

# Εξαρτώμενοι Τύποι Dependent Types

Συστήματα Τύπων  
των Γλωσσών Προγραμματισμού

Νίκος Παπασπύρου

*(με ιδέες και κομμάτια κλεμμένα από διαφόρους)*

## οι τύποι συναρτήσεων μέχρι τώρα

$\lambda x:T_1. t_2$	family of terms indexed by terms
$\lambda X::K_1. t_2$	family of terms indexed by types
$\lambda X::K_1. T_2$	family of types indexed by types.

- Τί λείπει;

$\lambda x:T_1. T_2$  family of types indexed by terms

## για ζέσταμα

- The *type* of a function can depend on the argument *value*
- Ποιος είναι ο τύπος αυτής της έκφρασης;  
**if x>2 then 1 else true**
- Θα μπορούσε να είναι:  
**if x>2 then Nat else Bool**
- αν επιτρέπαμε *υπολογισμούς τύπων*, ακριβώς όπως υπολογίζονται οι εκφράσεις

## για ζέσταμα

- και ο τύπος αυτής της συνάρτησης;

```
λx:Nat. if x>2 then 1 else true
```

- δεν αρκεί:

```
Nat → if x>2 then Nat else Bool
```

- γιατί η μεταβλητή **x** είναι η παράμετρος της συνάρτησης!

```
Πx:Nat. if x>2 then Nat else Bool
```

# εξαρτώμενες συναρτήσεις

- Dependent Products

$$\Pi x:T_1. T_2$$

$$\frac{\Gamma \vdash A \text{ type} \quad \Gamma, x:A \vdash M : B}{\Gamma \vdash \lambda x:A.M : \Pi x:A.B}$$
$$\frac{\Gamma \vdash M : \Pi x:A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : \{N/x\}B}$$

- Μη εξαρτώμενες συναρτήσεις:

$$T_1 \rightarrow T_2 \equiv \Pi x:T_1. T_2 \quad \text{για κάποιο } x \notin FV(T_2)$$

# εξαρτώμενα ζεύγη

- Dependent (strong) Sums

$$\Sigma x:T_1. T_2$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : \{M/x\}B}{\Gamma \vdash \langle M, N \rangle : \Sigma x:A.B}$$
$$\frac{\Gamma \vdash M : \Sigma x:A.B}{\Gamma \vdash \pi_1(M) : A}$$
$$\frac{\Gamma \vdash M : \Sigma x:A.B}{\Gamma \vdash \pi_2(M) : \{\pi_1(M)/x\}B}$$

- Μη εξαρτώμενα ζεύγη:

$$T_1 \times T_2 \equiv \Sigma x:T_1. T_2 \quad \text{για κάποιο } x \notin FV(T_2)$$

# εξαρτώμενα ζεύγη

- Dependent (strong) Sums

$$\Sigma x:T_1. T_2$$

- επιτρέπουν τις προβολές και των δύο:  $\pi_1$  και  $\pi_2$

- Existentials = weak sums

$$\exists x:T_1. T_2$$

- απαγορεύουν την  $\pi_1$
- επιτρέπουν την  $\pi_2$  υπό την προϋπόθεση να μην αποκαλύπτει έμμεσα το  $x$

$$\frac{\Gamma \vdash e_1 : \exists x:T_1. T_2 \quad \Gamma \vdash T :: *}{\Gamma, x:T_1, y:T_2 \vdash e_2 : T} \\ \Gamma \vdash \text{let } \{x,y\} = e_1 \text{ in } e_2 : T$$

είναι χρήσιμοι οι εξαρτώμενοι τύποι,  
ή παίζουμε;

- Μέγεθος πίνακα

```
sort = λx:Nat. λα:array(x) . κ.λπ.
```

- Τί τύπο έχει μια συνάρτηση σαν την printf;

```
printf("Hello world\n")
```

```
printf : String -> Unit
```

```
printf("Number = %d\n", n)
```

```
printf : String -> Int -> Unit
```



μπα, η printf έγινε type-safe;

```
printf : Πfmt:String. PrintfType fmt
```

```
PrintfType :: String => *
PrintfType (Nil)           = String
PrintfType ('%' : 'd' : cs) = Int -> PrintfType cs
PrintfType ('%' : 's' : cs) = String -> PrintfType cs
                          ...
PrintfType ('%' : _ : cs)   = PrintfType cs
PrintfType (_ : cs)        = PrintfType cs
```

```
printf "%s = %d" "five" 5
```

```
printf : String -> String -> Int -> String
```

μπα, η printf έγινε type-safe;

```
printf fmt = pr fmt "" where
  pr ""          res = res
  pr ('%': 'd': cs) res =
    \ i -> pr cs (res ++ show (i::Int))
  pr ('%': 's': cs) res =
    \ s -> pr cs (res ++ s)
  pr ('%': c : cs) res =
    pr cs (res ++ [c])
  pr (c:cs)      res =
    pr cs (res ++ [c])
```

# βρες το λάθος!

```
if null xs then tail xs else xs
```

- Λίστες με μέγεθος, χωρίς dependent types

```
data List0 x = Nil
```

```
data List1 x = Cons0 x (List0 x)
```

```
data List2 x = Cons1 x (List1 x)
```

```
...
```

