# 计算概论A—实验班 函数式程序设计 Functional Programming

胡振江,张伟

北京大学 计算机学院 2023年09~12月

#### 第9章: An Example: The Countdown Problem

Adapted from Graham's Lecture slides





## What Is Countdown?

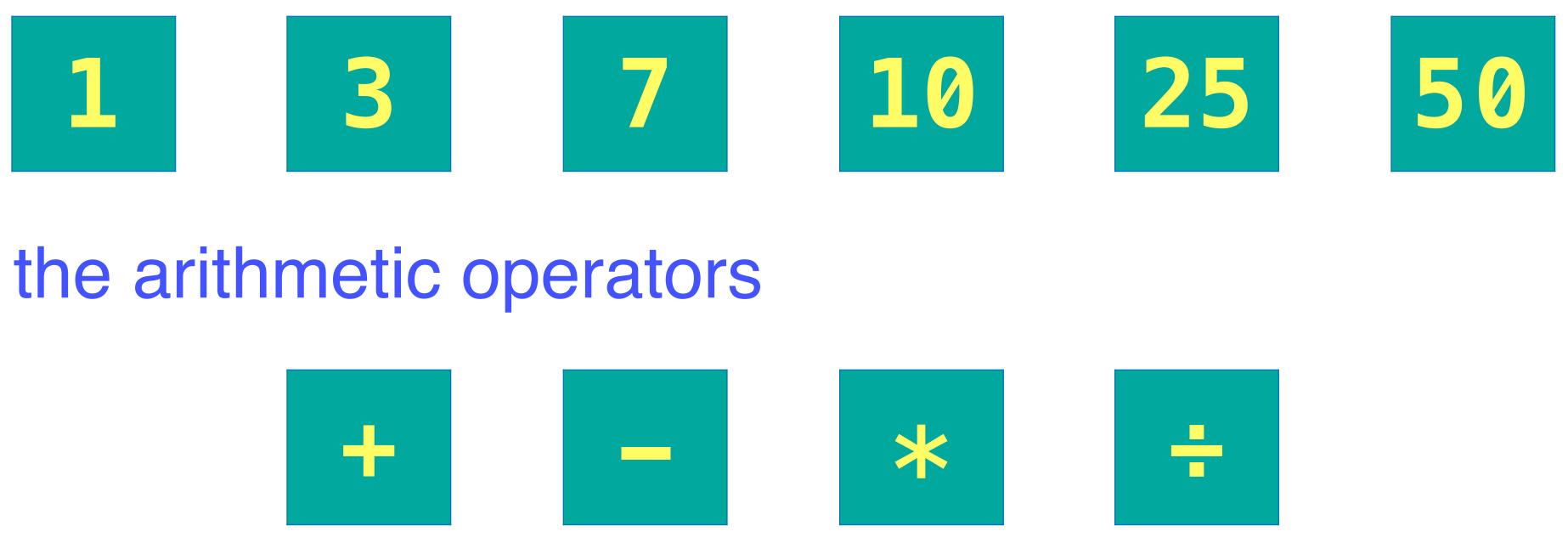
- A popular quiz programme on British television that has been running since 1982.
- et Des Lettres".
- Includes a numbers game that we shall refer to as the countdown problem.

