Cedille2: A proof theoretic redesign of the Calculus of Dependent Lambda Eliminations

by

Andrew Marmaduke

A thesis submitted in partial fulfillment
of the requirements for the Doctor of Philosophy degree
in Computer Science
in the Graduate College
of The University of Iowa

May 2024

ACKNOWLEDGMENTS

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

PUBLIC ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Type theory is a tool for reasoning about assertions of some domain of discourse. When applied to programming languages, that domain is the expressible programs and their properties. Of course, a type theory may be rich enough to express detailed properties about a program, such that it halts or returns an even number. Therein lies a tension between what properties a type theory can faithfully (i.e. consistently) encode and the complexity of the type theory itself. If the theory is too complex then it may be untenable to prove that the type theory is well-behaved. Indeed, the design space of type theories is vast, likely infinite. When incorporating features the designer must balance complexity against capability.

Modern type theory arguably began with Martin-Löf in the 1970s and 1980s when he introduced a dependent type theory with the philosophical aspirations of being an alternative foundation of mathematics [29, 30]. Soon after in 1985, the Calculus of Constructions (CC) was introduced by Coquand [11, 12]. Inductive data (e.g. natural numbers, lists, trees) was shown by Guevers to be impossible to derive in CC [20]. Nevertheless, inductive data was added as an extension by Pfenning [36] and the Calculus of Inductive Constructions (CIC) became the basis for the proof assistant Rocq [34].

In the early 1990s Barendregt introduced a generalization to Pure Type Systems (PTS) and studied CC under his now famous λ-cube [5, 4]. The λ-cube demonstrated how CC could be deconstructed into four essential sorts of functions. At its base was the Simply Typed Lambda Calculus (STLC) a type theory introduced in the 1940s by Church to correct logical consistency issues in his (untyped) λ-calculus [8]. The STLC has only basic functions found in all programming languages. System F, a type theory introduced by Girard [22, 23] and independently by Reynolds [40], is obtained from STLC by adding quantification over types (i.e. polymorphic functions). Adding a copy of STLC at the type-layer, functions from types to types, yields System F^ω. Finally, the addition of quantification over terms or functions from terms to types, completes CC. While this is not the only path through the λ-cube to arrive at CC it is the most well-known and the most immediately relevant.

Perhaps surprisingly, all the systems of the λ-cube correspond to a logic. In the 1970s Curry circulated his observations about the STLC corresponding to intuitionistic propositional logic [24]. Reynolds and Girard's combined work demonstrated that System F corresponds to second-order intuitionistic propositional logic [22, 40, 41]. Indeed, Barendregt extended the correspondence to all systems in his λ-cube noting System F^ω as corresponding to higher-order intuitionistic propositional logic and CC as corresponding to higher-order intuitionistic predicate logic [4]. Fundamentally, the Curry-Howard correspondence associates programs of a type theory with proofs of a logic, and types with formula. However, the correspondence is not an isomorphism because the logical view does not possess a unique assignment of proofs. The type theory contains potentially *more* information than the proof derivation.

Cedille is a programming language with a core type theory based on CC [43, 44]. However, Cedille took an alternative road to obtaining inductive data than what was done in the 1980s. Instead, CC was modified to add the implicit products of Miquel [31], the dependent intersections of Kopylov [26], and an equality type over untyped terms. The initial goal of Cedille was to find an efficient way to encode inductive data. This was achieved in 2018 with Mendler-style lambda encodings [14]. However, the design of Cedille sacrificed certain properties such as the decidability of type checking. Decidability of type checking was stressed by Kreisel to Scott as necessary to reduce proof checking to type checking because a proof does not, under Kreisel's philosophy, diverge [42]. This puts into contention if Cedille corresponds to a logic at all. What remains is to describe the redesign of Cedille such that it does have decidability of type checking and to argue why this state of affairs is preferable. However, completing this journey requires a deeper introduction into the type theories of the λ-cube.

$$
\begin{aligned}
t &::= x \mid \mathfrak{b}(\kappa_1, x : t_1, t_2) \mid \mathfrak{c}(\kappa_2, t_1, \dots, t_{\mathfrak{a}(\kappa_2)}) \\
\kappa_1 &::= \lambda \mid \Pi \\
\kappa_2 &::= \star \mid \Box \mid \text{app}
\end{aligned}
$$

$$
\begin{aligned}
\mathfrak{a}(\star) = \mathfrak{a}(\Box) &= 0 \\
\mathfrak{a}(\text{app}) &= 2
\end{aligned}
\qquad
\begin{aligned}
\lambda\, x{:}t_1.\, t_2 &:= \mathfrak{b}(\lambda, x : t_1, t_2) \\
(x : t_1) \to t_2 &:= \mathfrak{b}(\Pi, x : t_1, t_2) \\
t_1\ t_2 &:= \mathfrak{c}(\text{app}, t_1, t_2)
\end{aligned}
\qquad
\begin{aligned}
\star &:= \mathfrak{c}(\star) \\
\Box &:= \mathfrak{c}(\Box)
\end{aligned}
$$

Figure 1.1: Syntax for System F$^\omega$.

1.1 System F$^\omega$

The following description of System F$^\omega$ differs from the standard presentation in a few important ways:

1. the syntax introduced is of a generic form which makes certain definitions more economical,

2. a bidirectional PTS style is used but weakening is replaced with a well-formed context relation.

These changes do not affect the set of proofs or formula that are derivable internally in the system.

Syntax consists of three forms: variables $(x, y, z, \dots)$, binders $(\mathfrak{b})$, and constructors $(\mathfrak{c})$. Every binder and constructor has an associated discriminate or tag to determine the specific syntactic form. Constructor tags have an associated arity $(\mathfrak{a})$ which determines the number of arguments, or subterms, the specific constructor contains. A particular syntactic expression will be interchangeably called a syntactic form, a term, or a subterm if it exists inside another term in context. See Figure 1.1 for the complete syntax of F$^\omega$. Note that the grammar for the syntax is defined using a BNF-style [15] where $t ::= f(t_1, t_2, \dots)$ represents a recursive definition defining a category of syntax, t, by its allowed subterms. For convenience a shorthand form is defined for each tag to maintain a more familiar appearance with standard syntactic definitions. Thus, instead of writing $\mathfrak{b}(\lambda, (x : A), t)$ the more common form is used: $\lambda\, x :$

$$\begin{aligned}FV(x) &= \{x\}\\ FV(\mathfrak{b}(\kappa_1, x : t_1, t_2)) &= FV(t_1) \cup (FV(t_2) - \{x\})\\ FV(\mathfrak{c}(\kappa_2, t_1, \dots, t_{\mathfrak{a}(\kappa_2)})) &= FV(t_1) \cup \dots \cup FV(t_{\mathfrak{a}(\kappa_2)})\end{aligned}$$

$$\begin{aligned}[y := t]x &= x\\ [y := t]y &= t\\ [y := t]\mathfrak{b}(\kappa_1, x : t_1, t_2) &= \mathfrak{b}(\kappa_1, x : [y := t]t_1, [y := t]t_2)\\ [y := t]\mathfrak{c}(\kappa_2, t_1, \dots, t_{\mathfrak{a}(\kappa_2)}) &= \mathfrak{c}(\kappa_2, [y := t]t_1, \dots, [y := t]t_{\mathfrak{a}(\kappa_2)})\end{aligned}$$

Figure 1.2: Operations on syntax for System F$^\omega$, including computing free variables and susbtitution.

$A.\, t$. Whenever the tag for a particular syntactic form is known the shorthand will always be used instead.

Free variables of syntax is defined by a straightforward recursion that collects variables that are not bound in a set. Likewise, substitution is recursively defined by searching through subterms and replacing the associated free variable with the desired term. See Figure 1.2 for the definitions of substitution and computing free variables. However, there are issues with variable renaming that must be solved. A syntactic form is renamed by consistently replacing bound and free variables such that there is no variable capture. For example, the syntax $\lambda\, x : A.\, y\ x$ cannot be renamed to $\lambda\, y : A.\, y\ y$ because it captures the free variable y with the binder λ. More critically, variable capture changes the meaning of a term. There are several rigorous ways to solve variable renaming including (non-exhaustively): De Bruijn indices (or levels) [13], locally-nameless representations [7], nominal sets [38], locally-nameless sets [39], etc. All techniques incorporate some method of representing syntax uniquely with respect to renaming. For this work the variable bureaucracy will be dispensed with. It will be assumed that renaming is implicitly applied whenever necessary to maintain the meaning of a term. For example, $\lambda\, x : A.\, y\ x = \lambda\, z : A.\, y\ z$ and the substitution $[x := t]\lambda\, x : A.\, y\ x$ unfolds to $\lambda\, x : [x := t]A.\, [z := t](y\ x)$.

$$\frac{t_1 \rightsquigarrow t_1'}{\mathfrak{b}(\kappa, x : t_1, t_2) \rightsquigarrow \mathfrak{b}(\kappa, x : t_1', t_2)} \qquad \frac{t_2 \rightsquigarrow t_2'}{\mathfrak{b}(\kappa, x : t_1, t_2) \rightsquigarrow \mathfrak{b}(\kappa, x : t_1, t_2')}$$

$$\frac{t_i \rightsquigarrow t_i' \qquad i \in 1, \dots, \mathfrak{a}(\kappa)}{\mathfrak{c}(\kappa, t_1, \dots t_i, \dots t_{\mathfrak{a}(\kappa)}) \rightsquigarrow \mathfrak{c}(\kappa, t_1, \dots t_i', \dots t_{\mathfrak{a}(\kappa)})}$$

$$(\lambda\, x{:}A.\, b)\; t \rightsquigarrow [x := t]b$$

Figure 1.3: Reduction rules for System F^ω.

The syntax of F^ω has a well understood notion of reduction (or dynamics, or computation) defined in Figure 1.3. This is an *inductive* definition of a two-argument relation on terms. A given rule of the definition is represented by a collection of premises ($P_1, \dots, P_n$) written above the horizontal line and a conclusion (C) written below the line. An optional name for the rule (EXAMPLE) appears to the right of the horizontal line. An inductive definition induces a structural induction principle allowing reasoning by cases on the rules and applying the induction hypothesis on the premises. During inductive proofs it is convenient to name the derivation of a premise ($\mathcal{D}_1, \dots, \mathcal{D}_n$). Moreover, to minimize clutter during proofs the name of the rule is removed.

$$\frac{P_1 \qquad \dots \qquad P_n}{C}\ \textsc{Example} \qquad \frac{\overset{\mathcal{D}_1}{P_1} \qquad \dots \qquad \overset{\mathcal{D}_n}{P_n}}{C}$$

Inductive definitions build a finite tree of rule applications concluding with axioms (or leafs). Axioms are written without premises and optionally include the horizontal line. The reduction relation for F^ω consists of three rules and one axiom. Relations defined in this manner are always the *least* relation that satisfies the definition. In other words, any related terms must have a corresponding inductive tree witnessing the relation.

The reduction relation (or step relation) models function application anywhere in a term via its axiom, called the β-rule. This relation is antisymmetric.

$$\frac{}{t\ R^*\ t}\ \textsc{Reflexive} \qquad\qquad \frac{t\ R\ t' \qquad t'\ R^*\ t''}{t\ R^*\ t''}\ \textsc{Transitive}$$

Figure 1.4: Reflexive-transitive closure of a relation R.

There is a *source* term s and a *target* term t, $s \rightsquigarrow t$, where t is the result of one function evaluation in s. Alternatively, $s \rightsquigarrow t$ is read as s *steps* to t. Note that if there is no λ-term applied to an argument (i.e. no function ready to be evaluated) for a given term t then that term cannot be the source term in the reduction relation. A term that cannot be a source is called a *value*. If there exists some sequence of terms related by reduction that end with a value, then all source terms in the sequence are *normalizing*. If *all* possible sequences of related terms end with a value for a particular source term s, then s is *strongly normalizing*. Restricting the set of terms to a normalizing subset is critical to achieve decidability of the reduction relation.

For any relation $-R-$, the reflexive-transitive closure $(-R^*-)$ is inductively defined with two rules as shown in Figure 1.4. In the case of the step relation the reflexive-transitive closure, $s \rightsquigarrow^* t$, is called the *multistep relation*. Additionally, when $s \rightsquigarrow^* t$ then s *multisteps* to t. It is easy to show that any reflexive-transitive closure is itself transitive.

Lemma 1.1. *Let R be a relation on a set A and let $a, b, c \in A$. If $a\ R^*\ b$ and $b\ R^*\ c$ then $a\ R^*\ c$*

Proof. By induction on $a\ R^*\ b$.

Case: $\dfrac{}{t\ R^*\ t}$

It must be the case the $a = b$.

Case: $\dfrac{\overset{\mathcal{D}_1}{t\ R\ t'} \qquad \overset{\mathcal{D}_2}{t'\ R^*\ t''}}{t\ R^*\ t''}$

Let $z = t'$, then we have $a\ R\ z$ and $z\ R^*\ b$. By the inductive hypothesis (IH) we have $z\ R^*\ c$ and by the transitive rule we have $a\ R^*\ c$ as desired.

□

Two terms are *convertible*, written $t_1 \equiv t_2$, if $\exists\ t'$ such that $t_1 \rightsquigarrow^* t'$ and $t_2 \rightsquigarrow^* t'$. Note that this is not the only way to define convertibility in a type theory, but it is the standard method for a PTS. Convertibility is used in the typing rules to allow syntax forms to have continued valid types as terms reduce. It may be tempting to view conversion as the reflexive-symmetric-transitive closure of the step relation, but transitivity is not an obvious property. In fact, proving transitivity of conversion is often a significant effort, beginning with the confluence lemma.

