











Functional Domain Modelling Works



marcoemrich - last edited 6 days ago



Forked from /s/6tvWP1cCPsjwZJvvrWtcwe

Basic Building Blocks

Primitives



```
1 /* primitives (type alias) */
 2 type name = string
                                                                     type name = string;
3 type age = int
   type age = int;
                                                                     4 type size = float
                                                                     type size = float;
 5 type isAlive = bool
   type isAlive = bool;
                                                                     m
 6
 7
 8 let a: int = 27
   let a: int = 27;
                                                                     n
9 let a: age = 27
                                                                     let a: age = 27;
10 let a = 27
                                                                     let a: int = 27;
11 let a = 5.7;
   let a: float = 5.7;
                                                                     12
13 27;
                                                                     n
   -: int = 27
14
15 "foo";
   -: string = "foo"
```













Abstract Types

important usage:

- 1. reduce implementation detail
- 2. model top-down

```
<>+ ∅+ 🖺 -
```

```
17 type notYetSpecified;
   type notYetSpecified;
                                                                        h
18
```

Algebraic Data Types (ADT)

```
<>+ ◇+ ⋒-
```

```
### Sum Types (OR)
(also Variants, Unions)
```

```
<>+ ◇+ ⋒-
```

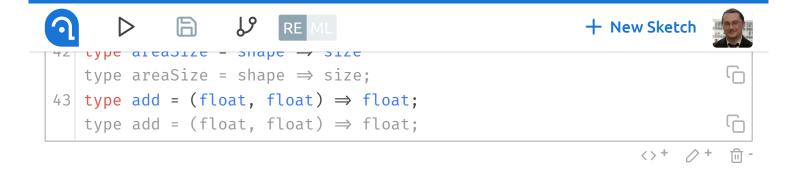
```
19 type color = Red | Green | Blue | Yellow;
                                                                      type color = Red | Green | Blue | Yellow;
20 type shape = Circle | Rectangle;
   type shape = Circle | Rectangle;
                                                                       n
21
22 /* 4 possible Values (sum) */
23 Red:
   - : color = Red
```

Product Types (AND)



```
RE ML
                                                       + New Sketch
    type coordinate = (int, int);
26 type coloredShape = (color, shape);
   type coloredShape = (color, shape);
27
28 /* 4 Colors * 2 Shapes = 8 possible Values (product) */
29 let x: coloredShape = (Red, Circle);
   let x: coloredShape = (Red, Circle);
                                                                      m
30 (7, 3)
   -: (int, int) = (7, 3)
                                                                      m
31
32 /* Product: Record */
33 type coloredShape = {shape, color};
   type coloredShape = { shape: shape, color: color, };
                                                                      m
34
35 {shape: Circle, color: Red};
   - : coloredShape = {shape: Circle, color: Red}
                                                                      m
36 {color: Green, shape: Rectangle};
    - : coloredShape = {shape: Rectangle, color: Green}
                                                             <>+
### Sum Type with Payload
                                                             <>+ />+ ⋒-
37 /* Sum with Payload aka Type Constructors */
38 type shape = Circle(int) | Square(int, int);
                                                                      type shape = Circle(int) | Square(int, int);
39
40 Square(3, 4); /* (3) */
    -: shape = Square(3, 4)
41 Circle(4); /* (3, 4) */
    - : shape = Circle(4)
```

Functions



Primitive Anti-Pattern

```
<>+ ∅ + ⑪ -
```

```
44 let durationToNextRoom = 10.0;
let durationToNextRoom: float = 10.;
45 let wayToNextRoom = 5.0;
let wayToNextRoom: float = 5.;
46
47 let speed = (way: float, time: float) ⇒ way /. time;
let speed: (float, float) ⇒ float = <fun>;
48
49 speed(durationToNextRoom, wayToNextRoom);
-: float = 2.
```

Tagged Types (Sum Type of 1)

