Discussion 02

Functions

Kenneth Fang (kwf37), Newton Ni (cn279)

Jan. 28, 2019

Key Concepts

Functions are ...

- **values**.

Key Concepts

Functions are ...

- **values**.
- characterized by their **type**.

Syntax

Definition

- `let f = fun x -> e`
- `let f x = e`

Syntax

Definition

- `let f = fun x -> e`
- `let f x = e`

Application

- `f x y z`

Static Semantics

Definition: `fun x1 ... xn -> e`

- If each parameter `xi` has type `ti`,
- If the body `e` has type `u` (knowing the types of each `xi`)
- Then conclude type `t1 -> ... -> tn -> u`

Static Semantics

Definition: `fun x1 ... xn -> e`

- If each parameter `xi` has type `ti`,
- If the body `e` has type `u` (knowing the types of each `xi`)
- Then conclude type `t1 -> ... -> tn -> u`

Application: `e0 e1 ... en`

- If `e0` has type `t1 -> ... -> tn -> u`
- If each argument `ei` has type `ti`
- Then conclude type `u`

Dynamic Semantics

Definition: `fun x1 ... xn -> e`

Dynamic Semantics

Definition: `fun x1 ... xn -> e`

- Already a value: no need to evaluate further!

Dynamic Semantics

Definition: `fun x1 ... xn -> e`

- Already a value: no need to evaluate further!

Application: `e0 e1 ... en`

- Evaluate each expression `ei` to a value `vi`
  - `e0` will evaluate to a function `v0`: why?
  - `ei` will evaluate to argument `vi`
- Substitute each argument into the body of `e` to get `e'`
- Evaluate `e'` to value `v`

Guess the Type

- `fun x -> x + 2`

Guess the Type

- `fun x -> x + 2`
- `fun x -> x +. 2.0`

Guess the Type

- `fun x -> x + 2`
- `fun x -> x +. 2.0`
- `fun x y -> x + y`

Guess the Type

- `fun x -> x + 2`
- `fun x -> x +. 2.0`
- `fun x y -> x + y`
- `fun x y z -> x + y`

Guess the Type

- `fun x -> x + 2`
- `fun x -> x +. 2.0`
- `fun x y -> x + y`
- `fun x y z -> x + y`
- `fun x -> x`

Partial Application

- Apply a function with `n` parameters to `m` arguments, `m < n`

Partial Application

- Apply a function with `n` parameters to `m` arguments, `m < n`
- Get back a function that takes `n - m` arguments

Currying

Uncurried

- `(fun x y z -> x + y + z) 1 2 3`

Currying

Uncurried

- `(fun x y z -> x + y + z) 1 2 3`
- `1 + 2 + 3`

Currying

Uncurried

- `(fun x y z -> x + y + z) 1 2 3`
- `1 + 2 + 3`
- `6`

Currying

Uncurried

- `(fun x y z -> x + y + z) 1 2 3`
- `1 + 2 + 3`
- `6`

Curried

- `(fun x -> fun y -> fun z -> x + y + z) 1 2 3`

Currying

Uncurried

- `(fun x y z -> x + y + z) 1 2 3`
- `1 + 2 + 3`
- `6`

Curried

- `(fun x -> fun y -> fun z -> x + y + z) 1 2 3`
- `(fun y -> fun z -> 1 + y + z) 2 3`

Currying

Uncurried

- `(fun x y z -> x + y + z) 1 2 3`
- `1 + 2 + 3`
- `6`

Curried

- `(fun x -> fun y -> fun z -> x + y + z) 1 2 3`
- `(fun y -> fun z -> 1 + y + z) 2 3`
- `(fun z -> 1 + 2 + z) 3`

Currying

Uncurried

- `(fun x y z -> x + y + z) 1 2 3`
- `1 + 2 + 3`
- `6`

Curried

- `(fun x -> fun y -> fun z -> x + y + z) 1 2 3`
- `(fun y -> fun z -> 1 + y + z) 2 3`
- `(fun z -> 1 + 2 + z) 3`
- `1 + 2 + 3`

Currying

Uncurried

- `(fun x y z -> x + y + z) 1 2 3`
- `1 + 2 + 3`
- `6`

Curried

- `(fun x -> fun y -> fun z -> x + y + z) 1 2 3`
- `(fun y -> fun z -> 1 + y + z) 2 3`
- `(fun z -> 1 + 2 + z) 3`
- `1 + 2 + 3`
- `6`

Exercises

- More type-checking exercises in `types.ml`

Exercises

- More type-checking exercises in `types.ml`
- Basic calculus in `calculus.ml`