COL100: Introduction to Computer Science

2.1: More about functions

Algorithms and programs

Algorithms in functional model

$$square : \mathbb{Z} \to \mathbb{Z}$$
$$square(n) = n \times n$$

Programs in SML

```
fun square(n) = n * n;
```

Then `square : int * int -> int`

Functions

Every function f has a type of the form $X \to Y$

- $square : \mathbb{Z} \to \mathbb{Z}$
- $sumOfSquares : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$

Any value $x \in X$ can be passed into f, and we can be sure that $f(x) \in Y$

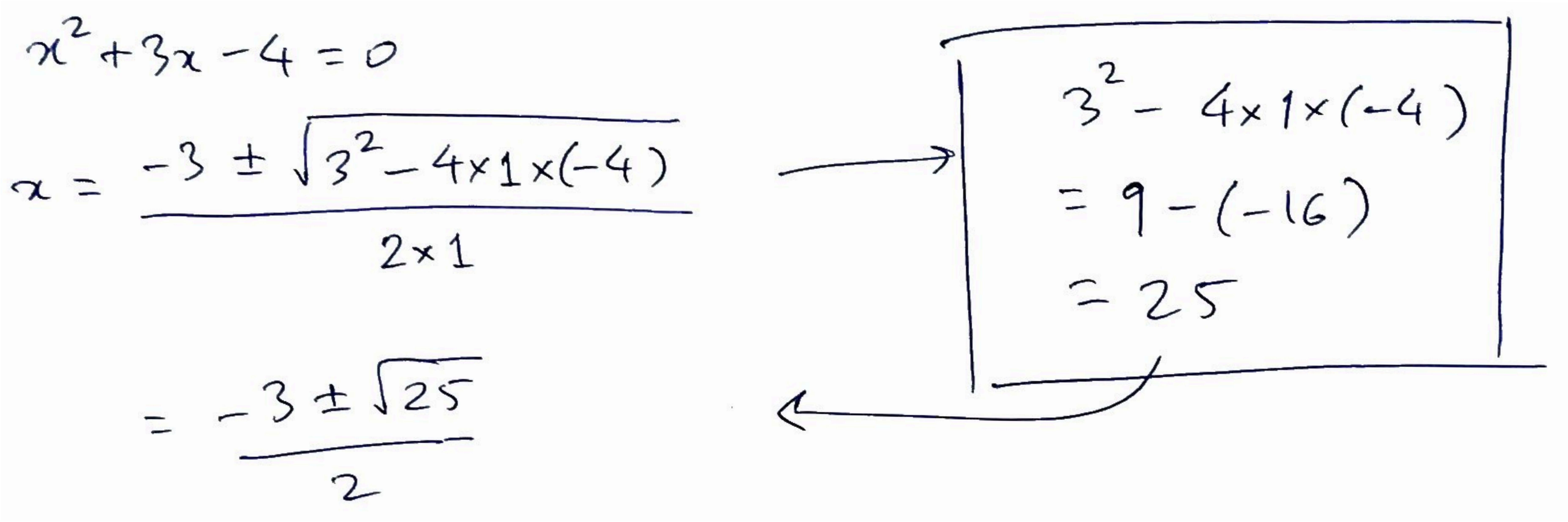
Function evaluation as substitution

- Defining variable $x = 5$ means x can always be replaced with 5
- Defining function $f(x) = \langle \ldots x \ldots \rangle$ means $f(\mathit{anything})$ can always be replaced with $\langle \ldots \mathit{anything} \ldots \rangle$

$$\begin{aligned} & \mathit{sumOfSquares}(3, 4) \\ &= \mathit{square}(3) + \mathit{square}(4) \\ &= 3 \times 3 + 4 \times 4 \\ &= 9 + 16 \\ &= 25 \end{aligned}$$

