

# Dependently Typed Languages in Statix

Jonathan Brouwer   Jesper Cockx   Aron Zwaan

Delft University of Technology, The Netherlands

March 17, 2023

## Background: What are Dependent Types?

- Types may depend on values!

### Example

```
concat : (A: Set) -> (n m : Nat) -> Vec A n -> Vec A m  
        -> Vec A (n + m)
```

- Proof assistants
- For example: Agda, Coq, Lean, ...

## Research Question

How suitable is Statix for defining a dependently-typed language?

- Defining system  $F$  was a challenge<sup>1</sup>, will it even be possible?
- Will it be easier than doing it in Haskell?

---

<sup>1</sup>Hendrik van Antwerpen et al. Scopes as types.

# Why is this important?

## Spoofax perspective

Developing a language with a complex type system tests the boundaries of what Spoofax can do.

## Dependent Types perspective

Using a language workbench helps with rapid prototyping.

# Primary Contribution: Calculus of Constructions in Statix

A lambda calculus with dependent types.

## Example 1

```
(\v: Type. v) T
```

## Example 2

```
let f = \T: Type. \x: T. x;  
f (T: Type -> Type) (\y: Type. y)
```

# Type Checking

## Type checking relation

$\text{typeOfExpr} : \text{scope} * \text{Expr} \rightarrow \text{Expr}$

## How do we use scopes?

A scope is used as a replacement for an environment and a context..  
One relation  $\text{name} \rightarrow \text{NameEntry}$ ,  $\text{NameEntry}$  is either:

- $\text{NType}$ : Stores a type (Corresponds to a context)
- $\text{NSubst}$ : Stores a substitution (Corresponds to an environment)

# Type Checking: Requires Evaluation

## Example 1

```
let T = if false then Int else Bool end;  
let b: T = true;
```

## Evaluation relation

```
betaHeadReduce : scope * Expr -> scope * Expr  
betaReduce : scope * Expr -> Expr  
exactBetaEq : (scope * Expr) * (scope * Expr)
```

## Type Checking: From inference rules to Statix code

$$\frac{\langle s \mid e \rangle : t_e \quad \langle \text{sPutSubst}(s, x, (s, e)) \mid b \rangle : t_b}{\langle s \mid \text{Let}(x, e, b) \rangle : t_b}$$

### Equivalent Statix code

```
typeOfExpr (s, Let(x, e, b)) = bt :-  
  typeOfExpr (s, e) == et,  
  typeOfExpr (sPutSubst (s, x, (s, e)), b) == bt
```



# Extra contributions

## Features

- ① Implemented Inference
- ② Implemented Inductive Data Types
- ③ Implemented Universes
- ④ Interpreter
- ⑤ Compiler to Clojure

## Evaluation

- ① Comparison with implementation in Haskell
- ② Comparison with implementation in LambdaPi
- ③ Evaluation of Spoofax

# Conclusions

Spoofax is a great tool for developing dependently typed languages!

- We can use scopes to represent environments and contexts
- Statix can still use improvements