Chapter 13 - 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.
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:

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

Apply an operator:

```haskell
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:

\[
\text{data Expr } = \text{Val Int} \mid \text{App Op Expr Expr Expr}
\]
Evaluate 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:

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

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

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

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

\[
\text{split :: } [a] \rightarrow [[[a],[a]]]
\]

For example:

\[
> \text{split [1,2,3,4]}
\]

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

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

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

```
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:

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

\[
\text{results ns} = [(e,n) | e \leftarrow \text{exprs ns}, n \leftarrow \text{eval e}]
\]

type Result = (Expr, Int)

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

\[
\text{results :: [Int] \rightarrow [Result]}
\]

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

```haskell
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]
```

where

```haskell
combine' :: Result -> Result -> [Result]
```
Combining results:

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

New function that solves countdown problems:

\[
solutions' :: [\text{Int}] \rightarrow \text{Int} \rightarrow [\text{Expr}]
solutions' ns n = \\
\quad [e \mid ns' \leftarrow \text{choices } ns \\
\quad , (e,m) \leftarrow \text{results } ns' \\
\quad , m == n]
\]
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:

\[
\begin{aligned}
\text{valid} & : \text{Op} \rightarrow \text{Int} \rightarrow \text{Int} \rightarrow \text{Bool} \\
\text{valid Add} \ x \ y & = x \leq y \\
\text{valid Sub} \ x \ y & = x > y \\
\text{valid Mul} \ x \ y & = x \leq y \land x \neq 1 \land y \neq 1 \\
\text{valid Div} \ x \ y & = x \ `\text{mod}` \ y = \ 0 \land y \neq 1
\end{aligned}
\]
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

All solutions: 0.80 seconds

Around 2 times faster.

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.