# λίστες με μέγεθος

- `data List :: * => Nat => *`  
  `= nil : ∀a::*. List a 0`  
  `| cons : Πn:Nat. ∀a::*.`  
    `α -> List a n -> List a (succ n)`

- **πράξεις με λίστες:**

`head : Πn:Nat. ∀a::*.`

`List a (succ n) -> a`

`tail : Πn:Nat. ∀a::*.`

`List a (succ n) -> List a n`

`mult : ∀i,j,k:Nat ⇒ Matrix i j → Matrix j k → Matrix i k`

# η συνάρτηση του Ackerman τερματίζει!

ack =  $\lambda m. \lambda n.$

$\langle i, j \rangle$

if  $m = 0$  then  $n+1$

else

if  $n = 0$  then ack  $(m-1)$  1

$\langle i-1, 1 \rangle$

else ack  $(m-1)$  (ack  $m$   $(n-1)$ )

$\langle i-1, k \rangle$

$\langle i, j-1 \rangle$

$\langle i, j \rangle$	>	$\langle i-1, 1 \rangle$
$\langle i, j \rangle$	>	$\langle i-1, k \rangle$
$\langle i, j \rangle$	>	$\langle i, j-1 \rangle$

with type

$\{i:\text{nat}, j:\text{nat}\} \langle i, j \rangle \Rightarrow \text{int}(i) \rightarrow \text{int}(j) \rightarrow [k:\text{nat}] \text{int}(k)$

# κάποιο λάκκο έχει η φάβα!

- Τί χάνουμε με τα dependent types?
  - **decidability** of the type system
- γιατί;
  - ο *έλεγχος τύπων* εξαρτάται από τον *έλεγχο ισότητας εκφράσεων*

`(λx: (if t=s then Nat else Bool) . 3) true`

# μπορούμε να σώσουμε το decidability;

- Ναι, αν μειώσουμε την ισχύ των dependent types
  - δεν μπορούν όλες οι εκφράσεις να χρησιμοποιούνται σε εξαρτήσεις
  - μόνο οι “καθαρές”
- Τί ιδιότητες έχουν οι *καθαρές εκφράσεις*;
  - πάντα τερματίζουν (strongly normalizing)
  - όχι side-effects
- Παρόλα αυτά, το type checking παραμένει *δύσκολο* και (γενικά) *αυθαίρετα χρονοβόρο*

# χμμ, τί άλλο μπορεί να εκφραστεί; (Dependent ML)

- Array

$$\Pi s : \text{nat}. \alpha \text{ array}(s)$$

- Integers between  $l$  and  $r$

$$\Sigma n : \{n : \text{nat} \mid l \leq n \leq r\}. \text{int}(n)$$

- Array of integers between  $l$  and  $r$

$$\Pi s : \text{size}. \left( \Sigma n : \{n : \text{nat} \mid l \leq n \leq r\}. \text{int}(n) \right) \text{array}(s)$$



## άλλο τίποτα;

- The “unchecked” subscript function

$$\text{sub} : \Pi s : \text{int}. \Pi i : \{i : \text{int} \mid 0 \leq i < s\}.$$

$$\alpha \text{ array}(s) \rightarrow \text{int}(i) \rightarrow \alpha$$

- Subscripting arrays of integers between  $l$  and  $r$

$$A : \left( \Sigma n : \{n : \text{nat} \mid l \leq n \leq r\}. \text{int}(n) \right) \text{array}(10)$$

$$\text{sub}[10][3] A 3 : \Sigma n : \{n : \text{nat} \mid l \leq n \leq r\}. \text{int}(n)$$

Required showing:

$$s \doteq 10, i \doteq 3 \models 0 \leq i \leq s$$

μπα, τι λες!

The type of “less-than”

$$\begin{aligned} & \Pi i, j : \mathbf{int}. \mathbf{int}(i) \rightarrow \mathbf{int}(j) \rightarrow \\ & \Sigma b : \{b : \mathbf{bool} \mid (i < j \wedge b) \vee (i \geq j \wedge \neg b)\}. \\ & \mathbf{bool}(b) \end{aligned}$$

και τα εξαρτώμενα ζεύγη  
πού αλλού χρησιμεύουν;

- `filter p nil = nil`  
`filter p (h :: t) =`  
    `if p h then h :: filter t`  
    `else filter t`

- με τύπο:

`Πn:nat. int list(n) ->`  
`Σm:nat. int list(m)`

- ή καλύτερα:

`Πn:nat. int list(n) ->`  
`Σm:{m:nat | m<=n}. int list(m)`

# ποιες γλώσσες παίζουν με dependent types;

- Cayenne  
<http://www.cs.chalmers.se/~augustss/cayenne/>
- Dependent ML (DML, deCaml)  
<http://www.cs.bu.edu/~hwxi/DML/DML.html>
- Epigram  
<http://sneezy.cs.nott.ac.uk/epigram/>
- Omega  
<http://www.cs.pdx.edu/~sheard/Omega/>
- Coq  
<http://coq.inria.fr/>

Θα δούμε τίποτα άλλο σήμερα;

- Hongwei **Xi** and Frank **Pfenning**,  
**Dependent types in practical programming**
- Thorsten **Altenkirch** , Conor **McBride** and  
James **McKinna**,  
**Why dependent types matter**
- Lennart **Augustsson**,  
**An exercise in dependent types:  
A well-typed interpreter**