

Data structures

The INFDEV@HR Team

Hogeschool Rotterdam
Rotterdam, Netherlands

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Introduction

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Lecture topics

- Let
- Tuples
- Discriminated unions (polymorphism)

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Let-in

Idea

- Sometimes we wish to give a name to a value or a computation, to reuse later
- This construct is called `let-in`
- We could then say something like `let age = 9 in age + age`
- We can nest `let-in` constructs, and then say something like `let age = 9 in (let x = 2 in age * x)`

Idea

- Sometimes we wish to give a name to a value or a computation, to reuse later
- This construct is called `let-in`
- We could then say something like `let age = 9 in age + age`
- We can nest `let-in` constructs, and then say something like `let age = 9 in (let x = 2 in age * x)`
- This makes code significantly more readable, as it looks like a series of declarations top-to-bottom

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
let age = 9 in (age + age)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
let age = 9 in (age + age)
```

```
let age = 9 in (age + age)
```


Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
let age = 9 in (age + age)
```

```
let age = 9 in (age + age)
```

```
((λage→(age + age)) 9)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λage→(age + age)) 9)
```

```
((λage→(age + age)) 9)
```

```
((λage→(age + age)) 9)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λage→(age + age)) 9)
```

```
((λage→(age + age)) 9)
```

```
( 9 + 9 )
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(9 + 9)$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$(9 + 9)$

$(9 + 9)$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

(9 + 9)

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

(9 + 9)

18

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Data types

Overview

- We now move on to ways to define data types
- The definitions will be both **minimal** and **composable**
- Classes, polymorphism, etc. can all be rendered under our definitions, so we miss nothing substantial

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Overview

Notice: from now on we will start ignoring the reduction steps for simple terms such as $3+3$, $x = 0$, etc. for brevity

Minimality

- The lambda calculus has so far proven very powerful, despite its size
- We do not need hundreds of different operators, we can simply build them^a
- The only extension needed is purely syntactic in nature to make it more mnemonic, but this is only skin-deep and requires no change to the underlying mechanisms of the lambda calculus

^aand more

Minimality

- In defining data types we wish to maintain this minimality
- We do not want dozens of separate, competing data types all slightly overlapping

Fundamental scenarios

- **Tuples:** storing multiple things together at the same time, like the fields and methods in a class
- **Unions:** storing either one of various things at a time, like an interface that is exactly one of its concrete implementors

The importance of composition

- We just need to cover the case of two items, higher numbers come through composition
- For example, given the ability to store a pair, we can build a pair of pairs to create arbitrary tuples
- Similarly, given the ability to store either of two values, we can build either of many values with nesting

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Tuples

- A pair of values is defined simply as something that stores these two values
- We can extract them by giving the pair a function that will receive the values

$$(\lambda x \ y \rightarrow (\lambda f \rightarrow ((f \ x) \ y)))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

(1, 2)

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$(1, 2)$

$((\underline{,}) 1) 2)$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((,) 1) 2)
```

```
(((,) 1) 2)
```

```
(((λx y → (λf → ((f x) y))) 1) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((\lambda x y \rightarrow (\lambda f \rightarrow ((f x) y))) 1) 2)
```



```
(((\lambda x y \rightarrow (\lambda f \rightarrow ((f x) y))) 1) 2)
```

```
(((\lambda x y \rightarrow (\lambda f \rightarrow ((f x) y))) 1) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λx y → (λf → ((f x) y))) 1) 2)
```

```
((λx y → (λf → ((f x) y))) 1) 2)
```

```
((λy f → ((f 1) y)) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λy f → ((f 1) y)) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λy f → ((f 1) y)) 2)
```

```
((λy f → ((f 1) y)) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λy f→((f 1) y)) 2)
```

```
((λy f → ((f 1) y)) 2)
```

```
(λf → ((f 1) 2))
```

- We can define two utility functions that, given a pair, extract the first or second value
- They are usually called π_1 and π_2 , or `fst` and `snd`

$$(\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x)))$$
$$(\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow y)))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(\pi_1 \ (1, 2))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(\pi_1 \ (1, 2))$$
$$(\underline{\pi_1} \ (1, 2))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(\pi_1 \ (1, 2))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(\pi_1 \ (1, 2))$$
$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (1, 2))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (1, \ 2))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (1, \ 2))$$
$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ ((\underline{(),} \ 1) \ 2))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λp→(p (λx y→x))) (((,) 1) 2))
```

