Syntax and Typing for Cedille Core

Aaron Stump

December 28, 2018

1 Motivation

This note proposes a syntax for annotated terms and type-computation rules for them, for a core version of Cedille, to which full Cedille is easily translatable. Additionally, Cedille Core may be a good target for other higher-level extrinsic type theories.

The goal is to have a version of Cedille for which typing can be implemented straightforwardly, in a small, trustworthy checker. Cedille as we are implementing it for interactive development of proofs has a lot of features that are not needed in a lower-level format intended just for confirming typings. The checker for core Cedille does not need to produce span information for the benefit of an IDE, as our full implementation of Cedille does. Furthermore, it is acceptable for core Cedille to require more annotations on terms, to make type checking easier to implement.

There is one downside of the approach taken here: there will be notably more calls to check definitional equality of types, and hence checking time might be a bit slower. This is because when checking (for instance) an application of $f : A \to B$ to $a : A'$, we must check that A and A' are definitionally equal. In bidirectional checking, we synthesize the type $A \to B$ for f , and then check the argument a against A. The latter check will generally decompose A, but could be quite a bit cheaper than testing definitional equality of A with some other (possibly identical!) type.

2 Syntax

Figure 1 gives the syntax for terms of Cedille Core, in the style of pure type systems [1]: we have just one syntactic category of terms, and we rely on the type system to distinguish terms, types, and kinds (and a sole superkind $\Box$). The constructs are listed with some short comments to give a hint to the reader of what they mean; the typing rules below should clarify the meaning further. Also, the constructs are listed in the figure in order from highest precedence (most aggressively binding arguments) to lowest. In the typing rules below, we will also use this definition:

$$sorts\ \mathcal{S} := \{\star, \Box\}$$

We will use s as a metavariable ranging over $\mathcal{S}$.

2.1 Syntactic check for being a term

It is important that we only form equations between terms, because Cedille's semantics does not support forming equations except where both sides of the equation are terms. Terms can be syntactically distinguished from types and kinds as long as we use different syntactic categories for term variables, type variables, and

term variables u			
type variables X			
kind variables k			
variables x	$::=$	$u \mid X \mid k$	
pure terms p	$::=$	$u \mid p\ p' \mid \lambda\, u.\, p$	
annotated terms t	$::=$	x	use of a variable
		$\star$	kind for types
		$\Box$	sole superkind
		$t.1$	project first view of a dependent intersection
		$t.2$	project second view of a dependent intersection
		$\beta\ t'\{t\}$	proof of $t' \simeq t'$, where the proof erases to t
		$\delta\ T\ t$	proves anything if$t$ proves a certain false equation
		$\varsigma\ t$	symmetry of equality
		$t\ t'$	application of term to term
		$t\ \text{-}t'$	application of a term to an erased argument
		$\rho\ t\ @\ x.t'\ \text{-}\ t''$	equality elimination by type-guided rewriting
		$\forall\, x{:}t.\, t'$	implicit product (quantify over erased argument)
		$\Pi\, x{:}t.\, t'$	explicit product (usual Π-type)
		$\iota\, x{:}T.\, T'$	dependent intersection
		$\lambda\, x{:}t.\, t'$	usual λ-abstraction
		$\Lambda\, x{:}t.\, t'$	erased λ-abstraction
		$[t, t'\ @\ x.t'']$	introduce dependent intersection
		$\phi\ t\ \text{-}\ t'\ \{t''\}$	when t proves t' and t'' are equal, erase to t''
		$[x = t : t']\ \text{-}\ t''$	let x equal t of type t' in t''
		$\{p \simeq p'\}$	equality between pure terms

Figure 1: Sytax for Cedille Core

$$
\begin{array}{lcl}
|x| & = & x \\
|\star| & = & \star \\
|\Box| & = & \Box \\
|t.1| & = & |t| \\
|t.2| & = & |t| \\
|\beta\ t'\{t\}| & = & |t| \\
|\delta\ T\ t| & = & |t| \\
|\varsigma\ t| & = & |t| \\
|t\ t'| & = & |t|\ |t'| \\
|t\ \text{-}t'| & = & |t| \\
|\rho\ t\ @\ x.t'\ \text{-}\ t''| & = & |t''| \\
|\forall\, x : t.\, t'| & = & \forall\, x : |t|.\, |t'| \\
|\Pi\, x : t.\, t'| & = & \Pi\, x : |t|.\, |t'| \\
|\iota\, x : T.\, T'| & = & \iota\, x : |t|.\, |t'| \\
|\lambda\, u : t.\, t'| & = & \lambda\, u.\, |t'| \\
|\lambda\, X : t.\, t'| & = & \lambda\, X : |t|.\, |t'| \\
|\Lambda\, x : t.\, t'| & = & |t'| \\
|[t, t'\ @\ x.t'']| & = & |t| \\
|\phi\ t\ \text{-}\ t'\ \{t''\}| & = & |t''| \\
|[x = t : t']\ \text{-}\ t''| & = & (\lambda\, x.\, |t''|)\ |t| \\
|\{t \simeq t'\}|| & = & \{|t| \simeq |t'|\}
\end{array}
$$

Figure 2: Erasure for annotated terms

kind variables. That is why the syntax of Figure 1 distinguishes these. The typing rule below for forming equations requires the sides to be pure (i.e., unannotated) terms.

3 Erasure

When comparing terms for definitional equality, the rules below will compare erased terms, using the erasure function defined in Figure 2.