Lemma 1.2 (Confluence)**.** *If $s \rightsquigarrow^* t_1$ and $s \rightsquigarrow^* t_2$ then $\exists\ t'$ such that $t_1 \rightsquigarrow^* t'$ and $t_2 \rightsquigarrow^* t'$*

Proof. See Appendix A for a proof of confluence involving a larger reduction relation. Note that F^ω's step relation is a subset of this relation and thus is confluent. □

Theorem 1.3 (Transitivity of Conversion)**.** *If $a \equiv b$ and $b \equiv c$ then $a \equiv c$*

Proof. By premises we know $\exists u, v$ such that $a \rightsquigarrow^* u$, $b \rightsquigarrow^* u$, $b \rightsquigarrow^* v$, and $c \rightsquigarrow^* v$. By confluence, $\exists z$ such that $u \rightsquigarrow^* z$ and $v \rightsquigarrow^* z$. By transitivity of multistep reduction, $a \rightsquigarrow^* z$ and $c \rightsquigarrow^* z$. Therefore, $a \equiv c$. □

Figure 1.5 defines the typing relation on terms for F^ω. As previously mentioned this formulation is different from standard presentations. Four relations are defined mutually:

1. $\Gamma \vdash t \rhd T$, to be read as T is the inferred type of the term t in the context Γ or, t infers T in Γ;

2. $\Gamma \vdash t \blacktriangleright T$, to be read as T is the inferred type, possibly after some reduction, of the term t in the context Γ or, t reduction-infers T in Γ;

$$\frac{\Gamma \vdash t \rhd A \qquad A \rightsquigarrow^* B}{\Gamma \vdash t \blacktriangleright B}\ \textsc{RedInf} \qquad \frac{\begin{array}{c} B = \square \lor \Gamma \vdash B \blacktriangleright K \\ \Gamma \vdash t \rhd A \qquad A \equiv B \end{array}}{\Gamma \vdash t \lhd B}\ \textsc{Chk}$$

$$\frac{}{\vdash \varepsilon}\ \textsc{CtxEm} \qquad \frac{\begin{array}{c} x \notin FV(\Gamma) \\ \vdash \Gamma \qquad \Gamma \vdash A \blacktriangleright K \end{array}}{\vdash \Gamma, x : A}\ \textsc{CtxApp}$$

$$\frac{\vdash \Gamma}{\Gamma \vdash \star \rhd \square}\ \textsc{Axiom} \qquad \frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x \rhd A}\ \textsc{Var}$$

$$\frac{\Gamma \vdash A \blacktriangleright \square \qquad \Gamma, x : A \vdash B \blacktriangleright \square}{\Gamma \vdash (x : A) \to B \rhd \square}\ \textsc{Pi1} \qquad \frac{\Gamma \vdash A \blacktriangleright K \qquad \Gamma, x : A \vdash B \blacktriangleright \star}{\Gamma \vdash (x : A) \to B \rhd \star}\ \textsc{Pi2}$$

$$\frac{\begin{array}{c} \Gamma \vdash (x : A) \to B \blacktriangleright K \\ \Gamma, x : A \vdash t \rhd B \end{array}}{\Gamma \vdash \lambda\, x{:}A.\, t \rhd (x : A) \to B}\ \textsc{Lam} \qquad \frac{\begin{array}{c} \Gamma \vdash f \blacktriangleright (x : A) \to B \\ \Gamma \vdash a \lhd A \end{array}}{\Gamma \vdash f\ a \rhd [x := a]B}\ \textsc{App}$$

Figure 1.5: Typing rules for System F$^\omega$. The variable K is a metavariable representing either $\star$ or $\square$.

3. $\Gamma \vdash t \lhd T$, to be read as T is checked against the inferred type of the term t in the context Γ or, t checks against T in Γ;

4. $\vdash \Gamma$, to be read as the context Γ is well-formed, and thus consists only of types that themselves have a type

Note that there are two Pi rules that restrict the domain and codomain pairs of function types to three possibilities: $(\square, \square)$, $(\star, \star)$, and $(\square, \star)$. This is exactly what is required by the λ-cube for this definition to be F$^\omega$. For the unfamiliar reading these rules is arcane, thus exposition explaining a small selected set is provided.

$$\frac{\vdash \Gamma}{\Gamma \vdash \star \rhd \square}\ \textsc{Axiom}$$

The axiom rule has one premise, requiring that the context is well-formed. It concludes that the constant term $\star$ has type $\square$. Intuitively, the term $\star$ should be viewed as a universe of types, or a type of types, often referred to as a *kind*. Likewise, the term $\square$ should be viewed as a universe of kinds, or a kind of kinds. An alternative idea would be to change the conclusion to $\Gamma \vdash \star \rhd \star$. This is called the *type-in-type* rule, and it causes the type theory to be inconsistent [22, 25]. Note that there is no way to determine a type for $\square$. It plays the role of a type only.

$$\frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x \rhd A}\ \textsc{Var}$$

The variable rule is a context lookup. It scans the context to determine if the variable is anywhere in context and then the associated type is what that variable infers. This rule is what requires the typing relation to mention a context. Whenever a type is inferred or checked it is always desired that the context is well-formed. That is why the variable rule also requires the context to be well-formed as a premise, because it is a leaf relative to the inference relation. Without this additional premise there could be typed terms in ill-formed contexts.

$$\frac{\Gamma \vdash f \blacktriangleright (x : A) \to B \qquad \Gamma \vdash a \lhd A}{\Gamma \vdash f\ a \rhd [x := a]B}\ \textsc{App}$$

The application rule infers the type of the term f and reduces that type until it looks like a function-type. Once a function type is required it is clear that the type of the term a must match the function-type's argument-type. Thus, a is checked against the type A. Finally, the inferred result of the application is the codomain of the function-type B with the term a substituted for any free occurrences of x in B. This substitution is necessary because this application could be a type application to a type function. For example, let $f = \lambda\, X : \star.\, \text{id}\ X$ where id is the identity term. The inferred type of f is then $(X : \star) \to X \to X$. Let $a = \mathbb{N}$ (any type constant), then $f\ \mathbb{N} \rhd [X := \mathbb{N}](X \to X)$ or $f\ \mathbb{N} \rhd \mathbb{N} \to \mathbb{N}$.

While this presentation of F^ω is not standard Lennon-Bertrand demonstrated that it is equivalent to the standard formulation [27]. In fact, Lennon-Bertrand showed that a similar formulation is logically equivalent for the stronger CIC. Thus, standard metatheoretical results such as preservation and strong normalization still hold.

Lemma 1.4 (Preservation of F^ω). *If* $\Gamma \vdash s \lhd T$ *and* $s \rightsquigarrow^* t$ *then* $\Gamma \vdash t \lhd T$

Proof. See Appendix **??** for a proof of preservation of a conservative extension of F^ω, and thus a proof of preservation for F^ω itself. $\square$

Theorem 1.5 (Strong Normalization of F^ω)**.** *If $\Gamma \vdash t \rhd T$ then t and T are strongly normalizing*

Proof. System F^ω is a subsystem of CC which has several proofs of strong normalization. See (non-exhaustively) proofs using saturated sets [19], model theory [45], realizability [33], etc. $\square$

With strong normalization the convertibility relation is decidable, and moreover, type checking is decidable. Let *red* be a function that reduces its input until it is either $\star$, $\square$, a binder, or in normal form. Note that this function is defined easily by applying the outermost reduction and matching on the resulting term. Let *conv* test the convertibility of two terms. Note that this function may be defined by reducing both terms to normal forms and comparing them for syntactic identity. Both functions are well-defined because F^ω is strongly normalizing. Then the functions *infer*, *check*, and *wf* can be mutually defined by following the typing rules. Thus, type inference and type checking is decidable for F^ω.

While it is true that F^ω only has function types as primitives several other data types are internally derivable using function types. For example, the type of natural numbers is defined:

$$\mathbb{N} = (X : \star) \to X \to (X \to X) \to X$$

Likewise, pairs and sum types are defined:

$$A \times B = (X : \star) \to (A \to B \to X) \to X$$

$$A + B = (X : \star) \to ((A \to X) \times (B \to X)) \to X$$

The logical constants true and false are defined:

$$\top = (X : \star) \to X \to X$$

$$\bot = (X : \star) \to X$$

Negation is defined as implying false:

$$\neg A = A \to \bot$$

These definitions are called *Church encodings* and originate from Church's initial encodings of data in the λ-calculus [9, 10]. Note that if there existed a term such that $\vdash t \lhd \bot$ then trivially for *any* type T we have $\vdash t\,T \lhd T$. Thus, $\bot$ is both the constant false and the proposition representing the principle of explosion from logic. Moreover, this allows a concise statement of the consistency of F^ω.

Theorem 1.6 (Consistency of System F^ω)**.** *There is no term t such that $\vdash t \lhd \bot$*

Proof. Suppose $\vdash t \lhd \bot$. Let n be the value of t after it is normalized. By preservation $\vdash n \lhd \bot$. Deconstructing the checking judgment we know that $\vdash n \rhd T$ and $T \equiv \bot$, but $\bot$ is a value and values like n infer types that are also values. Thus, $T = \bot$ and we know that $\vdash n \rhd \bot$. By inversion on the typing rules $n = \lambda\, X : \star.\, b$, and we have $X : \star \vdash b \rhd X$. The term b can only be $\star$, $\square$, or X, but none of these options infer type X. Therefore, there does not exist a term b, nor a term n, nor a term t. $\square$

Recall that induction principles cannot be derived internally for any encoding of data [20]. This is not only cumbersome but unsatisfactory as the natural numbers are in their essence the least set satisfying induction. Ultimately, the issue is that these encodings are too general. They admit theoretical elements that F^ω is not flexible enough to express nor strong enough to exclude.

1.2 Calculus of Constructions and Cedille

As previously mentioned, CC is one extension away from F^ω on the λ-cube. Indeed, the two rules Pi1 and Pi2 can be merged to form CC:

$$\frac{\Gamma \vdash A \blacktriangleright K_1 \qquad \Gamma, x : A \vdash B \blacktriangleright K_2}{\Gamma \vdash (x : A) \to B \rhd K_2}\ \textsc{Pi}$$

where now both K_1 and K_2 are metavariables representing either $\star$ or $\square$. Note that no other rules, syntax, or reductions need to be changed. Replacing Pi1 and Pi2 with this new Pi rule is enough to obtain a complete and faithful definition of CC.

With this merger types are allowed to depend on terms. From a logical point of view, this is a quantification over terms in formula. Hence, why CC is a predicate logic instead of a propositional one according to the Curry-Howard correspondence. Yet, there is a question about what exactly quantification over terms means. Surely it does not mean quantification over syntactic forms.

It means, at minimum, quantification over well-typed terms, but from a logical perspective these terms correspond to proofs. In first order predicate logic the domain of quantification ranges over a set of *individuals*. The set of individuals represents any potential set of interest with specific individuals identified through predicates expressing their properties. With proofs the situation is different. A proof has meaning relative to its formula, but this meaning may not be relevant as an individual in predicate logic. For example, the proof 2 for a Church encoded natural number is intuitively data, but a proof that 2 is even is intuitively not. In CC, both are merely proofs that can be quantified over.

Cedille alters the domain of quantification from proofs to (untyped) λ-caluclus terms. Thus, for Cedille, the proof 2 becomes the encoding of 2 and the proof that 2 is even can *also* be the encoding of 2. This is achieved through a notion of *erasure* which removes type information and auxiliary syntactic forms from a term. Additionally, convertibility is modified to be convertibility of λ-calculus terms. However, erasure as it is defined in Cedille enables diverging terms in inconsistent contexts. The result by Abel and Coquand, which applies to a wide range of type theories including Cedille, is one way to construct a diverging term [1].

If terms are able to diverge, in what sense are they a proof? What a proof is or is not is difficult to say. As early as Aristotle there are documented forms of argument, Aristotle's syllogisms [3]. More than a millennium later Euclid's *Elements* is the most well-known example of a mathematical text containing what a modern audience would call proofs. Moreover, visual renditions of *Elements*, initiated by Byrne, challenge the notion of a proof being an algebraic object [6]. However, the study of proof as a mathematical object dates first to Frege [16] followed soon after by Peano's formalism of arithmetic [35] and Whitehead and Russell's *Principia Mathematica* [48]. For the kinds of logics discussed by the Curry-Howard correspondence, structural proof theories, the originator is Gentzen [17, 18]. Gentzen's natural deduction describes proofs as finite trees labelled by rules. Note that this is, of course, a very brief history of mathematical proof.

All of these formulations may be justified as acceptable notions of proof, but the purpose of proof from an epistemological perspective is to provide justification. It is unsatisfactory to have a claimed proof and be unable to check that it is constructed only by the rules of the proof theory. This is the situation with Cedille, although rare, there are terms where reduction diverges making it impossible to check a type. However, it is unfair to levy this criticism against Cedille alone, as well-known type theories also lack decidability of type checking. For example, Nuprl with its equality reflection rule [2], and the proof assistant Lean with its notion of casts [32]. Moreover, Lean has been incredibly successful in formalizing research mathematics including the Liquid Tensor Experiment [28] and Tao's formalization of The Polynomial Freiman-Ruzsa Conjecture [46]. Indeed, not having decidability of type checking does to necessarily prevent a tool from producing convincing arguments.

Ultimately, the definition of proof is a philosophical one with no absolute answer, but this work will follow Gentzen and Kreisel in requiring that a proof is a finite tree, labelled by rules, supporting decidable proof checking. The reader need only asks themselves which proof they would prefer if the option was available: one that potentially diverges, or one that definitely does not. If it is the latter, then striving for decidable type theories that are capable enough to reproduce the results obtained by proof assistants like Lean is a worthy goal.

1.3 Thesis

Cedille is a powerful type theory capable of deriving inductive data with relatively modest extension and modification to CC. However, this capability comes at the cost of decidability of type checking and thus, in the opinion of Kreisel, the cost of a Curry-Howard correspondence to a proof theory. A redesign of Cedille that focuses on maintaining a proof-theoretic view recovers decidability of type checking while still solving the original goals of Cedille. Although this redesign does prevent some constructions from being possible, the new balance struck between capability and complexity is desirable because of a well-behaved metatheory.

1.4 Contributions

Chapter 2 defines the Cedille2 Core (CC2) theory, including its syntax, and typing rules. Erasure from Cedille is rephrased as a projection from proofs to objects. Basic metatheoretical results are proven including: confluence, preservation, and classification.

Chapter 3 models CC2 in F^ω obtaining a strong normalization result for proof normalization. This model is a straightforward extension of a similar model for CC. Critically, proof normalization is not powerful enough to show consistency nor object normalization. Additionally, CC2 is shown to be a conservative extension of F^ω.

Chapter 4 models CC2 in CDLE obtaining consistency for CC2. Although CDLE is not strongly normalizing it still possess a realizability model which justifies its logical consistency. CC2 is closely related to CDLE which makes this models straightforward to accomplish. Moreover, a selection of axioms added to CC2 is shown to recover much of CDLEs features.

Chapter 5 proves object normalization from proof normalization and consistency. The φ, or cast, rule is the only difficulty after proof normalization and consistency. However, any proof can be translated into a new proof that contains no cast rules. Applying this observation yields an argument to obtain full object normalization.

Chapter 6 with normalization for both proofs and objects a well-founded type checker is defined. This implementation leverages normalization-by-evaluation and other basic techniques like pattern-based unification. The tool it benchmarked to demonstrate reasonable performance.

