CSC 520, Spring 2020

Principles of Programming Languages

Michelle Strout

THE UNIVERSITY OF ARIZONA

Version: 1

January 15, 2020

Today's Plan

- **Impcore operational semantics**
- **How we know what code is supposed to do at runtime: valid derivations**
- **Wednesday, What do we know about valid derivations: metatheory**

For Reference: Concrete Syntax for Impcore

- **Definitions and Expressions**

```
def ::= (define f (x1 ... xn) exp)
      |   (val x exp)
      |   exp
      |   (use filename)
      |   (check-expect exp1 exp2)

exp ::= integer-literal     ;; atomic forms
      |   variable-name
      |  (set x exp)         ;; compound forms
      |  (if exp1 exp2 exp3)
      |  (while exp1 exp2)
      |  (begin exp1 ... expn)
      |  (function-name exp1 ... expn)
```

For reference: AST definition for Impcore

The University of Arizona
Computer Science

- The abstract-syntax tree (AST)

```
exp = LITERAL (Value)
    | VAR (Name)
    | SET (Name name, Exp exp)
    | IFX (Exp cond, Exp true, Exp false)
    | WHILEX (Exp cond, Exp exp)
    | BEGIN (Explist)
    | APPLY (Name name, Explist actuals)
```

- One kind of "application" for both user-defined and primitive functions.

For Reference: Impcore three environments

- **Global variables** ξ (or **\xi**)
- **Functions** φ (or **\phi**)
- **Formal parameters** ρ (or **\rho**)
- **There are no local variables**
  - Just like awk; if you need temps, use extra formal parameters
  - For HW2, you'll add local variables
- **Function environment** φ **not shared with variables – just like Perl**

Syntax and Environments determine behavior

- **Behavior is called evaluation**
  - Expression is evaluated in environment to produce value
  - "The environment" has three parts: globals, formals, functions
- **Evaluation is**
  - Specified using inference rules (math)
  - Implemented using interpreter (code)
- **You know code. You will learn math.**

Key ideas apply to any language

- **Expressions**
- **Values**
- **Rules**

Rules written using operational semantics

- **Evaluation on an abstract machine**
  - Concise, precise definition
  - Guide to build interpreter
  - Prove "evaluation deterministic" or "environments can be on a stack"
- **Idea: "mathematical interpreter" is set of formal rules for interpretation**

Syntax & environments determine meaning

- **Initial state of abstract machine:**

$$\langle e, \xi, \phi, \rho \rangle$$

- **State** $\langle e, \xi, \phi, \rho \rangle$ **is**
  - e — expression being evaluated
  - xi — values of global variables
  - phi — definitions of functions
  - rho — values of formal parameters

- **Three environments determine what is in scope**

Meaning written as "Evaluation judgement"

THE UNIVERSITY OF ARIZONA. Computer Science

- **We say**

$$\langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle$$

- **(Big-step judgement form)**

- **Notes:**
  - **xi and xi' may differ**
  - **rho and rho' may differ**
  - **phi must equal phi**

- **Question: what do we know about globals? functions?**

Impcore atomic form: Literal

- **"Literal" generalizes "numeral"**

$$\frac{\textsc{Literal}}{\langle \text{LITERAL}(v), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi, \phi, \rho \rangle}$$

- **Numeral converted to LITERAL(v) in parser**

- **Question: what is LITERAL(v)?**

Impcore atomic form: Variable

$$\frac{x \in \operatorname{dom} \rho}{\langle \mathrm{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \rho(x), \xi, \phi, \rho \rangle} \quad \textsc{FormalVar}$$

$$\frac{x \notin \operatorname{dom} \rho \qquad x \in \operatorname{dom} \xi}{\langle \mathrm{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \xi(x), \xi, \phi, \rho \rangle} \quad \textsc{GlobalVar}$$

- **Parameters hide global variables. Question: how do we know this?**

Impcore compound form: Assignment

- **In SET(x,e), e is any expression**

$$\textsc{FormalAssign}\quad \frac{x \in \operatorname{dom} \rho \qquad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \mathrm{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho'\{x \mapsto v\} \rangle}$$

$$\textsc{GlobalAssign}\quad \frac{x \notin \operatorname{dom} \rho \qquad x \in \operatorname{dom} \xi \qquad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \mathrm{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi'\{x \mapsto v\}, \phi, \rho' \rangle}$$

- **Impcore can assign only to existing variables, Question: how do we know that?**

Semantics corresponds to code

We compose rules to make proofs

Math	*Code*
Semantics	Interpreter
Evaluation judgment	Result of evaluation
Proof of judgment	Computation of result
Rule of semantics	Case in the interpreter

Interpreter succeeds if and only if a proof exists

(Homework: result is unique!)

