Principles of Programming Languages

Lecture 03 First-Class Functions and Closures

Albert-Ludwigs-Universität Freiburg

Peter Thiemann

University of Freiburg, Germany

thiemann@informatik.uni-freiburg.de

30 Apr 2018

UNI FREIBURG

Plan

First-Class Functions and Closures

- Core feature of **functional** programming languages
- Meanwhile adopted by many mainstream languages
- Essential component of reactive and callback-style programming
- Functional means
  - functions are values like any other value
  - functions can be passed as parameters, returned from functions, and stored in datastructures

Plan

Syntax of FUN

Everything is an expression

Expressions: $e \in \mathsf{Expr}$

$$
\begin{array}{lcll}
e & ::= & x \mid e{+}e \mid \ldots & \\
 & \mid & \texttt{function}\ (\overline{x})e & \text{function, aka lambda expression} \\
 & \mid & e(\overline{e}) & \text{application}
\end{array}
$$

Shorthands (aka **Syntactic Sugar**)

$$
\begin{aligned}
\texttt{let}\ x = e_1\ \texttt{in}\ e_2 &\equiv (\texttt{function}\ (x)e_2)(e_1) \\
e_1; e_2 &\equiv \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \\
&\quad \text{where } x \notin e_2
\end{aligned}
$$

UNI FREIBURG

Closures: Modeling function values

$$
\begin{array}{rcll}
\sigma \ni & \mathsf{Env} & = & \mathsf{Var} \hookrightarrow \mathsf{Val} & \text{environments} \\
\langle \sigma, \overline{x}, e \rangle \ni & \mathsf{Closure} & = & \mathsf{Env} \times \mathsf{Var}^* \times \mathsf{Expr} & \\
y \ni & \mathsf{Val} & = & \mathbf{Z} \uplus \mathsf{Closure} & \text{values}
\end{array}
$$

- A closure $\langle \sigma, \overline{x}, e \rangle$ represents a function
  - $f = \texttt{function } (\overline{x}) e$
  - defined in environment σ
- The environment only contains the **free variables** of f

Free and bound variables

- The expression `function` $(\overline{x})e$ **binds** variables $\overline{x}$ in scope e, the body of the function.
- Variable x **occurs free** in expression e if it is used in e without an enclosing binding function expression.

Formally: $fv(e)$ set of free variables of e

$$
\begin{aligned}
fv(x) &= \{x\} \\
fv(e_1 + e_2) &= fv(e_1) \cup fv(e_2) \\
fv(\texttt{function}\ (\overline{x})e) &= fv(e) \setminus \{\overline{x}\} \\
fv(e(\overline{e})) &= fv(e) \cup fv(\overline{e})
\end{aligned}
$$

Plan

Big-Step Semantics

As before . . .

Evaluation of expressions

- input: current state and expression
- output: value of expression
- need relation $\sigma, e \hookrightarrow y$

Big-Step Evaluation of Expressions

FFun

$$\sigma, \texttt{function}\ (\overline{x})e \hookrightarrow \langle \sigma, \overline{x}, e \rangle$$

FApp

$$\frac{\sigma, e \hookrightarrow \langle \sigma', \overline{x}, e' \rangle \qquad \sigma, \overline{e} \hookrightarrow \overline{y} \qquad \sigma'[\overline{x \mapsto y}], e' \hookrightarrow y'}{\sigma, e(\overline{e}) \hookrightarrow y'}$$

Function call

- executes in closure's declaration environment σ'
- extended with bindings of the parameters

Plan

References

- References: objects with one field
- Also first-class values

Syntax of FUN-Ref

Expressions: $e \in \text{Expr}$

$$
\begin{array}{lcll}
e & ::= & \ldots & \text{(as before)} \\
 & | & \texttt{new}\ e & \text{new reference} \\
 & | & \texttt{!}e & \text{dereference} \\
 & | & e\texttt{:=}e & \text{reference assignment}
\end{array}
$$

