State

CS242

Lecture 10

Recall: The Lambda Calculus

Grammar:

$e \rightarrow x \mid \lambda x.e \mid e\ e$

Beta reduction:

$(\lambda x.e_1)\ e_2 \rightarrow e_1\,[x := e_2]$

(Keep in mind the two arrows on the two lines mean different things!)

The Ugly Bit: Substitution

$x\ [x := e] = e$
$y\ [x := e] = y$
$(e_1\ e_2)\ [x := e] = (e_1\ [x := e])\ (e_2\ [x := e])$
$(\lambda x.e_1)\ [x := e] = \lambda x.e_1$
$(\lambda y.e_1)\ [x := e] = \lambda y.(e_1\ [x := e])$ if $x \neq y$ and $y \notin FV(e)$

$FV(x) = \{ x \}$
$FV(e_1\ e_2) = FV(e_1) \cup FV(e_2)$
$FV(\lambda x.e) = FV(e) - \{ x \}$

Is there a better way of explaining this stuff?

Recall: Type Rules

$$\frac{}{A,\ x{:}\ t \vdash x : t}\ [\text{Var}] \qquad \frac{A,\ x{:}\ t \vdash e : t'}{A \vdash \lambda x{:}t.e : t \to t'}\ [\text{Abs}]$$

$$\frac{}{A \vdash i : \text{int}}\ [\text{Int}] \qquad \frac{A \vdash e_1 : t \to t' \qquad A \vdash e_2 : t}{A \vdash e_1\, e_2 : t'}\ [\text{App}]$$

Structural Operational Semantics

- Logical systems have long been used to present type rules

- Why not do the same thing for evaluation rules?

- SOS were introduced by Gordon Plotkin
  - And are now the most popular method for describing language semantics

Judgments

Type judgments

$A \vdash e : t$

Under type assumptions A for the free variables of e, it is provable that e has type t.

Judgments

Value judgments

$E \vdash e \rightarrow v$

E is an *environment*: A mapping from variables to values.

In the environment E assigning values to the free variables of e, it is provable that e reduces to value v.

Comments

What are values?

A subset of the programs that can't be further reduced.

In the pure lambda calculus, lambda abstractions λx.e are the values.

If we add constants such as 1,2,..., true, false, +, *, ... these would also be values

- possible results of computations

Evaluation Rules

$$\frac{}{E \vdash x \rightarrow E(x)}\ [\text{Var}]$$

$$\frac{}{E \vdash \lambda x.e \rightarrow \lambda x.e}\ [\text{Abs}]$$

$$\frac{}{E \vdash i \rightarrow i}\ [\text{Int}]$$

$$\frac{E \vdash e_1 \rightarrow \lambda x.e_0 \qquad E \vdash e_2 \rightarrow v \qquad E[x: v] \vdash e_0 \rightarrow v'}{E \vdash e_1\, e_2 \rightarrow v'}\ [\text{App}]$$

Note: E[x: v] is the same environment as E, x:v. E is extended (or updated if x is already present) at point x to return v.

Example

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \qquad \dfrac{}{x: \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to \lambda y.y} \qquad \dfrac{}{\vdash 1 \to 1} \qquad \dfrac{}{y: 1 \vdash y \to 1}}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to 1}$$

Example

$\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \rightarrow$

Example

$$\frac{\vdash (\lambda x.x)\ (\lambda y.y) \rightarrow \qquad\qquad \vdash 1 \rightarrow \qquad\qquad \vdash}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \rightarrow}$$

Example

$$\dfrac{\dfrac{\vdash \lambda x.x \rightarrow \qquad \vdash \lambda y.y \rightarrow \qquad \vdash}{\vdash (\lambda x.x)\ (\lambda y.y) \rightarrow} \qquad \vdash 1 \rightarrow \qquad \vdash}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \rightarrow}$$

Example

$$\cfrac{\cfrac{\cfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \vdash \lambda y.y \to \qquad \vdash}{\vdash (\lambda x.x)\ (\lambda y.y) \to} \qquad \vdash 1 \to \qquad \vdash}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to}$$

Example

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \qquad \vdash}{\vdash (\lambda x.x)\ (\lambda y.y) \to} \qquad \vdash 1 \to \qquad \vdash}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to}$$

