Typsystem

Monomorfiska
Överlagrade
Polymorfa
Arv

Typkontroll

- Statisk kontra dynamisk typning
  - Statisk kontrolleras under kompilering
  - Dynamisk kontrolleras under exekvering
- Typekvivalens
  - Strukturekvivalens : innehåller samma delar
  - Namnekvivalens: Typerna deklarerade på samma ställe
- Typkompletthet
  - Ingen operation skall vara godtyckligt begränsad när det gäller de inblandade typerna

Monomorfiska typer

- Varje konstant och variabel har en entydig typ
- Alla parametrar till funktioner och procedurer och alla funktionsresultat har en entydig typ
- Tvingar oss att skriva flera exakt lika funktioner där enda skillnaden är parametertypen

Överlagring (Ad-hoc-polymorfism)

- En identifierare sägs vara överlagrad om den samtidigt betecknar två eller flera funktioner
- Acceptabelt endast om varje funktionsanrop är entydigt bestämbart med t ex typinformation
- ML och Pascal endast inbyggda operatorer
- Ada har användardefinierade

Överlagrade identifierare

- Om I betecknar f_1: $S_1 \rightarrow T_1$ och f_2 : $S_2 \rightarrow T_2$
- **Kontextoberoende** överlagring (ML, Pascal, Java)
  - S_1 och S_2 måste vara olika, skilja sig från varandra
  - Anropade funktionen bestäms entydigt av argumenten
- **Kontextberoende** överlagring (Ada)
  - S_1 och S_2 eller T_1 och T_2 måste vara distinkta.
  - Om S_1 och S_2 olika så bestämmer de entydigt.
  - Om S_1 och S_2 lika men T_1 och T_2 olika så bestämmer kontexten

Polymorfiska typer
(Parametriserad polymorfism)

- En funktion som kan ta argument av olika typer, men gör samma sak med alla är polymorfisk.
- Polytyp
  - Typ med typvariabler.
- Pascal: `eof(f)` som `File('a) -> bool`
- ML : `null : 'a list -> bool`

Polymorfisk abstraktion

- Använder typvariabler istället för specificerade typer

```
fun snd (x:int,y:int) = y
fun snd (x:'a,y:'b) = y
fun snd (x,y) = y

fun len [] = 0
  | len (x::xs) = 1 + len xs
val len = fn : 'a list -> int
len [1,2,3] ==> 3
len [(1.0,true),(2.0,false),(3.0,true)]
    ==> 3
```


Parameteriserade typer

- En parametriserad typ är en typ som har andra typer som argument (en typfunktion)
  - Pascal: endast inbyggda typkonstruerare
    ```
    type Intpair = record fst, snd : Integer end;
    ```
  - ML: generella instanser av fördefinierade konstruerare
    ```
    type 'a mylist = (int * 'a) list
    ```
  - Även parametriserade typer med egna typkonsturerare
    ```
    datatype 'a tree = Tip | Node of 'a * 'a tree * 'a tree
    type inttree = int tree
    fun mirror Tip = Tip
      | mirror (Node (a,l,r)) = Node (a,mirror r,mirror l)
    val mirror = fn : 'a tree -> 'a tree
    ```


Polytyp och monotyp

- En **polytyp** är en typ som innehåller typvariabler
- En **typvariabel** kan stå för vilken typ som helst
- En **monotyp** innehåller inga typvariabler
- En **polytyp** är en typ i sig och innehåller därför en mängd värden
- Mängden värden i en polytyp är snittet av alla typer som kan härleddas från den
- Ex `'a list`, snittet här blir tomma listan som är gemensam för alla instanser av `'a list`

Typinferens

- När ingen explicit typning görs
- Typen härleds istället av systemet
- Pascal: `const I = E;`
- **Monomorfisk**
  ```
  fun even n = n mod 2 = 0
  val even = fn : int -> bool
  ```
- **Polymorfisk**
  ```
  fun id x = x
  val id = fn : 'a -> 'a
  ```


Implicit typomvandling (coercion)

- Implicit omvandling av ett värde av en typ till ett värde av en annan typ
- Måste finnas en unik avbildning
  - **Widening**
  - **Dereferencing**
  - **Rowing**
  - **Uniting**
  - **Voiding**
- Moderna språk undviker alltför mycket coercion

Subtyper och arv

(Inclusion polymorfism)

- **Subtyp**
  - Om en typ omfattar en mängd värden så är en delmängd av den typen en subtyp (deltyp)
  - Formellt: S är en subtyp till T omm $S \subseteq T$
  - Ett värde av typ S kan användas på alla ställen där det förväntas ett värde av typ T
- **Arv**
  - En subtyp ärver alla operationer som fungerar på supertypen