Code: Evaluate by cases

One case per rule; multiple cases per form:

- `VAR` find binding for variable, use value
- `SET` rebind variable in `formals` or `globals`
- `IFX` (recursively) evaluate condition, then `t` or `f`
- `WHILEX` (recursively) evaluate condition, body
- `BEGIN` (recursively) evaluate each `Exp` of body
- `APPLY` look up function in `functions`
  - built-in `PRIMITIVE` — do by cases
  - `USERDEF` function — use arg values to build `formals` env, recursively evaluate fun body

Implementing evaluation

```
Value eval(Exp e, Valenv ξ, Funenv φ, Valenv ρ) {
  switch(e->alt) {
  case LITERAL: return e->u.literal;
  case VAR: ... /* look up in ρ and ξ */
  case SET: ... /* modify ρ or ξ */
  case IFX: ...
  case WHILEX: ...
  case BEGIN: ...
  case APPLY: ...
  }
}
```

More detail

THE UNIVERSITY OF ARIZONA
Computer Science

```
Value eval(Exp e, Valenv ξ, Funenv φ, Valenv ρ) {
  switch(e->alt) {
  case LITERAL: return e->u.literal;
  case VAR: ... /* look up in ρ and ξ */
  case SET: ... /* modify ρ or ξ */
  case IFX: ...
  case WHILEX: ...
  case BEGIN: ...
  case APPLY: if (!isfunbound(e->u.apply.name, φ))
                  runerror("undefined function %n",
                           e->u.apply.name);
              f = fetchfun(e->u.apply.name, φ);
              ... /* user fun or primitive */
  }
}
```

Variable-form math: two rules

FormalVar

$$\frac{x \in \operatorname{dom} \rho}{\langle \mathsf{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \rho(x), \xi, \phi, \rho \rangle}$$

GlobalVar

$$\frac{x \notin \operatorname{dom} \rho \qquad x \in \operatorname{dom} \xi}{\langle \mathsf{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \xi(x), \xi, \phi, \rho \rangle}$$

How do we tell them apart?

Variable-form code: three cases

THE UNIVERSITY OF ARIZONA
Computer Science

Consult `formals` ρ then `globals` ξ:

```
case VAR:
  if (isvalbound(e->u.var, formals))
    return fetchval(e->u.var, formals);
  else if (isvalbound(e->u.var, globals))
    return fetchval(e->u.var, globals);
  else
    runerror("unbound var %n", e->u.var);
```

Why a third case?

- When no proof, run-time error

Assignment-form math: two rules

$$\textsc{FormalAssign}\quad \frac{x \in \operatorname{dom} \rho \qquad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \mathsf{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho'\{x \mapsto v\} \rangle}$$

$$\textsc{GlobalAssign}\quad \frac{x \notin \operatorname{dom} \rho \qquad x \in \operatorname{dom} \xi \qquad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle}{\langle \mathsf{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi'\{x \mapsto v\}, \phi, \rho' \rangle}$$

Assignment-form code: three cases

```
case SET: {
  Value v = eval(e->u.set.exp, globals, functions,
                                       formals);

  if (isvalbound(e->u.set.name, formals))
    bindval(e->u.set.name, v, formals);
  else if (isvalbound(e->u.set.name, globals))
    bindval(e->u.set.name, v, globals);
  else
    runerror("set: unbound variable %n",
             e->u.set.name);
  return v;
}
```

Application math: user-defined function

ApplyUser

$$\frac{\begin{array}{c}\phi(f) = \mathsf{USER}(\langle x_1, \ldots, x_n \rangle, e) \\ x_1, \ldots, x_n \text{ all distinct} \\ \langle e_1, \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1 \rangle \\ \langle e_2, \xi_1, \phi, \rho_1 \rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2 \rangle \\ \vdots \\ \langle e_n, \xi_{n-1}, \phi, \rho_{n-1} \rangle \Downarrow \langle v_n, \xi_n, \phi, \rho_n \rangle \\ \langle e, \xi_n, \phi, \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\} \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle \end{array}}{\langle \mathsf{APPLY}(f, e_1, \ldots, e_n), \xi_0, \phi, \rho_0 \rangle \Downarrow \langle v, \xi', \phi, \rho_n \rangle}$$

Simpler math: function of two parameters

$$\textsc{ApplyUser}\quad \frac{\begin{array}{c}\phi(f) = \mathsf{USER}(\langle x_1, x_2\rangle, e) \\ x_1, x_2 \text{ distinct} \\ \langle e_1, \xi_0, \phi, \rho_0\rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1\rangle \\ \langle e_2, \xi_1, \phi, \rho_1\rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2\rangle \\ \langle e, \xi_2, \phi, \{x_1 \mapsto v_1, x_2 \mapsto v_2\}\rangle \Downarrow \langle v, \xi', \phi, \rho'\rangle\end{array}}{\langle \mathsf{APPLY}(f, e_1, e_2, \xi_0, \phi, \rho_0\rangle \Downarrow \langle v, \xi', \phi, \rho_2\rangle}$$

