Dependent Type Theory with Contextual Types

Francisco Ferreira, David Thibodeau, and Brigitte Pientka

McGill University, Montréal, Québec, Canada
`fferre8@cs.mcgill.ca, david.thibodeau@mail.mcgill.ca, bpientka@cs.mcgill.ca`

We present a type theory with support for Higher Order Abstract Syntax (HOAS) by extending Martin-Löf style type theory [3] with contextual types and a specification framework based on the logical framework LF [2]. This system can be seen as the extension of the theory of Beluga [5] into full dependent types. The resulting system supports first class syntax and substitutions. This simplifies proofs by providing for free the substitution lemma and lemmas about the equational theory of substitutions. We can embed computations inside syntactic objects via closures. As in Beluga, we mediate between specifications and computations via contextual types. However, unlike Beluga, we can embed and use computations directly in contextual objects and types, hence we allow the arbitrary mixing of specifications and computations following ideas from [1]. Moreover, we take advantage of dependent types in several ways: we can reason about proofs and computation and we have support for polymorphism. As a consequence, our framework is suitable to encode complex logical relations proofs.

The syntax of calculus is presented in a pure type-systems with only one grammar as computations and specifications are interleaved arbitrarily. For clarity, we use the following naming convention for terms. We use E for expressions, S and T for types, Ψ and Φ for contexts, α and β for specification types, and σ and ϕ for substitutions.

$$
\begin{array}{lll}
\text{Terms} & & \\
E, S, T, \Psi, \Phi, \alpha, \beta & ::= \texttt{set}_n \mid (x : S) \to T \mid \lambda x.E \mid E\,E' \mid x \mid \mathbf{c} \mid \hat{\Psi}.E & \text{T.T. terms} \\
& \mid \;\; [\Psi \vdash \alpha] \mid \texttt{ctx} & \text{Cont. types} \\
& \mid \;\; \star \mid (x : \alpha) \twoheadrightarrow \beta \mid \widehat{\mathbf{c}} & \text{Spec. types} \\
& \mid \;\; \widehat{\lambda} x.E \mid E \cdot E' \mid E[\sigma] \mid \widehat{x} \mid ^\wedge \mid \texttt{id} \mid \sigma; x :=\, E \mid \emptyset \mid \Psi,\, \widehat{x} : E & \text{Spec. terms} \\
\text{Contexts} & \Gamma ::= \cdot \mid \Gamma, x : T & \\
\text{Signature} & \Sigma ::= \cdot \mid \Sigma, \mathbf{c} : T \mid \Sigma, \widehat{\mathbf{c}} : \alpha &
\end{array}
$$

Type theory terms are the terms of a standard Martin-Löf style type theory with an infinite hierarchy of universes. We embed specifications through contextual types that refer to terms of type α in an open context Ψ and to contexts. Types for specifications are a single universe $\star$, together with an intensional function space $(x : \alpha) \twoheadrightarrow \beta$ and constants.

Type theory terms and specification terms are typed with two different judgments $\Gamma \vdash E : T$ and $\Gamma; \Psi \vdash E : \beta$, respectively. For instance, we type extensional and intensional functions in the following way:

$$
\frac{\Gamma, x : S \vdash E : T}{\Gamma \vdash \lambda x.E : (x : S) \to T}\;\texttt{t-fun}
\qquad
\frac{\Gamma; \Psi, x : \alpha \vdash E : \beta}{\Gamma; \Psi \vdash \widehat{\lambda} x.E : (x : \alpha) \twoheadrightarrow \beta}\;\texttt{s-lam}
$$

The two function spaces differ by the contexts they act on. Type theory λ-abstractions introduce variables in the computational context Γ while specification λ-abstractions use the specification context Ψ. The coexisting calculi each have a specific β-reduction. The one for computational terms follows conventional literature:

$$
\frac{\Gamma, x : S \vdash E : T \quad \Gamma, x : S \vdash T : \texttt{set}_n \quad \Gamma \vdash E' : S}{\Gamma \vdash (\lambda x.E)\, E' \equiv [^{E'}/_{x}]E : [^{E'}/_{x}]T}\;\texttt{e-beta}
$$