Semantic Domains for FUN-Ref

$$
\begin{array}{rlcll}
a \ni & \mathsf{Addr} & & & \\
\mu \ni & \mathsf{Memory} & = & \mathsf{Addr} \hookrightarrow \mathsf{Val} & \text{memories} \\
\sigma \ni & \mathsf{Env} & = & \mathsf{Var} \hookrightarrow \mathsf{Addr} & \text{environments} \\
\langle \sigma, \overline{x}, e \rangle \ni & \mathsf{Closure} & = & \mathsf{Env} \times \mathsf{Var}^* \times \mathsf{Expr} & \\
y \ni & \mathsf{Val} & = & \mathbf{Z} \uplus \mathsf{Closure} \uplus \mathsf{Addr} & \text{values}
\end{array}
$$

Big-Step Semantics

Evaluation of expressions

- (machine) state: current memory and environment
- evaluation relation / judgment: $\mu, \sigma, e \hookrightarrow \mu', y$

Big-Step Semantics

Evaluation rules for references

FNew

$$\frac{\mu, \sigma, e \hookrightarrow \mu', y \qquad a \notin dom(\mu') \qquad \mu'' = \mu'[a \mapsto y]}{\mu, \sigma, \texttt{new}\ e \hookrightarrow \mu'', a}$$

FDeref

$$\frac{\mu, \sigma, e \hookrightarrow \mu', a \qquad y = \mu'(a)}{\mu, \sigma, \texttt{!}\ e \hookrightarrow \mu', y}$$

FAssign

$$\frac{\mu, \sigma, e_1 \hookrightarrow \mu', a \qquad \mu', \sigma, e_2 \hookrightarrow \mu'', y \qquad \mu''' = \mu''[a \mapsto y]}{\mu, \sigma, e_1\ \texttt{:=}\ e_2 \hookrightarrow \mu''', y}$$

Big-Step Semantics

Evaluation rules for functions

FVar
$$\frac{\sigma(x) = a \qquad \mu(a) = y}{\mu, \sigma, x \hookrightarrow \mu, y}$$

FFun
$$\mu, \sigma, \texttt{function}\ (\overline{x})e \hookrightarrow \mu, \langle \sigma, \overline{x}, e \rangle$$

FApp
$$\frac{\begin{array}{c}\mu, \sigma, e \hookrightarrow \mu', \langle \sigma', \overline{x}, e' \rangle \qquad \mu', \sigma, \overline{e} \hookrightarrow \mu'', \overline{y} \\ \overline{a} \cap dom(\mu'') = \emptyset \qquad \mu''[\overline{a \mapsto y}], \sigma'[\overline{x \mapsto a}], e' \hookrightarrow \mu''', y'\end{array}}{\mu, \sigma, e(\overline{e}) \hookrightarrow \mu''', y'}$$

Related to JavaScript

```
var x = {ref=42};         // let x = new(42) in
var f = function () {     // let f = function ()
  return x.ref }          //   (!x) in
f(); // returns 42        // f();
x.ref = 21;               // x := 21;
f(); // returns 21        // f()
```

Recursive Functions

- JavaScript function expressions can be recursive!

```
function fact(n) {
  if (n==0) { return 1; }
  else { return n * fact (n-1); }
}
```

- Implemented with a **recursive closure** $\langle \sigma, f, \overline{x}, e \rangle$ with the intention that variable f is set up to refer to its own closure

Recursive Lambda

FFunRec

$$\mu, \sigma, \texttt{function}\ f(\overline{x})e \hookrightarrow \mu, \langle \sigma, f, \overline{x}, e \rangle$$

FAppRec

$$\frac{\begin{array}{c} \mu, \sigma, e \hookrightarrow \mu', y \\ y = \langle \sigma', f, \overline{x}, e' \rangle \qquad \mu', \sigma, \overline{e} \hookrightarrow \mu'', \overline{y} \qquad a, \overline{a} \cap dom(\mu'') = \emptyset \\ \mu''[a \mapsto y, \overline{a \mapsto y}], \sigma'[f \mapsto a, \overline{x \mapsto a}], e' \hookrightarrow \mu''', y' \end{array}}{\mu, \sigma, e(\overline{e}) \hookrightarrow \mu''', y'}$$

Plan

Objects

Syntax of FUN-Obj

Expressions: $e \in \text{Expr}$

e	$::=$	$\ldots$	(as before)
	$\mid$	$\texttt{new}\ (\overline{l = e})$	new object
	$\mid$	$e.l$	property access
	$\mid$	$e.l{:=}e$	property assignment
l	$\in$	Label	property names (labels)

