Bringing Back Disco

Callahan Hirrel

Disco

- Functional programming language implemented in Haskell
- Intended to be used in a discrete math course
- Features are ONLY those relevant to discrete math
- Syntax is carefully design to be as close to mathematical syntax as possible

Why is this important?

- Functional programming is generally taught as an elective
- Disco offers experience with functional programming early on in a required class
- Gives CS students a familiar environment to learn abstract (and sometimes uncomfortable and just plain confusing) math concepts
- Helps strengthen and illuminate the deep connection between math and computer science

What did I learn?

- What it is like to work on a large, ongoing development project
- CS work outside of a classroom setting, so it was a bit more relaxed and self-paced
- The kinds of decisions and thought-processes involved in development
- Some basic Haskell and functional programming in general (algebraic data types and lazy evaluation to name a few)
- Types, type systems, and some of their underlying theory in programming

Disco in action

Example Time: Finite Types

For each natural number *n*, there should be a type with exactly *n* different values.

Why are finite types important?

- We have other types with a finite number of values:
  - `Void` has exactly 0 values
  - `Unit` has exactly 1 value
  - `Bool` has exactly 2 values (true and false)

Why are finite types important?

- We can have combinations of types
  - sums and pairs, to name a few
- Finite types can be helpful for combinatorics problems
  - Counting/enumerating types or combinations of types

Needed to decide on syntax

- `Fin n` - similar to a function call, so it's familiar. This is also similar to finite types in other languages.
- `[n]` - makes sense, because it is kind of like a list with n inhabitants. However, we don't want to get it confused with actual lists.
- `1..n` - this goes a long with the finite type being similar to a list
- `ℤn` - much like the notation used for cyclic groups
- ***BUT*** *we also need to think about how these types will be used, maybe it will help us decide on the syntax?*

What should values of finite types look like?

- If we have a type with exactly five different values, how should the values be represented?
- Easy way is simply `0, 1, 2, 3, 4`
- So, what can we do with these values…

Should finite types support arithmetic?

- "No." Well, okay then, what's the point?
- "Yes!" Well, okay, then, what kind?
  - Saturating - `3+4` in the finite type with 5 values would just be 4 (i.e.: we cut off increasing arithmetic at 4 and decreasing arithmetic at 0)
  - Modular - `3+4` in the finite type with 5 values would instead be 2 (i.e.: 3+4=7, which we then mod by 5 to get 2

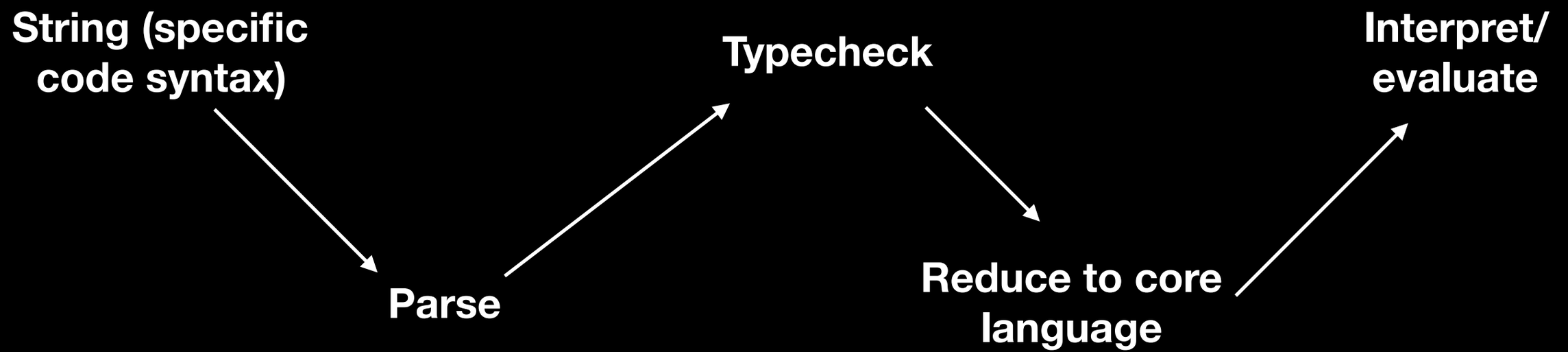
Decisions, decisions

- We ended up deciding for finite types to have *modular arithmetic*

- Because of this, we decided on `ℤn` as the syntax for these types (we also included `Fin n`, similar to other FLs)

- Their behavior will be very similar to cyclic groups, and we want to stay as close as possible to mathematical syntax.

Process of Adding a Feature



Enough with the slides, let's see some code…