

Langages et types

- typage statique : Ada, O'CamI, Eiffel, Haskell
- typage dynamique : Lisp, Scheme, CLOS, Smalltalk
- typage statique ET dynamique :
 - Java 1.4, C# 1.0
 - Java 1.5, C# 2.0

tendance typage statique : sûreté + généricité

Polymorphismes

Dans le cadre du typage statique :

- paramétrique (à la ML)
- *ad hoc* ou de surcharge
- objets ou d'inclusion

volonté d'intégration des différents polymorphismes dans les langages statiquement typés

Inférence de types

- totale en O'CamL, Haskell
 - sauf dans les interfaces .mli
 - peut être nécessaire de l'aider
 - extension objet, ...
- partielle pour la résolution des wildcard (<?>) en Java 1.5

Surcharge et inférence (1)

en O'Cam1 : difficultés liées à l'inférence

- polymorphisme paramétrique

```
# let g o = o#m();;
val g : < m : unit -> 'a; .. > -> 'a = <fun>
# let f x y = x#m y;;
val f : < m : 'a -> 'b; .. > -> 'a -> 'b = <fun>
```

Surcharge et inférence (2)

- application partielle :

Soit une classe c ayant 2 méthodes :

- $m : \text{int} \rightarrow \text{int} \rightarrow \text{int}$
- $m : \text{int} \rightarrow \text{float}$

```
let g (o:c) (i:int) = o#m i;;
```

Quelle méthode m choisir?

Surcharge et inférence - Haskell (3)

En Haskell :

- définir des types de classes regroupant des ensembles de fonctions surchargées.
- Une déclaration de classe définit une nouvelle classe et les opérateurs que celle-ci autorise.
- Une déclaration d'instance (d'une classe) indique qu'un certain type est une instance d'une classe. Cela inclue la définition des opérateurs surchargés de sa classe pour ce type.

```
class Num a where
  (+)      :: a -> a -> a
  negate  :: a -> a
```

Surcharge et inférence - Haskell (4)

On peut maintenant déclarer une instance `Int` de la classe `Num` de cette manière :

```
instance Num Int where
  x + y      = addInt x y
  negate x   = negateInt x
```

Et l'instance `Float`

```
instance Num Float where
  x + y      = addFloat x y
  negate x   = negateFloat x
```

L'application de `negate Num` aura un comportement différent si l'argument est de l'instance `Int` ou `Float`.

Surcharge et inférence - Haskell (5)

type de `negate`: `Num a => a -> a`

\Rightarrow : un argument supplémentaire (class restriction) à l'exécution pour effectuer le dispatch.

```
fib :: Num a, Num b => a -> b
```

```
fib n = fibGen 0 1 n
```

```
fibGen :: Num a, Num b => b -> b -> a -> b
```

```
fibGen a b n = case n of
```

```
    0 -> a
```

```
    n -> fibGen b (a + b) (n - 1)
```


Surcharge et inférence (6)

Extension du système de types en O'Caml

- Langage reFLect (Intel) et surcharge :

```
overload print : IO.output -> 'a -> unit  
overload _+_ : 'a -> 'a -> 'a
```

```
instance print Int.print Float.print  
instance _+_ Int._+_ Float._+_
```

voir : <http://gallium.inria.fr/~pouillar>

Génériques (1)

classes paramétrées :

- C++ : code spécialisé pour chaque instance de template
- O'Cam1 : reste dans le cadre du polymorphisme paramétrique de la couche fonctionnelle
- Java : même machine virtuelle, compatibilité ascendante, code engendré compatible avec la JVM, polymorphisme borné
- C# 2.0 : machine virtuelle avec instructions génériques
 - Design and Implementation of Generics for the .NET Common Language Runtime Andrew Kennedy and Don Syme. PLDI 2001



<http://research.microsoft.com/akenn/generics/index.htm>

Génériques (2)

- même code pour tout type :
 - représentation uniforme des données (α)
 - paramètre supplémentaire (class restriction) puis dispatch
 - monomorphisation : code spécialisé pour chaque instance :
 - ⇒ code plus important
 - casts sûrs dans le code :
 - ⇒ code moins rapide
- ⇒ conséquences sur les performances (GC, ...)

Intégration

- Styles de programmation
 - fonctionnel(paramétrique)/objet (à la O'Caml)
 - fonctionnel(paramétrique)/*ad hoc* : fonctions génériques (à la CLOS ou à la Haskell)
 - objet/*ad hoc* : Java 1.4, C# 1.0
 - objet/*ad hoc*/paramétrique : Java 1.5, C# 2.0
- Mélange
 - F# : C# pour l'objet + caml-light pour le fonctionnel/impératif
 - C# 3.0 : λ -expressions, inférence de types des variables locales
- Narbel Ph : Programmation fonctionnelle, generique et objet. Vuibert. 2005.

Interopérabilité

- FFI : foreign function interface (external O'CamI, JNI Java)
- IDL : fonctions, données (COM)
- IDL : objets (exemple O'Jacaré)

runtime commun : facilite l'interopérabilité (O'Jacaré.NET)

Typage statique ou dynamique

Tendance :

- vers le typage statique
- avec du typage dynamique pour les valeurs/programmes venant de l'extérieur (sérialisation, réseau)