DTU

Porpoise

contextual second-order abstract syntax in higher-order logic

Francisco Ferreira and Frederik Krogsdal Jacobsen

Higher order abstract syntax

Instead of writing:

$$\text{let } x = 1 + 2 \text{ in } x + 3$$

Write:

$$\text{let } (1 + 2)\ (\lambda y. y + 3)$$

Higher order abstract syntax

Instead of writing:

$$\text{let } x = 1 + 2 \text{ in } x + 3$$

Write:

$$\text{let } (1 + 2)\ (\lambda y.y + 3)$$

This is from the *meta*-logic

Higher order abstract syntax

Instead of writing:

$$\text{let } x = 1 + 2 \text{ in } x + 3$$

Write:

$$\text{let } (1 + 2)\ (\lambda y.y + 3)$$

This is from the *meta*-logic

Why?

- Alpha-equivalence by construction
- Type-preserving substitution “for free”

Contextual type theory

It is obvious that

$$\lambda x : \mathrm{nat} \to \mathrm{nat}.\lambda y : \mathrm{nat}.x\ y$$

is closed and well-typed with type $(\mathrm{nat} \to \mathrm{nat}) \to \mathrm{nat} \to \mathrm{nat}$.

But what about an incomplete term with a hole?

$$\lambda x : \mathrm{nat} \to \mathrm{nat}.\lambda y : \mathrm{nat}.\lfloor\ \rfloor$$

Contextual type theory allows us to characterize and instantiate holes

Contextual type theory

Contextual types internalize the typing judgment:

$$x : \mathrm{nat} \to \mathrm{nat}, y : \mathrm{nat} \vdash \lfloor\ \rfloor : \mathrm{nat}$$

The hole has the contextual type $\lceil x : \mathrm{nat} \to \mathrm{nat}, y : \mathrm{nat} \vdash \mathrm{nat} \rceil$

Contextual type theory

Contextual types internalize the typing judgment:

$$x : \mathrm{nat} \to \mathrm{nat}, y : \mathrm{nat} \vdash \lfloor\ \rfloor : \mathrm{nat}$$

The hole has the contextual type $\lceil x : \mathrm{nat} \to \mathrm{nat}, y : \mathrm{nat} \vdash \mathrm{nat} \rceil$

Advantages:

- Internalised support for incomplete terms when reasoning
- Substitutions become context-aware

Contextual modal type theory

- The contextual box modality says that a term is closed
- Behaves similar to S4
- The point is to separate syntactic and computational views on a term

With this, we essentially obtain the logic of the Beluga proof assistant
(if we add MLTT, we instead obtain the logic of the Orca proof assistant)

Expressivity is an issue

Type visible from meta-logic

$$\text{let } (a \wedge b)\ (\lambda x.\text{if } x \text{ then true else false})$$

... this is an exotic term

Other issues:

- Linearity is a problem because existing systems treat contexts structurally
- Relating to other theories is difficult because there are no libraries
- Encodings need to be very elaborate in some systems due to just having first-order reasoning logics
- To avoid exotic terms we need to restrict recursive functions and pattern matching

The syntactic framework SF

$$
\begin{array}{rrcl}
\text{Types} & A, B & ::= & \mathbf{a} \mid A \to B \mid \Box A \\
\text{Terms} & M, N & ::= & \mathbf{c}\,\vec{M} \mid \lambda x.\, M \mid \{M\} \mid x \\
\text{Substitutions} & \sigma & ::= & \cdot \mid \sigma, M \\
\text{Contexts} & \gamma & ::= & \cdot \mid \gamma, x
\end{array}
$$

- All terms are fully normalized by construction
- Babybel: embedding into OCaml following the approach of contextual modal type theory

Porpoise: SF with HOL term injection

$$\textsc{SpNil}\ \frac{}{\cdot : \gamma \vdash^{s} \boldsymbol{n} \ / \ \boldsymbol{n}} \qquad \textsc{SpCons}\ \frac{M : \gamma \vdash T \qquad \vec{s} : \gamma \vdash^{s} T' \ / \ \boldsymbol{n}}{M, \vec{s} : \gamma \vdash^{s} T \to T' \ / \ \boldsymbol{n}}$$

$$\textsc{TmLam}\ \frac{M : \gamma, (T, aux) \vdash T'}{\lambda x.\, M : \gamma \vdash T \to T' \ / \ \boldsymbol{n}} \qquad \textsc{TmBox}\ \frac{M : \cdot \vdash T}{\{M\} : \gamma \vdash T}$$

$$\textsc{TmVar}\ \frac{(T, aux) \in \gamma}{x : \gamma \vdash T} \qquad \textsc{TmC}\ \frac{sig(\boldsymbol{c}) = T \qquad \vec{s} : \gamma \vdash^{s} T \ / \ \boldsymbol{n}}{\boldsymbol{c}\ \vec{s} : \gamma \vdash \boldsymbol{n}}$$

- The type system forces all constructors to be fully applied

Work in progress!

- Are there classes of schemas and judgments where the substitution lemmas can be derived automatically?
- Classifying schemas is an open problem in general
- How nice can we make the experience of having to manually prove substitution lemmas?
- How easy is using other theories in practice? E.g. how annoying is it to work with real-valued semantics?