Solution to “all-fours?”

(check-assert (all-fours? 4))
(check-assert (not (all-fours? 5)))
(check-assert (all-fours? 44))
(check-assert (not (all-fours? 14)))

(define all-fours? (n)
  (if (< n 10)
      (= n 4)
      (and (= 4 (mod n 10))
           (all-fours? (/ n 10)))))

;; D2 recursion: n is d, where 0 < d < 10, or
;; n is 10 * m + d, where m > 0
Concrete syntax for Impcore (again)

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)
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?

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

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

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):

$$\text{Exp} = \text{LITERAL} \ (\text{Value})$$

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>VAR</td>
<td>(Name)</td>
</tr>
<tr>
<td>SET</td>
<td>(Name name, Exp exp)</td>
</tr>
<tr>
<td>IFX</td>
<td>(Exp cond, Exp true, Exp false)</td>
</tr>
<tr>
<td>WHILEX</td>
<td>(Exp cond, Exp exp)</td>
</tr>
<tr>
<td>BEGIN</td>
<td>(Explist)</td>
</tr>
<tr>
<td>APPLY</td>
<td>(Name name, Explist actuals)</td>
</tr>
</tbody>
</table>

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:

\[
\begin{align*}
\text{(define abs (n)} \\
\text{ (if (< n 0) (- 0 n) n))}
\end{align*}
\]
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

\( x \in \text{dom}\ \rho \) \( x \) 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, 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

\[(\ast \ y \ 3) \ ;; \ what \ does \ it \ mean?\]

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