An embedding of pure type systems in Agda
Blah blah blah blah blah blah blah blah blah blah
So what’s this all about?

There are two things to explain:

- Pure Type System (PTS)
- Agda
So what’s this all about?

There are two things to explain:

- Pure Type System (PTS)

- Agda

They’re actually very related!
Programming languages primer
Programming languages primer

Joe
Bob
Alex
Rob

SomeDocument.doc
Programming languages primer

Joe : Person
Bob : Person
Alex : Person
Rob : Person

SomeDocument.doc : FileHandle
Programming languages primer

Joe : Person
Bob : Person
Alex : Person
Rob : Person

SomeDocument.doc : FileHandle

Types!
Programming languages primer

What if we want more complicated types?

[Joe, Bob] : List Person
[Alex] : List Person
[] : List Person

[SomeDocument.doc] : List File
Programming languages primer

somedocument = fopen(“SomeDocument.doc”)

Say fopen : String -> File

This means somedocument : File here.

fwrite(somedocument, “Hi”)

Naturally, fwrite : File -> String -> ()
Programming languages primer

somedocument = fopen("SomeDocument.doc")
somedocument = fclose(somedocument)
fwrite(somedocument, "Hi!")
Programming languages primer

somedocument = fopen("SomeDocument.doc")

somedocument = fclose(somedocument)

fwrite(somedocument, "Hi!")

Uh oh!
Programming languages primer

We want more expressive types!

Consider the types File Open and File Close
We want more expressive types!

Consider the types `File Open` and `File Close`

```plaintext
fopen : String -> File Open
fwrite : File Open -> String -> ()
fclose : File Open -> File Close
```

Now we can verify at compile-time that it’s impossible to write to a closed file!
Programming languages primer

But wait, I just pulled a fast one on you.
We want more expressive types!

Consider the types File Open and File Close

\[
\begin{align*}
\text{fopen} & : \text{String} \to \text{File Open} \\
\text{fwrite} & : \text{File Open} \to \text{String} \to () \\
\text{fclose} & : \text{File Open} \to \text{File Close}
\end{align*}
\]

Now we can verify at compile-time that it’s impossible to write to a closed file!
We want more expressive types!

Consider the types `File Open` and `File Close`

```plaintext
fopen : String -> File Open
fwrite : File Open -> String -> ()
fclose : File Open -> File Close
```

Now we can verify at compile-time that it’s impossible to write to a closed file!
Who says I can’t have the type `File Animal`?

What are `Open` and `Close`? Are they types? They don’t feel like types...
Agda

Agda is “dependently-typed”, meaning that you can include values in types.

Turns out this is super powerful.
Pure Type System

A framework aimed at generalizing and unifying the idea of “dependently-typed” languages.
Embedding a PTS in Agda

Will it work?
Embedding a PTS in Agda

Will it work?

Who knows! Untrodden ground! But hopefully it will.
Why would you do this?

Agda is powerful enough to both program in, and prove mathematical theorems in.

Embedding PTS in Agda would allow us to prove all sorts of neat things about PTS!
If it works...

All sorts of neat proofs we can port into Agda:
- Substitution lemma
- Confluence
- Parametricity
- Strong normalization
- And more!
Questions?