<>+ Ø + 面 -

```
type secs = Secs(float);
type secs = Secs(float);

type meter = Meter(float);
type meter = Meter(float);

2

let speed: (meter, secs) ⇒ float =
    (Meter(m), Secs(s)) ⇒ m /. s;
let speed: (meter, secs) ⇒ float = <fun>;
/* let speed = (Meter(m), Secs(s)) ⇒ m /. s; */
```

```
RE ML
                                                     + New Sketch
58 let wayToNextRoom = Meter(5.0);
   let wayToNextRoom: meter = Meter(5.);
59
60 /* speed(durationToNextRoom, wayToNextRoom); */
                                                          <>+
                                                                   而 -
*/# Lists
                                                          <>+ ◇+ ⋒-
61 let x: list(shape) = [];
  let x: list(shape) = [];
                                                                   n
62
63 type simplePicture = { shape: list(shape), backgroundColor: color};
   type simplePicture = { shape: list(shape), backgroundColor: color, 🕞
```

Invariant Enforcement

```
Tennis Scoring Rules:

* Player points: Love, 15, 30, 40

* 40 points && win the ball ⇒ win game

* both player 40 ⇒ players are deuce

* deuce: the winner of a ball ⇒ advantage

* advantage && wins the ball ⇒ win game

* player without advantage wins ⇒ back at deuce

⟨>+ ∅+ Û-

### 1st Try

⟨>+ ∅+ Û-
```

```
type points = int;
type points = int;
65
```

let s: score = {player0ne: 1000, playerTwo: -15};

<>+ ∅ + 🛍

2nd Try

74 };

```
75 type points =
    Love
76
77
    Fifteen
78
    | Thirty
79
    | Fortv:
                                                                      type points = Love | Fifteen | Thirty | Forty;
80
81 type score = {
    playerOne: points,
82
    playerTwo: points,
83
84
   type score = { playerOne: points, playerTwo: points, };
                                                                      m
85
86 let s2 = {playerOne: Fifteen, playerTwo: Love};
   let s2: score = {playerOne: Fifteen, playerTwo: Love};
                                                                      m
87 let even = {playerOne: Forty, playerTwo: Forty};
   let even: score = {playerOne: Forty, playerTwo: Forty};
```

<>+ ∅ + ⑪

Better Solution



```
| PlayerTwo;
 90
    type player = PlayerOne | PlayerTwo;
 91
 92 type points =
     Love
 93
      | Fifteen
 94
      | Thirty;
 95
    type points = Love | Fifteen | Thirty;
                                                                       96
97 type score =
      | Points(points, points)
 98
 99
      | Forty(player, points /* of other player */)
100
      I Deuce
      | Advantage(player)
101
     | Game(player);
102
                                                                        h
    type score =
        Points(points, points)
      | Forty(player, points)
      Deuce
      | Advantage(player)
      | Game(player);
103
104 let startScore: score = Points(Love, Love);
                                                                       n
    let startScore: score = Points(Love, Love);
105 let anotherScore: score = Forty(PlayerTwo, Thirty);
    let anotherScore: score = Forty(PlayerTwo, Thirty);
                                                                        n
106 let anotherScore2: score = Deuce:
    let anotherScore2: score = Deuce;
                                                                        n
107 let anotherScore3: score = Advantage(PlayerOne);
    let anotherScore3: score = Advantage(PlayerOne);
                                                                        m
108
109 /*
110
    let impossibleScore1: score = Points(Seven, Eleven);
     let impossibleScore2: score = Points(Forty, Forty);
111
     let impossibleScore3: score = Forty(PlayerTwo, Forty);
112
     */
113
```















```
<>+ ∅ + 🛍 -
```

```
114 /* type option('a) = None | Some('a); */
115
116 let reciprocal = x \Rightarrow x = 0.0 ? None : Some(1.0 /. x);
    let reciprocal: float ⇒ option(float) = <fun>;
                                                                         m
117
118
119 type name = Name(string);
    type name = Name(string);
                                                                         n
120 type phone = Phone(string);
    type phone = Phone(string);
                                                                         n
121
122 type customer = {
|123| name,
     phone: option(phone),
124
125 };
    type customer = { name: name, phone: option(phone), };
```

Tactical Design Pattern and idioms

<>+ />+ ⋒-

Model Values explicitly!