Chapter 7 contains derivations of generic inductive data, quotient types, large eliminations, constructor subtyping, and inductive-inductive data. All of these constructions are possible in Cedille but require modest modifications to derive in Cedille2.

Chapter 8 concludes with a collection of open conjectures and questions. Cedille2 at the conclusion of this work is still in its infancy.

Chapter 2

THEORY DESCRIPTION AND BASIC METATHEORY

This chapter describes the syntax, reduction, and type judgment of the core system for Cedille2. Near the conclusion, this chapter also proves basic metatheoretic properties such as a weakening lemma, substitution lemma, classification, and preservation. The presentation is a classical PTS-style with a single inference judgment. As it stands it is not obvious how this judgment admits an inference algorithm, but this situation will be remedied in Chapter 6. This style is chosen in particular because it prevents proofs of basic facts from becoming undesirably complex.

2.1 Syntax and Reduction

Syntax for the system is defined generically as before. See Figure 2.1 for a complete description. For the moment the new syntactic forms are merely raw data with no logically or even computational meaning. Nevertheless, a basic fact about substitution on syntax is provable.

Lemma 2.1. *If $x \neq y$ and $y \notin FV(a)$ then*

$$[x := a][y := b]t = [y := [x := a]b][x := a]t$$

Proof. By induction on t. If t is a binder or a constructor, then substitution unfolds and the IH applied to subterms concludes those cases. Suppose t is a variable, z. If $z = x$, then $z \neq y$ and $t = a$ on both sides because $y \notin FV(a)$. If $z = y$, then $z \neq x$ and $t = [x := a]b$ on both sides. If $z \neq x$ and $z \neq y$, then $t = z$ on both sides. $\square$

Computational meaning is added via reduction rules described in Figure 2.2. The new reductions model projection of pairs (e.g. $[t_1, t_2, t_3].1 \rightsquigarrow t_1$), promotion of equalities (e.g. $\vartheta_1(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$) and an elimination form for equality. Note that conversion is different from a traditional

$$
\begin{aligned}
t &::= x \mid \mathfrak{b}(\kappa_1, x : t_1, t_2) \mid \mathfrak{c}(\kappa_2, t_1, \ldots, t_{\mathfrak{a}(\kappa_2)})\\
\kappa_1 &::= \lambda_m \mid \Pi_m \mid \cap\\
\kappa_2 &::= \diamond \mid \star \mid \square \mid \bullet_m \mid \text{pair} \mid \text{proj}_1 \mid \text{proj}_2 \mid \text{eq} \mid \text{refl} \mid \psi \mid \vartheta_1 \mid \vartheta_2 \mid \delta \mid \varphi\\
m &::= \omega \mid 0 \mid \tau\\
&\mathfrak{a}(\diamond) = \mathfrak{a}(\star) = \mathfrak{a}(\square) = 0\\
&\mathfrak{a}(\text{proj}_1) = \mathfrak{a}(\text{proj}_2) = \mathfrak{a}(\text{refl}) = \mathfrak{a}(\delta) = 1\\
&\mathfrak{a}(\bullet_m) = \mathfrak{a}(\psi) = \mathfrak{a}(\varphi) = 2\\
&\mathfrak{a}(\text{pair}) = \mathfrak{a}(\text{eq}) = \mathfrak{a}(\vartheta_1) = \mathfrak{a}(\vartheta_2) = 3
\end{aligned}
$$

$$
\begin{aligned}
\diamond &:= \mathfrak{c}(\diamond)\\
\star &:= \mathfrak{c}(\star)\\
\square &:= \mathfrak{c}(\square)\\
\lambda_m\, x{:}t_1.\, t_2 &:= \mathfrak{b}(\lambda_m, x : t_1, t_2)\\
(x : t_1) \to_m t_2 &:= \mathfrak{b}(\Pi_m, x : t_1, t_2)\\
(x : t_1) \cap t_2 &:= \mathfrak{b}(\cap, x : t_1, t_2)\\
t_1 \bullet_m t_2 &:= \mathfrak{c}(\bullet_m, t_1, t_2)\\
\varphi(t_1, t_2) &:= \mathfrak{c}(\varphi, t_1, t_2)\\
\psi(t_1, t_2) &:= \mathfrak{c}(\psi, t_1, t_2)
\end{aligned}
\qquad
\begin{aligned}
[t_1, t_2; t_3] &:= \mathfrak{c}(\text{pair}, t_1, t_2, t_3)\\
t.1 &:= \mathfrak{c}(\text{proj}_1, t)\\
t.2 &:= \mathfrak{c}(\text{proj}_2, t)\\
t_1 =_{t_2} t_3 &:= \mathfrak{c}(\text{eq}, t_1, t_2, t_3)\\
\text{refl}(t) &:= \mathfrak{c}(\text{refl}, t)\\
\vartheta_1(t_1, t_2, t_3) &:= \mathfrak{c}(\vartheta_1, t_1, t_2, t_3)\\
\vartheta_2(t_1, t_2, t_3) &:= \mathfrak{c}(\vartheta_2, t_1, t_2, t_3)\\
\delta(t) &:= \mathfrak{c}(\delta, t)
\end{aligned}
$$

Figure 2.1: Generic syntax, there are three constructors, variables, a generic binder, and a generic non-binder. Each are parameterized with a constant tag to specialize to a particular syntactic consruct. The non-binder constructor has a vector of subterms determined by an arity function computed on tags. Standard syntactic constructors are defined in terms of the generic forms.

$$\frac{t_1 \rightsquigarrow t_1'}{\mathfrak{b}(\kappa, x : t_1, t_2) \rightsquigarrow \mathfrak{b}(\kappa, x : t_1', t_2)} \qquad \frac{t_2 \rightsquigarrow t_2'}{\mathfrak{b}(\kappa, x : t_1, t_2) \rightsquigarrow \mathfrak{b}(\kappa, x : t_1, t_2')}$$

$$\frac{t_i \rightsquigarrow t_i' \qquad i \in 1, \dots, \mathfrak{a}(\kappa)}{\mathfrak{c}(\kappa, t_1, \dots t_i, \dots t_{\mathfrak{a}(\kappa)}) \rightsquigarrow \mathfrak{c}(\kappa, t_1, \dots t_i', \dots t_{\mathfrak{a}(\kappa)})}$$

$$\begin{aligned}
(\lambda_m\, x : A; B.\, b) \bullet_m t &\rightsquigarrow [x := t]b \\
[t_1, t_2; t_3].1 &\rightsquigarrow t_1 \\
[t_1, t_2; t_3].2 &\rightsquigarrow t_2 \\
\psi(\text{refl}(a), P) &\rightsquigarrow \lambda_\omega\, x : P \bullet_\tau a \bullet_\tau \text{refl}(a).\, x \\
\vartheta_1(\text{refl}(t_1), t_2, t_3) &\rightsquigarrow \text{refl}(t_2) \\
\vartheta_2(\text{refl}(t_1), t_2, t_3) &\rightsquigarrow \text{refl}(t_2)
\end{aligned}$$

$$s_1 \rightleftharpoons s_2 \text{ iff } \exists\, t.\ s_1 \rightsquigarrow^* t \text{ and } s_2 \rightsquigarrow^* t$$

$$s_1 \equiv s_2 \text{ iff } \exists\, t_1, t_2.\ s_1 \rightsquigarrow^* t_1, s_2 \rightsquigarrow^* t_2, \text{ and } |t_1| \rightleftharpoons |t_2|$$

Figure 2.2: Reduction and conversion for arbitrary syntax.

PTS. Convertibility with respect to reduction is written: $t \rightleftharpoons s$. A detailed discussion of conversion is delayed until Section 2.2.

Before more important facts about reduction can be discussed it is important to observe the interaction between reduction and substitution. First, note that multistep reduction (i.e. the reflexive-transitive closure of the reduction relation) is congruent with respect to syntax. Second, substitution is shown to commute with multistep reduction through a series of lemmas.

Lemma 2.2. *If $t_i \rightsquigarrow^* t_i'$ for any i then,*

1. $\mathfrak{b}(\kappa, (x : t_1), t_2) \rightsquigarrow^* \mathfrak{b}(\kappa, (x : t_1'), t_2')$
2. $\mathfrak{c}(\kappa, t_1, \dots, t_{\mathfrak{a}(\kappa)}) \rightsquigarrow^* \mathfrak{c}(\kappa, t_1', \dots, t_{\mathfrak{a}(\kappa)}')$

Proof. Pick any i and apply the reductions to the associate subterm. A straightforward induction on $t_i \rightsquigarrow^* t_i'$ demonstrates that the reductions apply only to the associated subterm. Repeat until all i reductions are applied. ☐

Lemma 2.3. *If $a \rightsquigarrow b$ then $[x := t]a \rightsquigarrow [x := t]b$*

Proof. By induction on $a \rightsquigarrow b$.

Case: $(\lambda_m\, x : A.\, b) \bullet_m t \rightsquigarrow [x := t]b$

$[x := s]((\lambda_m\, y : A.\, b) \bullet_m t) = (\lambda_m\, x : [x := s]A.\, [x := s]b) \bullet_m [x := s]t \rightsquigarrow [y := [x := s]t][x := s]b = [x := s][y := t]b$

Note that the final equality holds by Lemma 2.1.

Case: $[t_1, t_2; A].1 \rightsquigarrow t_1$

$[x := t][t_1, t_2, A].1 = [[x := t]t_1, [x := t]t_2, [x :=]A].1 \rightsquigarrow [x := t]t_1$

Case: $[t_1, t_2; A].2 \rightsquigarrow t_2$

$[x := t][t_1, t_2, A].2 = [[x := t]t_1, [x := t]t_2, [x :=]A].2 \rightsquigarrow [x := t]t_2$

Case: $\psi(\text{refl}(t), P) \rightsquigarrow \lambda_\omega\, x : P \bullet_\tau t.\, x$

$[x := s]\psi(\text{refl}(\text{t}), P) = \psi(\text{refl}([\text{x} := \text{s}]\text{t}), [x := s]P) \rightsquigarrow \lambda_\omega\, y : [x := s]P \bullet_\tau [x := s]t.\, y = [x := s](\lambda_\omega\, y : P \bullet_\tau t.\, y)$

Case: $\vartheta_1(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$

$[x := s]\vartheta_1(\text{refl}(t_1), t_2, t_3) = \vartheta_1(\text{refl}(([x := s]t_1)), [x := s]t_2, [x := s]t_3) \rightsquigarrow \text{refl}([x := s]t_2) = [x := s]\text{refl}(t_2)$

Case: $\vartheta_2(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$

$[x := s]\vartheta_2(\text{refl}(t_1), t_2, t_3) = \vartheta_2(\text{refl}(([x := s]t_1)), [x := s]t_2, [x := s]t_3) \rightsquigarrow \text{refl}([x := s]t_2) = [x := s]\text{refl}(t_2)$

Case: $$\frac{\overset{\mathcal{D}_1}{t_i \rightsquigarrow t_i'} \qquad i \in 1, \dots, \mathfrak{a}(\kappa)}{\mathfrak{c}(\kappa, t_1, \dots t_i, \dots t_{\mathfrak{a}(\kappa)}) \rightsquigarrow \mathfrak{c}(\kappa, t_1, \dots t_i', \dots t_{\mathfrak{a}(\kappa)})}$$

By the IH, $[x := t]t_i \rightsquigarrow [x := t]t'_i$. Note that

$$[x := t]\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}}(\kappa)) = \mathfrak{c}(\kappa, [x := t]t_1, \ldots, [x := t]t_{\mathfrak{a}}(\kappa))$$

Applying the constructor reduction rule and reversing the previous equality concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \rightsquigarrow t'_1}}{\mathfrak{b}(\kappa, x : t_1, t_2) \rightsquigarrow \mathfrak{b}(\kappa, x : t'_1, t_2)}$$

By the IH, $[x := t]t_1 \rightsquigarrow [x := t]t'_1$. Note that

$$[x := t]\mathfrak{b}(\kappa, (y : t_1), t_2) = \mathfrak{b}(\kappa, (y : [x := t]t_1), [x := t]t_2)$$

Applying the first binder reduction rule and reversing the previous equality concludes the case.

□

Lemma 2.4. *If $a \rightsquigarrow^* b$ then $[x := t]a \rightsquigarrow^* [x := t]b$*

Proof. By induction on $a \rightsquigarrow^* b$. The reflexivity case is trivial.

Case: $$\frac{\overset{\mathcal{D}_1}{t\ R\ t'} \qquad \overset{\mathcal{D}_2}{t'\ R^*\ t''}}{t\ R^*\ t''}$$

Let $z = t'$. By the IH applied to $\mathcal{D}_2$: $[x := t]z \rightsquigarrow^* [x := t]b$. By Lemma 2.3 applied to $\mathcal{D}_1$: $[x := t]a \rightsquigarrow [x := t]b$. Applying the transitivity rule yields $[x := t]a \rightsquigarrow^* [x := t]b$.

□

Lemma 2.5. *If $s \rightsquigarrow t$ then $[x := s]a \rightsquigarrow^* [x := t]a$*

Proof. By induction on a.

Case: x

Rename y. Suppose $x = y$, then $[x := s]y = s \rightsquigarrow t = [x := t]y$. Thus, $[x := s]y \rightsquigarrow^* [x := t]y$. Suppose $x \neq y$, then $[x := s]y = y \rightsquigarrow^* y = [x := t]y$.

Case: $\mathfrak{b}(\kappa_1, x : t_1, t_2)$

By the IH $[x := s]t_1 \rightsquigarrow^* [x := t]t_1$ and $[x := s]t_2 \rightsquigarrow^* [x := t]t_2$. Lemma 2.2 concludes the case.

Case: $\mathfrak{c}(\kappa_2, t_1, \ldots, t_{\mathfrak{a}(\kappa_2)})$

By the IH $[x := s]t_i \rightsquigarrow^* [x := t]t_i$ for all i. Lemma 2.2 concludes the case.

$\square$

Lemma 2.6. *If $s \rightsquigarrow^* t$ and $a \rightsquigarrow^* b$ then $[x := s]a \rightsquigarrow^* [x := t]b$*

Proof. By induction on $s \rightsquigarrow^* t$. The reflexivity case is Lemma 2.4.

Case: $$\frac{\overset{\mathcal{D}_1}{t\ R\ t'} \qquad \overset{\mathcal{D}_2}{t'\ R^*\ t''}}{t\ R^*\ t''}$$

Let $z = t'$. By the IH applied to $\mathcal{D}_2$: $[x := z]a \rightsquigarrow^* [x := t]b$. Lemma 2.5 yields $[x := s]a \rightsquigarrow^* [x := z]a$. Transitivity concludes with $[x := s]a \rightsquigarrow^* [x := t]b$.

