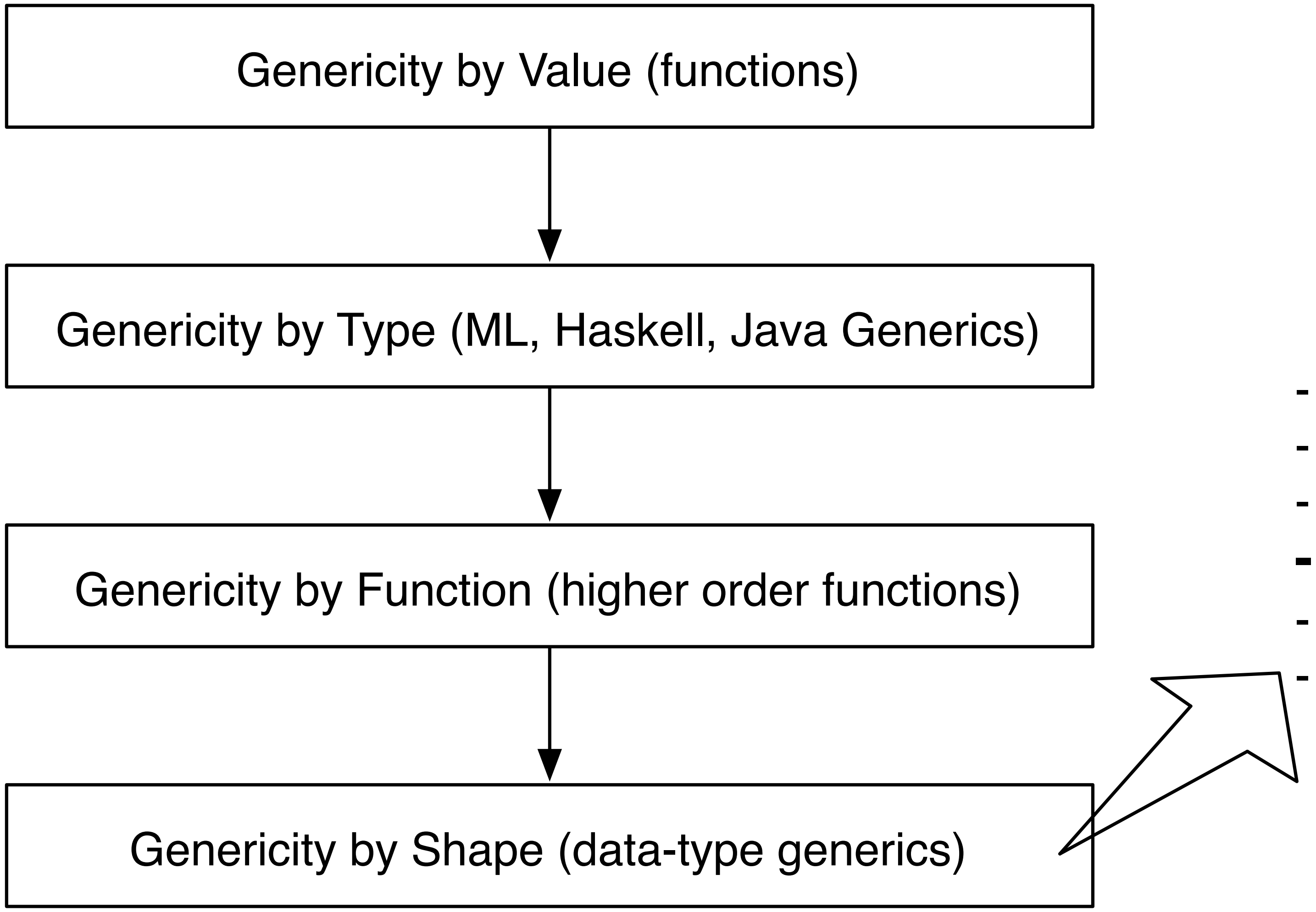
Implementing the Pattern Calculus

from theory to practice

plrg macquarie

Why?



