Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

•••

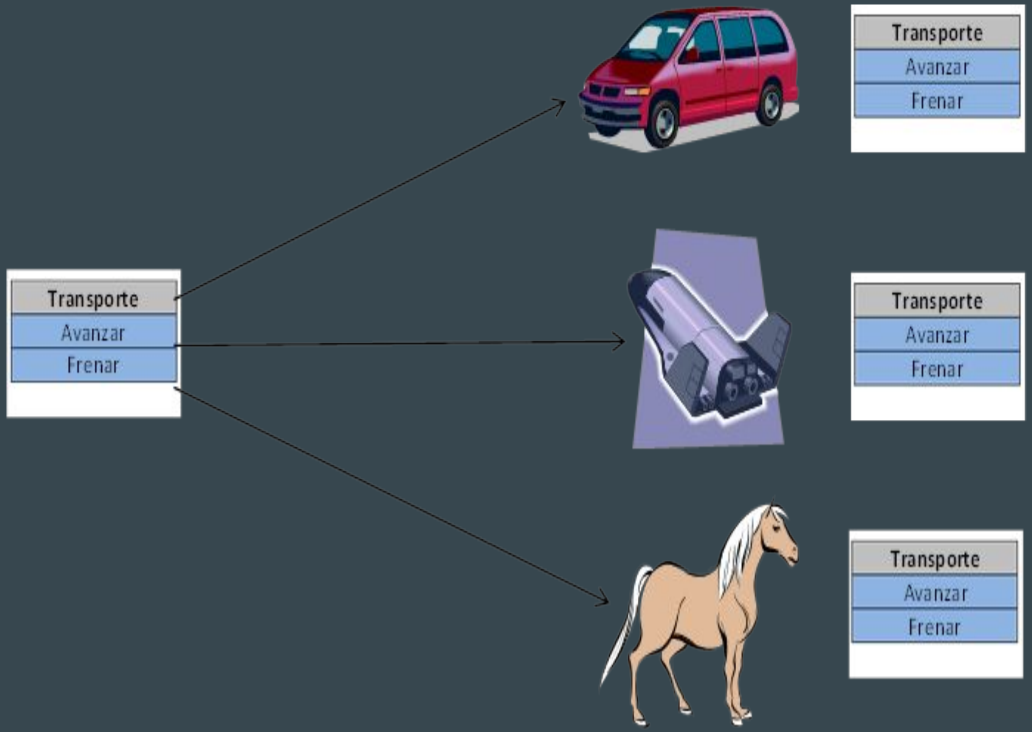
Karla Ramírez Pulido

Polimorfismo

Tipos de polimorfismo

1. Explícito

1. Implícito



Polimorfismo Explícito

Veamos la siguiente función lengthNum y lengthSym:

```
(define lengthNum
  (lambda (l : numlist) : number
    (cond
      [(numEmpty? l) 0]
      [(numCons? l) (add1 (lengthNum (numRest l)))])))
```

```
(define lengthSym
  (lambda (l : symlist) : number
    (cond
      [(symEmpty? l) 0]
      [(symCons? l) (add1 (lengthSym (symRest l)))])))
```

Polimorfismo Explícito

```
(define length Num
  (lambda (l : num list) : number
    (cond
      [( num Empty? l) 0]
      [( num Cons? l) (add1 (length Num ( num Rest l)))])))
```

```
(define length Sym
  (lambda (l : sym list) : number
    (cond
      [( sym Empty? l) 0]
      [( sym Cons? l) (add1 (length Sym ( sym Rest l)))])))
```

Si los tipos los representamos con τ tendríamos:

```
(define length
  (lambda (l : τ list) : number
    (cond
      [(τ Empty? l) 0]
      [(τ Cons? l) (add1 (length (τ Rest l)))])))
```

Función que verificará los tipos: $\Lambda(\tau)$

Entonces tendremos lista de elementos de tipo τ

Funciones del lenguaje se aplicarán a elementos de tipo τ

```
(define length
  <Λ (τ)
    (lambda (l : list(τ)) : number
      (cond
        [(Empty?_τ l) 0]
        [(Cons?_τ l) (add1 (length (Rest_τ l)))]))>)
```

Y la misma función lenght recibirá elementos de tipo τ

```
(define length
  <Λ (τ)
    (lambda (l : list(τ)) : number
      (cond
        [(Empty?<τ> l) 0]
        [(Cons?<τ> l) (add1 (length<τ> (Rest<τ> l)))]))>)
```

Llamadas a la función lenght

2 llamadas a la función lenght con elementos de distinto tipo

(*length*<num> (list 1 2 3))
(*length*<sym> (list 'a 'b 'c))

Variables de Tipo

Son introducidas por los procedimientos de tipo Λ

y son inicializadas por las aplicaciones de funciones.

Para la función length

> (lenght ‘(1 3))

2

Toma 1: $\text{lenght} : \text{type} \rightarrow (\text{list}(\text{type}) \rightarrow \text{number})$

Toma 2:

$$\mathit{length} : \forall \alpha .\ \text{list}(\alpha) \rightarrow \text{number}$$

Para la función map

```
> (map add1 ‘(1 3 5))

(2 4 6)
```

$$map : \forall \alpha, \beta.\ \mathsf{list}(\alpha) \times (\alpha \rightarrow \beta) \rightarrow \mathsf{list}(\beta)$$

Polimorfismo Implícito

Consideremos la función identidad:

(lambda (x) x)

Tipos: $\alpha \rightarrow \alpha$

¿En qué tiempo se conocen los tipos de "x"?

La siguiente función:

¿Qué regresa el código siguiente?

¿De qué tipo recibe la función identidad?

```
(let ([id (lambda (x) x)])
  (+ (id 5)
     (id 6)))
```

¿Qué regresa el siguiente código?

```
(let ([id (lambda(x) x ) ])
    (if (id true)
        (id  5)  ;then-expr
        (id  6)   ;else-expr
))
```

¿Qué tipos recibe y regresa “id”?

Siguiendo el ejemplo anterior:

Los tipos de la función identidad:

```
(let ([id (lambda (x) x)])
  (if ([1](lambda (x) x) true)
      ([2](lambda (x) x) 5)
      ([3](lambda (x) x) 6)))
```

En 1: $\alpha \rightarrow \alpha$

En 2: $\beta \rightarrow \beta$

En 3: $\gamma \rightarrow \gamma$

Juicio de tipo para LET

$$\frac{\Gamma \vdash v : T' \qquad\qquad \Gamma[\,x \leftarrow \mathrm{CLOSE}\,(T',\ \Gamma)\,] \vdash b : T}{\Gamma \vdash (\,let\ [x\ \ v]\ \ b) \ :\ T}$$

¿Dudas?

Gracias