4 Typing

The type-checking algorithm for Cedille Core is presented as "almost" algorithmic typing rules (more on this shortly), in Figure 3. The rules are almost algorithmic because we must understand a few conventions for applying the rules:

- The rules are applied bottom-up, and a judgment $\Gamma \vdash t : t'$ represents a call to compute a type t' given a context Γ and a term t.
- Some of the premises of the rules state that a computed type should have a certain form (e.g., be a Π-type). It may happen, though, that the computed type β-reduces to a Π-type but is not literally one. To handle this case, one should head-normalize the computed types for premises that require a specific form.
- Where two premises use the same meta-variable, one must check definitional equality of the terms in question (for example, the domain-part of a Π-type and the type of an argument, in the application typing rule). The definitional equality relation, which we can denote $\Gamma \vdash t =_{\beta\eta} t'$, is the usual relation

Common

$$\frac{}{\Gamma, x : t \vdash x : t} \qquad \frac{\Gamma \vdash t : \Pi\, x{:}t_1.\, t_2 \quad \Gamma \vdash t : t_1}{\Gamma \vdash t\ t' : [t'/x]t_2} \qquad \frac{\Gamma \vdash t : \star \quad \Gamma, x : t \vdash t' : \star}{\Gamma \vdash \iota\, u{:}t.\, t' : \star}$$

Types and kinds

$$\frac{}{\Gamma \vdash \star : \Box} \qquad \frac{\Gamma \vdash t : s \quad \Gamma, x : t \vdash t' : s' \quad \mathit{Var}(x, s)}{\Gamma \vdash \Pi\, x{:}t.\, t' : s'} \qquad \frac{\Gamma \vdash t : s \quad \Gamma, x : t \vdash t' : \star \quad \mathit{Var}(x, s)}{\Gamma \vdash \forall\, x{:}t.\, t' : \star}$$

$$\frac{\Gamma \vdash t : \iota\, x{:}t_1.\, t_2}{\Gamma \vdash t.2 : [t/x]t_2} \qquad \frac{\Gamma, x : t \vdash t' : t'' \quad \Gamma \vdash \Pi\, x{:}t.\, t'' : s}{\Gamma \vdash \lambda\, x{:}t.\, t' : \Pi\, x{:}t.\, t''} \qquad \frac{\Gamma \vdash t : \{t_1 \simeq t_2\}}{\Gamma \vdash \varsigma\ t : \{t_2 \simeq t_1\}}$$

$$\frac{\Gamma \vdash t : \iota\, x{:}t_1.\, t_2}{\Gamma \vdash t.1 : t_1} \qquad \frac{\Gamma \vdash t : \forall\, x{:}t_1.\, t_2 \quad \Gamma \vdash t : t_1}{\Gamma \vdash t\ \text{-}t' : [t'/x]t_2} \qquad \frac{\Gamma \vdash t : \{t_1 \simeq t_2\} \quad \Gamma \vdash t'' : [t_2/x]t'}{\Gamma \vdash \rho\ t\ @\ x.t'\ \text{-}\ t'' : [t_1/x]t'}$$

$$\frac{\Gamma \vdash \{t' \simeq t'\} : \star \quad \mathit{FV}(t) \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash \beta\ t'\{t\} : \{t' \simeq t'\}} \qquad \frac{\mathit{FV}(p\ p') \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash \{p \simeq p'\} : \star} \qquad \frac{\Gamma \vdash t_1 : \Box \quad \Gamma, k = t_1 : t' \vdash t_2 : t''}{\Gamma \vdash [k = t_1 : \Box]\ \text{-}\ t_2 : t''}$$

$$\frac{\Gamma \vdash t : t_1 \quad \Gamma \vdash t' : [t/x]t_2 \quad \Gamma \vdash \iota\, x{:}t_1.\, t_2 : \star \quad |t| = |t'|}{\Gamma \vdash [t, t'\ @\ x.t_2] : \iota\, x{:}t_1.\, t_2} \qquad \frac{\Gamma \vdash t : \{t_1 \simeq t_2\} \quad \Gamma \vdash t_1 : t'}{\Gamma \vdash \phi\ t\ \text{-}\ t_1\ \{t_2\} : t'}$$

$$\frac{\Gamma \vdash t_1 : t' \quad \Gamma \vdash t' : s \quad \mathit{Var}(x, s) \quad \Gamma, x = t_1 : t' \vdash t_2 : t''}{\Gamma \vdash [x = t_1]\ \text{-}\ t_2 : t''}$$

$$\frac{\Gamma, x : t \vdash t' : t'' \quad x \notin \mathit{FV}(|t'|) \quad \Gamma \vdash \forall\, x{:}t.\, t'' : s}{\Gamma \vdash \Lambda\, x{:}t.\, t' : \forall\, x{:}t.\, t''} \qquad \frac{\Gamma \vdash t : \{\lambda\, x.\, \lambda\, y.\, x \simeq \lambda\, x.\, \lambda\, y.\, y\}}{\Gamma \vdash \delta\ T\ t : T}$$

Figure 3: Type-checking rules for Cedille Core

of $\beta\eta$-equivalence of terms in pure untyped lambda calculus, extended congruentially for the typing constructs $\forall$, Π, ι, and $\simeq$, and also extended to make use of let-definitions $x = t : t'$ contained in Γ (by replacing x with t when checking definitional equality). This same mechanism can be used for global definitions, as well.

- Note that the formation rules for $\forall$- and Π-abstractions use a premise $\mathit{Var}(x, s)$ to check that the variable x is a legal form of variable given the kind s. This judgment is defined by these rules:

$$\frac{}{\mathit{Var}(u, \star)} \qquad \frac{}{\mathit{Var}(X, \Box)}$$

Note that ρ rewrites from the right-hand side of an equation to the left-hand side, for consistency when synthesizing a type for a ρ-term in full Cedille.

References

[1] H. Barendregt. Lambda Calculi with Types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992.