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?

\[ x = 3; \quad \text{(set } x 3) \]

\[ \text{while } (i \ast i < n) \quad \text{(while } (< \ast i i) \quad n) \]
\[ i = i + 1; \quad \text{(set } i (+ i 1)) \]

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):

\[\text{Exp} = \text{LITERAL} \quad \text{(Value)} \]
\[| \quad \text{VAR} \quad \text{(Name)} \]
\[| \quad \text{SET} \quad \text{(Name name, Exp exp)} \]
\[| \quad \text{IFX} \quad \text{(Exp cond, Exp true, Exp false)} \]
\[| \quad \text{WHILEX} \quad \text{(Exp cond, Exp exp)} \]
\[| \quad \text{BEGIN} \quad \text{(Explist)} \]
\[| \quad \text{APPLY} \quad \text{(Name name, Explist actuals)} \]

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:

\[(\text{define abs (n)} \quad (\text{if} \ (< \ n \ 0) \ (- \ 0 \ n) \ n))\]
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

$x \in \text{dom } \rho \quad x$ is defined in environment $\rho$

$\rho(x) \quad$ the value of $x$ in environment $\rho$

$\rho\{x \mapsto v\} \quad$ extends/modifies environment $\rho$ to map $x$ to $v$
Environments in C, abstractly

An abstract type:

```c
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

\((* y 3) \;; \text{ what does it mean?}\)

Your thoughts?
Impcore uses three environments

Global variables $\xi$

Functions $\phi$

Formal parameters $\rho$

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 $\phi$ 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
- \( \xi \) Values of global variables
- \( \phi \) Definitions of functions
- \( \rho \) Values of formal parameters

Three environments determine what is in scope.
Meaning written as “Evaluation judgment”

We say
\[ \langle e, \xi, \phi, \rho \rangle \downarrow \langle v, \xi', \phi, \rho' \rangle \]

(Big-step judgment 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”

\[ \langle \text{LITERAL}(v), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi, \phi, \rho \rangle \]

Numeral converted to \text{LITERAL}(v) in parser
Impcore atomic form: Variable name

\[ \begin{align*}
\text{FORMALVar} & \\
& \quad x \in \text{dom } \rho \\
& \quad \langle \text{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \rho(x), \xi, \phi, \rho \rangle
\end{align*} \]

\[ \begin{align*}
\text{GLOBALVar} & \\
& \quad x \notin \text{dom } \rho \quad x \in \text{dom } \xi \\
& \quad \langle \text{VAR}(x), \xi, \phi, \rho \rangle \Downarrow \langle \xi(x), \xi, \phi, \rho \rangle
\end{align*} \]

Parameters hide global variables.
Impcore compound form: Assignment

In \( \text{SET}(x, e) \), \( e \) is any expression

**FORMALASSIGN**

\[
x \in \text{dom} \rho \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle
\]

\[
\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \{x \mapsto v\} \rangle
\]

**GLOBALASSIGN**

\[
x \notin \text{dom} \rho \quad x \in \text{dom} \xi \quad \langle e, \xi, \phi, \rho \rangle \Downarrow \langle v, \xi', \phi, \rho' \rangle
\]

\[
\langle \text{SET}(x, e), \xi, \phi, \rho \rangle \Downarrow \langle v, \xi' \{x \mapsto v\}, \phi, \rho' \rangle
\]

Impcore can assign only to existing variables