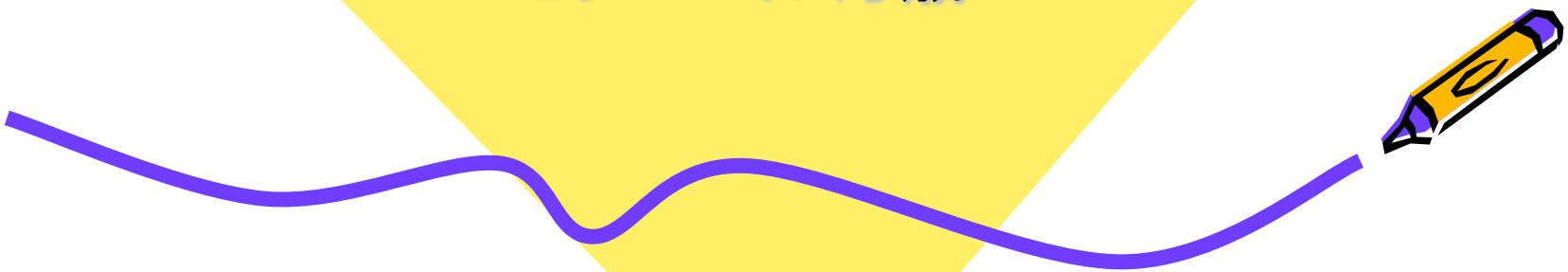
WEBシステムと Expression Problem

～モジュール万歳～



Expression Problemとは

- 種類がたくさんあって、操作がたくさんある物を抽象化したいときに発生する問題。
  - Ex.) 非終端記号、図形、UIオブジェクト
- 特に、一旦抽象化した物を、再利用しながら拡張するときに困難が発生する。

一番簡単なので図形を例に

- 設定。
  - 図形には、四角と丸がある。
  - 図形を書くdrawと消すclearがある。
  - どう書く？

まずレコードで

```
interface shape {
  void draw()
  void clear()
}
Class rectangle {
  void draw() {…}
  void clear() {…}
}
Class circle {
  void draw(){…}
  void clear(){…}
}
```

操作を抽象化。

これに手を加えずに、moveを加えるのは無理。

ヴァリアントで

```
Module Shape = struct
  type t = Rectangle | Circle
  let draw = function...
  let clear = function...
End
```

データを抽象化。

これに手を加えずに、Triangleを加えるのは無理。

じゃー、どうする？

- これがExpression Problem.(だと思う)
- http://www.daimi.au.dk/~madst/tool/papers/expression.txt
- 1998年のPhilip Wadler

OCamlの解
ポリモーフィックヴァリアント

```
Module Shape = struct
  type t =
    [`Rectangle|`Circle]
  let draw=function...
  let clear=function...
end
```

```
Module Shape' = struct
  type t = [ Shape.t | `Triangle]
  let draw=function
        #Shape.t -> Shape.draw
     | `Rectangle ->...
  let clear=function...
  let move x y = function...
end
```

すごいぜOCaml!

ところが、Shapeを利用するファンクターを考えると…

```
module Canvas(S:ShapeSig) = struct
  let click : S.t -> unit =function
      `Rectangle->...
    | `Circle->...
End
module DCanvas = Canvas(Shape)  <- OK
module ECanvas = Canvas(Shape')  <- NG
```

型が合わない！

操作はsubtyping的

```
module type A = sig
  type t
  let compare : t -> t -> int
End
Module Set(C:A) = struct
  type 'a t = 'a list
  let add : C.t -> 'a t -> 'a t
End
module OSet = Set(struct type t = int
                         let copare = compare
                         let something = "ok" end)
```

Compareさえあれば幸せ。

データはそうはいかない

```
module type Engy = sig
  type t = [`Eele|`Gas]
end
module Car (E:Engy) = struct
  let drive =function
     `Eele -> “clean”
   |`Gas -> “dirty”
End
module Matusda = Car(struct type t = [`Eele|`Gas] end)
module Toyota = Car(struct type t = [`Eele|`Gas|`Hybrid] end)
```

型が合わない！

そこでPrivateRowの登場！

```
module type Engy = sig
  type t = private [>`Eele|`Gas]
end
module Car (E:Engy) = struct
  let drive =function
     `Eele -> “clean”
   |`Gas -> “dirty”
End
module Matusda = Car(struct type t = [`Eele|`Gas] end)
module Toyota = Car(struct type t = [`Eele|`Gas|`Hybrid] end)
```

これだけで解決

やっぱりすごいぜOCaml!

ここまでが前振り。

- Webシステムにおいて、Expression Problemは現れるか？