CS 160 Compilers

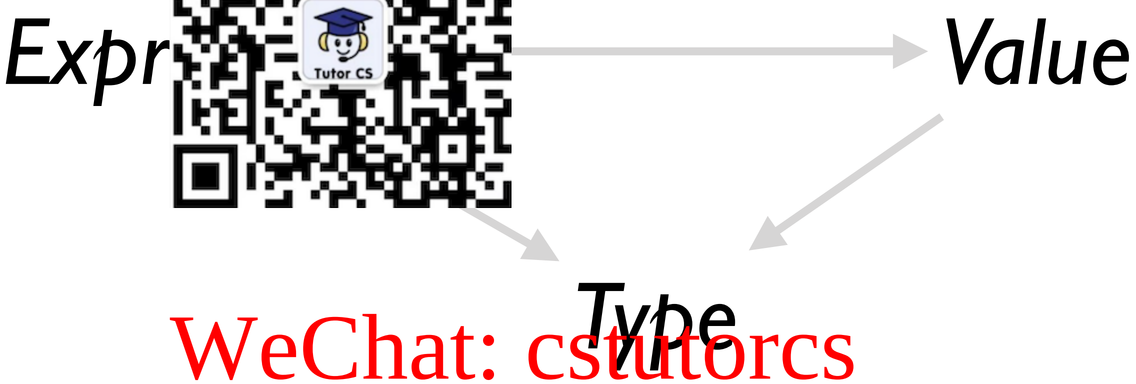
Lectur[illegible] OCaml Crash Course II

Yu Feng
Spring 2023

Outline for today

- Data types
- Pattern matching

ML’s holy grail



- Simple
- Variables
- Functions

Building datatypes

Three key ways complex types/values

- “Each-of” typ

  Value of T contains value of T1 and a value of T2

- “One-of” types:

  Value of T contains value of T1 or a value of T2

- “Recursive”

  Value of T contains (sub)-value of same type T

One of types

We’ve defined a “one-
named attrib

Elements are one of:

- string
- int
- int*int*int
- float
- bool

```
type attrib =
  Name of string
| Age of int
| DOB of int*int*int
| Address of string
| Height of real
| Alive of bool
| Phone of int*int
| Email of string;
```

Test & Take whats in box?

Is it a …

string?

or an int?

or …

Check the TAG!

Whats in the box

```
type attrib =
   Name of string
 | Age of int
 | DOB of int*int*int
 | Address of string
 | Height of real
 | Alive of bool
 | Phone of int*int
 | Email of string;
```

```
match e with
  | Name s -> printf "%s" s
  | Age i -> printf "%d" i
  | DOB(d,m,y) -> printf "%d/%d/%d" d m y
  | Address s -> printf "%s" s
  | Height h -> printf "%f" h
  | Alive b -> printf "%b" b s
  | Phone(a,r) -> printf "(%d)-%d" a r
```

Pattern-match expression: check if e is of the form …

- On match:
  - value in box bound to pattern variable
  - matching result expression is evaluated
- Simultaneously test and extract contents of box

Beware to handle all tags!

```
# match ([illegible]b”) with
| Age i -[illegible].printf "%d" I
| Email s -> Printf.printf "%s" s ;;

Exception: Match Failure!!
```

None of the cases matched the tag (Name)

Causes nasty Run-Time Error!

Compiler to rescue!

```
# # let printAt[illegible] match a with
| Name s -> Printf.printf "%s" s
| Age i -> Printf.printf "%d" I
| DOB (d,m,y) -> Printf.printf "%d / %d / %d" d m y
| Address addr -> Printf.printf "%s" addr
| Height h -> Printf.printf "%f" h
| Alive b -> Printf.printf "%b" b
| Email e -> Printf.printf "%s" e
;;
Warning P: this pattern-matching is not
exhaustive.Here is an example of a value that is
not matched:Phone (_, _)
```

Compile-time checks for:
missed cases: ML warns if you miss a case!