Notice that the substitution is the traditional capture avoiding substitution implemented as a meta operation on syntax. In contrast, a term may also contain specification-level β-redeces. These redeces require a substitution operation on LF-bound variables which we write as $\{E/x\}$. The rule for specification level β-redeces reflects this fact:

$$\frac{\Gamma;\Psi,x:\alpha\vdash E:\beta \quad \Gamma;\Psi,x:\alpha\vdash\beta\ \texttt{is kind} \quad \Gamma;\Psi\vdash E':\alpha}{\Gamma;\Psi\vdash(\widehat{\lambda}x.E)\cdot E'\equiv\{E'/x\}E:\{E'/x\}\beta}\ \texttt{es-beta}$$

In addition to the description of the theory we present a prototype[1] for our type theory. Our prototype implements Agda-style dependent pattern matching[4] extended to allow matching on specifications. In particular, we allow matching directly on contexts and substitutions, and also on specification-level λ-abstractions and thus abstract over binders.

```
lf tm where
| lam: (tm ↠ tm) ↠ tm
...
lf oft : tm ↠ tp ↠ * where
| t-lam: (e: tm ↠ tm) (s t: tp) ↠ ((x:tm) ↠ oft x s ↠ oft (e ' x) t) ↠
    oft (lam e) (arr s t)
...
thm tps : (m n : (⊢ tm)) (t : (⊢ tp)) → (⊢ oft m t) → (⊢ step m n) →
          (⊢ oft n t) where ...
```

The definition of `tm` defines `lam` using the intensional function space to represent the variable binding. We also use this function space to represent parametric derivations as in the `t-lam` rule. We can thus write, for example, a proof of type preservation via a single function by pattern matching.

In conclusion, we developed a theory that allows for embedding contextual LF specifications into a fully dependently typed language that simplifies proofs about structures with syntactic binders (such as programming languages and logics). Moreover, we have a prototype, the Orca system, in which we implement some example proofs.

References

[1] Francisco Ferreira and Brigitte Pientka. Programs using syntax with first-class binders. In *To appear at ESOP '17*, 2017.

[2] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

[3] Per Martin-Löf. *Intuitionistic Type Theory*, volume 1 of *Studies in Proof Theory Lecture Notes*. Bibliopolis, Napoli, 1984.

[4] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, September 2007. Technical Report 33D.

[5] Brigitte Pientka and Andrew Cave. Inductive beluga: Programming proofs. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, pages 272–281, 2015.

[1] Available at: http://github.com/orca-lang/orca

A The typing rules

$\boxed{\Gamma \vdash E : T}$: E is of type T in context Γ

$$\frac{}{\Gamma \vdash \mathtt{set}_n : \mathtt{set}_{(n+1)}}\ \texttt{t-set} \qquad \frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ \texttt{t-var} \qquad \frac{\mathbf{c} : T \in \Sigma}{\Gamma \vdash \mathbf{c} : T}\ \texttt{t-con}$$

$$\frac{\Gamma \vdash S : \mathtt{set}_n \quad \Gamma \vdash T : \mathtt{set}_{(m+1)}}{\Gamma \vdash (x : S) \to T : \mathtt{set}_{(\max\ n\ (m+1))}}\ \texttt{t-pi-2} \qquad \frac{\Gamma \vdash S : \mathtt{set}_n \quad \Gamma \vdash T : \mathtt{set}_0}{\Gamma \vdash (x : S) \to T : \mathtt{set}_0}\ \texttt{t-pi-1}$$

$$\frac{\Gamma, x : S \vdash E : T}{\Gamma \vdash \lambda x.E : (x : S) \to T}\ \texttt{t-fun} \qquad \frac{\Gamma \vdash E : (x : S) \to T \quad \Gamma \vdash E' : S}{\Gamma \vdash E\,E' : [{}^{E'}/_{x}]T}\ \texttt{t-app}$$

