- ((□^{1} f. □^{2} x. f x)(□^{3} a.a))(□^{4} b.b)
- f □ (□^{3} a.a) [2]
  - a □ x [2]
  - f x □ a [3]
- (□^{1} f. □^{2} x. f x)(□^{3} a.a) □ □^{2} x. f x [3]
  - x □ (□^{4} b.b) [2]
  - ((□^{1} f. □^{2} x. f x)(□^{3} a.a))(□^{4} b.b) □ f x [3]

- ((□^{1} f. □^{2} x. f x)(□^{3} a.a))(□^{4} b.b) □ f x □ a □ x □ (□^{4} b.b) [4]

Section 2.2

- Types given are not the only types of the subcomponents
- But those are the types needed to make the entire formula typecheck
- ((□^{1} f. □^{2} x. f x)(□^{3} a.a))(□^{4} b.b)
- (□^{1} f. □^{2} x. f x) : (□ □ □) □ □ □ □
- f : (□ □ □) !- □^{2} x. f x : □ □ □
- □^{3} a.a : □ □ □

Lackwit

Using type inference to find bugs

- Use refined types
- E.g., not all uses of int are the same
  - size in bytes of a character array
  - size in characters of a character array
  - Socket #
  - File descriptor
  - student ID
  - exam score

example

```
int x;
int p1;
void f(int a, int b, int * c, int * d)
{ x = a;
*c = *d;
}
void g(int * q, int * r, int * s)
{ int t1 = 2;
int c1 = 3, c2 = 4;
int p;
p = p1;
x++;
f(c1, p, &t1, q);
f(c2, c2, r, s);
}
```

Annotate with types

- lower case letters for ground types
  - non-polymorphic
- upper case letters for polymorphic types in functions

Annotated example

```
int^n x;
int^z p1;
void f(int^n a, int^A b, int^B * c, int^B * d)
{ x = a;
*c = *d;
}
void g(int^Y * q, int^X * r, int^X * s)
{ int^Y t1 = 2;
int^n c1 = 3, c2 = 4;
int^z p;
p = p1;
x++;
f(c1, p, &t1, q);
f(c2, c2, r, s);
}
```

Need polymorphic pointer analysis

- Steensgaard's analysis isn't polymorphic
  - int f(int *p) { return *p; }
  - int A,B;
  - int t = f(&A)
  - int u = f(&B)
- t and A should have the same type
- u and B should have the same type
- t and u should have distinct types

Polymorphic interference

- Basic type inference is the same
  - but at each call site, copy and unify type arguments

Annotated example

```
int^n x;
int^z p1;
void f(int^C a, int^A b, int^B * c, int^D * d)
{ x = a; // C = n
*c = *d; // B == D
}
void g(int^Y * q, int^X * r, int^W * s)
{ int^U t1 = 2;
int^V c1 = 3;
int^T c2 = 4;
int^S p;
p = p1; // S == z
x++;
f(c1, p, &t1, q); // V == n && U == Y
f(c2, c2, r, s); // T == n && X == W
}
```

```
int^n x;
int^z p1;
void f(int^n a, int^A b, int^B * c, int^B * d)
{ x = a;
*c = *d;
}
void g(int^Y * q, int^X * r, int^X * s)
{ int^Y t1 = 2;
int^n c1 = 3, c2 = 4;
int^z p;
p = p1;
x++;
f(c1, p, &t1, q);
f(c2, c2, r, s);
}
```

Who can directly access representation of component X of variable Y?

- Who can access the current vehicle field of map_manager_global?
- Shaded nodes constrain type by accessing fields of the structure

Augmented type system

- Augment types with a set of properties about how values of that type are used
  - e.g., allocated, deallocated, read, written, …
- For example,
  - a pointer to □ has type (□, read, write)
  - deref has type □□ □□. (□, yes, □) □ □