CS 3110

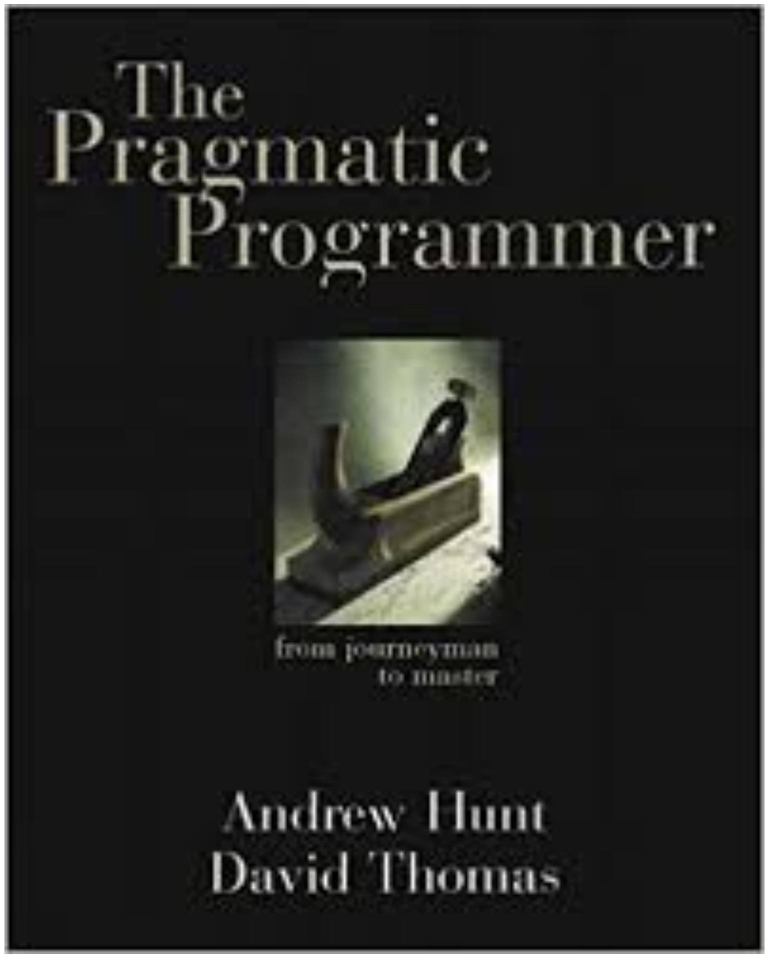
Variants

Nate Foster

Spring 2019

Today's music: *Union* by The Black Eyed Peas (feat. Sting)

Instant Access Textbook



Must opt out *today* if you don't want to purchase through Instant Access!

Review

Previously in 3110:

- Lists, records, tuples
- Pattern matching

Today:

- Variants

PATTERN MATCHING ON LISTS

Dem
o

Pattern matching

- Match shape of data
- Extract part(s) of data

Syntax:

```
match e with
 | p1 -> e1
 | p2 -> e2
 | …
 | pn -> en
```

`p1`..**`pn`**:
pattern expressions

Semantics of pattern matching

- `[]` matches `[]` and nothing else
- `h :: t`
  - matches `2::[]`, binding `h` to `2` and `t` to `[]`
  - matches `1::3::[]`, binding `h` to `1` and `t` to `3::[]`
- `_` matches everything

  underscore character, called wildcard
  (it's like a blank space)

Full details in textbook

Why pattern matching is THE GREATEST

1. You can’t forget a case
   (inexhaustive pattern-match warning)
2. You can’t duplicate a case
   (unused match case warning)
3. You can’t get an exception
   (e.g., `hd []`)
4. Pattern matching leads to elegant, concise, beautiful code

VARIANTS

Dem
o

Variant types

Type definition syntax:

```
type t =
| C1 of t1
| ...
| Cn of tn
```

Optional data carried by constructor

Constructors *aka* tags

Question

Which of the following would be better represented with records rather than variants?

