Lec 10 : ILFP

How to define a type class?

```
class C a where
  m :: ctx ⇒ a → b
```

- `C` — Class name
- `m` — Operation/method of C
- `a` — Polymorphic type : a as an instance of Eq if a has (overloaded) m defined on it
- `ctx ⇒` — context
- `a → b` — type of m

Eg:

```
class Eq a where
  (==) :: a → a → Bool
```

Thus, (Eq a) is a context or constraint on type a (it is not a type expression)

Instance declarations

```
instance Eq MyInt where
  x == y = myIntEq x y
```

myIntEq → your own defn overriding Eq's defn of (==)

Class Extn

```
class (Eq a) => Ord a where
```

(<)
(<=)
(>=)
(>)

→ Extra methods specific to Ord

Ord is a subclass of Eq

Modules

— dual purpose : Controlling name spaces
ADT
Creating

- name are alphanumeric & must begin with an uppercase letter

- Technically, it is one big declaration with keyword module

Eg :

```
module Tree where
  data BinTa = Empty T | Node a
                (BinTa)
                (BinTa)
  :
  ; funcs
  ,
```

module exports everything i.e all names bound at the top level

I could have a declaration

module Tree (BinT (EmphyT, Node), fnclist ...)

Visibility for Export

— Use import

module C where

import Tree (...)

Abstract Data Types

module TreeADT (BinT, ----) where

data BinT a = Constructors ++

⋮] Signature

→ Constructors are hidden

- Motivation:
  - on demand compilation (importing the module)
  - abstraction: only the interface is exposed, the implementations can change

Structural Induction

- Let U be the Universe set.
- Let $B \subseteq U$, be a non-empty subset
- Let K be the set of constructors
- Let $\mathcal{X}$ be a family of subsets of U s.t.
  - Basis. $B \subseteq X$
  - I.S. If $f \in K$ is of arity $n \geq 0$ and $a_1, a_2, \ldots, a_n \in X$ then $f(a_1, a_2, \ldots, a_n) \in X$

- A set A is said to be inductively defined from B, K, U if it is the smallest set satisfying the above conditions, i.e.

$$A = \bigcap_{X \in \mathcal{X}} X$$

Construction Sequence

A sequence $[a_1, a_2, \ldots, a_m]$ of elements of U is called a construction sequence for a_m if for all $i = 1, \ldots m$,

- Either $a_i \in B$ or
- $\exists f \in K$, s.t. $\alpha(f) = n \geq 0$ and $0 < i_1 \ldots i_n < i$ s.t. $f(a_{i_1} \ldots a_{i_n}) = a_i$
- a_i depends on a_j if $a_j \prec a_i$ (prefix ordering)

Eg: MyExpr:

- Basis. Every $n \in \mathbb{N}$ is a MyExpr
- I.S.
  - if e & e' are MyExpr then add(e,e') & mult(e,e') are also MyExpr.
  - Nothing else is a MyExpr

Basis: $\mathbb{N}$

$K = \{add, mult\}$, U = set of all finite symbols constructed from $\mathbb{N}$ and application of Constructors.

- Constructor sequence for

mult (add (0,1), mult (1,1))

$$= [\ 0,\ 1,\ \text{add}(0,1),\ \text{mult}(1,1),\ \text{mult}(\text{add}(0,1), \text{mult}(1,1))\ ]$$

BNF notation

$$e, e' ::= n \in \mathbb{N} \mid \text{add}(e, e') \mid \text{mult}(e, e')$$

Principle of Structural Induction

Let $A \subseteq U \neq \emptyset$ be an inductively defined set. using $B \neq 0$ & $K \neq \emptyset$

- A property P holds for every element of provided :

  - Basis. $P(b)$, $\forall b \in B$ is true

  - J.S. $\forall f \in K$, if $P(a_i)$ holds where $i \in \{1, \ldots, \alpha(f)\}$ and $a_i \in A$ then it implies $P(f(a_1, \ldots a_{\alpha(f)}))$ also holds !

Eg: BNF

$e, e' := n \in \mathbb{N} \mid (e + e') \mid (e - e') \mid (e * e')$

Show for every prefix e' of e (where e is an expr of the above BNF)

$\mathcal{L}(e') \geq R(e')$

where $\mathcal{L}(e')$: # of (in e'

$R(e')$ = # of) in e'