$$\frac{\Gamma; \Psi \vdash E : \alpha}{\Gamma \vdash \hat{\Psi}.E : [\Psi \vdash \alpha]}\ \texttt{t-box} \qquad \frac{\Gamma \vdash E : S \quad \Gamma \vdash S \equiv T}{\Gamma \vdash E : T}\ \texttt{t-conv}$$

$$\frac{\Gamma \vdash \Psi : \mathtt{ctx} \quad \Gamma; \Psi \vdash \alpha : \star}{\Gamma \vdash [\Psi \vdash \alpha] : \mathtt{set}_0}\ \texttt{t-box-1} \qquad \frac{\Gamma \vdash \Psi : \mathtt{ctx} \quad \Gamma; \Psi \vdash \alpha \ \mathtt{is\ kind}}{\Gamma \vdash [\Psi \vdash \alpha] : \mathtt{set}_0}\ \texttt{t-box-2}$$

$$\frac{}{\Gamma \vdash \mathtt{ctx} : \mathtt{set}_0}\ \texttt{t-ctx}$$

$$\frac{}{\Gamma \vdash \emptyset : \mathtt{ctx}}\ \texttt{t-empty} \qquad \frac{\Gamma \vdash \Psi : \mathtt{ctx} \quad \Gamma; \Psi \vdash \alpha : \mathtt{ctx}}{\Gamma \vdash \Psi, \widehat{x} : \alpha : \mathtt{ctx}}\ \texttt{t-snoc}$$

$\boxed{\Gamma; \Psi \vdash \alpha \ \mathtt{is\ kind}}$: α is a syntactic type in ctx. Γ and spec. context Ψ.

$$\frac{}{\Gamma; \Psi \vdash \star \ \mathtt{is\ kind}}\ \texttt{s-kind} \qquad \frac{\Gamma; \Psi \vdash \alpha : \star \quad \Gamma; \Psi, x : \alpha \vdash \beta \ \mathtt{is\ kind}}{\Gamma; \Psi \vdash (x : \alpha) \twoheadrightarrow \beta \ \mathtt{is\ kind}}\ \texttt{s-pi-k}$$

$\boxed{\Gamma; \Psi \vdash E : \alpha}$: E is of syntactic type α in ctx. Γ and spec. context Ψ.

$$\frac{\Gamma; \Psi \vdash \alpha : \star \quad \Gamma; \Psi, x : \alpha \vdash \beta : \star}{\Gamma; \Psi \vdash (x : \alpha) \twoheadrightarrow \beta : \star}\ \texttt{s-pi} \qquad \frac{x : \alpha \in \Psi}{\Gamma; \Psi \vdash \widehat{x} : \alpha}\ \texttt{s-var} \qquad \frac{\widehat{\mathbf{c}} : \alpha \in \Sigma}{\Gamma; \Psi \vdash \widehat{\mathbf{c}} : \alpha}\ \texttt{s-con}$$

$$\frac{\Gamma; \Psi, x : \alpha \vdash E : \beta}{\Gamma; \Psi \vdash \widehat{\lambda} x.E : (x : \alpha) \twoheadrightarrow \beta}\ \texttt{s-lam} \qquad \frac{\Gamma; \Psi \vdash E : (x : \alpha) \twoheadrightarrow \beta \quad \Gamma; \Psi \vdash E' : \alpha}{\Gamma; \Psi \vdash E \cdot E' : \{E'/x\}\beta}\ \texttt{s-app}$$

$$\frac{\Gamma; \Psi \vdash \sigma : \Phi \quad \Gamma \vdash E : [\Phi \vdash \alpha]}{\Gamma; \Psi \vdash E[\sigma] : \{\sigma\}\alpha}\ \texttt{s-clo} \qquad \frac{}{\Gamma; \Psi \vdash \ \hat{}\ : \emptyset}\ \texttt{s-emptysub}$$