$\square$

Lemma 2.6 is the only fact about the interaction of substitution and reduction that is needed moving forward. A straightforward consequence is a similar lemma about substitution commuting with convertibility w.r.t. reduction.

Lemma 2.7. *If $s \rightleftharpoons t$ and $a \rightleftharpoons b$ then $[x := s]a \rightleftharpoons [x := t]b$*

Proof. By definition $\exists\ z_1, z_2$ such that $t \rightsquigarrow^* z_1$, $s \rightsquigarrow^* z_1$, $a \rightsquigarrow^* z_2$, and $b \rightsquigarrow^* z_2$. Applying Lemma 2.6 twice yields $[x := s]a \rightsquigarrow^* [x := z_1]z_2$ and $[x := t]b \rightsquigarrow^* [x := z_1]z_2$. $\square$

Transitivity, as before, is a consequence of confluence. Confluence is not an obvious property to obtain and can also be an involved property to prove. For example, a natural variant for the ϑ_1 reduction rule is $\vartheta_1(\text{refl}(t.1)) \rightsquigarrow \text{refl}(t)$, but this breaks confluence. To see why, consider $\vartheta_1(\text{refl}([x, y, z].1))$. One choice leads to $\vartheta_1(\text{refl}(x))$, and the other leads to $\text{refl}(x)$. However, these terms are not joinable, hence confluence fails. The full proof of confluence is relegated to Appendix A, but note that the approach closely follows the PLFA book [47].

Lemma 2.8 (Confluence). *If $s \rightsquigarrow^* t_1$ and $s \rightsquigarrow^* t_2$ then $\exists\, t'$ such that $t_1 \rightsquigarrow^* t'$ and $t_2 \rightsquigarrow^* t'$*

Proof. See Appendix A. □

As with F^ω the important consequence of confluence is that conversion w.r.t. reduction is an equivalence relation. However, this is *not* the conversion relation that will be used in typing judgments. Thus, while important, it is still only a stepping stone to showing judgmental conversion is transitive.

Lemma 2.9. *For any s and t the relation $s \rightleftharpoons t$ is an equivalence.*

Proof. Reflexivity is immediate because $s \rightsquigarrow^* s$. Symmetry is also immediate because if $s \rightleftharpoons t$ then $\exists\, z$ such that $s \rightsquigarrow^* z$ and $t \rightsquigarrow^* z$, but logical conjunction is commutative. Transitivity is a consequence of confluence, see Theorem 1.3. □

Additionally, there is a final useful fact about convertibility w.r.t. reduction that is occasionally used throughout the rest of this work. That is, like reduction, conversion w.r.t. reduction of subexpressions yields conversion of the entire term.

Lemma 2.10. *If $t_i \rightleftharpoons t_i'$ for any i then,*

1. $\mathfrak{b}(\kappa, (x : t_1), t_2) \rightleftharpoons \mathfrak{b}(\kappa, (x : t_1'), t_2')$
2. $\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}(\kappa)}) \rightleftharpoons \mathfrak{c}(\kappa, t_1', \ldots, t_{\mathfrak{a}(\kappa)}')$

Proof. By Lemma 2.2 applied on both sides. □

$$
\begin{aligned}
|x| &= x & |\diamond| &= \diamond \\
|\star| &= \star & |[t_1, t_2; t_3]| &= |t_1| \\
|\square| &= \square & |t.1| &= |t| \\
|\lambda_0\, x : A.\, t| &= |t| & |t.2| &= |t| \\
|\lambda_\omega\, x : A.\, t| &= \lambda_\omega\, x : \diamond.\, |t| & |x =_A y| &= |x| =_{|A|} |y| \\
|\lambda_\tau\, x : A.\, t| &= \lambda_\tau\, x : |A|.\, |t| & |\text{refl}(t)| &= \lambda_\omega\, x : \diamond.\, x \\
|(x : A) \to_m B| &= (x : |A|) \to_m |B| & |\psi(e, P)| &= |e| \\
|(x : A) \cap B| &= (x : |A|) \cap |B| & |\vartheta_1(e, t_1, t_2)| &= |e| \\
|f \bullet_0 a| &= |f| & |\vartheta_2(e, t_1, t_2)| &= |e| \\
|f \bullet_\omega a| &= |f| \bullet_\omega |a| & |\delta(e)| &= |e| \\
|f \bullet_\tau a| &= |f| \bullet_\tau |a| & |\varphi(f, e)| &= \lambda_\omega\, x : \diamond.\, x
\end{aligned}
$$

Figure 2.3: Erasure of syntax, for type-like and kind-like syntax erasure is homomorphic, for term-like syntax erasure reduces to the untyped lambda calculus.

2.2 Erasure and Pseudo-objects

Cedille has a notion of erasure of syntax that transforms terms into the untyped λ-calculus. This concept is generalized in the core theory of Cedille2 to operate on general syntax. It still called erasure mostly as a holdover, but erasure no longer actually erases all type information of type annotations. Instead, erasure should be thought of as computing the raw syntactic forms of objects. In Section 2.3 the notion of proof will be defined. An object is the erasure of a proof. Erasure is defined in Figure 2.3.

Note that the only purpose of the syntactic constructor $\diamond$ is to be a placeholder for erased type annotations of λ_m syntactic forms. However, for λ_τ variants, the annotation is *not* erased. This is partly why calling this transformation *erasure* is a slight lie, because it does not always erase. Regardless, it is faithful to the interpretation from Cedille when focused on non-type-like syntactic forms. Indeed, any form that is not type-like does reduce to the untyped λ-calculus. For type-like syntax, erasure is instead locally homomorphic. Erasure of raw syntax does not possess much structure, but it does commute with substitution. Additionally, as a consequence an extension of Lemma 2.7 is possible.

Lemma 2.11. $|[x := t]b| = [x := |t|]|b|$

Proof. By induction on the size of b.

Case: $\mathfrak{b}(\kappa, (x : t_1), t_2)$

If $b = \lambda_0\, y\!:\!A.\, b'$, then $|b| = |b'|$ which is a smaller term. Then, by the IH $|[x := t]b'| = [x := |t|]|b'|$. Thus,

$$\begin{aligned}&|[x := t]\lambda_0\, y\!:\!A.\, b'| = |\lambda_0\, y\!:\![x := t]A.\, [x := t]b'|\\ &= |[x := t]b'| = [x := |t|]|b'| = [x := |t|]|\lambda_0\, y\!:\!A.\, b'|\end{aligned}$$

For the remaining tags, assume w.l.o.g. $\kappa = \cap$. Then $b = (y : A) \cap B$, and by the IH $|[x := t]A| = [x := |t|]|A|$ and $|[x := t]B| = [x := |t|]|B|$. Thus,

$$\begin{aligned}&|[x := t]((y : A) \cap B)| = |(y : [x := t]A) \cap [x := t]B|\\ &= (y : |[x := t]A|) \cap |[x := t]B| = (y : [x := |t|]|A|) \cap [x := |t|]|B|\end{aligned}$$

And, $[x := |t|]|(y : A) \cap B| = (y : [x := |t|]|A|) \cap [x := |t|]|B|$. Thus, both sides are equal.

Case: $\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}(\kappa)})$

If $\kappa \in \{\star, \square\}$ then the equality is trivial.

If $\kappa \in \{\bullet_0, \mathrm{pair}, \mathrm{proj}_1, \mathrm{proj}_2, \psi, \vartheta, \delta\}$ then $|\mathfrak{c}(\kappa, t_1, \ldots)| = |t_1|$. Moreover, substitution commutes and both sides of the equality are equal.

If $\kappa \in \{\mathrm{refl}, \varphi\}$ then the equality is trivial.

If $\kappa \in \{\bullet_\omega, \bullet_\tau, \mathrm{eq}\}$ then w.l.o.g. assume $\kappa = \mathrm{eq}$. Now $|[x := t](a =_A b)| = |[x := t]a| =_{|[x:=t]A|} |[x := t]b|$. By the IH this becomes $[x := |t|]|a| =_{[x:=|t|]|A|} [x := |t|]|b|$. On the right-hand side, $[x := |t|]|a =_A b| = [x := |t|]|a| =_{[x:=|t|]|A|} [x := |t|]|b|$. Thus, both sides are equal.

Case: b variable

Suppose $b = x$, then $|[x := t]x| = |t|$ and $[x := |t|]|x| = |t|$. Suppose $b = y$, then $|[x := t]y| = y$ and $[x := |t|]|y| = y$. Thus, both sides are equal.

$\square$

Lemma 2.12. *If* $|s| \rightleftharpoons |t|$ *and* $|a| \rightleftharpoons |b|$ *then* $|[x := s]a| \rightleftharpoons |[x := t]b|$

Proof. By definition $\exists\ z_1, z_2$ such that $|s| \rightsquigarrow^* z_1$, $|t| \rightsquigarrow^* z_1$, $|a| \rightsquigarrow^* z_2$ and $|b| \rightsquigarrow^* z_2$. By Lemma 2.6 applied twice $[x := |s|]|a| \rightsquigarrow^* [x := |z_1|]z_2$ and $[x := |t|]|b| \rightsquigarrow^* [x := |z_1|]z_2$. Finally, by Lemma 2.11 $[x := |s|]|a| = |[x := s]a|$ and $[x := |t|]|b| = |[x := t]b|$. $\square$

Beyond these lemmas more structure needs to be imposed on raw syntax to obtain better behavior with erasure. In particular, the pair case and the λ_0 case are problematic. Indeed, for pairs there is an assumption that the first and second component are convertible. This restriction is what transforms these pairs into something more, an element of an intersection. Likewise, the λ_0 binder is meant to signify that the bound variable does not appear free in the erasure of the body. Imposing these restrictions on syntax retains the spirit of what it means to be an object. However, because syntax is still not a proof, this restriction on syntax instead forms a set of *pseudo-objects*. The inductive definition of pseudo-objects is presented in Figure 2.4.

Note that the restriction for pairs is $|t_1| \rightleftharpoons |t_2|$ as opposed to $t_1 \equiv t_2$. The distinction here is subtle, but it enables proving one of the important properties for the structure of pseudo-objects, that $|t_1| \rightleftharpoons |t_2|$ if and only if $t_1 \equiv t_2$. To reach that goal requires a series of technical lemmas about pseudo-objects and the concepts introduced so far.

Lemma 2.13. *If* s pseobj *and* $s \rightsquigarrow t$ *then* $|s| \rightleftharpoons |t|$

Proof. By induction on s pseobj.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t_2 \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{\kappa \neq \lambda_0}}{\mathfrak{b}(\kappa, x : t_1, t_2) \text{ pseobj}}$$

$$\frac{t_1 \text{ pseobj} \qquad t_2 \text{ pseobj} \qquad \kappa \neq \lambda_0}{\mathfrak{b}(\kappa, x : t_1, t_2) \text{ pseobj}} \qquad \frac{\begin{array}{c}\forall\ i \in 1, \ldots, \mathfrak{a}(\kappa).\ t_i \text{ pseobj} \\ \kappa \neq \text{pair}\end{array}}{\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}(\kappa)}) \text{ pseobj}}$$

$$\frac{\begin{array}{c}A \text{ pseobj} \\ t \text{ pseobj} \qquad x \notin FV(|t|)\end{array}}{\lambda_0\, x : A.\, t \text{ pseobj}} \qquad \frac{\begin{array}{cc}t_1 \text{ pseobj} & t_2 \text{ pseobj} \\ t_3 \text{ pseobj} & |t_1| \rightleftharpoons |t_2|\end{array}}{[t_1, t_2; t_3] \text{ pseobj}}$$

$$x \text{ pseobj}$$

Figure 2.4: Definition of Pseudo Objects.

By cases on $s \rightsquigarrow t$, applying the IH and Corollary **??**.

Case: $$\frac{\overset{\mathcal{D}_1}{A \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{x \notin FV(|t|)}}{\lambda_0\, x : A.\, t \text{ pseobj}}$$

By cases on $s \rightsquigarrow t$, applying the IH and Corollary **??**.

Case: $$\frac{\overset{\mathcal{D}_1}{\forall\ i \in 1, \ldots, \mathfrak{a}(\kappa).\ t_i \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{\kappa \neq \text{pair}}}{\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}(\kappa)}) \text{ pseobj}}$$

By cases on $s \rightsquigarrow t$.

Case: $(\lambda_m\, x : A.\, b) \bullet_m t \rightsquigarrow [x := t]b$

Note that $\lambda_m\, x : A.\, b$ pseobj. If $m = 0$ then $x \notin FV(b)$ and $|[x := t]b| = |b|$. Thus, $|(\lambda_0\, x : A.\, b) \bullet_0 t| = |\lambda_0\, x : A.\, b| = |b|$. If $m = \omega$, then $|(\lambda_\omega\, x : A.\, b) \bullet_\omega t| = (\lambda_\omega\, x.\, b) \bullet_\omega |t|$. By definition of reduction $(\lambda_\omega\, x.\, b) \bullet_\omega |t| \rightleftharpoons [x := |t|]|b|$. Finally, by Lemma 2.11 the goal is obtained. The case of $m = \tau$ is almost exactly the same.

Case: $[t_1, t_2; A].1 \rightsquigarrow t_1$

$|[t_1, t_2; A].1| = |[t_1, t_2; A]| = |t_1|$

Case: $[t_1, t_2; A].2 \rightsquigarrow t_2$

Observe that $|[t_1, t_2; A].2| = |t_1|$ and $[t_1, t_2; A]$ pseobj. Thus, $|s| = |t_1| \rightleftharpoons |t_2|$.

Case: $\psi(\text{refl}(t), P) \rightsquigarrow \lambda_\omega\, x : P \bullet_\tau t.\, x$

$$|\psi(\text{refl}(t), P)| = |\text{refl}(t)| = \lambda_\omega\, x.\, x = |\lambda_\omega\, x : P \bullet_\tau t.\, x|$$

Case: $\vartheta_1(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$

$$|\vartheta_1(\text{refl}(t_1), t_2, t_3)| = |\text{refl}(t_1)| = \lambda_\omega\, x.\, x = |\text{refl}(t_2)|$$

Case: $\vartheta_2(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$

Same as previous case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_i \rightsquigarrow t_i'} \qquad i \in 1, \ldots, \mathfrak{a}(\kappa)}{\mathfrak{c}(\kappa, t_1, \ldots t_i, \ldots t_{\mathfrak{a}(\kappa)}) \rightsquigarrow \mathfrak{c}(\kappa, t_1, \ldots t_i', \ldots t_{\mathfrak{a}(\kappa)})}$$

By the IH, $|t_i| \rightleftharpoons |t_i'|$. The goal is achieved by Corollary **??**

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t_2 \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{t_3 \text{ pseobj}} \qquad \overset{\mathcal{D}_4}{|t_1| \rightleftharpoons |t_2|}}{[t_1, t_2; t_3] \text{ pseobj}}$$

By cases on $s \rightsquigarrow t$, applying the IH and Corollary **??**.