Another view of function evaluation

“Rough work” box



$x^2 + 3x - 4 = 0$

$$x = \frac{-3 \pm \sqrt{3^2 - 4\times 1\times(-4)}}{2\times 1}$$

$$= \frac{-3 \pm \sqrt{25}}{2}$$

$3^2 - 4\times 1\times(-4)$

$= 9 - (-16)$

$= 25$

$$sumOfSquares(3, 4)$$

$$\mathit{sumOfSquares}(3, 4)$$

$$\boxed{\begin{array}{c} \mathit{sumOfSquares}(x, y) \mid x = 3, y = 4 \\ \\ \mathit{square}(x) + \mathit{square}(y) \\ = \mathit{square}(3) + \mathit{square}(4) \end{array}}$$

$$sumOfSquares(3, 4)$$

$$sumOfSquares(x, y) \mid x = 3, y = 4$$

$$square(x) + square(y)$$
$$= square(3) + square(4)$$

$$square(n) \mid n = 3$$

$$n \times n$$
$$= 3 \times 3$$
$$= 9$$

$$\mathit{sumOfSquares}(3, 4)$$

$$\boxed{\begin{array}{c} \mathit{sumOfSquares}(x, y) \mid x = 3, y = 4 \\ \\ \mathit{square}(x) + \mathit{square}(y) \\ = \mathit{square}(3) + \mathit{square}(4) \\ = 9 + \mathit{square}(4) \end{array}}$$

$$sumOfSquares(3, 4)$$

$$sumOfSquares(x, y) \mid x = 3, y = 4$$

$$square(x) + square(y)$$
$$= square(3) + square(4)$$
$$= 9 + square(4)$$

$$square(n) \mid n = 4$$

$$n \times n$$
$$= 4 \times 4$$
$$= 16$$

$$\mathit{sumOfSquares}(3, 4)$$

$$\mathit{sumOfSquares}(x, y) \mid x = 3, y = 4$$

$$\begin{gathered}
\mathit{square}(x) + \mathit{square}(y) \\
= \mathit{square}(3) + \mathit{square}(4) \\
= 9 + \mathit{square}(4) \\
= 9 + 16 \\
= 25
\end{gathered}$$