Based upon an original French version called "Des Chiffres

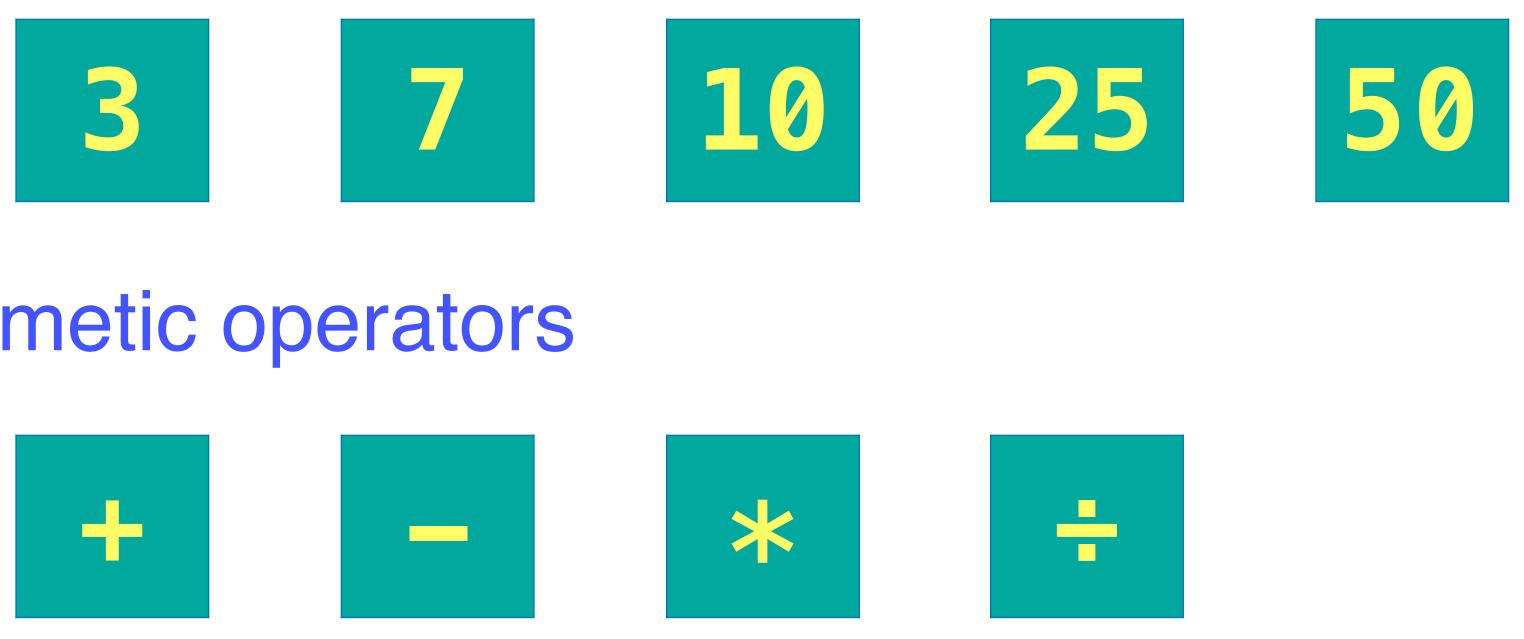


### Countdown: An example

#### Using the numbers



#### and the arithmetic operators



#### construct an expression whose value is



- positive naturals  $(1, 2, 3, \ldots)$ .
- 2. Each of the source numbers can be used at most once when constructing the expression.



# 1. All the numbers, including intermediate results, must be

#### Countdown: The example





\* There are 780 solutions for this example.

\* Changing the target number to that has no solutions.

# **831** gives an example





#### A type for Operators:

#### Apply an operator:

apply :: Op -> Int -> Int -> Int apply Add x y = x + yapply Sub x y = x - yapply Mul x y = x \* yapply Div x y = x div y

#### Evaluating Expressions

#### data Op = Add | Sub | Mul | Div deriving (Show)







# natural numbers is another such:

valid Add \_ \_ = True valid Sub x y = x > yvalid Mul \_ = True valid Div x y = x `mod` y == 0

#### A type for Expressions:

Decide if the result of applying an operator to two positive

# valid :: Op -> Int -> Int -> Bool

#### data Expr = Val Int | App Op Expr Expr deriving (Show)

a positive natural number:

#### eval :: Expr -> [Int] eval (Val n) = [n | n > 0]eval (App ol r) = [apply o x y | x < - eval l, y <- eval r , valid o x y]

#### Return the overall value of an expression, provided that it is

• Either: succeeds and returns a singleton list • Or: fails and returns the empty list







#### Returns all subsequences of a list.

## subs :: [a] -> [[a]] subs [] = [[]] subs (x:xs) = let yss = subs xs

#### > subs [1,2,3] [[],[3],[2],[2,3],[1],[1,3],[1,2],[1,2,3]]

in yss ++ map (x:) yss

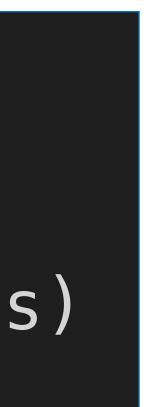
interleave :: a -> [a] -> [[a]] interleave x [] = [[x]]

> interleave 1 [2,3,4]

- Returns all possible ways of inserting a new element into a list.
- interleave x (y:ys) = (x:y:ys) : map (y:) (interleave x ys)

[[1,2,3,4],[2,1,3,4],[2,3,1,4],[2,3,4,1]]







Returns all permutations of a list.

