Discussion 04

Variants

Kenneth Fang (kwf37), Newton Ni (cn279)

Feb. 6, 2019

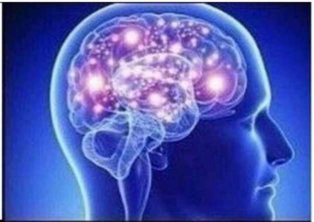
Key Concepts

- Sum types are **fundamental**.

Key Concepts

- Sum types are **fundamental**.
- Option types enable **static analysis**.

```
type coin =
| Penny
| Nickel
| Dime
| Quarter
```



```
type shape =
| Point
| Square of int
| Circle of
{ radius: int }
```

```
type ’a t =
| Leaf
| Node of
’a * ’a t * ’a t
```

Variants: Enumerations

```
type coin =
| Penny
| Nickel
| Dime
| Quarter
```

Variants: Enumerations

Useful when . . .

- You have a small number of constants

Variants: Enumerations

Useful when ...

- You have a small number of constants
- e.g. card suits, keyboard buttons, Crayon colors

Variants: Tagged Unions

```
type shape =
| Point
| Square of int
| Circle of { radius: int }
```

Useful when . . .

- You have different representations of a concept

Useful when . . .

- You have different representations of a concept
- e.g. state machines, errors, class hierarchies

Variants: Recursion and Polymorphism

```
type 'a tree =
| Leaf
| Node of 'a * 'a tree * 'a tree
```

Variants: Recursion and Polymorphism

Useful when . . .

- You have inductively defined (self-similar) data

Variants: Recursion and Polymorphism

Useful when . . .

- You have inductively defined (self-similar) data
- e.g. naturals, trees, languages, games

Exercise: Calculator

```
(** Represents a binary operator. *)
type bin =
| Add
| Sub
| Mul
| Div

(** Represents an expression. *)
type exp =
| Int of int
| Bin of exp * bin * exp
```

Exercise: Calculator

- You only have to worry about `calculator.ml`

Exercise: Calculator

- You only have to worry about `calculator.ml`
- Three increasingly interesting functions:

Exercise: Calculator

- You only have to worry about `calculator.ml`
- Three increasingly interesting functions:
- `let rec string_of_bin (b: bin) : string`

Exercise: Calculator

- You only have to worry about `calculator.ml`
- Three increasingly interesting functions:
- `let rec string_of_bin (b: bin) : string`
- `let rec string_of_exp (e: exp) : string`

Exercise: Calculator

- You only have to worry about `calculator.ml`
- Three increasingly interesting functions:
- `let rec string_of_bin (b: bin) : string`
- `let rec string_of_exp (e: exp) : string`
- `let rec eval (e: exp) : (int option)`

Exercise: Calculator

- Use `make` to compile and run the calculator

Exercise: Calculator

- Use `make` to compile and run the calculator
- Calculator will accept expressions in REPL (e.g. (5 * (2 - 3)))

Recitation Exercises