Case: s variable

By cases on $s \rightsquigarrow t$, t must be a variable. Thus, $|s| = |t|$.

□

Lemma 2.14. *If* s pseobj, $|s| \rightleftharpoons |b|$, *and* $s \rightsquigarrow t$ *then* $|t| \rightleftharpoons |b|$

Proof. By Lemma 2.13 $|s| \rightleftharpoons |t|$ and by Lemma 2.9 $|t| \rightleftharpoons |b|$. □

Lemma 2.15. *If* b pseobj *and* t pseobj *then* $[x := t]b$ pseobj

Proof. By induction on b pseobj. The λ_0 and pair cases are no different from the respective $\mathfrak{b}$ and $\mathfrak{c}$ cases.

Case: $$\dfrac{\overset{\mathcal{D}_1}{t_1 \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t_2 \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{\kappa \neq \lambda_0}}{\mathfrak{b}(\kappa, x : t_1, t_2) \text{ pseobj}}$$

By the IH $[x := t]t_1$ pseobj and $[x := t]t_2$ pseobj. Thus, $\mathfrak{b}(\kappa, (y : [x := t]t_1), [x := t]t_2)$ pseobj.

Case: $$\dfrac{\overset{\mathcal{D}_1}{\forall\ i \in 1, \ldots, \mathfrak{a}(\kappa).\ t_i \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{\kappa \neq \text{pair}}}{\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}(\kappa)}) \text{ pseobj}}$$

By the IH $[x := t]t_i$ pseobj.
Thus, $\mathfrak{c}(\kappa, [x := t]t_1, \ldots [x := t]t_{\mathfrak{a}(\kappa)})$ pseobj.

Case: s variable

If $s = x$ then $[x := t]x = t$, and t pseobj. Otherwise, $s = y$ with y a variable and y pseobj.

□

Lemma 2.16. *If s pseobj and $s \leadsto t$ then t pseobj*

Proof. By induction on s pseobj.

Case: $$\dfrac{\overset{\mathcal{D}_1}{t_1 \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t_2 \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{\kappa \neq \lambda_0}}{\mathfrak{b}(\kappa, x : t_1, t_2) \text{ pseobj}}$$

By cases on $s \leadsto t$. Suppose w.l.o.g. that $t_2 \leadsto t_2'$. Observe that t_2 pseobj because it is a subterm of s. Then by the IH t_2' pseobj. Thus, $\mathfrak{b}(\kappa, x : t_1, t_2')$ pseobj.

Case: $$\dfrac{\overset{\mathcal{D}_1}{A \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{x \notin FV(|t|)}}{\lambda_0\, x\!:\!A.\, t \text{ pseobj}}$$

By cases on $s \leadsto t$. Suppose w.l.o.g that $t \leadsto t'$. Note that if $x \notin FV(|t|)$ then $x \notin FV(|t'|)$, reduction only reduces the amount of free variables. Observe that t pseobj. Then by the IH t' pseobj. Thus, $\lambda_0\, x : A.\, t'$ pseobj.

Case: $$\dfrac{\overset{\mathcal{D}_1}{\forall\ i \in 1, \ldots, \mathfrak{a}(\kappa).\ t_i \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{\kappa \neq \text{pair}}}{\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}(\kappa)}) \text{ pseobj}}$$

By cases on $s \rightsquigarrow t$.

Case: $(\lambda_m\, x : A.\, b) \bullet_m t \rightsquigarrow [x := t]b$

Observe that b pseobj and t pseobj because both are subterms of s. By Lemma 2.15 $[x := t]b$ pseobj.

Case: $[t_1, t_2; A].1 \rightsquigarrow t_1$

Observe that t_1 pseobj because it is a subterm of s.

Case: $[t_1, t_2; A].2 \rightsquigarrow t_2$

Observe that t_2 pseobj.

Case: $\psi(\text{refl}(t), P) \rightsquigarrow \lambda_\omega\, x : P \bullet_\tau t.\, x$

Observe that t pseobj and P pseobj. By application of constructor and binder rules $\lambda_\omega\, x : P \bullet_\tau t.\, x$ pseobj.

Case: $\vartheta_1(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$

Observe that t_2 pseobj. By application of constructor rule $\text{refl}(t_2)$ pseobj.

Case: $\vartheta_2(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$

Same as previous case.

Case: $$\dfrac{\overset{\mathcal{D}_1}{t_i \rightsquigarrow t_i'} \qquad i \in 1, \ldots, \mathfrak{a}(\kappa)}{\mathfrak{c}(\kappa, t_1, \ldots t_i, \ldots t_{\mathfrak{a}(\kappa)}) \rightsquigarrow \mathfrak{c}(\kappa, t_1, \ldots t_i', \ldots t_{\mathfrak{a}(\kappa)})}$$

By the IH t_i' pseobj. By application of the constructor rule the goal is obtained.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t_2 \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{t_3 \text{ pseobj}} \qquad \overset{\mathcal{D}_4}{|t_1| \rightleftharpoons |t_2|}}{[t_1, t_2; t_3] \text{ pseobj}}$$

By cases on $s \rightsquigarrow t$. Suppose w.l.o.g. $t_1 \rightsquigarrow t_1'$. Note that t_1 pseobj because it is a subterm of s. By the IH t_1' pseobj. By Lemma 2.14 $|t_1'| \rightleftharpoons |t_2|$. Thus, $[t_1', t_2; A]$ pseobj.

Case: s variable

By cases on $s \rightsquigarrow t$, t must be a variable. Thus, t pseobj.

□

Lemma 2.17. *If s pseobj, $|s| \rightleftharpoons |b|$, and $s \rightsquigarrow^* t$ then $|t| \rightleftharpoons |b|$*

Proof. By induction on $s \rightsquigarrow^* t$. The reflexivity case is trivial. The transitivity case is obtained from Lemma 2.14 and Lemma 2.16 and applying the IH. □

Theorem 2.18. *If s pseobj and $s \rightsquigarrow^* t$ then t pseobj*

Proof. By induction on $s \rightsquigarrow^* t$. The reflexivity case is trivial. The transitivity case is obtained from Lemma 2.16 and applying the IH. □

Lemma 2.19. *If s pseobj, $|t| \rightleftharpoons |b|$, and $s \rightsquigarrow^* t$ then $|s| \rightleftharpoons |b|$*

Proof. By induction on $s \rightsquigarrow^* t$. Consequence of Lemma 2.13 and Lemma 2.18.

□

Lemma 2.20. *If s pseobj, $s \equiv b$, and $s \rightsquigarrow^* t$ then $t \equiv b$*

Proof. Note that $\exists\ z_1, z_2$ such that $s \rightsquigarrow^* z_1$, $b \rightsquigarrow^* z_2$, and $|z_1| \rightleftharpoons |z_2|$. By confluence $\exists\ z_1'$ such that $z_1 \rightsquigarrow^* z_1'$ and $t \rightsquigarrow^* z_1'$. Then, by Lemma 2.18 z_1 pseobj. Finally, by Lemma 2.17 $|z_1'| \rightleftharpoons |z_2|$. Therefore, $t \equiv b$. □

Unlike with convertibility w.r.t. reduction, obtaining transitivity of conversion requires the additional assumption that the inner syntax form is a pseudo-object. Indeed, the incorporation of erasure into the definition requires this extra structure, because otherwise reductions on pairs would not agree. For example, $|[x, y, t]|$ is not convertible with $|[y, x, t]|$ for variables x and y, but this situation is ruled out because $[x, y, t]$ is not a pseudo-object.

Theorem 2.21. *If b* pseobj*, $a \equiv b$, and $b \equiv c$ then $a \equiv c$*

Proof. Note that $\exists\ u_1, u_2$ such that $a \rightsquigarrow^* u_1$, $b \rightsquigarrow^* u_2$, and $|u_1| \rightleftharpoons |u_2|$. Additionally, $\exists\ v_1, v_2$ such that $b \rightsquigarrow^* v_1$, $c \rightsquigarrow^* v_2$, and $|v_1| \rightleftharpoons |v_2|$. By confluence, $\exists\ z$ such that $u_2 \rightsquigarrow^* z$ and $v_1 \rightsquigarrow^* z$. Then, by Lemma 2.18 u_2 pseobj and v_1 pseobj. Next, by Lemma 2.17 $|u_1| \rightleftharpoons |z|$ and $|z| \rightleftharpoons |v_2|$. Thus, $|u_1| \rightleftharpoons |v_2|$ by Lemma 2.9 and $a \equiv c$. □

Knowing that $|s| \rightleftharpoons |t|$ if and only if $s \equiv t$ is critical for maintaining the spirit of Cedille. While the core theory of Cedille2 is its own system the purpose is to refine the design of Cedille without losing its essential features. A critical feature of Cedille is that convertibility is done with the untyped λ-calculus (i.e. erased terms) not with annotated terms themselves. Having Theorem 2.22 means that whenever conversion is checked between terms it is safe to instead check conversion w.r.t. reduction of objects. Not only does this maintain the spirit of Cedille, but it also enables optimizations in type checking. Indeed, arbitrarily expensive sequences of reductions could potentially be erased and not considered when checking $|s| \rightleftharpoons |t|$ instead of $s \equiv t$.

Theorem 2.22. *Suppose s* pseobj *and t* pseobj*, then $|s| \rightleftharpoons |t|$ iff $s \equiv t$*

Proof. Case ($\Rightarrow$): Suppose $|s| \rightleftharpoons |t|$. By definition $s \rightsquigarrow^* s$ and $t \rightsquigarrow^* t$. Thus, $s \equiv t$. Case ($\Leftarrow$): Suppose $s \equiv t$, then $\exists\ z_1, z_2$ such that $s \rightsquigarrow^* z_1$, $t \rightsquigarrow^* z_2$, and $|z_1| \rightleftharpoons |z_2|$. By two applications of Lemma 2.19 $|s| \rightleftharpoons |t|$. □

Finally, a useful lemma about substitution's interaction with conversion is obtained from the effort of pseudo-objects. This lemma is necessary to prove metatheoretic results about the system.

Lemma 2.23. *If s, t, a, b pseobj, $s \equiv t$, and $a \equiv b$ then $[x := s]a \equiv [x := t]b$*

Proof. By Lemma 2.22 $|s| \rightleftharpoons |t|$ and $|a| \rightleftharpoons |b|$. Then, by Lemma 2.12 $|[x := s]a| \rightleftharpoons |[x := t]b|$. Finally, by Lemma 2.22 again, $[x := s]a \equiv [x := t]b$. $\square$

2.3 Inference Judgment

The typing judgments, presented in Figure 2.6; Figure 2.7; and Figure 2.8, delineate what syntax are *proofs*. As stated previously, the erasure of a proof is an *object*. Thus, for $\Gamma \vdash t : A$, t is a proof and $|t|$ it's object. The judgment follows a standard PTS style, but the rules are carefully chosen so that an inference algorithm is possible. Judgments of the form $\Gamma \vdash t : A$ should be read t infers A in Γ.

$$\frac{}{\Gamma \vdash \star : \square}\ \textsc{Axiom}$$

The axiom rule is the same as with F^ω. The constant $\star$ should be interpreted as a universe of types, and the constant $\square$ as a universe of kinds. Thus, the axiom rule states that the universe of types *is* a kind in any context.

$$\frac{(x_m : A) \in \Gamma}{\Gamma \vdash x : A}\ \textsc{Var}$$

The variable rule requires that a variable at a certain type is inside the context. Note that variables are annotated with a mode. Modes take three forms: free (ω); erased (0); or type (τ). The type mode is used for proofs that exist inside the type universe; the free mode for proofs that belong to some type; and the erased mode for proofs that belong to some type but whose bound variable is not computationally relevant in the associated object. Variables are annotated with modes primarily to enable reconstruction of the appropriate binders.

$$\frac{\Gamma \vdash A : \mathrm{dom}_\Pi(m, K) \qquad \Gamma, x_m : A \vdash B : \mathrm{codom}_\Pi(m)}{\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)}\ \textsc{Pi}$$

The function type formation rule is similar to the rule for CC, but the domain and codomain are restricted. Instead of being part of either a type or kind universe, the respective universes are restricted by the associated mode. If the mode is τ then the domain can be either a type or a kind, but the codomain must be a kind. If the mode is ω then the domain and codomain both must be types.

$$\begin{aligned}\mathrm{dom}_\Pi(\omega, K) &= \star & \qquad \mathrm{codom}_\Pi(\omega) &= \star\\ \mathrm{dom}_\Pi(\tau, K) &= K & \mathrm{codom}_\Pi(\tau) &= \square\\ \mathrm{dom}_\Pi(0, K) &= K & \mathrm{codom}_\Pi(0) &= \star\end{aligned}$$

Figure 2.5: Domain and codomains for function types. The variable K is either $\star$ or $\square$.

$$\frac{}{\Gamma \vdash \star : \square}\ \textsc{Axiom} \qquad\qquad \frac{(x_m : A) \in \Gamma}{\Gamma \vdash x : A}\ \textsc{Var}$$

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash t : B \qquad A \equiv B}{\Gamma \vdash t : A}\ \textsc{Conv}$$

$$\frac{\Gamma \vdash A : \mathrm{dom}_\Pi(m, K) \qquad \Gamma, x_m : A \vdash B : \mathrm{codom}_\Pi(m)}{\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)}\ \textsc{Pi}$$

$$\frac{\begin{array}{c}\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)\\ \Gamma, x_m : A \vdash t : B \qquad x \notin FV(|t|) \text{ if } m = 0\end{array}}{\Gamma \vdash \lambda_m\, x{:}A.\, t : (x : A) \to_m B}\ \textsc{Lam}$$

$$\frac{\Gamma \vdash f : (x : A) \to_m B \qquad \Gamma \vdash a : A}{\Gamma \vdash f \bullet_m a : [x := a]B}\ \textsc{App}$$

Figure 2.6: Inference rules for function types, including erased functions. The variable K is either $\star$ or $\square$.