Genericity by Value (functions)

Genericity by Type (ML, Haskell, Java Generics)

Genericity by Function (higher order functions)

Genericity by Shape (data-type generics)

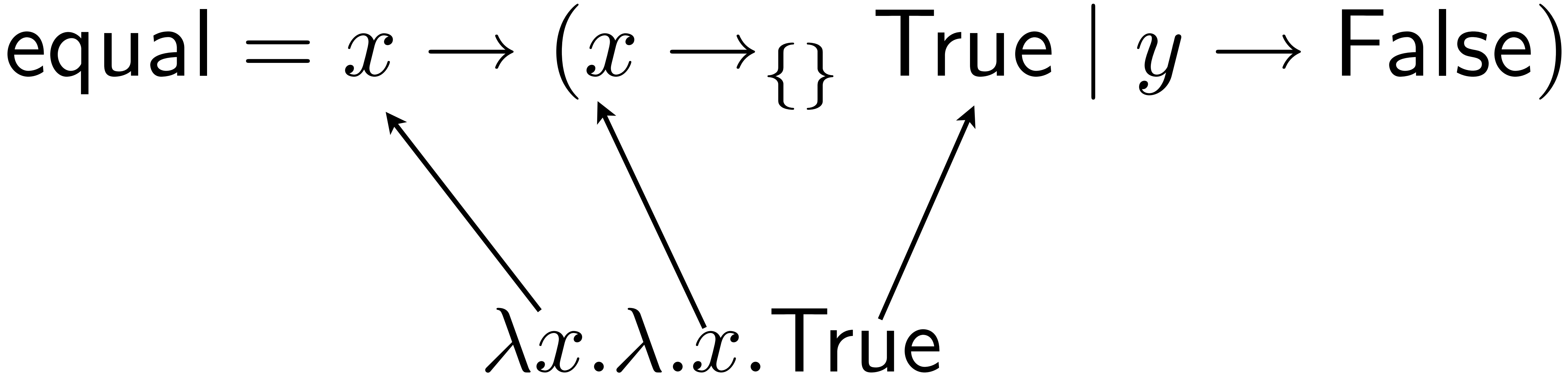
- Generic Haskell
- Generics in Clean
- Scrap Your Boilerplate
- **The Pure Pattern Calculus**
- Generics for the masses
- etc.

Interests

- **Datatype-generic programming**
- Compiler generators
- Program Transformations

Pattern Calculus World

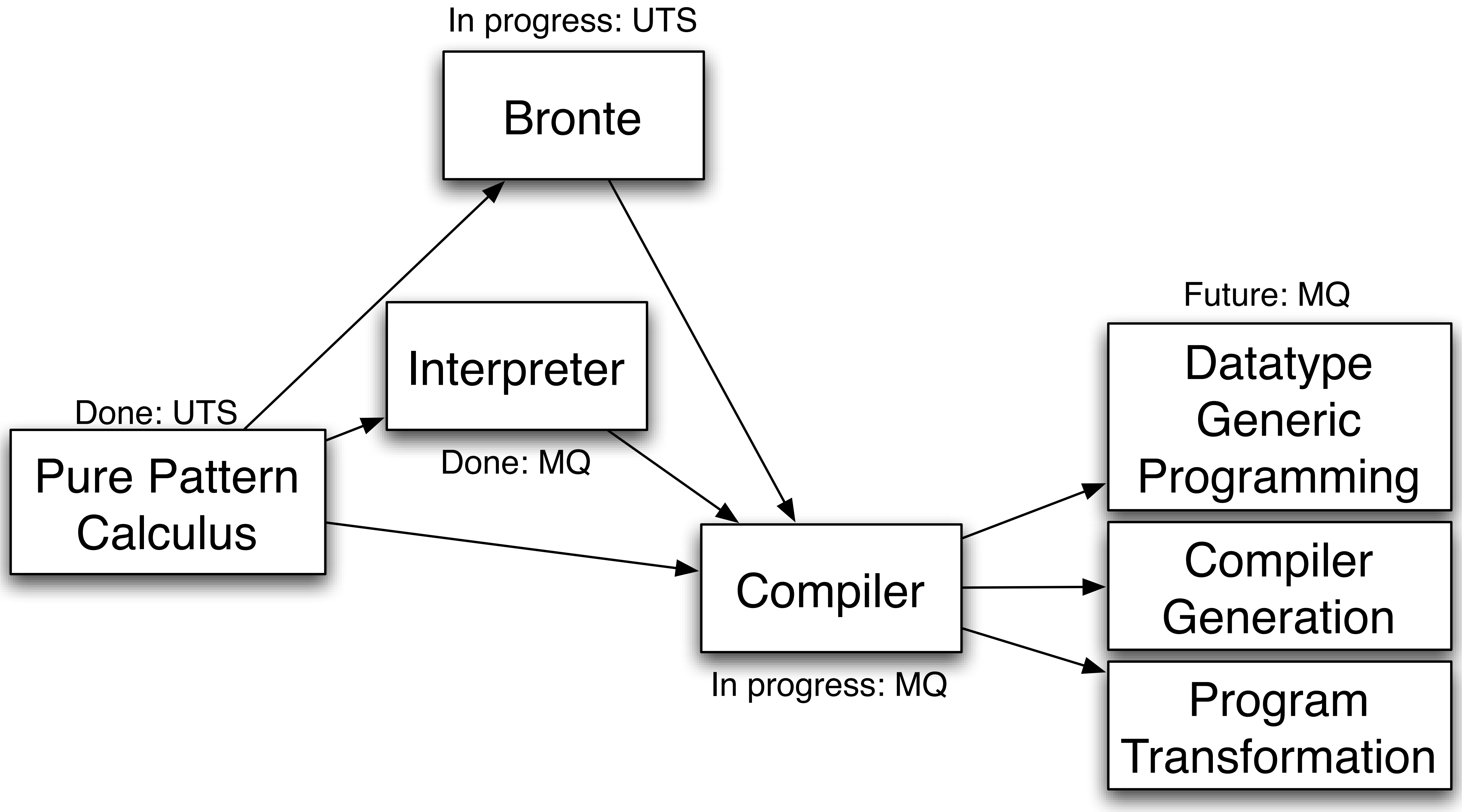
- Created by Barry Jay at UTS.
  - portable patterns
  - any expression can be a pattern
  - data, structure, path and pattern polymorphism
- A fair bit of work going on there as well.
- Macquarie is focussing on *implementation*.