A. *Coins*, which can be pennies, nickels, dimes, or quarters

B. *Students*, who have names and id numbers

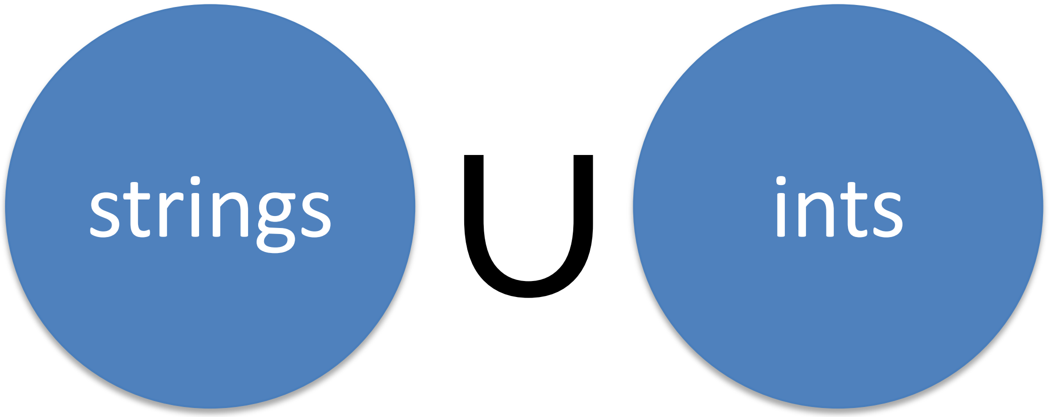
C. A *dessert*, which has a sauce, a creamy component, and a crunchy component

D. A and C

E. B and C

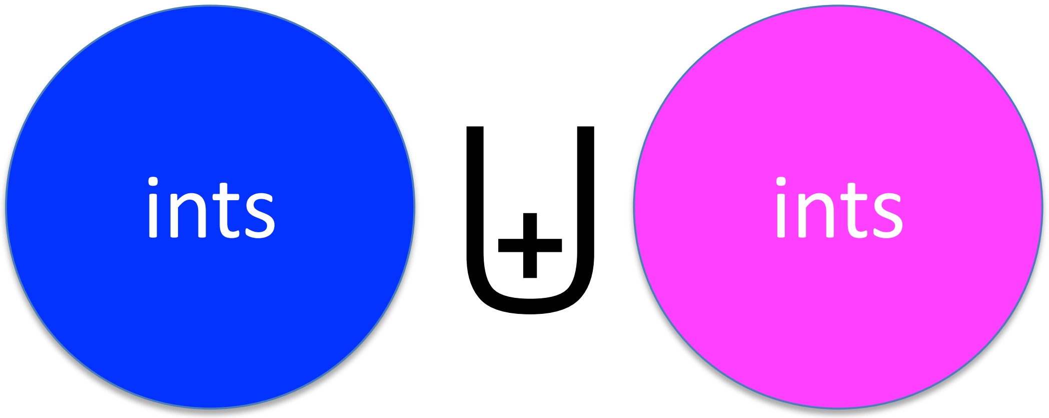
Variant: union

```
type stringOrInt =
    | String of string
    | Int of int
```



Variant: tagged union

```
type blueOrPinkInt =
  | Blue of int
  | Pink of int
```



$$+$$

One Of: Sum Type

×

Each Of: Product Type

Algebraic Data Types

RECURSIVE VARIANTS

Dem
o

PARAMETERIZED VARIANTS

Dem

o

Type variables

Variable: name standing for unknown value
Type variable: name standing for unknown type

Java example: **`List<T>`**

OCaml Syntax: single quote followed by identifier
e.g., **`'foo, 'key, 'value`**

But most often simply just: **`'a`**
Pronounced: "alpha"

Parametric polymorphism

- *poly* = many, *morph* = form
- write function that works for many arguments regardless of their type
- closely related to Java generics
- related to C++ template instantiation

VARIANTS ARE POWERFUL

Lists are just variants

OCaml effectively codes up lists as variants:

```
type 'a list = [] | :: of 'a * 'a list
```

- **`list`** is a type constructor parameterized on type variable **`'a`**
- **`[]`** and **`::`** are constructors
- Just a bit of syntactic magic in the compiler to use **`[]`** and **`::`** instead of alphabetic identifiers

Exceptions are (mostly) just variants

OCaml effectively codes up exceptions as slightly strange variants:

```
type exn
exception MyNewException of string
```

- Type **`exn`** is an extensible variant that may have new constructors added after its original definition
- Raise exceptions with **`raise e`**, where **`e`** is a value of type **`exn`**
- Handle exceptions with pattern matching, just like you would process any variant

OPTIONS

"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object-oriented language. My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."

– Sir Tony Hoare

Option: A built-in variant

```
type 'a option = None | Some of 'a
```

Dem
o

Null Pointer Exception
Pattern Match against None
imgflip.com

Upcoming events

- [Wed] A0 due
- [Thur] Level Up!

This is powerful.

THIS IS 3110