Otherwise, the mode is 0 and the domain may be either a type or kind, but the codomain must be a type. Note that this means polymorphic functions of data are not allowed to use their type argument computational in the object of a proof.

$$\frac{\begin{array}{c}\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)\\ \Gamma, x_m : A \vdash t : B \qquad x \notin FV(|t|) \text{ if } m = 0\end{array}}{\Gamma \vdash \lambda_m\, x{:}A.\, t : (x : A) \to_m B}\ \textsc{Lam}$$

The function formation rule is again similar to the rule for CC. Unlike the standard PTS CC rule, the codomain of the inferred function type is again restricted to $\mathrm{codom}_\Pi(m)$. Additionally, if the mode is erased then it must be explicitly shown that the bound variable does not appear in the associated object. Note that this is exactly the requirement imposed by pseudo-objects.

$$\frac{\Gamma \vdash f : (x : A) \to_m B \qquad \Gamma \vdash a : A}{\Gamma \vdash f \bullet_m a : [x := a]B}\ \textsc{App}$$

The application rule is not surprising, the only notable feature is that the mode of the function type and the application must match.

$$\frac{\Gamma \vdash A : \star \qquad \Gamma, x_\tau : A \vdash B : \star}{\Gamma \vdash (x : A) \cap B : \star}\ \textsc{Int}$$

The intersection type formation rule is similar to the function type formation rule, but the terms are all restricted to be types. Thus, there are no intersections of kinds in the core Cedille2 system.

$$\frac{\Gamma \vdash (x : A) \cap B : \star \qquad \Gamma \vdash t : A \qquad \Gamma \vdash s : [x := t]B \qquad t \equiv s}{\Gamma \vdash [t, s; (x : A) \cap B] : (x : A) \cap B}\ \textsc{Pair}$$

The pair formation rule is standard for formation of dependent pairs. A third type annotation argument is required in order to make the formula inferable from the proof. Otherwise, the annotation is required to be itself a type, the first component to match the first type, and the second component to match the second type with its free variable substituted with the first component. Additionally, the first and second component must be convertible. This restriction is what makes this a proof of an intersection, as opposed to merely a pair. Note that by Theorem 2.22 this condition is equivalent to $|t| \rightleftharpoons |s|$ which is the restriction imposed by pseudo-objects.

$$\frac{\Gamma \vdash t : (x : A) \cap B}{\Gamma \vdash t.2 : [x := t.1]B}\ \textsc{Snd}$$

The first and second projection rules are unsurprising. Both rules model projection from a pair as expected.

$$\frac{\Gamma \vdash A : \star \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \star}\ \textsc{Eq}$$

The equality type formation rule requires that the type annotation is a type and that the left and right-hand sides infer that type. Note that a typed equality like this is standard from the perspective of modern type theory but significantly different from the *untyped* equality of Cedille. Indeed, the equality rules are the area of significant deviation from the original Cedille design.

$$\frac{\Gamma \vdash A : \star \qquad \Gamma, x_\tau : A \vdash B : \star}{\Gamma \vdash (x : A) \cap B : \star}\ \textsc{Int}$$

$$\frac{\Gamma \vdash (x : A) \cap B : \star \qquad \Gamma \vdash t : A \qquad \Gamma \vdash s : [x := t]B \qquad t \equiv s}{\Gamma \vdash [t, s; (x : A) \cap B] : (x : A) \cap B}\ \textsc{Pair}$$

$$\frac{\Gamma \vdash t : (x : A) \cap B}{\Gamma \vdash t.1 : A}\ \textsc{Fst} \qquad \frac{\Gamma \vdash t : (x : A) \cap B}{\Gamma \vdash t.2 : [x := t.1]B}\ \textsc{Snd}$$

Figure 2.7: Inference rules for intersection types.

$$\frac{\Gamma \vdash A : \star \qquad \Gamma \vdash t : A}{\Gamma \vdash \mathrm{refl}(t) : t =_A t}\ \textsc{Refl}$$ The reflexivity rule is the only value for equality types. It is the standard inductive formulation of the equality type.

$$\frac{\Gamma \vdash e : a =_A b \qquad \Gamma \vdash P : (y : A) \to_\tau (p : a =_A y) \to_\tau \star}{\Gamma \vdash \psi(e, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a) \to_\omega P \bullet_\tau b \bullet_\tau e}\ \textsc{Subst}$$ The substitution rule is dependent variation of the Leibniz's Law. It is a variation of Martin-Löf's J rule introduced by Pfenning and Paulin-Mohring [37]. Notice that the only critical difference between this rule and a standard variation of Leibniz's Law is that the predicate may depend on the equality proof as well.

$$\frac{\begin{array}{c}\Gamma \vdash a : (x : A) \cap B \\ \Gamma \vdash b : (x : A) \cap B \qquad \Gamma \vdash e : a.1 =_A b.1\end{array}}{\Gamma \vdash \vartheta_1(e, a, b) : a =_{(x:A)\cap B} b}\ \textsc{PrmFst}$$ The first and second promotion rules enable equational reasoning about intersections. Indeed, because intersections are not inductive it is difficult to reason about them without some auxiliary rule. The first promotion rule in particular states that two elements of an intersection are equal if their first projections are equal. Second projection promotion is very similar except it equates two elements of an intersection if their second projections are equal.

$$\frac{\begin{array}{c}\Gamma \vdash f : (a : A) \to_\omega (x : A') \cap B \\ A \equiv A' \qquad \Gamma \vdash e : (a : A) \to_\omega a =_A (f \bullet_\omega a).1 \qquad FV(|e|) = \varnothing\end{array}}{\Gamma \vdash \varphi(f, e) : (a : A) \to_\omega (x : A) \cap B}\ \textsc{Cast}$$ The cast rule asserts that a new function $\varphi(f, e)$ exists at the associate type if there

$$\dfrac{\begin{array}{c}\Gamma \vdash A : \star \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : A\end{array}}{\Gamma \vdash a =_A b : \star}\ \textsc{Eq} \qquad \dfrac{\Gamma \vdash A : \star \qquad \Gamma \vdash t : A}{\Gamma \vdash \mathrm{refl}(t) : t =_A t}\ \textsc{Refl}$$

$$\dfrac{\Gamma \vdash e : a =_A b \qquad \Gamma \vdash P : (y : A) \to_\tau (p : a =_A y) \to_\tau \star}{\Gamma \vdash \psi(e, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a) \to_\omega P \bullet_\tau b \bullet_\tau e}\ \textsc{Subst}$$

$$\dfrac{\begin{array}{c}\Gamma \vdash a : (x : A) \cap B \\ \Gamma \vdash b : (x : A) \cap B \qquad \Gamma \vdash e : a.1 =_A b.1\end{array}}{\Gamma \vdash \vartheta_1(e, a, b) : a =_{(x:A)\cap B} b}\ \textsc{PrmFst}$$

$$\dfrac{\begin{array}{c}\Gamma \vdash a : (x : A) \cap B \\ \Gamma \vdash b : (x : A) \cap B \qquad \Gamma \vdash e : a.2 =_{[x:=a.1]B} b.2\end{array}}{\Gamma \vdash \vartheta_2(e, a, b) : a =_{(x:A)\cap B} b}\ \textsc{PrmSnd}$$

$$\dfrac{\begin{array}{c}\Gamma \vdash f : (a : A) \to_\omega (x : A') \cap B \\ A \equiv A' \qquad \Gamma \vdash e : (a : A) \to_\omega a =_A (f \bullet_\omega a).1 \qquad FV(|e|) = \varnothing\end{array}}{\Gamma \vdash \varphi(f, e) : (a : A) \to_\omega (x : A) \cap B}\ \textsc{Cast}$$

$$\dfrac{\Gamma \vdash e : \mathrm{ctt} =_{\mathrm{cBool}} \mathrm{cff}}{\Gamma \vdash \delta(e) : (X : \star) \to_0 X}\ \textsc{Sep}$$

Figure 2.8: Inference rules for equality types where cBool := $(X : \star) \to_0 (x : X) \to_\omega (y : X) \to_\omega X$; ctt := $\lambda_0\, X : \star.\, \lambda_\omega\, x : X.\, \lambda_\omega\, y : X.\, x$; and cff := $\lambda_0\, X : \star.\, \lambda_\omega\, x{:}X.\, \lambda_\omega\, y{:}X.\, y$. Also, $i, j \in \{1, 2\}$

is another function, f, that is extensionally the identity in an erased context. The requirement on e is of particular interest to understand. In general, it states only that f is extensionally the identity function. However, the additional restriction that $FV(|e|)$ is empty means that variables from the context may not appear in the object of e. Thus, computationally, e cannot become stuck at the fault of the context. Of course, the input to e may still be a variable and thus prevent reduction.

$$\dfrac{\Gamma \vdash e : \mathrm{ctt} =_{\mathrm{cBool}} \mathrm{cff}}{\Gamma \vdash \delta(e) : (X : \star) \to_0 X}\ \textsc{Sep}$$

The separation rule states only that the equational theory is not degenerate, i.e. that there are at least two distinct proofs.

The context of a judgment is said to be *well-formed*, written $\vdash \Gamma$, if all variables in Γ are distinct and for every $\Gamma, x : A, \Delta$ it is the case that $\Gamma \vdash A : K$. In other words, all types in a context must be proofs in the associated context prefix. This condition is not automatically met by the judgment, but there are no proofs of interest where this condition fails that will be considered. Thus, whenever $\Gamma \vdash t : A$ it is assumed that $\vdash \Gamma$.

An important observation is that proofs and their types are a richer form of pseudo-objects. Thus, conversion is an equivalence relation for proofs and their types. Other basic lemmas of importance are the admissibility of a weakening rule, and a substitution rule.

Lemma 2.24. *If* $\Gamma \vdash t : A$ *then* t pseobj

Proof. Straightforward by induction. The only interesting case is the pair case, but it is discharged by Theorem 2.22. $\square$

Lemma 2.25. *If* $\Gamma \vdash t : A$ *then* A pseobj

Proof. By induction. The Ax, Pi, Int and Eq rules are trivial. Rules Lam, Pair, and Conv rules are immediate by applying Lemma 2.24 to a sub-derivation. The Cast rule is immediate by applying the IH to a sub-derivation. The Fst and App rules are omitted because it is similar to the Snd rule. Likewise, the PrmFst and Refl rules are omitted because it is similar to the PrmSnd rule.

Case: $$\frac{\overset{\mathcal{D}_1}{(x_m : A) \in \Gamma}}{\Gamma \vdash x : A}$$

Note that it is assumed that $\vdash \Gamma$. Thus, there is some prefix of Γ, call it Δ, such that $\Delta \vdash A : K$. By Lemma 2.24: A pseobj.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash t : (x : A) \cap B}}{\Gamma \vdash t.2 : [x := t.1]B}$$

By the IH applied to $\mathcal{D}_1$: B pseobj. Using Lemma 2.24 gives t pseobj and thus $t.1$ pseobj. Now by Lemma 2.15: $[x := t.1]B$ pseobj.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{e : a =_A b} \qquad \Gamma \vdash \overset{\mathcal{D}_2}{P : (y : A) \to_\tau (p : a =_A y) \to_\tau \star}}{\Gamma \vdash \psi(e, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

By Lemma 2.24: P, e pseobj. Applying the IH to $\mathcal{D}_1$ gives A, a, b pseobj. Now building up the subexpressions using pseudo-object rules concludes the proof.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{a : (x : A) \cap B} \qquad \Gamma \vdash \overset{\mathcal{D}_2}{b : (x : A) \cap B} \qquad \Gamma \vdash \overset{\mathcal{D}_3}{e : a.2 =_{[x := a.1]B} b.2}}{\Gamma \vdash \vartheta_2(e, a, b) : a =_{(x:A)\cap B} b}$$

Applying the IH to $\mathcal{D}_1$ gives that $(x : A) \cap B$ pseobj. Now, by Lemma 2.24: a, b pseobj. Using the pseudo-object rule for equality concludes the case.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{e : \mathrm{ctt} =_{\mathrm{cBool}} \mathrm{cff}}}{\Gamma \vdash \delta(e) : (X : \star) \to_0 X}$$

Immediate by a short sequence of pseobjrules.

$\square$

Lemma 2.26 (Weakening). *If* $\Gamma, \Delta \vdash t : A$ *and* $\Gamma \vdash B : K$ *then* $\Gamma, x_m : B, \Delta \vdash t : A$ *for* x *fresh*

Proof. By induction. Most cases are a direct consequence of applying the IH to sub-derivations and applying the associated rule. Note that it is assumed that $\vdash \Gamma, \Delta$, and thus $\vdash \Gamma$. Now, because B is a proof it is obvious that $\vdash \Gamma, x_m : B, \Delta$.

Case: $$\frac{}{\Gamma \vdash \star : \square}$$

Trivial by axiom rule.

Case: $$\frac{\overset{\mathcal{D}_1}{(x_m : A) \in \Gamma}}{\Gamma \vdash x : A}$$

If $(x_m : A) \in \Gamma, \Delta$ then $(x_m : A) \in \Gamma, y : B, \Delta$.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \mathrm{dom}_\Pi(m, K)} \qquad \overset{\mathcal{D}_2}{\Gamma, x_m : A \vdash B : \mathrm{codom}_\Pi(m)}}{\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)}$$

The IH applied to $\mathcal{D}_1$ and $\mathcal{D}_2$ and the pi-rule concludes the case.

□

Lemma 2.27 (Substitution)**.** *Suppose* $\Gamma \vdash b : B$.

1. *If* $\Gamma, x : B, \Delta \vdash t : A$ *then* $\Gamma, [x := b]\Delta \vdash [x := b]t : [x := b]A$
2. *If* $\vdash \Gamma, x : B, \Delta$ *then* $\vdash \Gamma, [x := b]\Delta$

Proof. By mutual recursion. The Ax rule is trivial and omitted. The rules Lam and Int are very similar to the Pi rule. The rules Fst, Eq, Refl, Subst, PrmFst, PrmSnd, Cast and Sep rules are proven by applying *1.* to sub-derivations and using the associated rule. Rule Snd is very similar to App and thus omitted. Likewise, Conv is very similar to Pair and thus omitted. Note that the context cannot be empty.

Case: $\vdash \Gamma, x : B, \Delta', y : A$

Note that Δ' is a smaller context, thus by *2.* $\vdash [x := b]\Delta'$. Moreover, it is the case that $\Gamma, x : B, \Delta' \vdash A : K$. Now, using *1.* with the previous derivation gives $\Gamma, [x := b]\Delta' \vdash [x := b]A : K$. Thus, the context remains well-formed.