$$\frac{\Gamma \vdash \Phi : \mathtt{ctx}}{\Gamma; \Psi, \Phi \vdash \mathtt{id} : \Psi}\ \texttt{s-id} \qquad \frac{\Gamma; \Psi \vdash \sigma : \Phi \quad \Gamma; \Psi \vdash E : \{\sigma\}\alpha}{\Gamma; \Psi \vdash \sigma; x := E : (\Phi, x : \alpha)}\ \texttt{s-dot}$$

$$\frac{\Gamma; \Psi \vdash E : \beta \quad \Gamma; \Psi \vdash \alpha \equiv \beta}{\Gamma; \Psi \vdash E : \alpha}\ \texttt{t-conv}$$

B Equality

For brevity, we show only the rules that highlight computation:

$\boxed{\Gamma \vdash E \equiv E' : T}$: E is equal to E' at type T in context Γ.

$\boxed{\Gamma; \Psi \vdash E \equiv E' : \alpha}$: E is equal to E' at type α in contexts Γ and Ψ.

$\boxed{\Gamma; \Psi \vdash E \overset{kind}{\equiv} E'}$: Kinds E and E' are equal in contexts Γ and Ψ.

$$\frac{\Gamma, x : S \vdash E : T \quad \Gamma, x : S \vdash T : \mathtt{set}_n \quad \Gamma \vdash E' : S}{\Gamma \vdash (\lambda x.E)\, E' \equiv [{}^{E'}/_{x}]E : [{}^{E'}/_{x}]T}\ \texttt{e-beta}$$

$$\frac{\Gamma; \Psi, x : \alpha \vdash E : \beta \quad \Gamma; \Psi, x : \alpha \vdash \beta\ \mathtt{is\ kind} \quad \Gamma; \Psi \vdash E' : \alpha}{\Gamma; \Psi \vdash (\widehat{\lambda} x.E) \cdot E' \equiv \{E'/x\}E : \{E'/x\}\beta}\ \texttt{e-beta}$$

$$\frac{\Gamma; \Psi \vdash \sigma : \Phi \quad \Gamma; \Phi \vdash E : \alpha}{\Gamma; \Psi \vdash (\hat{\Phi}.E)[\sigma] \equiv \{\sigma\}E : \{\sigma\}\alpha}\ \texttt{e-clo}$$

$$\frac{\Gamma; \Psi \vdash E \equiv E' : \alpha}{\Gamma \vdash \hat{\Psi}.E \equiv \hat{\Psi}.E' : [\Psi \vdash \alpha]}\ \texttt{e-box-1}$$

$$\frac{\Gamma \vdash \Psi \equiv \Phi : \mathtt{ctx} \quad \Gamma; \Psi \vdash \alpha \equiv \beta : \star}{\Gamma \vdash [\Psi \vdash \alpha] \equiv [\Phi \vdash \beta] : \mathtt{set}_0}\ \texttt{e-box-2} \qquad \frac{\Gamma \vdash \Psi \equiv \Phi : \mathtt{ctx} \quad \Gamma; \Psi \vdash \alpha \overset{kind}{\equiv} \beta}{\Gamma \vdash [\Psi \vdash \alpha] \equiv [\Phi \vdash \beta] : \mathtt{set}_0}\ \texttt{e-box-k}$$

$$\frac{\Gamma \vdash E \equiv E' : \Phi[\alpha] \quad \Gamma; \Psi \vdash \sigma \equiv \sigma' : \Phi}{\Gamma; \Psi \vdash E[\sigma] \equiv E'[\sigma'] : \{\sigma\}\alpha}\ \texttt{e-clo}$$

$$\frac{\Gamma; \Psi \vdash \sigma \equiv \sigma' : \Phi \quad \Gamma; \Psi \vdash E \equiv E' : \{\sigma\}\alpha}{\Gamma; \Psi \vdash \sigma; x :=\ E \equiv \sigma'; x :=\ E' : \Phi, x : \alpha}\ \texttt{e-dot}$$