Andromedan type theory

Andrej Bauer
University of Ljubljana

Philipp Haselwarter
University of Ljubljana

Matija Pretnar
University of Ljubljana

Christopher A. Stone
Harvey Mudd College

December 4, 2018

Abstract

An outline of the type theory implemented by Andromeda.

1 The declarative formulation

In this section we give the formulation of type theory in a declarative way which minimizes the number of judgments, is better suited for a semantic account, but is not susceptible to an algorithmic treatment.

1.1 Syntax

Contexts:

$$\begin{array}{rcll} \Gamma & ::= & \bullet & \text{empty context} \\ & \mid & \Gamma, x : T & \text{context } \Gamma \text{ extended with } x : T \end{array}$$

Terms (e) and types (T, U):

$$\begin{array}{rcll} e, T, U & ::= & \mathsf{Type} & \text{universe} \\ & \mid & \prod_{(x:T)} U & \text{product} \\ & \mid & \mathsf{Eq}_T(e_1, e_2) & \text{equality type} \\ & \mid & x & \text{variable} \\ & \mid & \lambda x{:}T_1.T_2 \,.\, e & \lambda\text{-abstraction} \\ & \mid & e_1 \mathbin{@}^{x:T_1.T_2} e_2 & \text{application} \\ & \mid & \mathsf{refl}_T\ e & \text{reflexivity} \end{array}$$

Note that λ-abstraction and application are tagged with extra types not usually seen in type theory. An abstraction $\lambda x{:}T_1.T_2 \,.\, e$ speficies not only the type T_1 of x but also the type T_2 of e, where x is bound in T_2 and e. Similarly, an application $e_1 \mathbin{@}^{x:T_1.T_2} e_2$ specifies that e_1 and e_2 have types $\prod_{(x:T_1)} T_2$ and T_2, respectively. This is necessary because in the presence of exotic equalities (think "$\mathsf{nat} \to \mathsf{bool} \equiv \mathsf{nat} \to \mathsf{nat}$") we must be *very* careful about β-reductions.

The annotations on an application matter also for determining when two terms are equal. For example, if $X, Y : \mathsf{Type}$, $f : \mathsf{nat} \to X$ and

$$e : \mathsf{Eq}_{\mathsf{Type}}(\mathsf{nat}{\to}X, \mathsf{nat}{\to}Y),$$

then $(f \mathbin{@}^{\_:\mathsf{nat}.X}\ 0) : X$ and $(f \mathbin{@}^{\_:\mathsf{nat}.Y}\ 0) : Y$, so the two identical-but-for-annotations terms have different types and thus cannot be equivalent.

1.2 Judgments

$\Gamma\ \mathsf{ctx}$	Γ is a well formed context
$\Gamma \vdash e : T$	e is a well formed term of type T in context Γ
$\Gamma \vdash e_1 \equiv e_2 : T$	e_1 and e_2 are equal terms of type T in context Γ

The judgement "T is a type in context Γ" is a special case of term formation, namely $\Gamma \vdash T : \mathsf{Type}$. Similarly, equality of types is just equality of terms at Type.

1.3 Contexts

$$\frac{}{\bullet\ \mathsf{ctx}}\ \textsc{ctx-empty} \qquad \frac{\Gamma\ \mathsf{ctx} \qquad \Gamma \vdash T : \mathsf{Type}}{\Gamma,\, x{:}T\ \mathsf{ctx}}\ \textsc{ctx-extend}$$

1.4 Terms and types

General rules

$$\frac{\Gamma \vdash e : T \qquad \Gamma \vdash T \equiv U : \mathsf{Type}}{\Gamma \vdash e : U}\ \textsc{term-conv} \qquad \frac{\Gamma\ \mathsf{ctx} \qquad (x{:}T) \in \Gamma}{\Gamma \vdash x : T}\ \textsc{term-var}$$

Universe

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathsf{Type} : \mathsf{Type}}\ \textsc{ty-type}$$

Products

$$\frac{\Gamma \vdash T : \mathsf{Type} \qquad \Gamma,\, x{:}T \vdash U : \mathsf{Type}}{\Gamma \vdash \prod_{(x:T)} U : \mathsf{Type}}\ \textsc{ty-prod} \qquad \frac{\Gamma,\, x{:}T \vdash e : U}{\Gamma \vdash (\lambda x{:}T.U \,.\, e) : \prod_{(x:T)} U}\ \textsc{term-abs}$$

$$\frac{\Gamma \vdash e_1 : \prod_{(x:T)} U \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash e_1 \mathbin{@}^{x:T.U}\ e_2 : U[e_2/x]}\ \textsc{term-app}$$