```
((λp→(p (λx y→x))) (((,) 1) 2))
```

```
((λp→(p (λx y→x))) ((λx y→ (λf→((f x) y))) 1)  
2))
```


Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ ((\lambda x \ y \rightarrow (\lambda f \rightarrow ((f \ x) \ y))) \ 1) \ 2))$$

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ 1) \ 2)) \ ((\lambda x \ y \rightarrow (\lambda f \rightarrow ((f \ x) \ y)))$$
$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ 1) \ 2)) \ ((\lambda x \ y \rightarrow (\lambda f \rightarrow ((f \ x) \ y))) \ 1)$$

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (\lambda x \ y \rightarrow (\lambda f \rightarrow ((f \ x) \ y))) \ 1) \ 2)$$

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (\underline{((\lambda x \ y \rightarrow (\lambda f \rightarrow ((f \ x) \ y))) \ 1)} \ 2))$$
$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ ((\lambda y \ f \rightarrow ((f \ 1) \ y)) \ 2))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ ((\lambda y \ f \rightarrow ((f \ 1) \ y)) \ 2))$$

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ ((\lambda y \ f \rightarrow ((f \ 1) \ y)) \ 2))$$
$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ \underline{((\lambda y \ f \rightarrow ((f \ 1) \ y)) \ 2)})$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (\lambda y \ f \rightarrow ((f \ 1) \ y)) \ 2))$$

```
((λp→(p (λx y→x))) ((λy f→((f 1) y)) 2))
```

```
((λp→(p (λx y→x))) (λf→((f 1) 2)))
```


Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (\lambda f \rightarrow ((f \ 1) \ 2)))$$

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (\lambda f \rightarrow ((f \ 1) \ 2)))$$
$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (\lambda f \rightarrow ((f \ 1) \ 2)))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (\lambda f \rightarrow ((f \ 1) \ 2)))$$

$$((\lambda p \rightarrow (p \ (\lambda x \ y \rightarrow x))) \ (\lambda f \rightarrow ((f \ 1) \ 2)))$$
$$((\lambda f \rightarrow ((f \ 1) \ 2)) \ (\lambda x \ y \rightarrow x))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λf → ((f 1) 2)) (λx y → x))
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λf→((f 1) 2)) (λx y→x))
```

```
((λf→((f 1) 2)) (λx y→x))
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda f \rightarrow ((f \ 1) \ 2)) \ (\lambda x \ y \rightarrow x))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λf→((f 1) 2)) (λx y→x))
```

```
(( (λx y→x) 1) 2)
```


Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((\lambda x y \rightarrow x) 1) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((\lambda x y \rightarrow x) 1) 2)
```

```
((((\lambda x y \rightarrow x) 1)) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((λx y→x) 1) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((λx y→x) 1) 2)
```

```
((λy→1) 2)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda y \rightarrow 1) \ 2)$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$((\lambda y \rightarrow 1) \ 2)$

$((\lambda y \rightarrow 1) \ 2)$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$((\lambda y \rightarrow 1) \ 2)$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$((\lambda y \rightarrow 1) \ 2)$

1

Pair of values

We should expect that π_1 and π_2 are inverse operations to constructing a pair, as they destroy it

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
let p = (1, 2) in (( $\pi_1$  p), ( $\pi_2$  p))
```

```
let p = (1, 2) in (( $\pi_1$  p), ( $\pi_2$  p))
```

```
(( $\pi_1$  (1, 2)), ( $\pi_2$  (1, 2)))
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\pi_1 \text{ (1, 2) }), (\pi_2 \text{ (1, 2) }))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\pi_1 \text{ (1, 2) }), (\pi_2 \text{ (1, 2) }))$$
$$((\pi_1 \text{ (1, 2) }), (\pi_2 \text{ (1, 2) }))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\pi_1 \ (1, 2)), (\pi_2 \ (1, 2)))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\pi_1 (1, 2)), (\pi_2 (1, 2)))$$
$$(1, (\pi_2 (1, 2)))$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(1, ( $\pi_2$  (1, 2)))
```


Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(1, ( $\pi_2$  (1, 2)))
```

```
(1, ( $\pi_2$  (1, 2)))
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(1, \underline{(\pi_2 (1, 2))})$$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$(1, \underline{\pi_2 (1, 2)})$

$(1, 2)$

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

(1, 2)

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

(1, 2)

(1, 2)

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Discriminated unions

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

- A choice of values is defined simply as something that stores either of two possible values
- We call such a choice a **discriminated union**
- We build a discriminated union with either of two functions to build the first or the second value
- They are usually called `inl` and `inr`^a

^a*in* stands for injection, and *l* and *r* stand for left and right

$$(\lambda x \rightarrow (\lambda f \ g \rightarrow (f \ x)))$$
$$(\lambda y \rightarrow (\lambda f \ g \rightarrow (g \ y)))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(inl 1)
```


Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(inl 1)
```

```
(inl 1)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(inl 1)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(inl 1)
```

```
( (λx→ (λf g→(f x))) 1)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda x \rightarrow (\lambda f \ g \rightarrow (f \ x))) \ 1)$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λx→ (λf g→(f x))) 1)
```

```
((λx→ (λf g→(f x))) 1)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda x \rightarrow (\lambda f \ g \rightarrow (f \ x))) \ 1)$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λx→ (λf g→(f x))) 1)
```

```
(λf g→(f 1))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(inr TRUE)
```


Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(inr TRUE)
```

```
(inr TRUE)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(inr TRUE)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(inr TRUE)
```

```
( (λy→ (λf g→(g y))) TRUE)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λy→ (λf g→(g y))) TRUE)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λy→ (λf g→(g y))) TRUE)
```

```
((λy→ (λf g→(g y))) TRUE)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λy→ (λf g→(g y))) TRUE)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λy→ (λf g→(g y))) TRUE)
```

```
(λf g→(g TRUE))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

- Extracting the input of a discriminated union is a process known as `match`^a
- Given a union and two functions (one per case), if the union was the first case we apply the first function, otherwise we apply the second function

^awhich is a sort of `switch`, just on steroids

$$(\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g)))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inl 1)) ( $\lambda x \rightarrow (x + 1)$ )) ( $\lambda y \rightarrow (y \wedge$   
FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inl 1)) (λx→(x + 1))) (λy→(y ∧  
  FALSE)))
```

```
((match (inl 1)) (λx→(x + 1))) (λy→(y ∧  
  FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inl 1)) ( $\lambda x \rightarrow (x + 1)$ )) ( $\lambda y \rightarrow (y \wedge$   
FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inl 1)) (λx→(x + 1))) (λy→(y ∧  
FALSE)))
```

```
(( (λu→ (λf g→((u f) g))) (inl 1)) (λx→(x + 1)  
)) (λy→(y ∧ FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((((λu→ (λf g→((u f) g))) (inl 1)) (λx→(x +  
1))) (λy→(y ∧ FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((((λu→ (λf g→((u f) g))) (inl 1)) (λx→(x +  
1))) (λy→(y ∧ FALSE)))
```

```
((((λu→ (λf g→((u f) g))) (inl 1)) (λx→(x +  
1))) (λy→(y ∧ FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((((λu→ (λf g→((u f) g))) (inl 1)) (λx→(x +  
1))) (λy→(y ∧ FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((((λu→ (λf g→((u f) g))) (inl 1)) (λx→(x +  
1))) (λy→(y ∧ FALSE)))
```

```
((((λu→ (λf g→((u f) g))) (  
(λx→ (λf g→(f x))) 1)) (λx→(x + 1))) (λy→(  
y ∧ FALSE)))
```


Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) \ ((\lambda x \rightarrow (\lambda f \ g \rightarrow (f \ x)) \ 1)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) \ ((\lambda x \rightarrow (\lambda f \ g \rightarrow (f \ x))) \ 1)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$
$$\frac{(((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) \ ((\lambda x \rightarrow (\lambda f \ g \rightarrow (f \ x))) \ 1)) \ (\lambda x \rightarrow (x + 1)))}{y \wedge \text{FALSE}})$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$\frac{\begin{array}{l} (((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) \\ ((\lambda x \rightarrow (\lambda f \ g \rightarrow (f \ x))) \ 1)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})) \end{array}}{}$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$\frac{\begin{array}{l} (((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) \\ ((\lambda x \rightarrow (\lambda f \ g \rightarrow (f \ x))) \ 1)) \ (\lambda x \rightarrow (x + 1))) \end{array}}{(\lambda y \rightarrow (y \wedge \text{FALSE}))}$$
$$(((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) \ (\lambda f \ g \rightarrow (f \ 1))) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE}))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((((λu→ (λf g→((u f) g))) (λf g→(f 1))) (λx→  
  (x + 1))) (λy→(y ∧ FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) (\lambda f \ g \rightarrow (f \ 1))) (\lambda x \rightarrow (x + 1))) (\lambda y \rightarrow (y \wedge \text{FALSE}))$$
$$\frac{(((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) (\lambda f \ g \rightarrow (f \ 1))) (\lambda x \rightarrow (x + 1))) (\lambda y \rightarrow (y \wedge \text{FALSE}))}{}$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$\frac{(((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) \ (\lambda f \ g \rightarrow (f \ 1))) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))}{}$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$\frac{(((\lambda u \rightarrow (\lambda f \ g \rightarrow ((u \ f) \ g))) \ (\lambda f \ g \rightarrow (f \ 1))) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE}))}{}$$