Example

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \rightarrow \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \rightarrow \lambda y.y} \qquad x: \lambda y.y \vdash x \rightarrow}{\vdash (\lambda x.x)\ (\lambda y.y) \rightarrow} \qquad \vdash 1 \rightarrow \qquad \vdash}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \rightarrow}$$

Example

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \qquad \dfrac{}{x: \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to} \qquad \vdash 1 \to \qquad \vdash}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to}$$

Example

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \qquad \dfrac{}{x : \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to \lambda y.y} \qquad \vdash 1 \to \qquad \vdash}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to}$$

Example

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \qquad \dfrac{}{x: \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to \lambda y.y} \qquad \dfrac{}{\vdash 1 \to 1} \qquad \vdash}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to}$$

Example

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \quad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \quad \dfrac{}{x: \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to \lambda y.y} \qquad \dfrac{}{\vdash 1 \to 1} \qquad y: 1 \vdash y \to}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to}$$

Example

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \qquad \dfrac{}{x: \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to \lambda y.y} \qquad \dfrac{}{\vdash 1 \to 1} \qquad \dfrac{}{y: 1 \vdash y \to 1}}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to}$$

Example

$$\frac{\dfrac{\dfrac{}{\vdash \lambda x.x \rightarrow \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \rightarrow \lambda y.y} \qquad \dfrac{}{x: \lambda y.y \vdash x \rightarrow \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \rightarrow \lambda y.y} \qquad \dfrac{}{\vdash 1 \rightarrow 1} \qquad \dfrac{}{y: 1 \vdash y \rightarrow 1}}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \rightarrow 1}$$

Discussion

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \qquad \dfrac{}{x: \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to \lambda y.y} \qquad \dfrac{}{\vdash 1 \to 1} \qquad \dfrac{}{y: 1 \vdash y \to 1}}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to 1}$$

The tree structure of the derivation reflects the structure of the computation, not the structure of the expression. Interior nodes are function calls (applications), leaves are constants and abstractions that evaluate to themselves or variables that evaluate to their binding in the environment.

Discussion

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \qquad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \qquad \dfrac{}{x: \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to \lambda y.y} \qquad \dfrac{}{\vdash 1 \to 1} \qquad \dfrac{}{y: 1 \vdash y \to 1}}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to 1}$$

Note how substitution is built in to the rules: The binding of a variable is passed down through the tree from the point of definition (at the function application) to the point where the variable is used via the environment.

Discussion

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.x \to \lambda x.x} \quad \dfrac{}{\vdash \lambda y.y \to \lambda y.y} \quad \dfrac{}{x: \lambda y.y \vdash x \to \lambda y.y}}{\vdash (\lambda x.x)\ (\lambda y.y) \to \lambda y.y} \qquad \dfrac{}{\vdash 1 \to 1} \qquad \dfrac{}{y: 1 \vdash y \to 1}}{\vdash ((\lambda x.x)\ (\lambda y.y))\ 1 \to 1}$$

The evaluation rules also capture aspects of implementations. The environment, for example, represents the call stack.

Another Example

$$\dfrac{\dfrac{}{\vdash \lambda x.x\ x \to \lambda x.x\ x} \qquad \dfrac{}{\vdash \lambda x.x\ x \to \lambda x.x\ x} \qquad \dfrac{\dfrac{}{x: \lambda x.x\ x \vdash x \to \lambda x.x\ x} \quad \dfrac{}{x: \lambda x.x\ x \vdash x \to \lambda x.x\ x} \quad x: \lambda x.x\ x \vdash x\ \ x \to\ ?}{x: \lambda x.x\ x \vdash x\ x}}{\vdash (\lambda x.x\ x)\ (\lambda x.x\ x) \to}$$

Proofs are finite objects and as a result infinite computations have no derivations in this semantics! There are more elaborate versions of SOS that address this issue.

But isn’t this lecture supposed to be about state?

The Lambda Calculus with State

$e \rightarrow x \mid \lambda x.e \mid e\ e \mid i \mid new \mid !e \mid e := e$

new	allocate a new memory location *x* and return a pointer to *x* (initialize *x* to 0)
!e	dereference a pointer
$e_1 := e_2$	assign to a pointer, value is e_2

Example

$e \rightarrow x \ | \ \lambda x.e \ | \ e\ e \ | \ i \ | \ \text{new} \ | \ !e \ | \ e := e$

$!((\lambda y.K\ y\ (y := 1))\ \text{new}) \rightarrow 1$

Rules?

$$\frac{}{E \vdash \text{new} \rightarrow} \quad [\text{New}]$$

What is state?