- **What order are actual parameters evaluated?**
- **What if formal param names are duplicated?**
- **What var changes in f can be seen by caller?**

Evaluating function application

The math demands these steps:

- Find function in old environment

```
f = fetchfun(e->u.apply.name, functions);
```

- Using old ρ, evaluate actuals

```
vs = evallist(e->u.apply.actuals,globals,
              functions, formals);
```

N.B. `actuals` evaluated in current environment

- Make a new environment: bind formals to actuals

```
new_formals = mkValenv(f.u.userdef.formals, vs);
```

- Evaluate body in new environment

```
return eval(f.u.userdef.body, globals, functions,
                                     new_formals);
```

Using Operation Semantics

- **Valid derivations**
  - "How do I know what this program should evaluate to?"
  - Code example

```
(define and (p q)
   (if p q 0))

(define digit? (n)
    (and (<= 0 n) (< n 10)))
```

- **Questions:**
  - In body of digit?, what expressions are evaluated in what order?
  - For the and function application, template is (f e1 e2). Matches?
  - Result of (digit? 7)?

Simpler math: function of two parameters

APPLYUSER

$$\frac{\begin{array}{c}\phi(f) = \mathsf{USER}(\langle x_1, x_2\rangle, e) \\ x_1, x_2 \text{ distinct} \\ \langle e_1, \xi_0, \phi, \rho_0\rangle \Downarrow \langle v_1, \xi_1, \phi, \rho_1\rangle \\ \langle e_2, \xi_1, \phi, \rho_1\rangle \Downarrow \langle v_2, \xi_2, \phi, \rho_2\rangle \\ \langle e, \xi_2, \phi, \{x_1 \mapsto v_1, x_2 \mapsto v_2\}\rangle \Downarrow \langle v, \xi', \phi, \rho'\rangle\end{array}}{\langle \mathsf{APPLY}(f, e_1, e_2, \xi_0, \phi, \rho_0\rangle \Downarrow \langle v, \xi', \phi, \rho_2\rangle}$$

Exercise: Which judgements are valid?

Which of these judgments *correctly* describes what code does at run time?

- $\langle$ (+ 2 2), $\xi, \phi, \rho\rangle \Downarrow \langle 99, \xi, \phi, \rho\rangle$
- $\langle$ (+ 2 2), $\xi, \phi, \rho\rangle \Downarrow \langle 0, \xi\{x \mapsto 10\}, \phi, \rho\rangle$
- $\langle$ (+ 2 2), $\xi, \phi, \rho\rangle \Downarrow \langle 4, \xi, \phi, \rho\rangle$
- $\langle$ (while 1 0), $\xi, \phi, \rho\rangle \Downarrow \langle 77, \xi, \phi, \rho\rangle$
- $\langle$ (begin (set n (+ n 1)) 17), $\xi, \phi, \rho\rangle \Downarrow \langle 17, \xi, \phi, \rho\rangle$

To know for sure, we *need a proof*.

Judgement is valid when "derivable"

- **Special kind of proof: derivation**
  - It's a data structure (derivation tree)
  - Made inductively, by composing rules
  - Valid derivation matches rules (by substitution)
  - Spacelike representation of timelike behavior (think flip-book animation)
- **A form of "syntactic proof"**

Recursive evaluation for inductive proof

THE UNIVERSITY OF ARIZONA
Computer Science

- Root of derivation at the bottom (surprise!)
- Build
  - Start on the left, go up
  - Cross the evaluation judgment relation down arrow
  - Finish on the right, go down
- First let's see a movie

Evaluating $(10+1) \times (10-1)$

$$\overline{\langle \texttt{(* (+ 10 1) (- 10 1))}, \xi, \phi, \rho \rangle \Downarrow}$$

Evaluating $(10+1) \times (10-1)$

$$\dfrac{\overline{\langle \texttt{(+ 10 1)}, \xi, \phi, \rho \rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))}, \xi, \phi, \rho \rangle \Downarrow}$$

Evaluating $(10+1) \times (10-1)$

$$\dfrac{\dfrac{\overline{\langle \texttt{10}, \ldots \rangle \Downarrow}}{\langle \texttt{(+ 10 1)}, \xi, \phi, \rho \rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))}, \xi, \phi, \rho \rangle \Downarrow}$$

