Data structures

The INFDEV@HR Team

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Introduction

Lecture topics

- Tuples
- Discriminated unions (polymorphism)
- Lists

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 types

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

[a] and more

Data types

Minimality

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

Data types

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

Tuples

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

```
(λx→y→f→((f x) y))
```

Tuples


```
(1, 2)
```

Tuples


```
(1, 2)
```

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

Tuples


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

Tuples

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

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

Tuples

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

Tuples

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

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

Tuples

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

Tuples

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

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

Tuples


```
((λy→f→((f 1) y)) 2)
```

Tuples

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

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

Tuples

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

Tuples

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

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

Tuples

- 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`

```
(λp→(p (λx→y→x)))
```

```
(λp→(p (λx→y→y)))
```

Tuples

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

Tuples

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

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

Tuples

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

Tuples

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

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

Tuples

```
((λp→(p (λx→y→x))) (1, 2))
```

Tuples

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

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

Tuples

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

Tuples

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

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

Tuples

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

Tuples

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

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

Tuples

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

Tuples

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

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

Tuples

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

Tuples


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



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

Tuples


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

Tuples


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

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

Tuples

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

Tuples


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

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

Tuples

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

Tuples


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

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

Tuples

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

Tuples

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

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

Tuples


```
((λf→((f 1) 2)) (λx→y→x))
```

Tuples

```
((λf→((f 1) 2)) (λx→y→x))
```

```
((λf→((f 1) 2)) (λx→y→x))
```

Tuples

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

Tuples

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

$$(((\lambda x \to y \to x)\ 1)\ 2)$$

Tuples

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

Tuples


```
(((λx→y→x) 1) 2)
```

```
(((λx→y→x) 1) 2)
```

Tuples

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

Tuples

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

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

Tuples

```
((λy→1) 2)
```

Tuples

$$((\lambda \mathtt{y} \rightarrow \mathtt{1})\ \ \mathtt{2})$$

$$\underline{((\lambda \mathtt{y} \rightarrow \mathtt{1})\ \ \mathtt{2})}$$

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

Tuples

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

$$1$$

Pair of values

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

Tuples

let p = (1, 2) in ((π_1 p), (π_2 p))

Tuples

$$\texttt{let p = (1, 2) in } ((\pi_1\ \texttt{p}),\ (\pi_2\ \texttt{p}))$$

$$\underline{\texttt{let p = (1, 2) in } ((\pi_1\ \texttt{p}),\ (\pi_2\ \texttt{p}))}$$

Tuples

$$\texttt{let p = (1, 2) in ((}\pi_1\texttt{ p), (}\pi_2\texttt{ p))}$$

Tuples

$\underline{\texttt{let p = (1, 2) in ((}\pi_1\texttt{ p), (}\pi_2\texttt{ p))}}$

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

Tuples

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

Tuples

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

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

Tuples

$$\boxed{\texttt{(}\underline{\texttt{(}\pi_1\ \texttt{(1, 2))}}\texttt{, (}\pi_2\ \texttt{(1 , 2)))}}$$

Tuples

$$\texttt{(}\underline{\texttt{(}\pi_1\ \texttt{(1, 2))}}\texttt{, (}\pi_2\ \texttt{(1 , 2)))}$$

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

Tuples

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

Tuples

$\boxed{\texttt{(1 , (}\pi_2\texttt{ (1 , 2)))}}$

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

Tuples

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

Tuples

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

$(1,\ 2)$

Tuples


```
(1, 2)
```

Data structures

The INFDEV@HR Team

Tuples

```
(1, 2)
```

```
(1, 2)
```

Discriminated unions

- 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

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

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

Discriminated unions


```
(inl 1)
```

Discriminated unions


```
(inl 1)
```

```
(inl 1)
```

Discriminated unions


```
(inl 1)
```

Discriminated unions

```
(inl 1)
```

```
((λx→f→g→(f x)) 1)
```

Discriminated unions


```
((λx→f→g→(f x)) 1)
```

Discriminated unions


```
((λx→f→g→(f  x))  1)
```

```
((λx→f→g→(f  x))  1)
```

Discriminated unions

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

Discriminated unions

$$\underline{((\lambda x \rightarrow f \rightarrow g \rightarrow (f\ x))\ 1)}$$

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

Discriminated unions


```
(λf→g→(f  1))
```

Discriminated unions

$$\boxed{(\lambda f \rightarrow g \rightarrow (f\ \ 1))}$$

$$\boxed{\underline{(\lambda f \rightarrow g \rightarrow (f\ \ 1))}}$$

Discriminated unions

$\underline{(\lambda \mathtt{f} \rightarrow \mathtt{g} \rightarrow (\mathtt{f}\ \ 1))}$

Discriminated unions

```
(λf→g→(f 1))
```

```
(inl 1)
```

Discriminated unions


```
(inr TRUE)
```

Discriminated unions


```
(inr TRUE)
```

```
(inr TRUE)
```

```
(inr TRUE)
```

