Variants

Nate Foster
Spring 2019

Today’s music: *Union* by The Black Eyed Peas (feat. Sting)
Instant Access Textbook

The Pragmatic Programmer

Andrew Hunt
David Thomas

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
Pattern matching

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

Syntax:
```
match e with
| p1  -> e1
| p2  -> e2
| ... 
| pn  -> en
```
Semantics of pattern matching

• [ ] matches [ ] and nothing else
• \texttt{h :: t}
  – matches \texttt{2::[ ]}, binding \texttt{h} to \texttt{2} and \texttt{t} to [ ]
  – matches \texttt{1::3::[ ]}, binding \texttt{h} to \texttt{1} and \texttt{t} to \texttt{3::[ ]}
• \texttt{\_} matches everything
  underscore character, called \texttt{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
Variant types

Type definition syntax:

```haskell
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

define type stringOrInt =
| String of string
| Int of int
Variant: tagged union

```haskell
type blueOrPinkInt =
    | Blue of int
    | Pink of int
```
One Of: Sum Type
Each Of: Product Type
Algebraic Data Types
RECURSIVE VARIANTS
PARAMETERIZED VARIANTS
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

• $poly = \text{many}$, $morph = \text{form}$
• 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:

```ocaml
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:

```ocaml
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
Null Pointer Exception

Pattern Match against None
Upcoming events

• [Wed] A0 due
• [Thur] Level Up!

This is powerful.

THIS IS 3110