Evaluating $(10+1) \times (10-1)$

$$\dfrac{\dfrac{\overline{\langle \texttt{10}, \ldots \rangle \Downarrow \langle 10, \ldots \rangle}}{\langle \texttt{(+ 10 1)}, \xi, \phi, \rho \rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))}, \xi, \phi, \rho \rangle \Downarrow}$$

Evaluating $(10+1)\times(10-1)$

$$\cfrac{\cfrac{\overline{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle}}{\langle \texttt{(+ 10 1)},\xi,\phi,\rho\rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))},\xi,\phi,\rho\rangle \Downarrow}$$

Evaluating $(10+1)\times(10-1)$

$$\cfrac{\cfrac{\overline{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle} \qquad \overline{\langle 1,\ldots\rangle \Downarrow}}{\langle \texttt{(+ 10 1)},\xi,\phi,\rho\rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))},\xi,\phi,\rho\rangle \Downarrow}$$

Evaluating $(10+1)\times(10-1)$

$$\cfrac{\cfrac{\overline{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle} \qquad \overline{\langle 1,\ldots\rangle \Downarrow \langle 1,\ldots\rangle}}{\langle \texttt{(+ 10 1)},\xi,\phi,\rho\rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))},\xi,\phi,\rho\rangle \Downarrow}$$

Evaluating $(10+1)\times(10-1)$

$$\cfrac{\cfrac{\overline{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle} \qquad \overline{\langle 1,\ldots\rangle \Downarrow \langle 1,\ldots\rangle}}{\langle \texttt{(+ 10 1)},\xi,\phi,\rho\rangle \Downarrow \langle 11,\xi,\phi,\rho\rangle}}{\langle \texttt{(* (+ 10 1) (- 10 1))},\xi,\phi,\rho\rangle \Downarrow}$$

Evaluating $(10+1)\times(10-1)$

$$\dfrac{\dfrac{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle \qquad \langle 1,\ldots\rangle \Downarrow \langle 1,\ldots\rangle}{\langle \texttt{(+ 10 1)},\xi,\phi,\rho\rangle \Downarrow \langle 11,\xi,\phi,\rho\rangle}}{\langle \texttt{(* (+ 10 1) (- 10 1))},\xi,\phi,\rho\rangle \Downarrow}$$

Evaluating $(10+1)\times(10-1)$

$$\dfrac{\dfrac{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle \qquad \langle 1,\ldots\rangle \Downarrow \langle 1,\ldots\rangle}{\langle \texttt{(+ 10 1)},\xi,\phi,\rho\rangle \Downarrow \langle 11,\xi,\phi,\rho\rangle} \qquad \dfrac{}{\langle \texttt{(- 10 1)},\xi,\phi,\rho\rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))},\xi,\phi,\rho\rangle \Downarrow}$$

Evaluating $(10+1)\times(10-1)$

$$\dfrac{\dfrac{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle \qquad \langle 1,\ldots\rangle \Downarrow \langle 1,\ldots\rangle}{\langle \texttt{(+ 10 1)},\xi,\phi,\rho\rangle \Downarrow \langle 11,\xi,\phi,\rho\rangle} \qquad \dfrac{\langle 10,\ldots\rangle \Downarrow}{\langle \texttt{(- 10 1)},\xi,\phi,\rho\rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))},\xi,\phi,\rho\rangle \Downarrow}$$

Evaluating $(10+1)\times(10-1)$

$$\dfrac{\dfrac{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle \qquad \langle 1,\ldots\rangle \Downarrow \langle 1,\ldots\rangle}{\langle \texttt{(+ 10 1)},\xi,\phi,\rho\rangle \Downarrow \langle 11,\xi,\phi,\rho\rangle} \qquad \dfrac{\langle 10,\ldots\rangle \Downarrow \langle 10,\ldots\rangle}{\langle \texttt{(- 10 1)},\xi,\phi,\rho\rangle \Downarrow}}{\langle \texttt{(* (+ 10 1) (- 10 1))},\xi,\phi,\rho\rangle \Downarrow}$$

Algorithm for building derivations

Want to solve

$$\langle e, \xi, \phi, \rho \rangle \Downarrow ?$$

What rule can I use to prove it?

1. Syntactic form of e narrows to a few choices (usually 1 or 2)
2. Look for form in conclusion
3. Now check premises
4. When premise is evaluation judgment, build sub-derivation recursively

Derivation is written $\mathcal{D}$

Exercise: Evaluate `(digit? 7)`

THE UNIVERSITY OF ARIZONA
Computer Science

$$\frac{\vdots}{\langle \texttt{(and (<= 0 n) (< n 10))}, \xi, \phi, \{\texttt{n} \mapsto 7\} \rangle \Downarrow ?}$$