5118014 Principles of Programming Language

Lecture 6. First-order Functions

Shin Hong

Function

- a function is a sequence of operations taking inputs and producing outputs, working as a high-level operation
  - a fundamental aspect of program abstraction

- first-order functions cannot be treated as data (values), so that these cannot be passed as an input or output to another function
  - while first-class function can do both

F1VAE

- add first-order functions to VAE
- a program consists of a set of function definitions and an expression
  - a function must be defined at top-level, not inside an expression
  - closure is not possible with first-order function

Syntax

- `<program> ::= <func-list> <expr>`
- `<func-list> ::= <func> <func-list> | ϵ`
- `<func> ::= ‘fun’ <id> ‘(‘ <id> ‘)’ ‘=’ <expr> ‘;’`
- `<expr> ::= … | <id> ‘(‘ <expr> ‘)’`

- currently, we restrict a function to have only one parameter for simplicity

Example

```
“fun twice(x)=(x+x) ;
 fun inc(x)=(x+1) ;
 let x=1 in (twice(inc(x))”
```

Semantics (1/3)

- function environment
  - a map from identifiers to function definitions
  - $FEnv = Id \nrightarrow FunDef$
  - $\Lambda \in FEnv$
- adding function environment to semantics function
  - $\Rightarrow \subseteq Env \times FEnv \times E \times \mathbb{Z}$
  - $(\sigma, \Lambda, e, n) \in \Rightarrow$
  - $\sigma, \Lambda \vdash e \Rightarrow n$

Semantics (2/3)

$$\sigma, \Lambda \vdash \ n \ \Rightarrow n \quad \text{[Num]}$$

$$\frac{x \in Domain(\sigma)}{\sigma, \Lambda \vdash \ x \Rightarrow \sigma(x)} \ \text{[Id]}$$

$$\frac{\sigma, \Lambda \vdash \ e_1 \ \Rightarrow n_1 \qquad \sigma, \Lambda \vdash e_2 \Rightarrow n_2}{\sigma, \Lambda \vdash \ (e_1 + e_2) \ \Rightarrow \ n_1 + n_2} \ \text{[Add]}$$

$$\frac{\sigma, \Lambda \vdash \ e_1 \ \Rightarrow n_1 \qquad \sigma, \Lambda \vdash e_2 \Rightarrow n_2}{\sigma, \Lambda \vdash \ (e_1 - e_2) \ \Rightarrow \ n_1 - n_2} \ \text{[Sub]}$$

$$\frac{\sigma, \Lambda \vdash \ e_1 \ \Rightarrow n_1 \qquad \sigma[x \mapsto n_1], \Lambda \vdash e_2 \Rightarrow n_2}{\sigma, \Lambda \vdash \ \text{let } x = e_1 \text{ in } e_2 \ \Rightarrow \ n_2} \ \text{[Let]}$$

Semantics (3/3)

$$\frac{}{\sigma, \Lambda \vdash \quad x(e) \quad \Rightarrow \quad n} \text{[Call]}$$

Semantics (3/3)

$$\frac{\sigma, \Lambda \vdash \ e \ \Rightarrow n' \quad x \in Domain(\Lambda) \quad \Lambda(x) = \text{fun } x(x') = e' \quad \emptyset\,[x' \mapsto n'], \Lambda \vdash e' \Rightarrow \ n}{\sigma, \Lambda \vdash \quad x(e) \quad \Rightarrow \quad n} \text{ [Call]}$$

- $x(e)$ evaluates to n under σ and Λ if
  - $x \in Domain(\Lambda)$ and let's say that $\Lambda(x)$ is $\text{fun } x(x') = e'$, and
  - $\sigma, \Lambda \vdash e \Rightarrow n'$, and
  - $\emptyset[x' \mapsto n'], \Lambda \vdash e' \Rightarrow n$ // the function is evaluated regardless of σ (static scoping)

Exercise

```
fun twice(x) = (x + x) ;
fun x(y) = y ;
fun f(x) = (x + 1) ;
fun g(g) = g ;
```

show the result of evaluating each of the following F1VAE expressions under the empty environment and the given function definitions, and explain why:

- `twice(twice)`
- `let x=5 in x(x)`
- `g(3)`
- `g(f)`
- `g(g)`