutions ns n = [e | ns' <- choices ns, e <- exprs ns', eval e == [n]]
```
How Fast Is It?

System: 2.8GHz Core 2 Duo, 4GB RAM

Compiler: GHC version 7.10.2

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

One solution: 0.108 seconds

All solutions: 12.224 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:

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

\[
\text{results } [] = [] \\
\text{results } [n] = [(\text{Val } n, n) | n > 0] \\
\text{results } ns = \\
[\text{res } | (ls, rs) \leftarrow \text{split } ns \\
, lx \leftarrow \text{results } ls \\
, ry \leftarrow \text{results } rs \\
, res \leftarrow \text{combine'} lx ry]
\]

where

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

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

New function that solves countdown problems:

```haskell
solutions' :: [Int] → Int → [Expr]
solutions' ns n = 
  [e | ns' ← choices ns 
    , (e,m) ← results ns' 
    , m == n]
```
How Fast Is It Now?

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

One solution: 0.014 seconds

All solutions: 1.312 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:

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
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: 0.007 seconds

All solutions: 0.119 seconds

More generally, our program usually returns all solutions in a fraction of a second, and is around 100 times faster that the original version.