



# Programming with Joinads

Tomáš Petříček ([tomas.petricek@cl.cam.ac.uk](mailto:tomas.petricek@cl.cam.ac.uk))

University of Cambridge, UK

**Advisors:** Alan Mycroft, Don Syme

# Introduction

## □ Useful programming models

- Parallel programming
- Event-based GUI programming
- Parsers for textual input



## □ Instances of the same pattern

- Similar library structure
- But used differently

## □ Capture the pattern in a language!

# Multiplying future values

## Future values

- Compute result in background

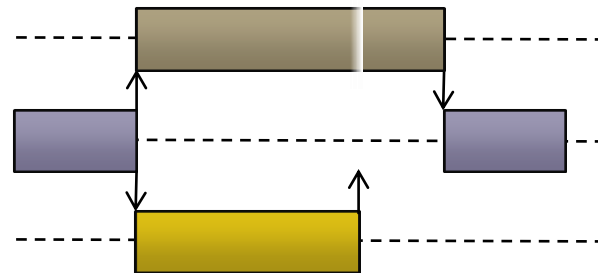
## Pattern matching

- “a” waits for a value

Suspended computation

Run both

```
multiply f1 f2 =  
  docase (f1, f2) of  
    (a, b) -> return (a * b)
```



# Multiplying future values

## Future values

- Compute result in background

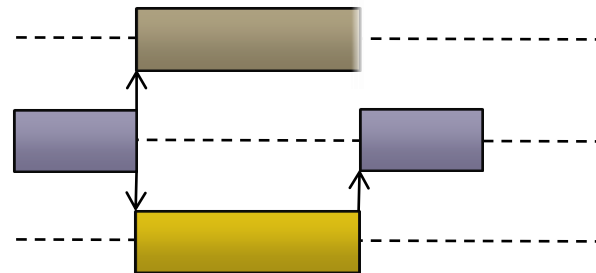
## Pattern matching

- “a” waits for a value
- “?” does not need a value to match
- “0” waits for a specific value

```
multiply f1 f2 =  
  docase (f1, f2) of  
    (0, ?) -> return 0  
    (?, 0) -> return 0  
    (a, b) -> return (a * b)
```

Run both

Choice



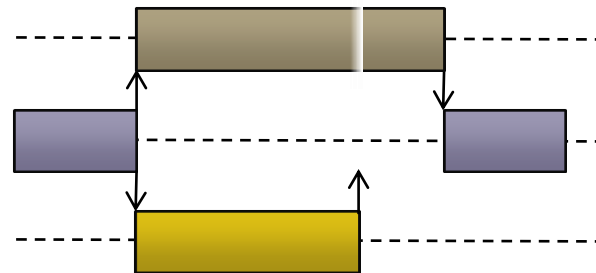
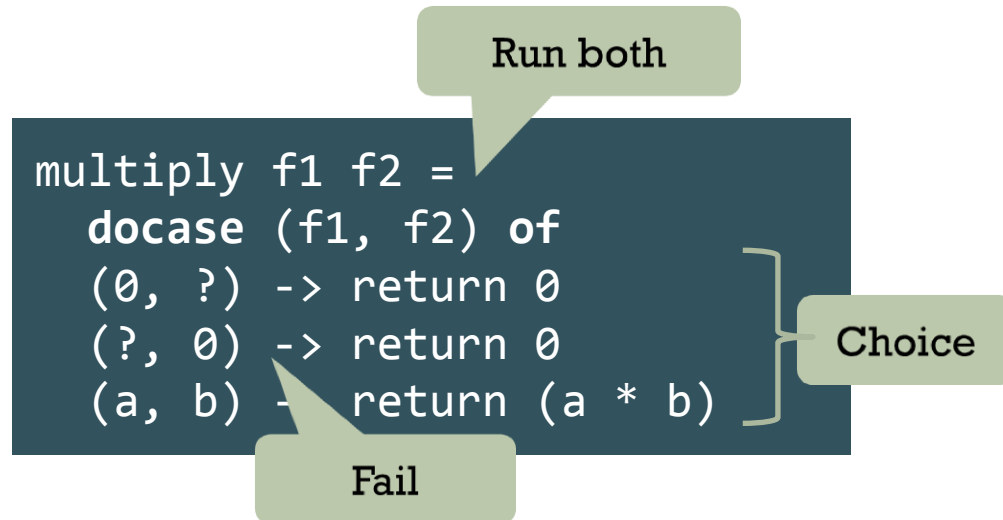
# Multiplying future values

## Future values

- Compute result in background

## Pattern matching

- “a” waits for a value
- “?” does not need a value to match
- “0” waits for a specific value



# Joinads as an algebraic structure

- ▣ Set  $\mathcal{J}$  representing “joinadic” computations
- ▣ Constant 0 and associative operators  $\otimes$ ,  $\oplus$ :
  - ▣  $a \otimes 0 = 0$
  - ▣  $a \oplus 0 = a$
  - ▣  $a \otimes b = b \otimes a$
  - ▣  $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
- ▣  $(\mathcal{J}, \oplus, \otimes, 0)$  is a *commutative near-semiring*

# Parsing using Joinads

## ▣ Validating Cambridge phone numbers

```
valid = docase ( many (satisfies isDigit),  
                multiple 10 character,  
                startsWith (string "1223") )  
  of (str, _, _) -> return str
```

- ▣ Contain only digits
  - ▣ Consists of 10 characters
  - ▣ Start with a prefix “1223”
- ▣  $r_1 \otimes r_2$  represents intersection of languages
- ▣ Returns results of all three parsers

# The two key points of the talk

## ▣ Reusable language extension

- ▣ Works with instances of a pattern
- ▣ Uniform and easy to use
- ▣ Think overloaded “+” operator

## ▣ Numerous examples

- ▣ Parallel, concurrent and interactive programming
- ▣ Writing validation rules and parsers
- ▣ Reduced line count (21 lines to just 6)

# Printing buffer using joins

## Join calculus

- Channels store values
- Joins specify reactions

```
let buffer() =  
  docase (get, putInt, putString) of  
    (r, n, ?) ->  
      reply r (intToString n)  
    (r, ?, s) ->  
      reply r s
```

First clause

Second clause

Second clause

## Pattern matching

- Use clauses to encode joins