#### perms :: [a] -> [[a]] perms [] = [[]] perms (x:xs) = concat \$ map (interleave x) (perms xs)

#### > perms [1,2,3] [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]





#### Return a list of all possible ways of choosing zero or more elements from a list in any order.

# choices :: [a] -> [[a]]

> 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]]

- choices = concat . map perms . subs

## Formalising The Problem

#### 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 = (values e) `elem` (choices ns) & eval e == [n]

## Brute Force Solution

#### 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) : [ 0

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

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

#### [([1],[2,3,4]),([1,2],[3,4]),([1,2,3],[4])]



## Brute Force Solution

# precisely a given list of numbers.

#### exprs :: [Int] -> [Expr] exprs [] = [] exprs [n] = [Val n] exprs ns = [e | (ls,rs) <- split ns

# combine :: Expr -> Expr -> [Expr]

- Return a list of all possible expressions whose values are

  - l <- exprs ls
  - r <- exprs rs
  - e <- combine l r]
  - combine l r = [App o l r | o <- [Add,Sub,Mul,Div]]

### Brute Force Solution

# 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]]</pre>



Hardware:	2.8GHz
Compiler:	GHC ver
Example:	solutio
One solution:	0.108 s
All solutions:	12.224 s

### How Fast Is It?

- Core 2 Duo, 4GB RAM
- sion 7.10.2
- ns [1,3,7,10,25,50] 765
- econds
- econds

#### 如果在ghci中运行,时间估计会增加一个数量级

invalid - fail to evaluate.

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

rejection of invalid expressions.

#### Can We Do Better?

#### Many of the expressions that are considered will typically be

- Combining generation with evaluation would allow earlier



#### Fusing generation and evaluation

#### A type for Valid expressions and their values:

A function without fusion

#### results :: [Int] -> [Result] results ns = [(e,n) | e <- exprs ns , n < - eval e]

#### type Result = (Expr, Int)

## Fusing generation and evaluation

#### A function without fusion

#### A function with fusion

results :: [Int] -> [Result] results [] = [] results [n] = [(Val n, n) | n > 0]results ns = [res | (ls,rs) <- split ns

combine' :: Result -> Result -> [Result]

results :: [Int] -> [Result] results ns = [(e,n) | e <- exprs ns , n <- eval e]

- lx <- results ls
- , ry <- results rs
- , res <- combine' lx ry]
- combine' (l,x) (r,y) = [(App o l r, apply o x y)| o <- [Add,Sub,Mul,Div]</pre> , valid o x y]



#### A better solution

# solutions' :: [Int] -> Int -> [Expr]

solutions' ns n = [e | ns' <- choices ns , (e,m) <- results ns' m == n]





Hardware:	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 s	0.014 s	
All solutions:	12.224 s	<b>1.312</b> s	
	Brute Force	Fusion	

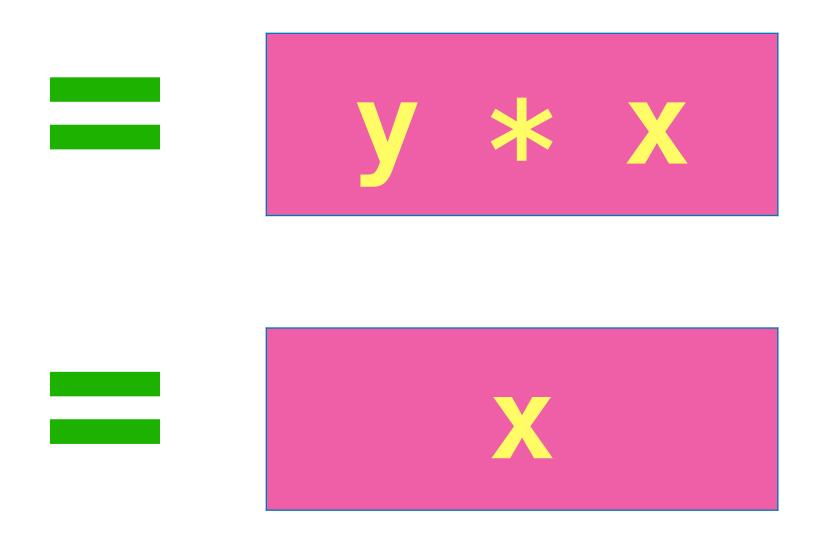
#### How Fast Now?