Case: $$\frac{\overset{\mathcal{D}_1}{(x_m : A) \in \Gamma}}{\Gamma \vdash x : A}$$

Rename to y. If $y \neq x$ then suppose wlog that $(y : A) \in \Delta$. Then $y : [x := b]A \in [x := b]\Delta$. Thus, $\Gamma, [x := b]\Delta \vdash y : [x := b]A$. Suppose $y = x$, then $[x := b]y = b$. Note that $[x := b]B = B$, because $\vdash \Gamma, x : B, \Delta$ forces $x \notin FV(B)$. Moreover, $A = B$ because $y = x$. Thus, $\Gamma, [x := b]\Delta \vdash [x := b]y : [x := b]A$.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : \mathrm{dom}_\Pi(m, K) \qquad \Gamma, x_m : A \vdash \overset{\mathcal{D}_2}{B} : \mathrm{codom}_\Pi(m)}{\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)}$$

Applying *1.* to the sub-derivations gives:

$\mathcal{D}_1$. $\Gamma, [x := b]\Delta \vdash [x := b]A : \mathrm{dom}_\Pi(m, K)$

$\mathcal{D}_2$. $\Gamma, [x := b]\Delta, y_m : [x := b]A \vdash [x := b]B : \mathrm{codom}_\Pi(m)$

Thus, $\Gamma, [x := b]\Delta \vdash (y : [x := b]A) \to_m [x := b]B : \mathrm{codom}_\Pi(m)$.

Case: $$\frac{\Gamma \vdash f : (x \overset{\mathcal{D}_1}{:} A) \to_m B \qquad \Gamma \vdash \overset{\mathcal{D}_2}{a} : A}{\Gamma \vdash f \bullet_m a : [x := a]B}$$

Applying *1.* to $\mathcal{D}_1$ and $\mathcal{D}_2$ gives $\Gamma, [x := b]\Delta \vdash [x := b]f : (y : [x := b]A) \to_m [x := b]B$ and $\Gamma, [x := b]\Delta, y_m : [x := b]A \vdash [x := b]a : [x := b]A$. By the App rule $\Gamma, [x := b]\Delta \vdash [x := b]f \bullet_m [x := b]a : [y := a][x := b]B$. Note that y is fresh to Γ, thus $y \notin FV(b)$. By Lemma 2.1 $[y := a][x := b]B = [x := b][y := a]B$.

Case: $$\frac{\Gamma \vdash (x : \overset{\mathcal{D}_1}{A}) \cap B : \star \qquad \Gamma \vdash \overset{\mathcal{D}_2}{t} : A \qquad \Gamma \vdash s : \overset{\mathcal{D}_3}{[x := t]}B \qquad \overset{\mathcal{D}_4}{t \equiv s}}{\Gamma \vdash [t, s; (x : A) \cap B] : (x : A) \cap B}$$

Applying *1.* to the sub-derivations gives:

$\mathcal{D}_1$. $\Gamma, [x := b]\Delta \vdash (y : [x := b]A) \cap [x := b]B : \star$

$\mathcal{D}_2$. $\Gamma, [x := b]\Delta \vdash [x := b]t : [x := b]A$

$\mathcal{D}_3$. $\Gamma, [x := b]\Delta \vdash [x := b]s : [x := b][y := t]B$

Note that y is locally-bound and thus $y \notin FV(\Gamma)$, thus by Lemma 2.1

$$[x := b][y := t]B = [y := [x := b]t][x := b]B$$

Now by Lemma 2.23: $[x := b]t \equiv [x := b]s$. Thus, by the Pair rule $\Gamma, [x := b]\Delta \vdash [[x := b]t, [x := b]s] : (y : [x := b]A) \cap [x := b]B$.

□

2.4 Classification and Preservation

Lemma 2.28. *If $\Gamma \vdash t : A$ then $A = \Box$ or $\Gamma \vdash A : K$*

Proof. By induction. The Ax, Pi, Lam, Int, Pair, Eq, and Conv rules are trivial. The Fst and PrmFst rules are omitted because they are very similar to Snd and PrmSnd respectively.

Case: $$\frac{\overset{\mathcal{D}_1}{(x_m : A) \in \Gamma}}{\Gamma \vdash x : A}$$

Because $x_m : A \in \Gamma$ then $\Gamma = \Delta_1, x_m : A, \Delta_2$. By $\vdash \Gamma$: $\Delta_1 \vdash A : K$. Now using weakening $\Gamma \vdash A : K$.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash f : (x : A) \to_m B} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A}}{\Gamma \vdash f \bullet_m a : [x := a]B}$$

Applying the IH to $\mathcal{D}_1$ gives $\Gamma \vdash (x : A) \to_m B : K$. Now $\Gamma, x : A \vdash B : K$. Using the substitution lemma gives $\Gamma \vdash [x := a]B : K$.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash t : (x : A) \cap B}}{\Gamma \vdash t.2 : [x := t.1]B}$$

By the IH applied to $\mathcal{D}_1$ gives $\Gamma \vdash (x : A) \cap B : K$. Thus, $\Gamma, x : A \vdash B : K$. Applying the substitution lemma gives $\Gamma \vdash [x := t.1]B : K$.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash t : A}}{\Gamma \vdash \text{refl}(t) : t =_A t}$$

Immediate by applying the Eq rule.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash e : a =_A b} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash P : (y : A) \to_\tau (p : a =_A y) \to_\tau \star}}{\Gamma \vdash \psi(e, P) : P \bullet_\tau a \bullet_\tau \text{refl}(a) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

Applying the IH to $\mathcal{D}_1$ gives $\Gamma \vdash a =_A b : K$. By inversion this gives $\Gamma \vdash A : \star$, $\Gamma \vdash a : A$, and $\Gamma \vdash b : A$. Now by the App rule $\Gamma \vdash P \bullet_\tau a : \star$ and $\Gamma \vdash P \bullet_\tau b : \star$. Using weakening gives $\Gamma, x : P \bullet_\tau a \vdash P \bullet_\tau b : \star$. Now the Pi rule concludes the case.

Case:
$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash a : (x : A) \cap B} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash b : (x : A) \cap B} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash e : a.2 =_{[x:=a.1]B} b.2}}{\Gamma \vdash \vartheta_2(e, a, b) : a =_{(x:A)\cap B} b}$$

By the IH applied to $\mathcal{D}_1$: $\Gamma \vdash (x : A) \cap B : K$. Note that K must be $\star$. Now applying the Eq rule gives $\Gamma \vdash a =_{(x:A)\cap B} b : \star$.

Case:
$$\frac{\begin{array}{c}\overset{\mathcal{D}_1}{\Gamma \vdash f : (a : A) \to_\omega (x : A') \cap B} \\ \overset{\mathcal{D}_2}{A \equiv A'} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash e : (a : A) \to_\omega a =_A (f \bullet_\omega a).1} \qquad \overset{\mathcal{D}_4}{FV(|e|) = \varnothing}\end{array}}{\Gamma \vdash \varphi(f, e) : (a : A) \to_\omega (x : A) \cap B}$$

Immediate by the IH applied to $\mathcal{D}_1$.

Case:
$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash e : \text{ctt} =_{\text{cBool}} \text{cff}}}{\Gamma \vdash \delta(e) : (X : \star) \to_0 X}$$

Have $\Gamma \vdash (X : \star) \to_\omega X : \star$ via short sequence of rules.

$\square$

Lemma 2.29. *If* $\vdash \Gamma$, $\Gamma \vdash t : A$, $\Gamma \rightsquigarrow \Gamma'$, *and* $t \rightsquigarrow t'$ *then* $\vdash \Gamma'$ *and* $\Gamma' \vdash t' : A$

Chapter 3

PROOF NORMALIZATION AND RELATIONSHIP TO SYSTEM F$^\omega$

Chapter 4

CONSISTENCY AND RELATIONSHIP TO CDLE

Chapter 5

OBJECT NORMALIZATION

A φ_i-proof is a proof that allows i nested φ syntactic constructs. For example, a φ_0-proof allows no φ subterms, a φ_1-proof allows φ subterms but no nested φ subterms, and a φ_2-proof allows φ_1 subterms. Defined inductively, a φ_0-proof is a proof with no φ syntactic constructs and a φ_{i+1}-proof is a proof with φ_i-proof subterms.

For any φ_i-proof p there is a strictification $s(p)$ that is a φ_0-proof in Figure 5.1.

Lemma 5.1 (Strictification Preserves Inference). *Given* $\Gamma \vdash t \rhd A$ *then* $\Gamma \vdash s(t) \rhd A$

Proof. By induction on the typing rule, the φ rule is the only one of interest:

Case:
$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash f : (a : A) \to_\omega (x : A') \cap B} \qquad \overset{\mathcal{D}_2}{A \equiv A'} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash e : (a : A) \to_\omega a =_A (f \bullet_\omega a).1} \qquad \overset{\mathcal{D}_4}{FV(|e|) = \varnothing}}{\Gamma \vdash \varphi(f, e) : (a : A) \to_\omega (x : A) \cap B}$$

Need to show that $\Gamma \vdash s(\varphi(a, f, e)) \rhd (x : A) \cap B$ which reduces to: $\Gamma \vdash s(f) \bullet_\omega s(a) \rhd (x : A) \cap B$. By the IH we know that $s(f)$ infers the same function type, and that $s(a)$ infers the same argument type, therefore the application rule concludes the proof.

□

Lemma 5.2 (Strict Proofs are Normalizing). *Given* $\Gamma \vdash t \rhd A$ *then* $s(t)$ *is strongly normalizing*

Proof. Direct consequence of strong normalization of proofs □

Lemma 5.3 (Strict Objects are Normalizing). *Given* $\Gamma \vdash t \rhd A$ *then* $|s(t)|$ *is strongly normalizing*

$$
\begin{aligned}
s(x) &= x & s([s,t,T]) &= [s(s), s(t), s(T)] \\
s(\star) &= \star & s(t.1) &= s(t).1 \\
s(\square) &= \square & s(t.2) &= s(t).2 \\
s(\lambda_m\, x{:}A.\, t) &= \lambda_m\, x{:}s(A).\, s(t) & s(x =_A y) &= s(x) =_{s(A)} s(y) \\
s((x : A) \to_m B) &= (x : s(A)) \to_m s(B) & s(\mathrm{refl}(t)) &= \mathrm{refl}(s(t)) \\
s((x : A) \cap B) &= (x : s(A)) \cap s(B) & s(\vartheta(e)) &= \vartheta(s(e)) \\
s(f \bullet_m a) &= s(f) \bullet_m s(a) & s(\delta(e)) &= \delta(s(e))
\end{aligned}
$$

$$
\begin{aligned}
s(J(A,P,x,y,r,w)) &= J(s(A), s(P), s(x), s(y), s(r), s(w)) \\
s(\varphi(a,f,e)) &= s(f) \bullet_\omega s(a)
\end{aligned}
$$

Figure 5.1: Strictification of a proof.

Proof. Proof Idea:

Proof reduction tracks object reduction in the absence of φ constructs. Thus, the normalization of a proof provides an upper-bound on the number of reductions an object can take to reach a normal form. $\square$

A proof, $\Gamma \vdash t_1 \rhd A$, is contextually equivalent to another proof, $\Gamma \vdash t_2 \rhd A$, if there is no context with hole of type A whose object reduction diverges for t_1 but not t_2. In other words, if a context can be constructed that distinguishes the terms based on their object reduction.

Lemma 5.4. *A φ_1-proof, p, is contextually equivalent to its strictification, $s(p)$*

Proof. Proof by induction on the typing rule for p, focus on the application rule:

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash f : (x : A) \to_m B} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A}}{\Gamma \vdash f \bullet_m a : [x := a]B}$$

In particular, we care about when $f = \varphi(v, b, e).2$ and $m = \omega$. Note that the first projection has a proof-reduction that yields a which makes it unproblematic.

We know that $s(v) = v$ because f is a φ_1-proof. Let v_n be the normal form of v and note that $|v_n|$ is also normal. Likewise, we have e_n and $|e_n|$ normal.

Suppose there is a context $C[\cdot]$ where $|p|$ diverges but $|s(p)|$ normalizes. (Note that the opposite assumption is impossible). If $|v_n|$ is a variable, then reduction in $|p|$ is blocked (contradiction). Otherwise $|v_n| = \lambda\, x.\, x\ t_1\ \cdots\ t_n$ where t_i are normal.

Now it must be the case that $|e \bullet_\omega v| = |e_n| \bullet_\omega |v_n|$ is normalizing. Thus, we have a refl proof that $v_n = (f \bullet_\omega v_n).1$. (Note, this proof *must* be refl because $\mathrm{FV}(|e|) = \varnothing$). But, this implies convertibility, thus $|v_n| =_\beta |f| \bullet_\omega |v_n|$, but this must mean more concretely that $|f| \bullet_\omega |v_n| \rightsquigarrow |v_n|$. Yet $|f| \bullet_\omega |v_n| \bullet_\omega a$ is strongly normalizing because it is $s(p)$. Therefore, p in this case is strongly normalizing which refutes the assumption yielding a contradiction.

□

Lemma 5.5. *If t_1 is strongly normalizing and contextually equivalent to t_2 then t_2 is strongly normalizing*

Proof. Immediate by the definition of contextual equivalence. □

Theorem 5.6. *A φ_i-proof p is strongly normalizing for all i*

Proof. By induction on i.

Case: $i = 0$

Immediate because $s(p) = p$ and strict proofs are strongly normalizing.

Inductive Case:

Suppose that φ_i-proof is strongly normalizing. Goal: show that φ_{i+1}-proof is strongly normalizing.

□

Chapter 6

CEDILLE2: SYSTEM IMPLEMENTATION

Chapter 7

CEDILLE2: INTERNALLY DERIVABLE CONCEPTS

Chapter 8

CONCLUSION AND FUTURE WORK

Appendix A

PROOF OF CONFLUENCE

Lemma A.1. *For any t, $t \Rrightarrow t$*

Proof. Straightforward by induction on t. □

Lemma A.2. *If $s \rightsquigarrow t$ then $s \Rrightarrow t$*

Proof. By induction on $s \rightsquigarrow t$.

Case: $(\lambda_m\, x : A.\, b) \bullet_m t \rightsquigarrow [x := t]b$

By Lemma A.1: $t \Rrightarrow t$ and $b \Rrightarrow b$. Applying the ParBeta rule concludes the case.

Case: $[t_1, t_2; A].1 \rightsquigarrow t_1$

As above, $t_1 \Rrightarrow t_1$, applying the ParFst rule concludes the case.

Case: $[t_1, t_2; A].2 \rightsquigarrow t_2$

Same as previous case using ParSnd rule.