$$\text{equal} = x \to (x \to_{\{\}} \text{True} \mid y \to \text{False})$$

$$\lambda x.\lambda.x.\text{True}$$

Where are we going?


In progress: UTS

Bronte

Interpreter

Done: UTS

Pure Pattern Calculus

Done: MQ

Compiler

In progress: MQ

Future: MQ

Datatype Generic Programming

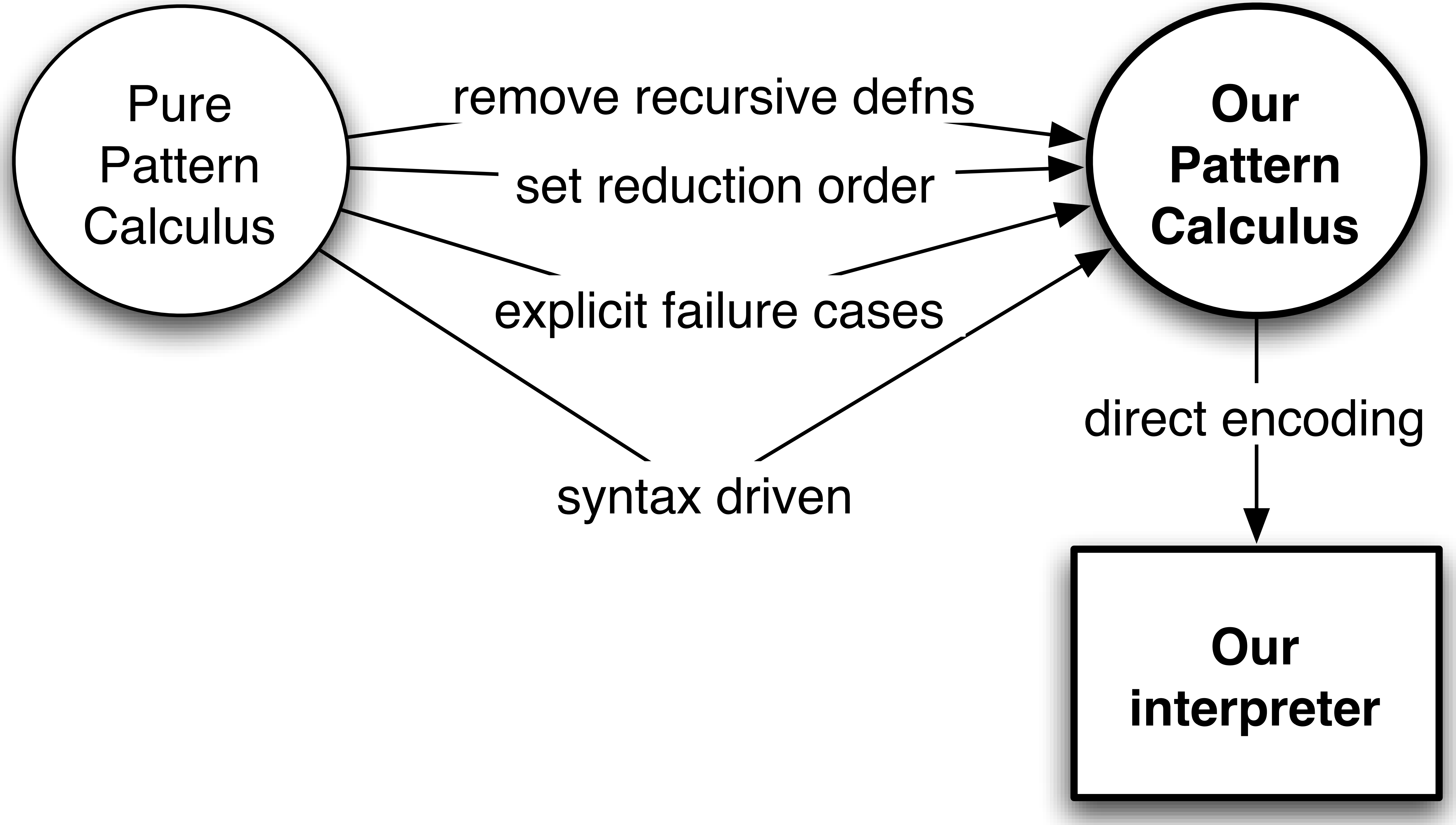
Compiler Generation

Program Transformation

Approach

- Interpreters to *explore the space of solutions*.
- Interpreters are easy right? - Yay for Haskell!
- Compiler(s) once we get settled.
- Compilers are hard work right? - Yay for Haskell.......

A working interpreter



Pure Pattern Calculus

remove recursive defns

set reduction order

explicit failure cases

syntax driven

Our Pattern Calculus

direct encoding

Our interpreter

Untyped (so far)

- We are on the case.
- System-F-like (but not System-F)
- My changes push in the direction of System-F anyway
- I’m confident ...

(E-App1)

$$\frac{t_1 \Rightarrow t_1'}{t_1\ t_2 \Rightarrow t_1'\ t_2}$$

(E-App2)

$$\frac{t_2 \Rightarrow t_2'}{v_1\ t_2 \Rightarrow v_1\ t_2'}$$

(E-Patt)

$$\frac{p \Rightarrow p'}{(p \to_\theta s \mid r) \Rightarrow (p' \to_\theta s \mid r)}$$

(E-AppAbsVar)

$$\frac{}{(x \to_\theta s \mid r)\ v \Rightarrow [x \mapsto v]\ s}$$

(E-AppAbsConstr1)

$$\frac{}{(C \to_\theta s \mid r)\ C \Rightarrow s}$$

