CSC 520, Spring 2020

Principles of Programming Languages

Michelle Strout

THE UNIVERSITY OF ARIZONA

Version: 1

January 15, 2020

Today's Plan

- **Introduction to Semantics**
- **Abstract Syntax**
- **Impcore operational semantics**
- **But first, what are two things you learned last class?**
  - ➔ **Student responses**

Programming-Language Semantics

- **Semantics means meaning**
- **Ways of knowing what happens when you run code**
  - Learn from examples
  - Build intuition from words describing what will happen
  - To know exactly, *unambiguously*, you need more precision

Why bother with precise semantics?

- **Distill understanding**
- **Express it in a sharable way**
- **Prove useful properties. For example,**
  - Private information doesn't leak
  - Device driver can't crash the OS kernel
  - Compiler optimizations preserve program meaning
  - Most important for you: things that look different are actually the same

Behavior decomposes

THE UNIVERSITY OF ARIZONA
Computer Science

- What happens when we run

```
(* y 3)
```

- Question: what pieces do we need to know about?
  - ➔ student answers

- Knowledge is expressed inductively
  - Atomic forms: describe behavior directly (e.g., constants, variables)
  - Compound forms: behavior specified by composing behaviors of parts

Review: Concrete Syntax for Impcore

THE UNIVERSITY OF ARIZONA
Computer Science

- **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)
```

How to define behaviors inductively

- **Expressions only**
  - Base cases (plural): numerals, names
  - Inductive steps: compound forms

- **To determine behavior of a compound form, look at behaviors of its parts**

First, simplify the task of definition

- **What's different? What's the same?**

```
x = 3;

while (i*i < n) {
  i = i + 1;
}
```

```
(set x 3)

(while (< (* i i) n)
  (set i (+ i 1)))
```

- **Abstract away gratuitous differences**

Abstract Syntax

- **Same inductive structure as BNF**
- **More uniform notation**
- **Good representation in computer**
- **Concrete syntax: sequence of symbols**
- **Abstract syntax: ???**

The abstraction is a tree

- 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.

Assigning behavior to AST of program

- An AST is a data structure that represents a program
- A parser converts program text into an AST
- Question: how can we represent all while loops?

```
while (i < n && a[i] < x) { i++ }
```

  – ➔ student answers

- Question: what about all function applications?
  – ➔ student answers

In C, trees are a bit fiddly

```
typedef struct Exp *Exp;
typedef enum {
  LITERAL, VAR, SET, IFX, WHILEX, BEGIN, APPLY
} Expalt; /* which alternative is it? */
struct Exp { // only two fields: 'alt' and 'u'!
  Expalt alt;
  union {
    Value literal;
    Name var;
    struct { Name name; Exp exp; } set;
    struct { Exp cond; Exp true; Exp false; } ifx;
    struct { Exp cond; Exp exp; } whilex;
    Explist begin;
    struct { Name name; Explist actuals; } apply;
} u;
};
```

- **In class: Draw the while loop example tree and apply example tree (abstract and C rep).**

Let’s picture some more trees

- An expression

```
(f x (* y 3))
```

- A definition

```
(define abs (n)
    (if (< n 0) (- 0 n) n))
```

Behaviors of ASTs, Part 1: Atomic forms

THE UNIVERSITY OF ARIZONA
Computer Science

- **Numeral: stands for a value**
- **Name: stands for what?**

In Impcore, a name stands for a value

THE UNIVERSITY OF ARIZONA
Computer Science

- **Environment associates each variable with one value**
- **Written** $\rho = \{x_1 \mapsto n_1, \ldots x_k \mapsto n_k\}$
- **Associate variable** x_i **with value** n_i
- **Environment is finite map, aka partial function**
  - **x in dom \rho,** **x is defined in environment \rho**
  - **\rho(x),** **the value of x in environment \rho**
  - **\rho{ x \mapsto v }, extends/modifies environment \rho to map x to v**

Environments in C, abstractly

- **An abstract type:**

```
typedef struct Valenv *Valenv;

Valenv mkValenv (Namelist vars, Valulist vals);

bool isvalbound (Name name, Valenv env);

Value fetchval (Name name, Valenv env);

void bindval (Name name, Value val, Valenv env);
```

- **Question: guess what does each of these do?**

"Environment" is point-headed theory

- **You may also hear:**
  - Symbol table
  - Name space
- **Influence of environment is "scope rules", in what part of the code does the environment govern/hold?**

Find behavior using environment

- **Recall**

```
(* y 3)
```

- **Question: what does this mean?**

Impcore uses 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

The University of Arizona
Computer Science

- **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

THE UNIVERSITY OF ARIZONA
Computer Science

- **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"

- **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

THE UNIVERSITY OF ARIZONA.
Computer Science

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

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

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

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

Impcore atomic form: Variable

THE UNIVERSITY OF ARIZONA
Computer Science

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

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

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

Impcore compound form: Assignment

THE UNIVERSITY OF ARIZONA
Computer Science

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

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

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

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