6.035: Semantics

Closures

What is a Closure?

Python

```
f = lambda x, y : x + y
```

C++

```
auto f = [](int x, int y) {
   return x + y;
};
```

Java

```
BinaryOperator<int> f =
(int x, int y) -> {
   return x + y;
};
```

Javascript

```
var f = function(x, y) {
   return x + y;
}
```

```
var f = (x, y) => {
   return x + y;
}
```

What is a Closure?

```
var f = fun(x) {
   print(x);
};
f(1)
```

Output: 1

```
var x = 1;
var f = fun(y) {
   print(x);
};
f(2);
```

Output: 1

```
var x = 1;
var f = fun() {
   x = 2
};
var g = fun(x) {
   x();
}
g(f);
print(x);
```

Outputs: 2

- A function and a scope (in which to run the function)
- First-class: a closure is a *value*
- *Higher-Order*: functions may take *closures* as arguments

Why?

Challenge: Managing Scopes

```
var f = 0;
{
  var x = 1;
  f = fun(y) {
    print(x);
    x = x + 1;
  };
}
f();
f();
f();
```

Output:
1
2
3

A closure may refer to value of variable in from a context that has been popped from the stack

Scope's frame may live longer than the the single execution of the scope's code

Strategy: closure contains a pointer to the frame in which it was created

Extending IMP with Closures

- Syntax: create and call closures
- Representation: closures are values and frames a long-lived
- Semantics: creating and calling closures

Extending IMP with Closures

- Syntax: create and call closures
- Representation: closures are values and frames a long-lived
- Semantics: creating and calling closures

Add Closures: Grammar

$E \rightarrow n \mid x \mid -E \quad \mid True \mid False$
$\quad \mid E + E \mid E - E \quad \mid E < E \mid E == E$
$\quad \mid E * E \mid E / E \quad \mid !E$
$\quad \mid E \,\&\&\, E \mid E \mid\mid E$

$S \rightarrow$ `var` x = E | x = E
$\quad \mid$ `var` x = `fun`$(x)\{\ S*\ \}$ | x =`fun`$(x)\{\ S*\ \}$ (create closure)
$\quad \mid x(E)$ (call closure)
$\quad \mid$ `if` (E) BS `else` BS
$\quad \mid BS$
$\quad \mid$ `while` (E) BS

$BS \rightarrow \{\ S^*\}$

Extending IMP with Closures

- Syntax: create and call closures
- Representation: closures are values and frames a long-lived
- Semantics: creating and calling closures

Linked Stacks (Boardwork)

```
var x = 1;
{
  var x = 2;
  {
      x = 3;
  }
}
```

Frames, Heaps, and Linked Stacks

- Domain of addresses: A
  - Location of an integer value in memory
- Domain of parent fields: $P = \{ \rho \}$
  - Field to access parent of frame
- Domain of frames: $\sigma \in \Sigma = (X \cup P) \rightarrow A$
  - A *frame* σ is an element of the domain Σ, which is set of all (partial) functions that map a variable x from the domain of all variables X to an address
- Domain of closures: $f \in F = A \times X \times S^*$
  - A closure is a tuple consisting of an frame address, variable, and sequence of statements
- Domain of heaps: $h \in H = A \rightarrow V$ where $V = \mathbb{N} \cup \mathbb{B} \cup \Sigma \cup F$
  - A *heap* h is an element of the domain H, which is set of all (partial) functions that map an address to a value (integer or boolean)

Evaluation Relations (with heaps and stacks)

- Define the semantics of each term in our language (*E*, *B*, and *S*) with an *evaluation relation*:

$$(\gamma, \boldsymbol{h}, \boldsymbol{e}) \rightarrow \boldsymbol{v} \qquad (\gamma, \boldsymbol{h}, \boldsymbol{s}) \rightarrow (\boldsymbol{\gamma}, \boldsymbol{h}) \qquad (\gamma, \boldsymbol{h}, \boldsymbol{bs}) \rightarrow (\boldsymbol{\gamma}, \boldsymbol{h})$$

- Meaning: given a ***stack***, and ***heap***, the ***term*** evaluates to a **result**

Evaluation Relations (with linked stacks)

- Define the semantics of each term in our language (*E*, *B*, and *S*) with an *evaluation relation*:

$$(a, \boldsymbol{h}, \boldsymbol{e}) \rightarrow \boldsymbol{v} \qquad (a, \boldsymbol{h}, \boldsymbol{s}) \rightarrow \boldsymbol{h} \qquad (a, \boldsymbol{h}, \boldsymbol{bs}) \rightarrow \boldsymbol{h}$$

- Meaning: given a ***frame pointer***, and ***heap***, the ***term*** evaluates to a **result**

Expressions: Inference Rules

Integer Constant

$$\frac{n = n_r}{(\gamma, h, n) \rightarrow n_r}$$

$$\frac{n = n_r}{(a, h, n) \rightarrow n_r}$$

Variable Reference

$$\frac{lookup(\gamma, x) = \sigma \quad \sigma(x) = a \quad h(a) = v_r}{(\gamma, h, x) \rightarrow v_r}$$

$$\frac{lookup(a_f, x) = \sigma \quad \sigma(x) = a_x \quad h(a_x) = v_r}{(a_f, h, x) \rightarrow v_r}$$

Unary Minus

$$\frac{(\gamma, h, e) \rightarrow v \quad type(v) = int \quad -int(v) = n_r}{(\gamma, h, -e) \rightarrow n_r}$$

Lookup

- $lookup(a, x) = \sigma$: lookup variable binding by traversing the stack

$$\frac{\neg(a = 0) \qquad h(a) = \sigma \qquad x \in dom(\sigma)}{lookup(a, x) = \sigma}$$

$$\frac{\neg(a = 0) \qquad h(a) = \sigma \qquad \neg\big(x \in dom(\sigma)\big) \qquad lookup(\sigma(\rho), x) = \sigma'}{lookup(a, x) = \sigma'}$$

