Background

- What are modes?

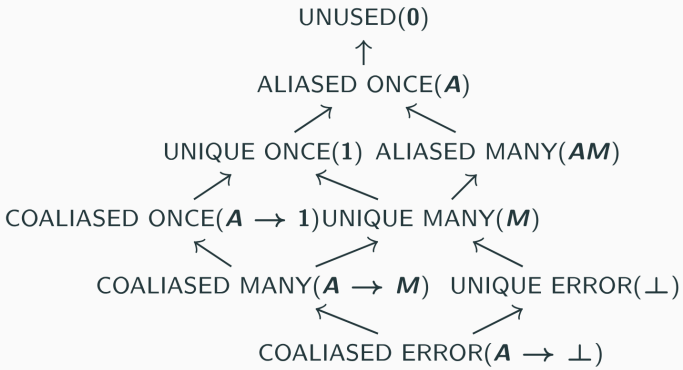
```
let process_parts_wrong (pair @ unique) =
let x @ unique = pair.fst in
let y @ unique = pair.fst in ...
```

- Modes as semirings (with orderings).

  many $\leq$ once

  once $+$ once $=$ many

  unique many $\cdot$ aliased once $=$ aliased many



- Where the semiring goes wrong...

```
let y @ aliased many = ... in
```

```
(y @@ aliased, y @@ aliased) @ unique                    ✓
  (* aliased + aliased = aliased many *)

let x = y @@ aliased in (x, x) @ unique                   ×
  (* (unique + unique) · aliased = aliased many *)
```

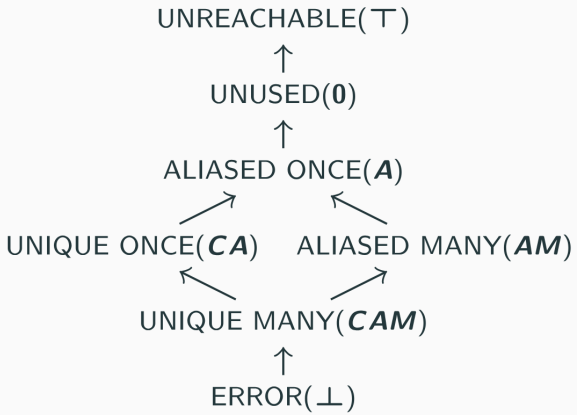
- Again, what are modes?

What I did

Proved useful properties (substitution, beta and eta law type preservation) of various modal calculi to compile a minimal set of laws needed for the modes that we wanted to express.

Takeaways:

- Splitting $\cdot$ into $\triangleleft$ (scale) and $\triangleright$ (box).
- Converting equalities to inequalities.
  $(a \triangleleft b) \triangleleft c = a \triangleleft (b \triangleleft c)$ might not hold,
  but $(a \triangleleft b) \triangleleft c \leq a \triangleleft (b \triangleleft c)$ does!
- Locks expressed as a division operator in terms of $\triangleleft$.



What else you can do:

- Show other mode systems follow our structure.
- Take advantage of the weaker laws to do other fun things (Kleene star for grades!)