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
- \( ((\lambda f. \lambda x. f x)(\lambda^1 a.a))(\lambda^1 b.b) \)
- \( f \in (\lambda^1 a.a) \quad [2] \)
  - \( a \in x \quad [2] \)
  - \( f x \in a \quad [3] \)
- \( ((\lambda f. \lambda x. f x)(\lambda^1 a.a))(\lambda x. f x) \quad [3] \)
  - \( x \in (\lambda^1 b.b) \quad [2] \)
  - \( ((\lambda f. \lambda x. f x)(\lambda^1 a.a))(\lambda b.b) \in f x \quad [3] \)
- \( ((\lambda f. \lambda x. f x)(\lambda^1 a.a))(\lambda^1 b.b) \in f x \)
  - \( a \in x \in (\lambda^1 b.b) \quad [4] \)
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

```c
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

```c
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);
    }
```
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

```c
int x;
int y;
valid(int a, int b, int c, int d, int e) {
  if (x && c)
    y = y + z;
  valid(g(x, a, f(b), a), x, a, b, c, d, e);
}
```

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, 🗄 🗄 🗄