Adapted from Graham's Lecture slides.

# 第九章: 倒计时问题



## What Is Countdown?

A popular <u>quiz programme</u> on British television that has been running since 1982.

Based upon an original <u>French</u> version called "Des Chiffres et Des Lettres".

Includes a numbers game that we shall refer to as the <u>countdown problem</u>.





#### Using the numbers



#### and the arithmetic operators



# construct an expression whose value is 765



#### **Rules**

- All the numbers, including intermediate results, must be <u>positive naturals</u> (1,2,3,...).
- Each of the source numbers can be used at <u>most once</u> when constructing the expression.
- We <u>abstract</u> from other rules that are adopted on television for pragmatic reasons.



For our example, one possible solution is

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

Notes:

■ There are <u>780</u> solutions for this example.

Changing the target number to 831 gives an example that has <u>no</u> solutions.



#### **Evaluating Expressions**

**Operators:** 

Apply an operator:

apply :: Op  $\rightarrow$  Int  $\rightarrow$  Int  $\rightarrow$  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:

valid :: 
$$0p \rightarrow Int \rightarrow Int \rightarrow Bool$$
  
valid Add \_ \_ = True  
valid Sub x y = x > y  
valid Mul \_ = True  
valid Div x y = x `mod` y == 0

Expressions:

7

data Expr = Val Int | App Op Expr Expr



Return the overall value of an expression, provided that it is a positive natural number:

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

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



# **Combinatorial functions**

-- subs : returns all subsequences of a list.

-- interleave : returns all possible ways of inserting a new element into a list.

interleave :: a -> [a] -> [[a]] > interleave 1 [2,3,4]
interleave x [] = [[x]] [[1,2,3,4],[2,1,3,4],[2,3,1,4],[2,3,4,1]]
interleave x (y:ys) = (x:y:ys) : map (y:) (interleave x ys)

-- perms : returns all permutations of a list.

| perms | :: [a] | -> [[a]] | > perms [1,2,3]                                   |
|-------|--------|----------|---|
| perms | []     | = [[]]   | [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]] |
| perms | (x:xs) | = concat | (map (interleave x) (perms xs))                   |

#### **Combinatorial functions**

-- choices : return a list of all possible ways of choosing zero or more elements from a list in any order:

choices :: [a] -> [[a]]
choices = concat . map perms . subs

> choices [1,2,3]
[[],[3],[2],[2,3],[3,2],[1],[1,3],[3,1],[1,2],[2,1],
[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]

## **Formalising The Problem**

-- values : return a list of all the values in an expression.

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

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

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

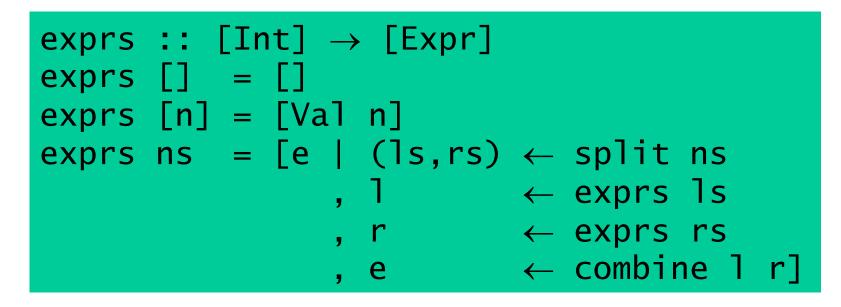
# **Brute Force Solution**

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

```
split :: [a] -> [([a],[a])]
split [] = []
split [_] = []
split (x:xs) = ([x],xs) : [(x:ls,rs) | (ls,rs) <- split xs]</pre>
```

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

-- exprs : return a list of all possible expressions whose values are precisely a given list of numbers



-- combine : 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]  $\rightarrow$  Int  $\rightarrow$  [Expr] solutions ns n = [e | ns'  $\leftarrow$  choices ns , e  $\leftarrow$  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 <u>invalid</u> - fail to evaluate.
- For our example, only around <u>5 million</u> of the 33 million possible expressions are valid.
- Combining generation with evaluation would allow <u>earlier rejection</u> of invalid expressions.