Equality types

TY-EQ
$$\frac{\Gamma \vdash T : \mathsf{Type} \qquad \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{Eq}_T(e_1, e_2) : \mathsf{Type}}$$

TERM-REFL
$$\frac{\Gamma \vdash e : T}{\Gamma \vdash \mathsf{refl}_T\ e : \mathsf{Eq}_T(e, e)}$$

1.5 Equality

General rules

EQ-REFL
$$\frac{\Gamma \vdash e : T}{\Gamma \vdash e \equiv e : T}$$

EQ-SYM
$$\frac{\Gamma \vdash e_2 \equiv e_1 : T}{\Gamma \vdash e_1 \equiv e_2 : T}$$

EQ-TRANS
$$\frac{\Gamma \vdash e_1 \equiv e_2 : T \qquad \Gamma \vdash e_2 \equiv e_3 : T}{\Gamma \vdash e_1 \equiv e_3 : T}$$

EQ-CONV
$$\frac{\Gamma \vdash e_1 \equiv e_2 : T \qquad \Gamma \vdash T \equiv U : \mathsf{Type}}{\Gamma \vdash e_1 \equiv e_2 : U}$$

Equality reflection

EQ-REFLECTION
$$\frac{\Gamma \vdash e : \mathsf{Eq}_T(e_1, e_2)}{\Gamma \vdash e_1 \equiv e_2 : T}$$

Computations

PROD-BETA
$$\frac{\begin{array}{c}\Gamma \vdash T_1 \equiv U_1 : \mathsf{Type} \qquad \Gamma, x : T_1 \vdash T_2 \equiv U_2 : \mathsf{Type} \\ \Gamma, x : T_1 \vdash e_1 : T_2 \qquad \Gamma \vdash e_2 : U_1\end{array}}{\Gamma \vdash \big((\lambda x{:}T_1.T_2 \,.\, e_1) \mathbin{@}^{x:U_1.U_2} e_2\big) \equiv e_1[e_2/x] : T_2[e_2/x]}$$

Extensionality

EQ-ETA
$$\frac{\Gamma \vdash e_1' : \mathsf{Eq}_T(e_1, e_2) \qquad \Gamma \vdash e_2' : \mathsf{Eq}_T(e_1, e_2)}{\Gamma \vdash e_1' \equiv e_2' : \mathsf{Eq}_T(e_1, e_2)}$$

PROD-ETA
$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \prod_{(x:T)} U \qquad \Gamma \vdash e_2 : \prod_{(x:T)} U \\ \Gamma, x : T \vdash (e_1 \mathbin{@}^{x:T.U} x) \equiv (e_2 \mathbin{@}^{x:T.U} x) : U\end{array}}{\Gamma \vdash e_1 \equiv e_2 : \prod_{(x:T)} U}$$

1.5.1 Congruences

Type formers

$$\frac{\Gamma \vdash T_1 \equiv U_1 : \mathsf{Type} \qquad \Gamma,\, x : T_1 \vdash T_2 \equiv U_2 : \mathsf{Type}}{\Gamma \vdash \prod_{(x:T_1)} T_2 \equiv \prod_{(x:U_1)} U_2 : \mathsf{Type}}\ \text{CONG-PROD}$$

$$\frac{\Gamma \vdash T \equiv U : \mathsf{Type} \qquad \Gamma \vdash e_1 \equiv e_1' : T \qquad \Gamma \vdash e_2 \equiv e_2' : T}{\Gamma \vdash \mathsf{Eq}_T(e_1, e_2) \equiv \mathsf{Eq}_U(e_1', e_2') : \mathsf{Type}}\ \text{CONG-EQ}$$

Products

$$\frac{\Gamma \vdash T_1 \equiv U_1 : \mathsf{Type} \qquad \Gamma,\, x : T_1 \vdash T_2 \equiv U_2 : \mathsf{Type} \qquad \Gamma,\, x : T_1 \vdash e_1 \equiv e_2 : T_2}{\Gamma \vdash (\lambda x{:}T_1.T_2 \,.\, e_1) \equiv (\lambda x{:}U_1.U_2 \,.\, e_2) : \prod_{(x:T_1)} T_2}\ \text{CONG-ABS}$$

$$\frac{\begin{array}{c}\Gamma \vdash T_1 \equiv U_1 : \mathsf{Type} \qquad \Gamma,\, x : T_1 \vdash T_2 \equiv U_2 : \mathsf{Type} \\ \Gamma \vdash e_1 \equiv e_1' : \prod_{(x:T_1)} T_2 \qquad \Gamma \vdash e_2 \equiv e_2' : T_1\end{array}}{\Gamma \vdash (e_1 \mathbin{@^{x:T_1.T_2}} e_2) \equiv (e_1' \mathbin{@^{x:U_1.U_2}} e_2') : T_2[e_2/x]}\ \text{CONG-APP}$$

