Our common framework

Goal: eliminate superficial differences
  • Makes comparisons easy
  • Differences that remain must be important!

No new language ideas.

Imperative programming with an IMPerative CORE:
  • Has features found in most languages
    (loops and assignment)
  • Trivial syntax (from LISP)
Idea of LISP syntax

Parenthesized prefix syntax:

• Names and numerals are basic atoms
• Other constructs bracketed with (…) or […]  
  (Possible keyword after opening bracket)

Examples:

(+ 2 2)
(if (isbound? x rho) (lookup rho x) (error 99))

(For now, we use just the round brackets)
Impcore structure

Two syntactic categories: expressions, definitions

No statements!—expression-oriented (compositional)

(if e1 e2 e3)
(while e1 e2)
(set x e)
(begin e1 ... en)
(f e1 ... en)

Evaluating e has value, may have side effects

Functions f named (e.g., + − * / = < > print)

The only type of data is “machine integer”
(deliberate oversimplification)
Syntactic structure of Impcore

An Impcore program is a sequence of definitions

(define mod (m n) (- m (* n (/ m n))))

Compare

int mod (int m, int n) {
    return m - n * (m / n);
}

Impcore variable definition

Example

(val n 99)

Compare

int n = 99;
Concrete syntax for Impcore

Definitions and expressions:

```
def ::= (define f (x1 ... xn) exp) ;; "true" defs
| (val x exp)
| exp
| (use filename) ;; "extended" defs
| (check-expect exp1 exp2)
| (check-assert exp)
| (check-error exp)

exp ::= integer-literal
| variable-name
| (set x exp)
| (if exp1 exp2 exp3)
| (while exp1 exp2)
| (begin exp1 ... expn)
| (function-name exp1 ... expn)
```
Example function shows every form

(define even? (n) (= (mod n 2) 0))

(define 3n+1-sequence (n) ; from Collatz
    (begin
        (while (!= n 1)
            (begin
                (println n)
                (if (even? n)
                    (set n (/ n 2))
                    (set n (+ (* 3 n) 1)))))
        n))