Expressions: Inference Rules

Integer Constant

$$\frac{n = n_r}{(\gamma, h, n) \rightarrow n_r} \longrightarrow \frac{n = n_r}{(a, h, n) \rightarrow n_r}$$

Variable Reference

$$\frac{lookup(\gamma, x) = \sigma \quad \sigma(x) = a \quad h(a) = v_r}{(\gamma, h, x) \rightarrow v_r}$$

$$\frac{lookup(a_f, x) = \sigma \quad \sigma(x) = a_x \quad h(a_x) = v_r}{(a_f, h, x) \rightarrow v_r}$$

Unary Minus

$$\frac{(\gamma, h, e) \rightarrow v \quad type(v) = int \quad -int(v) = n_r}{(\gamma, h, -e) \rightarrow n_r}$$

$$\frac{(a, h, e) \rightarrow v \quad type(v) = int \quad -int(v) = n_r}{(a, h, -e) \rightarrow n_r}$$

Inference Rules: Declaration

var x = E

$$\frac{h(a_f) = \sigma \qquad \neg(x \in dom(\sigma)) \qquad (a, h, e) \to v \qquad \neg(a_x \in dom(h)) \qquad \sigma[x : a_x] = \sigma' \qquad h[a_f : \sigma', a_x : v] = h'}{(a_f, h, \mathsf{var}\ x = e) \to h'}$$

Inference Rules: Assignment

$x = E$

$$\frac{(a, h, e) \rightarrow v \qquad update(a,\ h,\ x,\ v, h')}{(a,\ h,\ x = e) \rightarrow h'}$$

Update

- $update(a,\ h,\ x,\ v,\ h') =$

$$\frac{h(a_f) = \sigma \qquad x \in dom(\sigma) \quad \neg(a_x \in dom(h)) \qquad \sigma[x : a_x] = \sigma' \qquad h[a_f : \sigma',\ a : v] = h'}{update(a_f,\ h,\ x,\ v,\ h')}$$

$$\frac{h(a_f) = \sigma \qquad \neg(x \in dom(\sigma)) \quad update(\sigma(\rho),\ h,\ x,\ v,\ h')}{update(a_f,\ h,\ x,\ v,\ h')}$$

- Recursively search the stack for x, and update x, return new heap

Inference Rules: Block Scope

Block Scope $\{ S^* \}$:

$$\frac{\sigma = \{ \rho : a_f \} \qquad \neg(a_\sigma \in dom(h)) \qquad h[\, a_\sigma \ : \ \sigma] = h' \quad (a_{\sigma,} h', S^*) \rightarrow h''}{(a_f, h, bs\ S^*) \rightarrow h''}$$

Sequential Composition S^*:

$$\frac{}{(a, h, []) \rightarrow h} \qquad \frac{(a, h, S) \rightarrow h' \quad (a, h', S_r) \rightarrow h''}{(a, S :: S_r) \rightarrow h''}$$

Extending IMP with Closures

- Syntax: create and call closures
- Representation: closures are values and frames a long-lived
- Semantics: creating and calling closures

Inference Rules: Closure Creation (Declaration)

Declaration `var` x = `fun(`x`){` $S*$ `}`:

$$\frac{h(a_f) = \sigma \qquad \neg(x \in dom(\sigma)) \qquad (a_f, x_p, S^*) = f \qquad \neg(a_x \in dom(h)) \qquad \sigma[x : a_x] = \sigma' \qquad h[a_f : \sigma', a_x : f] = h'}{(a_f, h, \texttt{var}\ x = \texttt{fun}(x_p)\{\ S*\ \}\ ) \rightarrow h'}$$

- Executing a closure: capture frame pointer and store closure in heap

Managing Scope (Revisited)

```
1:  var f = 0;
2:  {
3:    var x = 1;
4:    f = fun(y) {
5:      print(x);
6:      x = x + 1;
7:    };
8:  }
9:  f();
10: f();
```

Line	FP (a)	Heap (h)
1	200	{ 100 : 0, 200: {p : 0, f : 100}}
3	208	{ 100 : 0, 108 : 1, 200 : {p : 0, f : 100}, 208 : {p : 200, x : 108} }
7	208	{ 100 : 0, 108 : 1, 116 : (208, y, {print(x); x = x + 1}), 200 : {p : 0, f : 116}, 208 : {p : 200, x : 108} }
8	200	{ 100 : 0, 108 : 1, 116 : (208, y, {print(x); x = x + 1}), 200 : {p : 0, f : 116}, 208 : {p : 200, x : 108} }
9	200	{ 100 : 0, 108 : 1, 116 : (208, y, {print(x); x = x + 1}), 124: 2, 200 : {p : 0, f : 116}, 208 : {p : 200, x : 124} }
10	200	{ 100 : 0, 108 : 1, 116 : (208, y, {print(x); x = x + 1}), 124: 2, 132:3 200 : {p : 0, f : 116}, 208 : {p : 208, x : 132} }

Inference Rules: Closure Execution

Closure Call: $x(e)$

$(a_f, h, x) \rightarrow (a_c, x_p, S^*)$ x Evaluates to a closure

$(a_f, h, e) \rightarrow v$ E evaluates to a value

$\neg(a_\sigma \in dom(h))$ $\sigma = \{\rho : a_c, x_p : v\}$ $h' = h[a_\sigma : \sigma]$ Create a new frame

$(a_\sigma, h', S^*) \rightarrow h''$ Evaluate body of closure

---

$$(a_f, h, x(e)) \rightarrow h''$$

The End