ie. replace booleans with variants

<>+ ◇+ ⋒-

```
126 /* Don't */
127
```



Smart Constructors

```
https://github.com/ostera/reason-design-
patterns/blob/master/patterns/smart-constructors.md
```

```
135 type name = Name(string); /* Business Contraint: Length < 20 */
    type name = Name(string);
                                                                              136
137 type makeName = string \Rightarrow option(name);
    type makeName = string \Rightarrow option(name);
                                                                              n
138 let smallerThan20 = s \Rightarrow String.length(s) < 20
    let smallerThan20: string ⇒ bool = <fun>;
                                                                              n
139
140 let makeName: makeName = str \Rightarrow (smallerThan20(str))?
    Some(Name(str)) : None
                                                                              n
    let makeName: makeName = <fun>;
141
142 /* Pretty:
143
144 let makeName: makeName = str \Rightarrow (str \triangleright String.length < 20) ?
    Some(Name(str)) : None
145
146 */
147
148 makeName("Sven");
    - : option(name) = Some((Name("Sven")))
                                                                              n
```













```
- : option(name) = None
```

Model State Changes explicitly!

```
* Move Runtime to Compile Time
* Bool → Type
```

<>+ ∅ + ⑪ -

```
151 /* Bad Idea */
152 type emailAdress = {
153 email: string,
154 verified: bool,
155 };
    type emailAdress = { email: string, verified: bool, };
                                                                      156
157 /* Better */
158 type unverifiedEmail =
159 | UnverifiedEmailAdress(string);
    type unverifiedEmail = UnverifiedEmailAdress(string);
                                                                      m
160 type verifiedEmail =
161 | VerifiedEmailAdress(string);
    type verifiedEmail = VerifiedEmailAdress(string);
                                                                      m
162
163 type email =
164 | UnverifiedEmailAdress(string)
     | VerifiedEmailAdress(string);
165
    type email = UnverifiedEmailAdress(string) | VerifiedEmailAdress(string)
166
167 type verifyEmail = unverifiedEmail ⇒ option(verifiedEmail);
    type verifyEmail = unverifiedEmail ⇒ option(verifiedEmail);
```

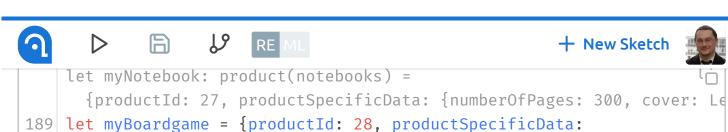
Model with Abstract Types;



Generics

```
<>+ ∅ + 🛍 -
```

```
174 type numberOfPlayers = | NumberOfPlayers(int);
                                                                       h
    type numberOfPlayers = NumberOfPlayers(int);
175 type duration = Minutes(int) | Hours(int);
    type duration = Minutes(int) | Hours(int);
                                                                       h
176 type boardgame = {numberOfPlayers: int, duration: duration};
    type boardgame = { numberOfPlayers: int, duration: duration, };
                                                                       m
177
178 type cover = Leather | Cardboard;
    type cover = Leather | Cardboard;
                                                                       h
179 type notebooks = {numberOfPages: int, cover: cover};
    type notebooks = { numberOfPages: int, cover: cover, };
                                                                       Ĺ
180
181 type id = int;
                                                                       n
    type id = int;
182
183 type product('productKind) = {
     productId: id,
184
      productSpecificData: 'productKind,
185
186 }:
    type product('productKind) = {
                                                                       n
      productId: id,
      productSpecificData: 'productKind,
    };
187
```



{productId: 28,

190 191

```
+ New Sketch
```



```
192 let productIdOf = product ⇒ product.productId;
    let productIdOf: product('a) ⇒ id = <fun>;
                                                                    193
```

{numberOfPlayers: 5, duration: Minutes(90)}};

let myBoardgame: product(boardgame) =

194 let numberOfPlayersOf = product ⇒

product.productSpecificData.numberOfPlayers; let numberOfPlayersOf: product(boardgame) ⇒ int = <fun>;