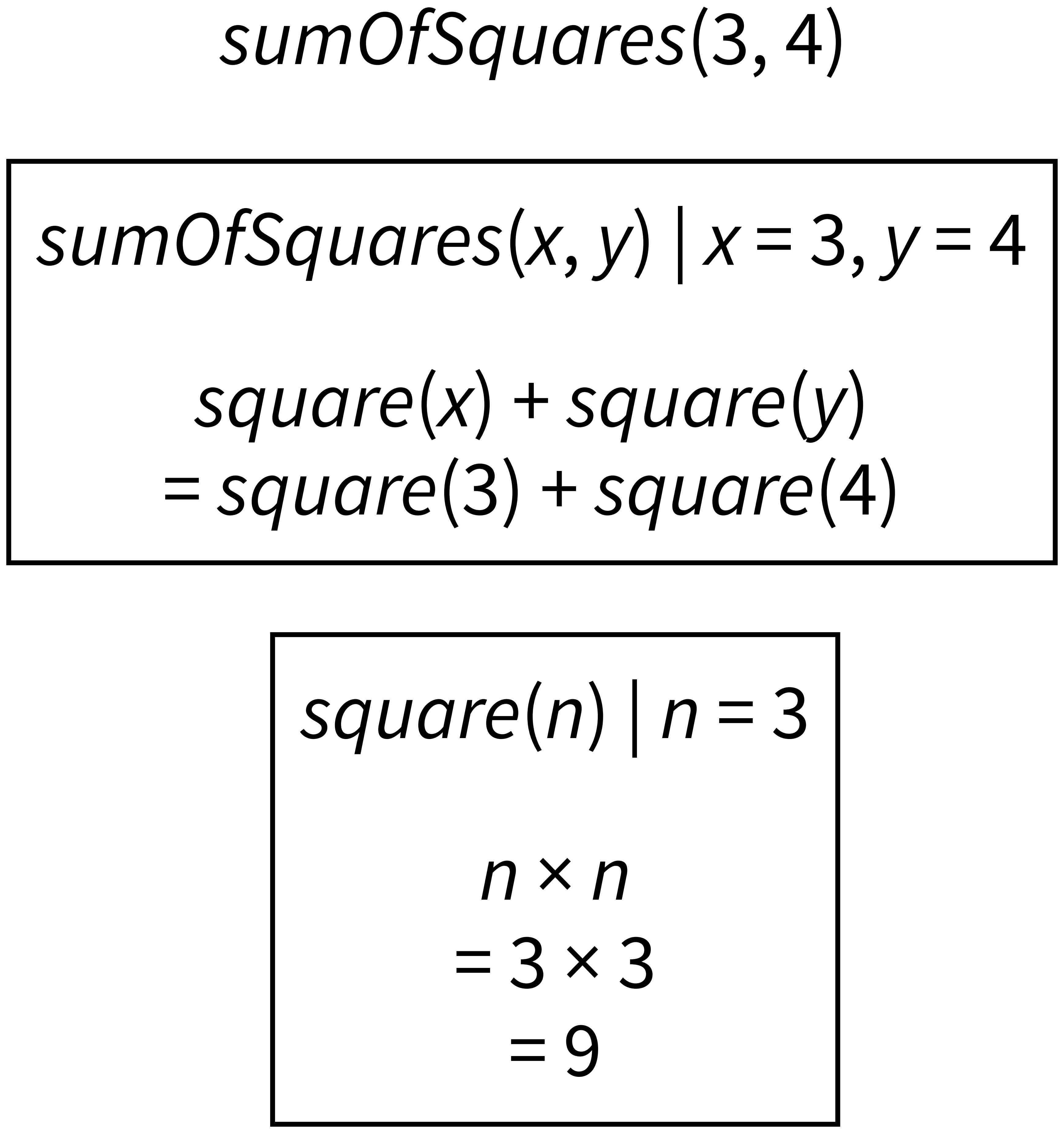
$$\mathit{sumOfSquares}(3, 4)$$

$$= 25$$

Function evaluation

Functions that call functions result in a stack of *frames*

Parameter variables e.g. x, y, n are only defined inside their frame



$$sumOfSquares(3, 4)$$

$$sumOfSquares(x, y) \mid x = 3, y = 4$$

$$square(x) + square(y)$$
$$= square(3) + square(4)$$

$$square(n) \mid n = 3$$

$$n \times n$$
$$= 3 \times 3$$
$$= 9$$

Local variables

$$var(a, b, c) = ((a - m)^2 + (b - m)^2 + (c - m)^2)/3$$
$$\text{where } m = (a + b + c)/3$$

In SML, `let … in … end`:

```
fun var(a, b, c) =
  let
    val m = (a + b + c)/3.0
  in
    ((a - m)*(a - m) + (b - m)*(b - m) + (c - m)*(c - m))/3.0
  end
```

The *scope* of `m` extends from its definition to the end of the `let…in…end` block.

Defining functions by cases

$$\max(a, b) = \begin{cases} a & \text{if } a > b, \\ b & \text{otherwise} \end{cases}$$

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{if } x < 0, \\ 0 & \text{otherwise} \end{cases}$$

In SML, `if … then … else …`:

```
fun max(a, b) =
  if a > b then a
  else b
```

```
fun sign(x) =
  if x > 0 then 1
  else if x < 0 then -1
  else 0
```

Conditional expressions

```
if … then … else …
```

All three blanks can be filled by any expression:

`if` [any expr. of type `bool`]
`then` [any expr. of some type]
`else` [any expr. of *same* type]

```
fun sign(x) =
  if x > 0
  then 1
  else if x < 0
       then -1
       else 0
```

`if…then…else…` is also an expression! e.g.

```
val payable = price - (if coupon then 0.10 * price else 0.0)
```

Partial functions and exceptions

What if the function you want to write is undefined on some values?

- e.g. $f : \mathbb{N} \to \mathbb{N}$, but no natural number type in SML!
  Have to write `f : int -> int`, but now `f` might be passed a negative number

Not-so-great solution: It’s the user’s fault, just return a junk value

Better solution: Raise an *exception* and abort the computation

```
fun f(n) =
  if n < 0
  then raise Fail("argument must be nonnegative")
  else …
```

Afterwards

- Read Sec. 3.2.1 and 3.3 of the lecture notes.
- Write a function `isLeapYear` that checks if a year is a leap year. A year is a leap year if it is divisible by 4, unless it is divisible by 100, in which case it is a leap year if it is divisible by 400. So 2020: yes, 2021: no, 2100: no, 2400: yes.