Stores

A *store* is a mapping from memory locations to values.

$S = [\ l_1 = 1, l_2 = 42\ ]$

1 is stored at location l_1

42 is stored at location l_2

Note that locations are values.

New

$$\frac{l \notin dom(S)}{E, S \vdash \text{new} \rightarrow l, S[l = 0]} \quad \text{[New]}$$

Evaluating an expression takes place in an initial state and produces a value and new state.

Dereference

$$\frac{E, S_0 \vdash e \rightarrow l, S_1}{E, S_0 \vdash !e \rightarrow S_1(l), S_1}\ [\text{Ref}]$$

Assignment

$$\frac{E, S_0 \vdash e_1 \rightarrow l, S_1 \qquad E, S_1 \vdash e_2 \rightarrow v, S_2}{E, S_0 \vdash e_1 := e_2 \rightarrow v, S_2[l = v]} \quad \text{[Assign]}$$

Note how the presence of a store requires us to specify the order of evaluation so that store updates happen in a predictable order. Also note how the store is "threaded" through the computation.

Modified Evaluation Rules

$$\frac{}{E, S \vdash x \rightarrow E(x), S}\ [\text{Var}]$$

$$\frac{}{E, S \vdash \lambda x.e \rightarrow \lambda x.e, S}\ [\text{Abs}]$$

$$\frac{}{E, S \vdash i \rightarrow i, S}\ [\text{Int}]$$

$$\frac{E, S_0 \vdash e_1 \rightarrow \lambda x.e_0, S_1 \quad E, S_1 \vdash e_2 \rightarrow v, S_2 \quad E[x: v], S_2 \vdash e_0 \rightarrow v', S_3}{E, S_0 \vdash e_1\, e_2 \rightarrow v', S_3}\ [\text{App}]$$

Even expressions that don't use the store directly need to thread it through the evaluation in case subexpressions do use the store.

Example with State

$$\dfrac{\dfrac{}{[y{:}l_0],\ [\ l_0 = 0\ ] \vdash y \to l_0,\ [\ l_0 = 0\ ]} \qquad \dfrac{[y{:}l_0],\ [\ l_0 = 0\ ] \vdash y \to l_0,\ [\ l_0 = 0\ ] \qquad [y{:}l_0],\ [\ l_0 = 0\ ] \vdash 1 \to 1,\ [\ l_0 = 0\ ]}{[y{:}l_0],\ [\ l_0 = 0\ ] \vdash y := 1 \to 1,\ [\ l_0 = 1\ ]}}{[y{:}l_0],\ [\ l_0 = 0\ ] \vdash K\ y\ (y := 1) \to l_0,\ [\ l_0 = 1\ ]}$$

$$\dfrac{\dfrac{\dfrac{}{[],\ [] \vdash \lambda y.K\ y\ (y := 1) \to \lambda y.K\ y\ (y := 1),\ []} \qquad \dfrac{}{[],\ [] \vdash \text{new} \to l_0,\ [\ l_0 = 0\ ]}}{[],\ [] \vdash (\lambda y.K\ y\ (y := 1))\ \text{new} \to l_0,\ [l_0 = 1]}}{[],\ [] \vdash\ !((\lambda y.K\ y\ (y := 1))\ \text{new}) \to 1,\ [l_0 = 1]}$$

Note: Shortened evaluation K e e' to: evaluate e, evaluate e', return value of e.

Discussion

- The operational semantics provides a precise model of state.
- The model is also close to how most programming languages work.
- Observe
  - The state is like an extra argument passed to every expression evaluation.
  - The state is never copied: No evaluation step uses more than one state, and every state is modified at most once.
  - The state is unstructured – the whole state is passed to every step of evaluation.

Summary

- Explaining state requires a significant extension of simple functional semantics
  - Need a state object with allocation, reads, and writes
  - Behavior of programs is much different than pure functions

- Semantics of state shows the costs
  - Sequential order of evaluation must be defined
  - Uncontrolled aliasing
  - Unstructured state exposes too much