# **Fusing Two Functions**

Valid expressions and their values:

type Result = (Expr,Int)

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

 $\begin{array}{rll} \mbox{results :: [Int]} \rightarrow [\mbox{Result]} \\ \mbox{results ns = [(e,n) & | & e \ \leftarrow & exprs ns \\ & & , & n \ \leftarrow & eval \ e] \end{array}$ 





This behaviour is achieved by defining

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

where

combine' :: Result  $\rightarrow$  Result  $\rightarrow$  [Result]



Combining results:

combine' (l,x) (r,y) =  
[(App o l r, apply o x y)  

$$\mid o \leftarrow [Add, Sub, Mul, Div]$$
  
, valid o x y]

#### New function that solves countdown problems:

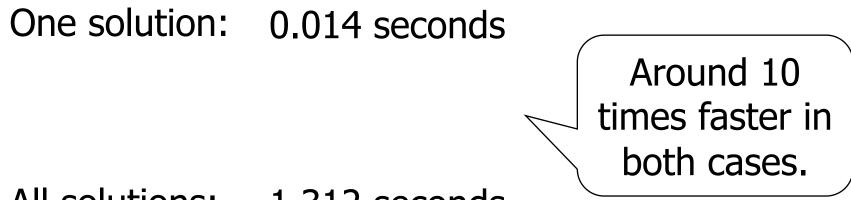
solutions' :: [Int]  $\rightarrow$  Int  $\rightarrow$  [Expr] solutions' ns n = [e | ns'  $\leftarrow$  choices ns , (e,m)  $\leftarrow$  results ns' , m == n]





## **How Fast Is It Now?**









## **Can We Do Better?**

Many expressions will be <u>essentially the same</u> using simple arithmetic properties, such as:

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



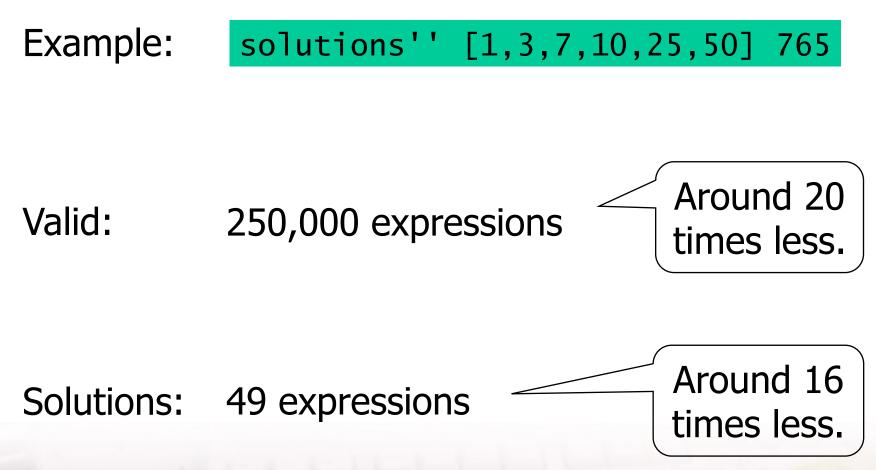
# **Exploiting Properties**

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

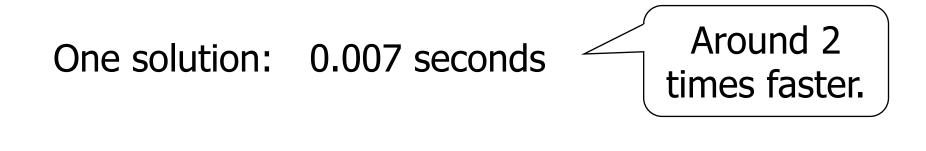
valid ::  $0p \rightarrow Int \rightarrow Int \rightarrow Bool$ valid Add x y =  $x \le y$ valid Sub x y = x > y valid Mul x y =  $x \le y \&\& x \ne 1 \&\& y \ne 1$ valid Div x y = x `mod` y ==  $0 \&\& y \ne 1$ 



### **How Fast Is It Now?**







All solutions: 0.119 seconds

Around 11 times faster.

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





#### 9-1.

Modify the final program to:

- a. allow the use of exponentiation in expressions;
- b. produce the nearest solutions if no exact solution is possible;
- c. order the solutions using a suitable measure of simplicity.