Semantic Domains for FUN-Obj

$$
\begin{array}{rlcll}
a \ni & \mathsf{Addr} & & & \\
\mu \ni & \mathsf{Memory} & = & \mathsf{Addr} \hookrightarrow \mathsf{Val} & \text{memories} \\
\sigma \ni & \mathsf{Env} & = & \mathsf{Var} \hookrightarrow \mathsf{Addr} & \text{environments} \\
\langle \sigma, \overline{x}, e \rangle \ni & \mathsf{Closure} & = & \mathsf{Env} \times \mathsf{Var}^* \times \mathsf{Expr} & \\
\{\overline{l \mapsto a}\} \ni & \mathsf{Object} & = & \mathsf{Label} \hookrightarrow \mathsf{Addr} & \\
y \ni & \mathsf{Val} & = & \mathbf{Z} \uplus \mathsf{Closure} \uplus \mathsf{Object} & \text{values}
\end{array}
$$

Big-Step Semantics

Evaluation rules for objects

FNewObj

$$\frac{\mu, \sigma, \overline{e} \hookrightarrow \mu', \overline{y} \qquad \overline{a} \cap dom(\mu) = \emptyset \qquad \mu'' = \mu'[\overline{a \mapsto y}]}{\mu, \sigma, \texttt{new}\ (\overline{l = e}) \hookrightarrow \mu'', \{\overline{l \mapsto a}\}}$$

FDerefObj

$$\frac{\mu, \sigma, e \hookrightarrow \mu', \{\overline{l \mapsto a_l}\} \qquad y = \mu'(a_l)}{\mu, \sigma, e.l \hookrightarrow \mu', y}$$

FAssignObj

$$\frac{\begin{array}{c}\mu, \sigma, e_1 \hookrightarrow \mu', \{\overline{l \mapsto a_l}\} \\ \mu', \sigma, e_2 \hookrightarrow \mu'', y \qquad \mu''' = \mu''[a_l \mapsto y]\end{array}}{\mu, \sigma, e_1.l\ \ :=\ \ e_2 \hookrightarrow \mu''', y}$$

Plan

Call-by-name

- Parameters are passed **unevaluated**
- Parameter evaluation happens when a parameter is accessed
- Problem: parameters may be evaluated multiple times
- Solution: lazy evaluation (see below)
- Problem: parameters must be evaluated **in the environment of the call site**
- Solution: represent a call-by-name parameter by a (parameterless) closure $\langle \sigma, e \rangle$
- Consequence: variables are bound to such closures

Call-by-name semantics

FApp-Name

$$\frac{\sigma, e \hookrightarrow \langle \sigma', \overline{x}, e' \rangle \qquad \sigma'[\overline{x \mapsto \langle \sigma, e \rangle}], e' \hookrightarrow y'}{\sigma, e(\overline{e}) \hookrightarrow y'}$$

FVar-Name

$$\frac{\sigma(x) = \langle \sigma', e' \rangle \qquad \sigma', e' \hookrightarrow y}{\sigma, x \hookrightarrow y}$$

- Application only evaluates the function part
- Problem: parameters are evaluated as often as they are used
- Solution: Lazy evaluation — required memory

Call-by-need / lazy semantics

FApp-Need

$$\frac{\overline{a} \cap dom(\mu') = \emptyset \qquad \begin{array}{c} \mu, \sigma, e \hookrightarrow \mu', \langle \sigma', \overline{x}, e' \rangle \\ \mu'[\overline{a \mapsto \langle \sigma, e \rangle}], \sigma'[\overline{x \mapsto a}], e' \hookrightarrow \mu'', y' \end{array}}{\mu, \sigma, e(\overline{e}) \hookrightarrow \mu'', y'}$$

FVar-Need

$$\frac{\begin{array}{cc} \sigma(x) = a & \mu(a) = \langle \sigma', e' \rangle \\ \mu, \sigma', e' \hookrightarrow \mu', y & \mu'' = \mu'[a \mapsto y] \end{array}}{\mu, \sigma, x \hookrightarrow \mu'', y}$$

FVar-Need-Value

$$\frac{\begin{array}{c} \sigma(x) = a \\ \mu(a) = y \neq \langle \sigma', e' \rangle \end{array}}{\mu, \sigma, x \hookrightarrow \mu, y}$$

- Laziness: evaluate at most once
- First evaluation **updates** the closure with the value
- Optimization: FVar-Name if variable is used at most once