## Can We Do Better Further?

# arithmetic properties, such as:

search and solution spaces.

Many expressions will be essentially the same using simple



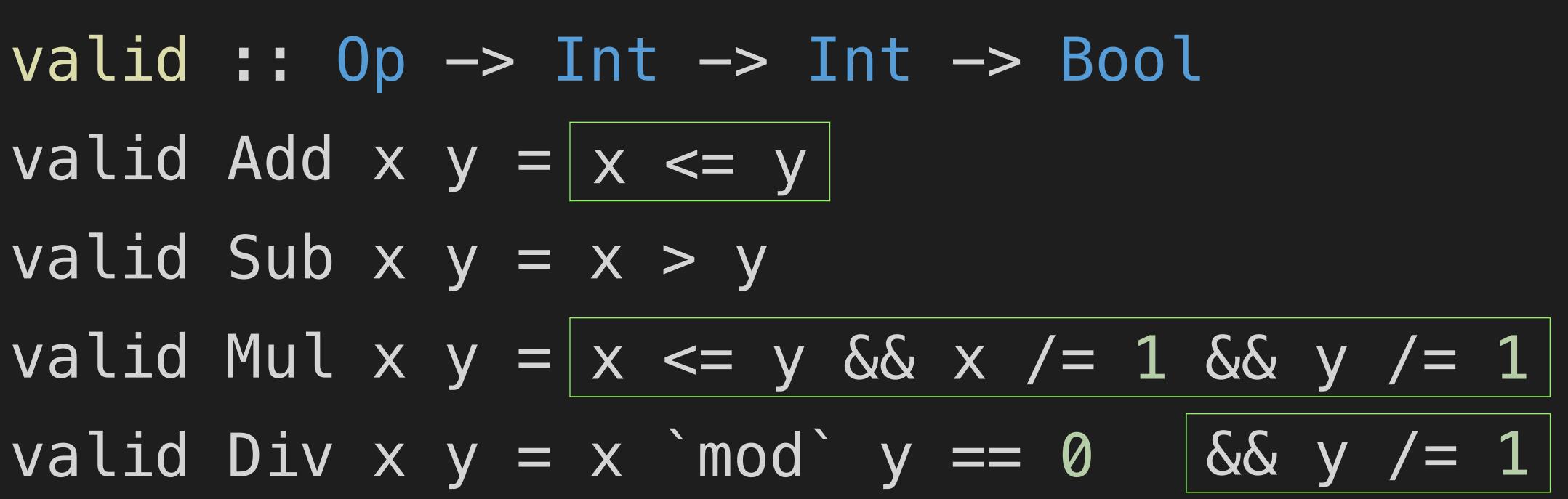
# Exploiting such properties would considerably reduce the



## A better value function

#### In Haskell, a new name for an existing type can be defined using a type declaration.

valid :: Op -> Int -> Int -> Bool valid Add x y =  $x \le y$ valid Sub x y = x > yvalid Div x y = x `mod` y == 0







Hardware:	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 s	0.014 s	0.007 s
All solutions:	12.224 s	<b>1.312</b> s	0.119 s
	Brute Force	Fusion	better valid

#### How Fast Now?





#### 9-1

Modify the final program to:

- 1. allow the use of exponentiation in expressions;

## 2. produce the nearest solutions if no exact solution is possible; 3. order the solutions using a suitable measure of simplicity.

#### 第9章: An Example: The Countdown Problem



Adapted from Graham's Lecture slides