Equality types

$$\frac{\Gamma \vdash e_1 \equiv e_2 : T \qquad \Gamma \vdash T \equiv U : \mathsf{Type}}{\Gamma \vdash \mathsf{refl}_T\ e_1 \equiv \mathsf{refl}_U\ e_2 : \mathsf{Eq}_T(e_1, e_1)}\ \text{CONG-REFL}$$

2 Algorithmic version

Andromeda should be thought of as a programming language for deriving judgments. At the moment the language is untyped. We can hope to have it *simply typed* one day.

2.1 Syntax

Expressions:

$$\begin{array}{rll} E ::= & \mathsf{Type} & \text{universe} \\ \mid & x & \text{variable} \end{array}$$

Computations:

$$
\begin{array}{rlll}
C & ::= & \mathsf{return}\ E & \text{pure expressions} \\
 & | & \mathsf{let}\ x := C_1\ \mathsf{in}\ C_2 & \text{let binding} \\
 & | & C :: E & \text{ascription} \\
 & | & \prod_{(x:E)} C & \text{product} \\
 & | & \mathsf{Eq}(C, C) & \text{equality type} \\
 & | & \lambda x{:}E \,.\, C & \lambda\text{-abstraction} \\
 & | & E\, C & \text{application} \\
 & | & \mathsf{refl}\ C & \text{reflexivity}
\end{array}
$$

The result of a computation is a value, which is a pair (e, T) where e and T are terms of type theory, as described in Section 1.1. The correctness guarantee which we want is that a computation only ever evaluates to derivable judgments.

2.1.1 Operational semantics

Operational semantics is given by *two* versions of evaluation of computations, called *inference* and *checking*, of the forms:

$$
\begin{array}{rl}
\text{Inference:} & \Gamma;\, \eta \vdash C \Rightarrow (e, T) \\
\text{Checking:} & \Gamma;\, \eta \vdash C \Leftarrow T \mapsto e
\end{array}
$$

These are read as "in the given context Γ and environment η command C infers that e has type T" and "in the given context Γ and environment η command C checks that e has the given type T."

[EXPLAIN THAT AN ENVIRONMENT MAPS VARIABLES TO VALUES.]

$$
\frac{\Gamma;\, \eta \vdash C \Rightarrow (e, U) \qquad \Gamma;\, \eta \vdash T \approx U}{\Gamma;\, \eta \vdash C \Leftarrow T \mapsto e}\ \textsc{check-infer}
\qquad
\frac{}{\Gamma;\, \eta \vdash \mathsf{return}\ \mathsf{Type} \Rightarrow (\mathsf{Type}, \mathsf{Type})}\ \textsc{infer-type}
$$

$$
\frac{\Gamma;\, \eta \vdash C_1 \Leftarrow \mathsf{Type} \mapsto T_1 \qquad \Gamma,\, x : T_1;\, \eta \vdash C_2 \Leftarrow \mathsf{Type} \mapsto T_2}{\Gamma;\, \eta \vdash \prod_{(x:C_1)} C_2 \Rightarrow (\prod_{(x:T_1)} T_2, \mathsf{Type})}\ \textsc{infer-product}
$$

$$
\frac{\Gamma;\, \eta \vdash C_1 \Rightarrow (e_1, T) \qquad \Gamma;\, \eta \vdash C_2 \Leftarrow T \mapsto e_2}{\Gamma;\, \eta \vdash \mathsf{Eq}(C_1, C_2) \Rightarrow (\mathsf{Eq}_T(e_1, e_2), \mathsf{Type})}\ \textsc{infer-eq}
$$

$$
\frac{\Gamma;\, \eta \vdash E \Leftarrow \mathsf{Type} \mapsto T \qquad \Gamma;\, \eta \vdash C \Leftarrow T \mapsto e}{\Gamma;\, \eta \vdash C :: E \Rightarrow (e, T)}\ \textsc{infer-ascription}
$$

Inference of $\mathsf{Eq}(C_1, C_2)$ keeps the type of the first argument.

