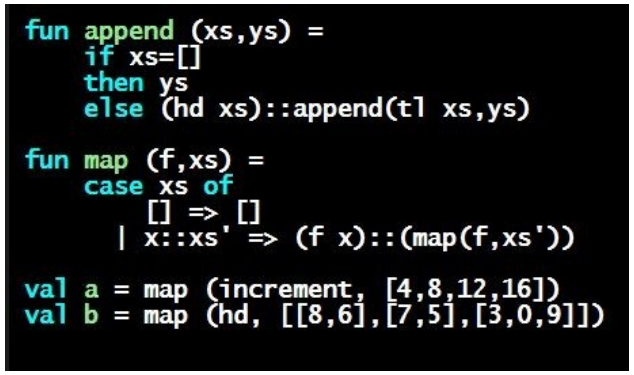

Programming Languages

Dan Grossman

2013

Static Versus Dynamic Typing, Part One

Now can argue…

Having carefully stated facts about static checking,
can *now* consider arguments about which is *better*:
static checking or dynamic checking

Remember most languages do some of each

- For example, perhaps types for primitives are checked statically, but array bounds are not

Claim 1a: Dynamic is more convenient

Dynamic typing lets you build a heterogeneous list or return a "number or a string" without workarounds

```
(define (f y)
  (if (> y 0) (+ y y) "hi"))

(let ([ans (f x)])
  (if (number? ans) (number->string ans) ans))
```

```
datatype t = Int of int | String of string
fun f y = if y > 0 then Int(y+y) else String "hi"

case f x of
   Int i => Int.toString i
 | String s => s
```

Claim 1b: Static is more convenient

Can assume data has the expected type without cluttering code with dynamic checks or having errors far from the logical mistake

```
(define (cube x)
  (if (not (number? x))
      (error "bad arguments")
      (* x x x)))

(cube 7)
```

```
fun cube x = x * x * x

cube 7
```

Claim 2a: Static prevents useful programs

Any sound static type system forbids programs that do nothing wrong, forcing programmers to code around limitations

```
(define (f g)
  (cons (g 7) (g #t)))

(define pair_of_pairs
  (f (lambda (x) (cons x x))))
```

```
fun f g = (g 7, g true) (* does not type-check *)

val pair_of_pairs = f (fn x => (x,x))
```

Claim 2b: Static lets you tag as needed

Rather than suffer time, space, and late-errors costs of tagging everything, statically typed languages let programmers “tag as needed” (e.g., with datatypes)

In the extreme, can use "TheOneRacketType" in ML

- Extreme rarely needed in practice

```
datatype tort = Int of int
              | String of string
              | Cons of tort * tort
              | Fun of tort -> tort
              | …

if e1
then Fun (fn x => case x of Int i => Int (i*i*i))
else Cons (Int 7, String "hi")
```

Claim 3a: Static catches bugs earlier

Static typing catches many simple bugs as soon as "compiled"

- Since such bugs are always caught, no need to test for them
- In fact, can code less carefully and "lean on" type-checker

```
(define (pow x) ; curried
  (lambda (y)
    (if (= y 0)
        1
        (* x (pow x (- y 1))))))) ; oops
```

```
fun pow x y =  (* does not type-check *)
  if y = 0
  then 1
  else x * pow (x,y-1)
```

Claim 3b: Static catches only easy bugs

But static often catches only "easy" bugs, so you still have to test your functions, which should find the "easy" bugs too

```
(define (pow x) ; curried
  (lambda (y)
    (if (= y 0)
        1
        (+ x ((pow x) (- y 1))))))) ; oops
```

```
fun pow x y =  (* curried *)
  if y = 0
  then 1
  else x + pow x (y-1) (* oops *)
```