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:

\[
\text{def} ::= (\text{define } f \ (x_1 \ldots x_n) \ \text{exp}) \quad ;; \text{"true" defs} \\
\quad | \quad (\text{val } x \ \text{exp}) \\
\quad | \quad \text{exp} \\
\quad | \quad (\text{use } \text{filename}) \quad ;; \text{"extended" defs} \\
\quad | \quad (\text{check-expect } \text{exp1} \ \text{exp2}) \\
\quad | \quad (\text{check-assert } \text{exp}) \\
\quad | \quad (\text{check-error } \text{exp}) \\
\]

\[
\text{exp} ::= \text{integer-literal} \\
\quad | \quad \text{variable-name} \\
\quad | \quad (\text{set } x \ \text{exp}) \\
\quad | \quad (\text{if } \text{exp1} \ \text{exp2} \ \text{exp3}) \\
\quad | \quad (\text{while } \text{exp1} \ \text{exp2}) \\
\quad | \quad (\text{begin } \text{exp1} \ldots \ \text{expn}) \\
\quad | \quad (\text{function-name } \text{exp1} \ldots \ \text{expn})
\]
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?

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

\[
\text{while } (i \times i < n) \quad (\text{while } (< (\ast i i) \ n)
\]

\[
i = i + 1; \quad (\text{set } i (+ i 1))
\]

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):

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

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;
    Expplist begin;
    struct { Name name; Expplist actuals; } apply;
  } u;
};
Let’s picture some trees

An expression:

\((f \ x \ (* \ y \ 3))\)

(Representation uses Explist)

A definition:

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

$x \in \text{dom } \rho$  \hspace{1cm} $x$ is defined in environment $\rho$

$\rho(x)$  \hspace{1cm} the value of $x$ in environment $\rho$

$\rho\{x \mapsto v\}$  \hspace{1cm} 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

(* 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