Ascription only has an infer rule, and will always switch to a checking phase. It breaks the inward information flow and has to use the check-infer when encountered during checking.

$$\textsc{infer-var}\quad \frac{\eta(x) = (e, T)}{\Gamma;\, \eta \vdash \mathsf{return}\ x \Rightarrow (e, T)}$$

$$\textsc{check-}\lambda\textsc{-tagged}\quad \frac{\begin{array}{c}\Gamma;\, \eta \vdash U \rightsquigarrow^* \prod_{(x:U_1)} U_2 \not\rightsquigarrow \\ \Gamma;\, \eta \vdash E \Leftarrow \mathsf{Type} \mapsto T_1 \qquad \Gamma;\, \eta \vdash T_1 \approx U_1 \qquad \Gamma,\, x : T_1;\, \eta \vdash C \Leftarrow U_2 \mapsto e\end{array}}{\Gamma;\, \eta \vdash \lambda x{:}E \,.\, C \Leftarrow U \mapsto \lambda x{:}U_1.U_2 \,.\, e}$$

$$\textsc{check-}\lambda\textsc{-untagged}\quad \frac{\Gamma;\, \eta \vdash U \rightsquigarrow^* \prod_{(x:U_1)} U_2 \not\rightsquigarrow \qquad \Gamma,\, x : U_1;\, \eta \vdash C \Leftarrow U_2 \mapsto e}{\Gamma;\, \eta \vdash \lambda x \,.\, C \Leftarrow U \mapsto \lambda x{:}U_1.U_2 \,.\, e}$$

$$\textsc{infer-}\lambda\quad \frac{\Gamma;\, \eta \vdash E \Leftarrow \mathsf{Type} \mapsto U_1 \qquad \Gamma,\, x : U_1;\, \eta(x) := (x, U_1) \vdash C \Rightarrow (e, U_2)}{\Gamma;\, \eta \vdash \lambda x{:}E \,.\, C \Rightarrow (\lambda x{:}U_1.U_2 \,.\, e, \prod_{(x:U_1)} U_2)}$$

$$\textsc{infer-app}\quad \frac{\Gamma;\, \eta \vdash E \Rightarrow (e_1, T) \qquad \Gamma;\, \eta \vdash T \rightsquigarrow^* \prod_{(x:U_1)} U_2 \not\rightsquigarrow \qquad \Gamma;\, \eta \vdash C \Leftarrow U_1 \mapsto e_2}{\Gamma;\, \eta \vdash E\, C \Rightarrow (e_1 \mathbin{@^{x:U_1.U_2}} e_2, U_2[e_2/x])}$$

$$\textsc{check-app-non-dep}\quad \frac{\Gamma;\, \eta \vdash C \Rightarrow (e_2, U) \qquad \Gamma;\, \eta \vdash E \Leftarrow \prod_{(\_:U)} T \mapsto e_1}{\Gamma;\, \eta \vdash E\, C \Leftarrow T \mapsto e_1 \mathbin{@^{\_:U.T}} e_2}$$

$$\textsc{infer-refl}\quad \frac{\Gamma;\, \eta \vdash C \Rightarrow (e, T)}{\Gamma;\, \eta \vdash \mathsf{refl}\ C \Rightarrow (\mathsf{refl}_T\ e, \mathsf{Eq}_T(e, e))}$$

$$\textsc{check-refl}\quad \frac{\begin{array}{c}\Gamma;\, \eta \vdash T \rightsquigarrow^* \mathsf{Eq}_U(e_1, e_2) \not\rightsquigarrow \\ \Gamma;\, \eta \vdash C \Leftarrow U \mapsto e \qquad \Gamma;\, \eta \vdash e \approx e_1 \Leftarrow U \qquad \Gamma;\, \eta \vdash e \approx e_2 \Leftarrow U\end{array}}{\Gamma;\, \eta \vdash \mathsf{refl}\ C \Leftarrow T \mapsto \mathsf{refl}_T\ e}$$

$$\textsc{check-let}\quad \frac{\Gamma;\, \eta \vdash C_1 \Rightarrow (e_1, U) \qquad \Gamma;\, \eta(x) := (e, U) \vdash C_2 \Leftarrow T \mapsto e_2}{\Gamma;\, \eta \vdash \mathsf{let}\ x := C_1\ \mathsf{in}\ C_2 \Leftarrow T \mapsto e_2}$$

$$\textsc{infer-let}\quad \frac{\Gamma;\, \eta \vdash C_1 \Rightarrow (e_1, U) \qquad \Gamma;\, \eta(x) := (e, U) \vdash C_2 \Rightarrow (e_2, T)}{\Gamma;\, \eta \vdash \mathsf{let}\ x := C_1\ \mathsf{in}\ C_2 \Rightarrow (e_2, T)}$$

TODO: check the freshness (and other side-conditions?).