Review: Concrete syntax for Impcore

Definitions and expressions:

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

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;                          (set x 3)

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

Abstract away gratuitous differences

(See the bones beneath the flesh)

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.

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;
};
```

Let's picture some trees

An expression:

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

(Representation uses `Explist`)

A definition:

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

Behaviors of ASTs, part I: Atomic forms

Numeral: stands for a value

Name: stands for what?

In Impcore, a name stands for a value

Environment associates each variable with one value

Written $\rho = \{x_1 \mapsto n_1, \ldots x_k \mapsto n_k\}$,
associates variable x_i with value n_i.

Environment is finite map, aka partial function

$x \in \mathrm{dom}\,\rho$	**x is defined in environment ρ**
$\rho(x)$	**the value of x in environment ρ**
$\rho\{x \mapsto v\}$	**extends/modifies environment ρ to map x to v**

Environments in C, abstractly

An abstract type:

```
typedef struct Valenv *Valenv;

Valenv mkValenv(Namelist vars, Valuelist vals);
bool isvalbound(Name name, Valenv env);
Value fetchval (Name name, Valenv env);
void bindval   (Name name, Value val, Valenv env);
```

“Environment” is pointy-headed theory

You may also hear:

- Symbol table
- Name space

Influence of environment is “scope rules”

- In what part of code does environment govern?

Find behavior using environment

Recall

```
(* y 3)    ;; what does it mean?
```

Your thoughts?

Impcore uses three environments

Global variables ξ

Functions ϕ

Formal parameters ρ

There are no local variables

- **Just like `awk`; if you need temps, use extra formal parameters**
- **For homework, 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"

- **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**
- ξ **Values of global variables**
- ϕ **Definitions of functions**
- ρ **Values of formal parameters**

Three environments determine what is in scope.

Meaning written as "Evaluation judgment"

We say

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

(Big-step judgment form.)

Notes:

- ξ and ξ' may differ
- ρ and ρ' may differ
- ϕ must equal ϕ

Question: what do we know about globals, functions?

Impcore atomic form: Literal

"Literal" generalizes "numeral"

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

Numeral converted to $\textsc{literal}(v)$ **in parser**

Impcore atomic form: Variable name

FORMALVAR

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

GLOBALVAR

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

Parameters hide global variables.

Impcore compound form: Assignment

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

FORMALASSIGN

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

GLOBALASSIGN

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

Impcore can assign only to existing variables