Discriminated unions

$(\underline{\texttt{inr}}\ \texttt{TRUE})$

$((\lambda \texttt{y} \rightarrow \texttt{f} \rightarrow \texttt{g} \rightarrow (\texttt{g}\ \ \texttt{y}))\ \ \texttt{TRUE})$

Discriminated unions

$$\boxed{((\lambda y \rightarrow f \rightarrow g \rightarrow (g\ \ y))\ \ \mathtt{TRUE})}$$

Discriminated unions


```
((λy→f→g→(g y)) TRUE)
```

```
((λy→f→g→(g y)) TRUE)
```

Discriminated unions

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

Discriminated unions

$$\boxed{(\underline{(\lambda y \rightarrow f \rightarrow g \rightarrow (g\ y))\ \mathtt{TRUE}})}$$

$$\boxed{(\lambda f \rightarrow g \rightarrow (g\ \mathtt{TRUE}))}$$

Discriminated unions


```
(λf→g→(g TRUE))
```

Discriminated unions

$(\lambda f \rightarrow g \rightarrow (g \;\; \texttt{TRUE}))$

$(\lambda f \rightarrow g \rightarrow (g \;\; \texttt{TRUE}))$

Discriminated unions

$(\lambda f \rightarrow g \rightarrow (g\ \mathtt{TRUE}))$

Discriminated unions

$(\lambda f \rightarrow g \rightarrow (g\ \mathtt{TRUE}))$

```
(inr TRUE)
```

Discriminated unions

- 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

[a] which is a sort of `switch`, just on steroids

$$(\lambda \mathtt{u} \rightarrow \mathtt{f} \rightarrow \mathtt{g} \rightarrow ((\mathtt{u}\ \mathtt{f})\ \mathtt{g}))$$

Discriminated unions

$$(((\texttt{match (inl 1)) } (\lambda\texttt{x}\rightarrow\texttt{(x} + \texttt{1))) } (\lambda\texttt{y}\rightarrow\texttt{(y } \wedge \texttt{ FALSE)))}$$

Discriminated unions

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

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

Discriminated unions

$$(((\underline{\texttt{match}}\ (\texttt{inl}\ 1))\ (\lambda x \rightarrow (x + 1)))\ (\lambda y \rightarrow (y \wedge \texttt{FALSE})))$$

Discriminated unions

$(((\underline{\texttt{match}}\ (\texttt{inl}\ 1))\ (\lambda x \rightarrow (x + 1)))\ (\lambda y \rightarrow (y \wedge \texttt{FALSE})))$

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

Discriminated unions

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

Discriminated unions

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

$$\boxed{\texttt{((((}\lambda\texttt{u}\rightarrow\texttt{f}\rightarrow\texttt{g}\rightarrow\texttt{((u f) g)) (}\underline{\texttt{inl}}\texttt{ 1)) (}\lambda\texttt{x}\rightarrow\texttt{(x + 1))) (}\lambda\texttt{y}\rightarrow\texttt{(y} \wedge \texttt{FALSE)))}}$$

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions


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

Discriminated unions


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

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

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions

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

Discriminated unions


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



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

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions

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

Discriminated unions

$$\boxed{((\underline{((\lambda u \to f \to g \to ((u\ f)\ g))\ (\lambda f \to g \to (f\ 1)))}\ (\lambda x \to (x + 1)))\ (\lambda y \to (y \wedge \mathtt{FALSE})))}$$

$$\boxed{(((\lambda f \to g \to (((\lambda f \to g \to (f\ 1))\ f)\ g))\ (\lambda x \to (x + 1)))\ (\lambda y \to (y \wedge \mathtt{FALSE})))}$$

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions

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

Discriminated unions


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

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

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions

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

Discriminated unions

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

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

Discriminated unions

$$\underline{((\lambda g \rightarrow ((\lambda x \rightarrow (x + 1))\ 1))\ (\lambda y \rightarrow (y \land \mathtt{FALSE})))}$$

Discriminated unions

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

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

Discriminated unions


```
((λx→(x + 1)) 1)
```

Discriminated unions

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

$$\underline{((\lambda x \rightarrow (x + 1))\ 1)}$$

Discriminated unions

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

Discriminated unions

$\underline{((\lambda x \rightarrow (x + 1))\ 1)}$

$(1 + 1)$

Discriminated unions

```
(1 + 1)
```

Discriminated unions


```
(1 + 1)
```

```
(1 + 1)
```

Discriminated unions

$\underline{(1\ +\ 1)}$

Discriminated unions


```
(1 + 1)
```

```
2
```

Discriminated unions

Choice between a pair of values

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

Discriminated unions


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

Discriminated unions


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

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

Discriminated unions


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

Discriminated unions


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

```
(inl 1)
```

Discriminated unions


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

Discriminated unions


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

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

Discriminated unions

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

Discriminated unions


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

```
(inr TRUE)
```

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

Conclusion

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

This is it!

The best of luck, and thanks for the attention!