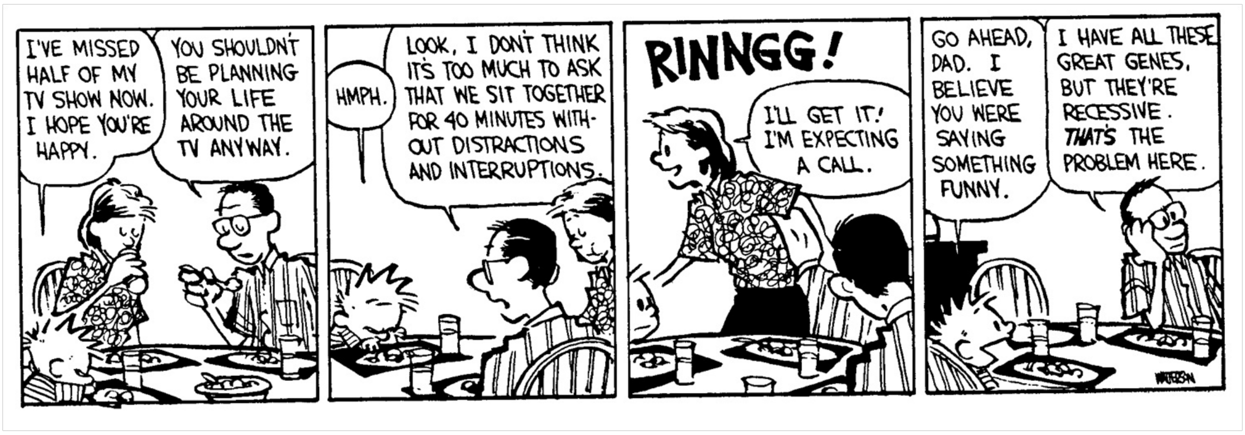
Meeting 13: Data Types and Polymorphism


I'VE MISSED HALF OF MY TV SHOW NOW. I HOPE YOU'RE HAPPY.
YOU SHOULDN'T BE PLANNING YOUR LIFE AROUND THE TV ANYWAY.
HMPH.
LOOK, I DON'T THINK IT'S TOO MUCH TO ASK THAT WE SIT TOGETHER FOR 40 MINUTES WITHOUT DISTRACTIONS AND INTERRUPTIONS.
RINNGG!
I'LL GET IT! I'M EXPECTING A CALL.
GO AHEAD, DAD. I BELIEVE YOU WERE SAYING SOMETHING FUNNY.
I HAVE ALL THESE GREAT GENES, BUT THEY'RE RECESSIVE. THAT'S THE PROBLEM HERE.
WATTERSON

Announcements

HW4 due Friday 10/13

start early! utilize tutoring hours!

Questions

1. printing and explicate — print polymorphic print ok
2. run func
3. what needs to be explicated? ✓
4. Go over the type checking
5. GetTag vs. in run func ✓
6. input() — PI — just for int ✓

input returns int

input_int() returns pyobj in tagged value

7. flattening if-expressions?

x = [] if input() else 3

⋮

- x

if expressions in P1 but not if statements

---

⑤ Get Tag vs. ~~is_int(. . _) in the runtime~~

~~project to : (type, tagged) → untagged~~

~~injection : (type, untagged) → tagged (pyobj)~~

not don't call from your compiler

Rule for HW4 for what is ok to do in the runtime

and what to produce compiled code:

P0 (int / bool) should run fast — don't go into memory unnecessarily

calls are expensive

Big things like lists can go the runtime

Why?

meta of the runtime object

pyobj (in C) ≈ values in P1 (tagged)

Rule for simplifying P1 is to (types) converting

~~IntToBool (- - - -)~~

IfExp(e_1, e_2, e_3)

atomic ::= variable
| constant

flatten_expr : expression $\rightarrow$ ($\underline{\text{statements}}$, $\underline{\text{atomic}}$)

$\boxed{e_2 \text{ if } e_1 \text{ else } e_3}$

P_1 input

(fs_1, fa_1) = flatten_expr(e_1

$\downarrow$ flatten

```
fs_1
if fa_1:
    fs_2
    tmp = fa_2
else:
    fs_3
    tmp = fa_3
```

flattened P_1 output

tmp

---

fs

fa

IfStmt[a, S_1, S_2)