$$(((\lambda f \ g \rightarrow ((\lambda f \ g \rightarrow (f \ 1)) \ f) \ g)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE}))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λf g→(((λf g→(f 1)) f) g)) (λx→(x + 1)))  
  (λy→(y ∧ FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(((\lambda f \ g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ f) \ g)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$
$$\frac{(((\lambda f \ g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ f) \ g)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))}{y \wedge \text{FALSE}}$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$\frac{((\lambda f \ g \rightarrow ((\lambda f \ g \rightarrow (f \ 1)) \ f) \ g)) \ (\lambda x \rightarrow (x + 1))}{y \wedge \text{FALSE}} \ (\lambda y \rightarrow ($$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$\frac{((\lambda f \ g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ f) \ g)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE}))}{y \wedge \text{FALSE}})$$
$$((\lambda g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ g)) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ g)) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ g)) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$
$$\frac{((\lambda g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ g)) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))}{\quad}$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$\frac{((\lambda g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ g)))}{(\lambda y \rightarrow (y \wedge \text{FALSE}))}$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$\frac{((\lambda g \rightarrow (((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ g)))}{(\lambda y \rightarrow (y \wedge \text{FALSE}))}$$
$$(((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(( (λf g → (f 1)) (λx → (x + 1))) (λy → (y ∧ FALSE)) )
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$
$$(((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(((\lambda f \ g \rightarrow (f \ 1)) \ (\lambda x \rightarrow (x + 1))) \ (\lambda y \rightarrow (y \wedge \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λf g→(f 1)) (λx→(x + 1))) (λy→(y ∧ FALSE))
```

```
((λg→( (λx→(x + 1)) 1)) (λy→(y ∧ FALSE)))
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda g \rightarrow ((\lambda x \rightarrow (x + 1)) \ 1)) \ (\lambda y \rightarrow (y \ \wedge \ \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda g \rightarrow ((\lambda x \rightarrow (x + 1)) 1)) (\lambda y \rightarrow (y \wedge \text{FALSE})))$$
$$((\lambda g \rightarrow ((\lambda x \rightarrow (x + 1)) 1)) (\lambda y \rightarrow (y \wedge \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda g \rightarrow ((\lambda x \rightarrow (x + 1)) 1)) (\lambda y \rightarrow (y \wedge \text{FALSE})))$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda g \rightarrow ((\lambda x \rightarrow (x + 1)) 1)) (\lambda y \rightarrow (y \wedge \text{FALSE})))$$
$$((\lambda x \rightarrow (x + 1)) 1)$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda x \rightarrow (x + 1)) \ 1)$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((λx→(x + 1)) 1)
```

```
((λx→(x + 1)) 1)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$((\lambda x \rightarrow (x + 1)) \ 1)$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$((\lambda x \rightarrow (x + 1)) \ 1)$

$(1 + 1)$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$$(1 + 1)$$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

$(1 + 1)$

$(1 + 1)$

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

(1 + 1)

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

(1 + 1)

2

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Choice between a pair of values

We should expect that `inl` and `inr` are inverse operations to `match`

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inl 1)) inl) inr)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inl 1)) inl) inr)
```

```
((match (inl 1)) inl) inr)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inl 1)) inl) inr)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((match (inl 1)) inl) inr)
```

```
(inl 1)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inr TRUE)) inl) inr)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
((match (inr TRUE)) inl) inr)
```

```
((match (inr TRUE)) inl) inr)
```

Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((match (inr TRUE)) inl) inr)
```


Discriminated unions

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

```
(((match (inr TRUE)) inl) inr)
```

```
(inr TRUE)
```

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

Conclusion

Recap

- Lambda terms can be used to encode arbitrary basic data types
- The terms are always lambda expression which, when they get parameters passed in, identify themselves somehow
- Identification can be done by applying something (possibly even a given number of times), or returning one of the parameters

Recap

- The data types we have seen cover an impressive range of applications
- Tuples cover grouping data together (like the fields of a class)
- Unions cover choosing different things (like the polymorphism of an interface that might be implemented by various concrete classes)
- Combining these two covers all possible programming needs, even for more complex data structures

Data
structures

The
INFDEV@HR
Team

Introduction

Let-in

Data types

Tuples

Discriminated
unions

Conclusion

The best of luck, and thanks for the
attention!