

# Meeting 12: Data Types and Polymorphism

PI  
HW4



## Announcements

HW4 due Friday 10/13

### Explicit Questions

- • What's Let(...)? ✓
- • Explicitation - typing fails? ✓
- • Flattening PI
- • Project to / Inject From ✓
- • AST Node design - two diff  
Add ✓

compile-time  
 - execution of your compiler  
meta

run-time  
 - execution of the output of your compiler  
object

Expr code

Project To  
 Inject From

$e_1 + e_2$   
 expression  $e$   
 $e_1 = \text{input}(k)$

Add( $e_1, e_2$ )

explicit

```

Let(x, e1)
Let(y, e2, AND(Expr
  IF(Expr(GetVal(x), INT),
    Expr(GetVal(y), INT)))
  InjectFrom Int AddInt(x, y))
  AND(Expr
  IF(Expr(GetVal(x), BIG),
    Expr(GetVal(y), BIG)))
  InjectFrom Int AddBig(x, y))
  
```

INT=0  
 BIG=3

```

def explicit(node):
  if isinstance(node, Add):
    e1 = node.left
    e2 = node.right
    x = make_temp()
    y = make_temp()
    return
  
```

→ AddBig(x, y) ← Project To  
 InjectFrom ProjectTo(big, -)

CallFunc('abort', - - -) //

Let  $(x, e_1, e_2)$

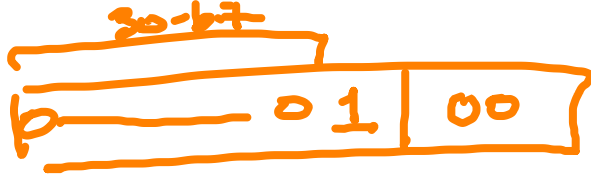
$\text{IfExp}(e_1, e_2, e_3)$

---

every (explicit python) AST is  
a tagged value

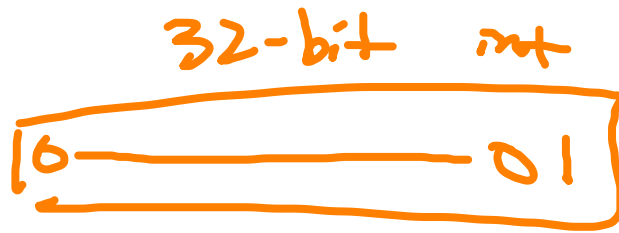
project to :: Tagged value  $\rightarrow$  untagged  
value  
of some  
specified  
type

inject from :: untagged  
value of  
specified  
type  $\rightarrow$  tagged  
value



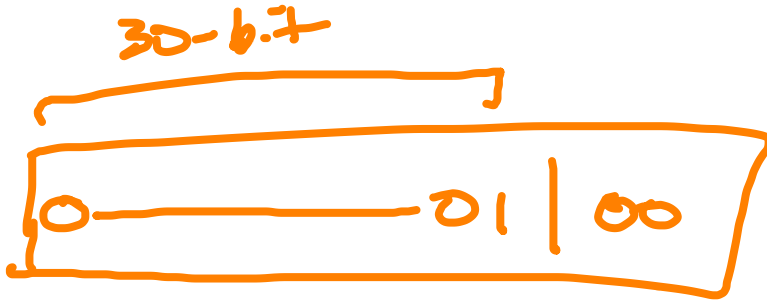
tagged value

↓ project to (int)



untagged value  
≡

↓ inject from (int)



---

IfExp( $e_1, e_2, e_3$ )

# Flatten PD (HW1)

expression

flatten  
→

[ tup =  
...  
],  
tup

expression

flatten  
→

not only assignments  
but also  
if statements













