Dependently Typed Languages in Statix

Jonathan Brouwer Jesper Cockx Aron Zwaan

Delft University of Technology, The Netherlands

March 15, 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?

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

tododododododo

Calculus of Constructions

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

`typeOfExpr : scope * Expr -> Expr`

How do we use scopes?

A scope is used as a combination of an environment and a context.
One relation `name` → `NameEntry`, NameEntry is either:

- NType: Stores a type -> Corresponds with context
- NSubst: Stores a substitution -> Corresponds with 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
exectBetaEq : (scope * Expr) * (scope * Expr)
```

Type Checking: From inference rules to Statix code

tododododododod From inference rules to Statix code

Beta head-reduction rules $\boxed{\langle s_1 \mid e_1\rangle\ \overline{p} \underset{\beta h}{\Rightarrow} \langle s_2 \mid e_2\rangle}$

$$\frac{}{\langle s \mid \mathsf{Type}()\rangle\ [] \underset{\beta h}{\Rightarrow} \langle s \mid \mathsf{Type}()\rangle} \qquad \frac{\langle \mathsf{sPutSubst}(s, x, (s, e)) \mid b\rangle\ \overline{p} \underset{\beta h}{\Rightarrow} \langle s' \mid b'\rangle}{\langle s \mid \mathsf{Let}(x, e, b)\rangle\ \overline{p} \underset{\beta h}{\Rightarrow} \langle s' \mid b'\rangle}$$

$$\frac{\mathsf{sGetName}(s, x) = \mathsf{NSubst}(s_e, e) \qquad \langle s_e \mid e\rangle\ \overline{p} \underset{\beta h}{\Rightarrow} \langle s_{e'} \mid e'\rangle}{\langle s \mid \mathsf{Var}(x)\rangle\ \overline{p} \underset{\beta h}{\Rightarrow} \langle s_{e'} \mid e'\rangle}$$

$$\frac{\mathsf{sGetName}(s, x) = \mathsf{NType}(t)}{\langle s \mid \mathsf{Var}(x)\rangle\ \overline{p} \underset{\beta h}{\Rightarrow} \mathsf{rebuild}(s, \mathsf{Var}(x), \overline{p})} \qquad \frac{}{\langle s \mid \mathsf{FnType}(x, a, b)\rangle\ [] \underset{\beta h}{\Rightarrow} \langle s \mid \mathsf{FnType}(x, a, b)\rangle}$$

$$\frac{}{\langle s \mid \mathsf{FnConstruct}(x, a, b)\rangle\ [] \underset{\beta h}{\Rightarrow} \langle s \mid \mathsf{FnConstruct}(x, a, b)\rangle}$$

$$\frac{\langle \mathsf{sPutSubst}(s, x, p) \mid b\rangle\ \overline{p} \underset{\beta h}{\Rightarrow} \langle s' \mid e'\rangle}{\langle s \mid \mathsf{FnConstruct}(x, \_, b)\rangle\ (p :: \overline{p}) \underset{\beta h}{\Rightarrow} \langle s' \mid e'\rangle} \qquad \frac{\langle s \mid f\rangle\ (a :: \overline{p}) \underset{\beta h}{\Rightarrow} \langle s' \mid e'\rangle}{\langle s \mid \mathsf{FnDestruct}(f, a)\rangle\ \overline{p} \underset{\beta h}{\Rightarrow} \langle s' \mid e'\rangle}$$

Figure 4.2: Rules for beta head reducing the Calculus of Constructions

Type checking rules $\boxed{\langle s \mid e\rangle : t}$

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

$$\frac{\mathsf{sGetName}(s, x) = \mathsf{NType}(t)}{\langle s \mid \mathsf{Var}(x)\rangle : t} \qquad \frac{\mathsf{sGetName}(s, x) = \mathsf{NSubst}(s_e, e) \qquad \langle s_e \mid e\rangle : t}{\langle s \mid \mathsf{Var}(x)\rangle : t}$$

$$\frac{\langle s \mid a\rangle : t_a \qquad t_a \underset{\beta}{=} \mathsf{Type}() \qquad \langle s \mid a\rangle \underset{\beta}{\Rightarrow} a' \qquad \langle \mathsf{sPutType}(s, x, a') \mid b\rangle : t_b \qquad t_b \underset{\beta}{=} \mathsf{Type}()}{\langle s \mid \mathsf{FnType}(x, a, b)\rangle : \mathsf{Type}()}$$

$$\frac{\langle s \mid a\rangle : t_a \qquad t_a \underset{\beta}{=} \mathsf{Type}() \qquad \langle s \mid a\rangle \underset{\beta}{\Rightarrow} a' \qquad \langle \mathsf{sPutType}(s, x, a') \mid b\rangle : t_b}{\langle s \mid \mathsf{FnConstruct}(x, a, b)\rangle : \mathsf{FnType}(x, a', t_b)}$$

$$\langle s \mid f\rangle : t_f \qquad \langle s \mid t_f\rangle\ [] \underset{\beta h}{\Rightarrow} \langle s_f \mid \mathsf{FnType}(x, t_{da}, t_b)\rangle$$

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![1]
- using scopes as env+context todo beperkingen noemen - statix enforces syntactic uniqueness of metavariable solutions but dependently typed languages use a weaker notion of equality - full thesis coming soonTM

[1] But there is still room for improvement