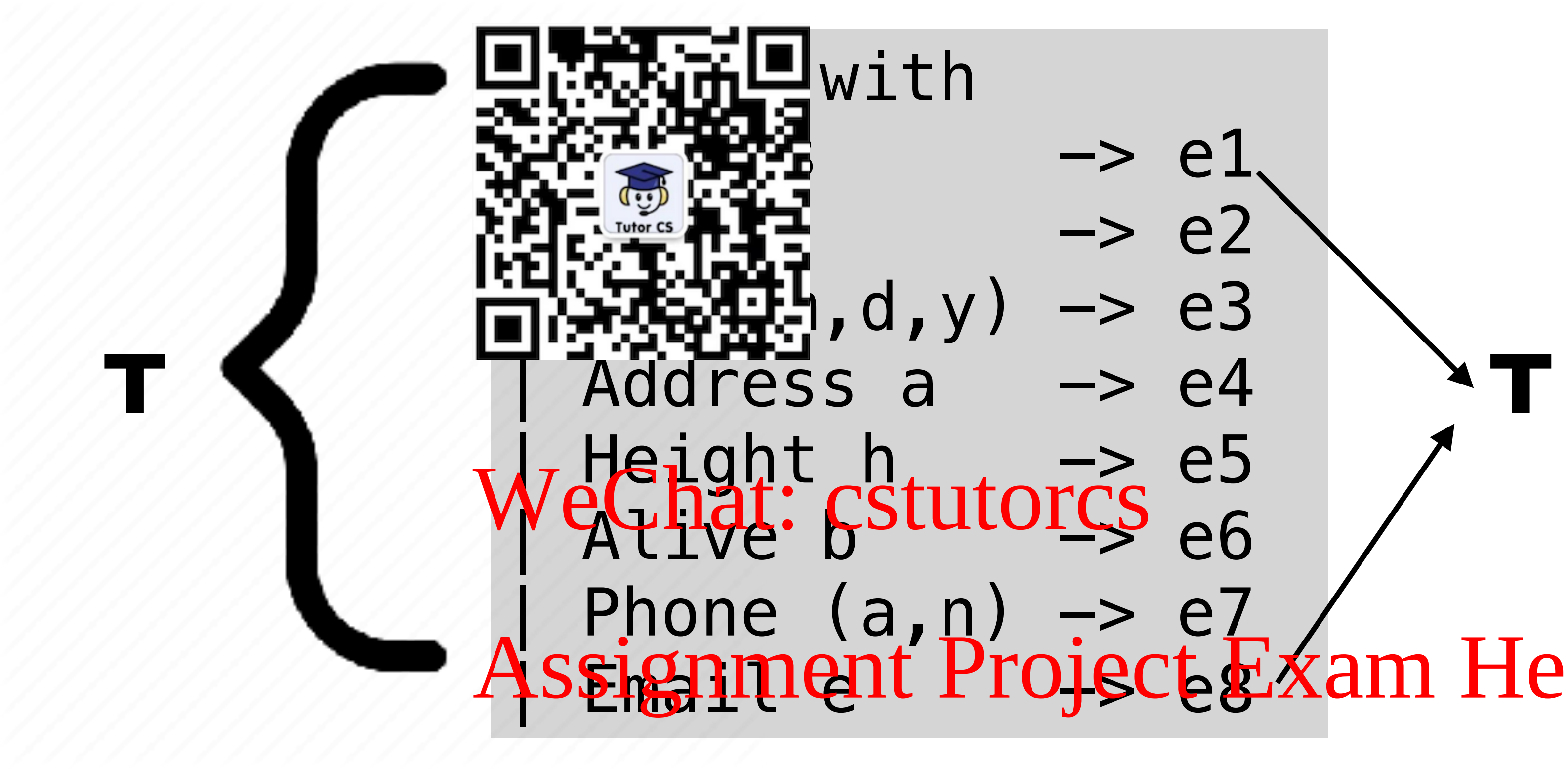
match-with is an Expression

```
        e with
       x1 -> e1
    | C2 x2 -> e2
    | ...
    | Cn xn -> en
```

Type Rule

- e1, e2,...,en must have same type T
- Type of whole expression is T

match-with is an Expression



```
             with
                      -> e1
                      -> e2
         ,d,y)        -> e3
 | Address a          -> e4
 | Height h           -> e5
 | Alive b            -> e6
 | Phone (a,n)        -> e7
 | Email e            -> e8
```

Type Rule

- e1, e2,...,en must have same type T
- Type of whole expression is T

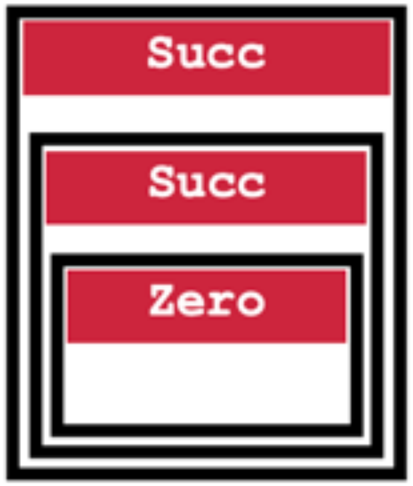
Recursive types

```
type nat = Zero | Succ of nat
```

What are values of nat ?

One nat contains another!

nat = recursive type



plus: nat*nat -> nat

```
nat =
ro
cc of nat
```

Base patte

Inductive patt

```
let rec plus n m =
match m with
| Zero -> n
| Succ m’ -> Succ (plus n m’)
```

Base pattern

Inductive pattern

Base expression

Inductive expression

List datatype

```
type          ist =
  N
| C           int * int_list
```

Lists are a derived type: built using elegant core!

1. Each-of
2. One-of
3. Recursive

:: is just a syntactic sugar for “Cons”

[] is a syntactic sugar for “Nil”

List function: length

```
let rec len l =
    match l with
    | Nil -> 0
    | Cons(h,t) -> 1 + (len t)
```

Base pattern

Base expression

Inductive pattern

Inductive expression

List function: list_max

```
let rec li        l =
    match l with
  | Nil -> 0
  | Cons(h,t) -> max h (list_max t)
```

Base pattern

Inductive pattern

Base expression

Inductive expression

```
let max x y = if x > y then x else y;;
```

TODOs by next lecture

- Come to the discussion if you have questions
- Start to work on HW1