(E-AppAbsConstr2)

$$\frac{C_1 \neq C_2}{(C_1 \to_\theta s \mid r)\ C_2 \Rightarrow\ r\ C_2}$$

(E-AppAbsConstr3)

$$\frac{}{(C \to_\theta s \mid r)\ v \Rightarrow r\ v}$$

(E-AppAbsApp)

$$\frac{\theta' = \theta \setminus d_1}{(d_1\ d_2 \to s \mid r)\ (v_1\ v_2) \Rightarrow (d_1 \to ((d_2 \to s \mid (v' \to r\ (v_1\ v_2)))\ v_2) \mid (v' \to r\ (v_1\ v_2)))\ v_1\ v_2}$$

Where to now?

- We have a simpler semantics that does what we need and will be (relatively) easy to implement.
- Then choose/build a type system.
- Next is the IL/Abstract machine that best suits the pattern calculus.

That’s not much

- Yeah, but... a whole world opens up from there.
- How much benefit/cost do we get from laziness?
- Compare this pattern matching mechanism to others in use (fat-bar, rho-stratego, etc.)
- What coverage of datatype-generic programming can you achieve?

- Can we embed this approach (these semantics) in some existing language?
- How can we use this as a term-rewriting system?
- How can we use this in compiler generation?
- Can we find any interesting optimisations?
- Can we target existing IL?

Comments?

- Some of these ideas are interesting, some are probably not.
- The point is, a real implementation opens up options for us.
- It makes new questions feasible to explore.