Case: $\psi(\text{refl}(t), P) \rightsquigarrow \lambda_\omega\, x : P \bullet_\tau t.\, x$

Using Lemma A.1: $t \Rrightarrow t$ and $P \Rrightarrow P$. Applying the ParSubst rule concludes the case.

Case: $\vartheta_1(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$

As with previous cases, $t_2 \Rrightarrow t_2$. Applying the ParPrmFst rule concludes the case.

Case: $\vartheta_2(\text{refl}(t_1), t_2, t_3) \rightsquigarrow \text{refl}(t_2)$

Same as above using ParPrmSnd.

$$\frac{}{x \Rrightarrow x}\ \textsc{ParVar}$$

$$\frac{t_i \Rrightarrow t_i' \quad \forall\, i \in \{1, \ldots, \mathfrak{a}(\kappa)\}}{\mathfrak{c}(\kappa, t_1, \ldots, t_i, \ldots, t_{\mathfrak{a}(\kappa)}) \Rrightarrow \mathfrak{c}(\kappa, t_1', \ldots, t_i', \ldots, t_{\mathfrak{a}(\kappa)}')}\ \textsc{ParCtor}$$

$$\frac{t_1 \Rrightarrow t_1' \qquad t_2 \Rrightarrow t_2'}{\mathfrak{b}(\kappa, x : t_1, t_2) \Rrightarrow \mathfrak{b}(\kappa, x : t_1', t_2')}\ \textsc{ParBind}$$

$$\frac{t_1 \Rrightarrow t_1' \qquad t_2 \Rrightarrow t_2' \qquad t_3 \Rrightarrow t_3'}{(\lambda_m\, x : t_1.\, t_2) \bullet_m t_3 \Rrightarrow [x := t_3']t_2'}\ \textsc{ParBeta}$$

$$\frac{t_1 \Rrightarrow t_1' \qquad t_2 \Rrightarrow t_2'}{\psi(\text{refl}(t_1), t_2) \Rrightarrow \lambda_\omega\, x : t_2' \bullet_\tau t_1'.\, x}\ \textsc{ParSubst}$$

$$\frac{t_1 \Rrightarrow t_1' \qquad t_2 \Rrightarrow t_2' \qquad t_3 \Rrightarrow t_3'}{[t_1, t_2; t_3].1 \Rrightarrow t_1'}\ \textsc{ParFst}$$

$$\frac{t_1 \Rrightarrow t_1' \qquad t_2 \Rrightarrow t_2' \qquad t_3 \Rrightarrow t_3'}{[t_1, t_2; t_3].2 \Rrightarrow t_2'}\ \textsc{ParSnd}$$

$$\frac{t_1 \Rrightarrow t_1' \qquad t_2 \Rrightarrow t_2' \qquad t_3 \Rrightarrow t_3'}{\vartheta_1(\text{refl}(t_1), t_2, t_3) \Rrightarrow \text{refl}(t_2')}\ \textsc{ParPrmFst}$$

$$\frac{t_1 \Rrightarrow t_1' \qquad t_2 \Rrightarrow t_2' \qquad t_3 \Rrightarrow t_3'}{\vartheta_2(\text{refl}(t_1), t_2, t_3) \Rrightarrow \text{refl}(t_2')}\ \textsc{ParPrmSnd}$$

Figure A.1: Parallel reduction rules for arbitrary syntax.

$$\begin{aligned}
\llparenthesis(\lambda_m\, x : t_1.\, t_2) \bullet_m t_3\rrparenthesis &= [x := \llparenthesis t_3\rrparenthesis]\llparenthesis t_2\rrparenthesis \\
\llparenthesis\psi(\text{refl}(t_1), t_2)\rrparenthesis &= \lambda_\omega\, x : \llparenthesis t_2\rrparenthesis \bullet_\tau \llparenthesis t_1\rrparenthesis.\, x \\
\llparenthesis[t_1, t_2; t_3].1\rrparenthesis &= \llparenthesis t_1\rrparenthesis \\
\llparenthesis[t_1, t_2; t_3].2\rrparenthesis &= \llparenthesis t_2\rrparenthesis \\
\llparenthesis\vartheta_1(\text{refl}(t_1), t_2, t_3)\rrparenthesis &= \text{refl}(\llparenthesis t_2\rrparenthesis) \\
\llparenthesis\vartheta_2(\text{refl}(t_1), t_2, t_3)\rrparenthesis &= \text{refl}(\llparenthesis t_2\rrparenthesis) \\
\llparenthesis\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}(\kappa)})\rrparenthesis &= \mathfrak{c}(\kappa, \llparenthesis t_1\rrparenthesis, \ldots, \llparenthesis t_{\mathfrak{a}(\kappa)}\rrparenthesis) \\
\llparenthesis\mathfrak{b}(\kappa, (x : t_1), t_2)\rrparenthesis &= \mathfrak{b}(\kappa, (x : \llparenthesis t_1\rrparenthesis), \llparenthesis t_2\rrparenthesis) \\
\llparenthesis x\rrparenthesis &= x
\end{aligned}$$

Figure A.2: Definition of a reduction completion function $\llparenthesis - \rrparenthesis$ for parallel reduction. Note that this function is defined by pattern matching, applying cases from top to bottom. Thus, the cases at the very bottom are catch-all for when the prior cases are not applicable.

Case: $$\frac{\overset{\mathcal{D}_1}{t_i \rightsquigarrow t_i'} \qquad i \in 1, \ldots, \mathfrak{a}(\kappa)}{\mathfrak{c}(\kappa, t_1, \ldots t_i, \ldots t_{\mathfrak{a}(\kappa)}) \rightsquigarrow \mathfrak{c}(\kappa, t_1, \ldots t_i', \ldots t_{\mathfrak{a}(\kappa)})}$$

By the IH applied to $\mathcal{D}_1$: $t_i \Rrightarrow t_i'$. Note that there is only one subderivation. For all $j \neq i$ $t_j \Rrightarrow t_j$ by Lemma A.1. Using the ParCtor rule concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \rightsquigarrow t_1'}}{\mathfrak{b}(\kappa, x : t_1, t_2) \rightsquigarrow \mathfrak{b}(\kappa, x : t_1', t_2)}$$

Applying the IH to $\mathcal{D}_1$ yields $t_1 \Rrightarrow t_1'$. By Lemma A.1: $t_2 \Rrightarrow t_2$. Using the ParBind rule concludes the case.

□

Lemma A.3. *If $s \rightsquigarrow^* t$ then $s \Rrightarrow^* t$*

Proof. By induction on $s \rightsquigarrow^* t$ applying Lemma A.2 in the inductive case. □

Lemma A.4. *If $s \Rrightarrow t$ then $s \rightsquigarrow^* t$*

Proof. By induction on $s \Rrightarrow t$.

Case: $$\frac{}{x \Rrightarrow x}$$

By reflexivity of reduction.

Case: $$\frac{\overset{\mathcal{D}_i}{t_i \Rrightarrow t'_i} \quad \forall\, i \in \{1, \dots, \mathfrak{a}(\kappa)\}}{\mathfrak{c}(\kappa, t_1, \dots, t_i, \dots, t_{\mathfrak{a}(\kappa)}) \Rrightarrow \mathfrak{c}(\kappa, t'_1, \dots, t'_i, \dots, t'_{\mathfrak{a}(\kappa)})}$$

By the IH applied to each $\mathcal{D}_i$: $t_i \rightsquigarrow^* t'_i$ for all i. Applying Lemma 2.2 concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2}}{\mathfrak{b}(\kappa, x : t_1, t_2) \Rrightarrow \mathfrak{b}(\kappa, x : t'_1, t'_2)}$$

As the previous case, the IH yields $t_1 \rightsquigarrow^* t_1$ and $t_2 \rightsquigarrow^* t'_2$. Again using Lemma 2.2 concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t'_3}}{(\lambda_m\, x : t_1.\, t_2) \bullet_m t_3 \Rrightarrow [x := t'_3]t'_2}$$

Applying the IH to all available derivations and using Lemma 2.2 gives $(\lambda_m\, x : t_1.\, t_2) \bullet_m t_3 \rightsquigarrow^* (\lambda_m\, x : t'_1.\, t'_2) \bullet_m t'_3$. Applying the beta rule of reduction with transitivity concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2}}{\psi(\text{refl}(t_1), t_2) \Rrightarrow \lambda_\omega\, x : t'_2 \bullet_\tau t'_1.\, x}$$

Same as the previous case but using the substitution rule.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t'_3}}{[t_1, t_2; t_3].1 \Rrightarrow t'_1}$$

Same as the previous case but using the first rule.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'}}{[t_1, t_2; t_3].2 \Rrightarrow t_2'}$$

Same as the previous case but using the second rule.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'}}{\vartheta_1(\mathrm{refl}(t_1), t_2, t_3) \Rrightarrow \mathrm{refl}(t_2')}$$

Same as the previous case but using the first promotion rule.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'}}{\vartheta_2(\mathrm{refl}(t_1), t_2, t_3) \Rrightarrow \mathrm{refl}(t_2')}$$

Same as the previous case but using the second promotion rule.

□

Lemma A.5. *If $s \Rrightarrow^* t$ then $s \rightsquigarrow^* t$*

Proof. By induction on $s \Rrightarrow^* t$ applying Lemma A.4 in the inductive case. □

Lemma A.6. *If $s \Rrightarrow s'$ and $t \Rrightarrow t'$ then $[x := s]t \Rrightarrow [x := s']t'$*

Proof. By induction on $t \Rrightarrow t'$.

Case: $$\frac{}{x \Rrightarrow x}$$

Rename to y. If $x = y$ then $s \Rrightarrow s'$ which is a premise. If $x \neq y$ then no substitution is performed and $y \Rrightarrow y$.

Case: $$\frac{\overset{\mathcal{D}_i}{t_i \Rrightarrow t_i' \quad \forall\, i \in \{1, \ldots, \mathfrak{a}(\kappa)\}}}{\mathfrak{c}(\kappa, t_1, \ldots, t_i, \ldots, t_{\mathfrak{a}(\kappa)}) \Rrightarrow \mathfrak{c}(\kappa, t_1', \ldots, t_i', \ldots, t_{\mathfrak{a}(\kappa)}')}$$

Applying the IH to $\mathcal{D}_i$ yields $[x := s]t_i \Rrightarrow [x := s']t_i'$ for all i. Unfolding substitution for $\mathfrak{c}$ and applying the ParCtor rule concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'}}{\mathfrak{b}(\kappa, x : t_1, t_2) \Rrightarrow \mathfrak{b}(\kappa, x : t_1', t_2')}$$

As above the IH gives $[x := s]t_i \Rrightarrow [x := s']t_i'$ for $i = 1$ and $i = 2$. Unfolding substitution for $\mathfrak{b}$ and applying the ParBind rule concludes.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'}}{(\lambda_m\, x : t_1.\, t_2) \bullet_m t_3 \Rrightarrow [x := t_3']t_2'}$$

By the IH: $[x := s]t_i \Rrightarrow [x := s']t_i'$ for $i = 1, 2, 3$. The ParBeta rule gives the following: $[x := s](\lambda_m\, y : t_1.\, t_2) \bullet_m t_3 = (\lambda_m\, y : [x := s]t_1.\, [x := s]t_2) \bullet_m [x := s]t_3 \Rrightarrow [y := t_3'][x := s']t_2'$. Note that y is bound and thus not a free variable in s' and, moreover, by implicit renaming $x \neq y$. Thus, by Lemma 2.1 $[y := t_3'][x := s']t_2' = [x := s'][y := t_3']t_2'$.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'}}{\psi(\text{refl}(t_1), t_2) \Rrightarrow \lambda_\omega\, x : t_2' \bullet_\tau t_1'.\, x}$$

By the IH: $[x := s]t_i \Rrightarrow [x := s']t_i'$ for $i = 1, 2$. The ParSubst rule gives: $[x := s](\psi(\text{refl}(t_1), t_2)) = \psi(\text{refl}([x := s]t_1), t_2) \Rrightarrow \lambda_\omega\, x : [x := s']t_2' \bullet_\tau [x := s']t_1'.\, x = [x := s']\lambda_\omega\, x : t_2' \bullet_\tau t_1'.\, x$.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'}}{[t_1, t_2; t_3].1 \Rrightarrow t_1'}$$

By the IH: $[x := s]t_i \Rrightarrow [x := s']t_i'$ for $i = 1, 2, 3$. The ParFst rule gives: $[x := s][t_1, t_2; t_3].1 = [[x := s]t_1, [x := s]t_2; [x := s]t_3].1 \Rrightarrow [x := s']t_1'$.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'}}{[t_1, t_2; t_3].2 \Rrightarrow t_2'}$$

Similar to previous case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'}}{\vartheta_1(\text{refl}(t_1), t_2, t_3) \Rrightarrow \text{refl}(t_2')}$$

By the IH: $[x := s]t_i \Rrightarrow [x := s']t_i'$ for $i = 1, 2, 3$. The ParFst rule gives: $[x := s]\vartheta_1(\text{refl}(t_1), t_2, t_3) = \vartheta_1(\text{refl}([x := s]t_1), [x := s]t_2, [x := s]t_3) \Rrightarrow \text{refl}([x := s']t_2') = [x := s']\text{refl}(t_2')$.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'}}{\vartheta_2(\text{refl}(t_1), t_2, t_3) \Rrightarrow \text{refl}(t_2')}$$

Similar to previous case.

□

Lemma A.7 (Parallel Triangle)**.** *If $s \Rrightarrow t$ then $t \Rrightarrow (\!|s|\!)$*

Proof. By induction on $s \Rrightarrow t$.

Case: $$\frac{}{x \Rrightarrow x}$$

Have $(\!|x|\!) = x$. Thus, this case is trivial.

Case: $$\frac{\overset{\mathcal{D}_i}{t_i \Rrightarrow t_i' \quad \forall\, i \in \{1, \ldots, \mathfrak{a}(\kappa)\}}}{\mathfrak{c}(\kappa, t_1, \ldots, t_i, \ldots, t_{\mathfrak{a}(\kappa)}) \Rrightarrow \mathfrak{c}(\kappa, t_1', \ldots, t_i', \ldots, t_{\mathfrak{a}(\kappa)}')}$$

By the IH applied to $\mathcal{D}_i$: $t_i' \Rrightarrow (\!|t_i|\!)$ for all i. Proceed by cases of $(\!|\mathfrak{c}(\kappa, t_1, \ldots t_{\mathfrak{a}(\kappa)})|\!)$.

Case: $(\!|(\lambda_m\, x : t_1.\, t_2) \bullet_m t_3|\!) = [x := (\!|t_3|\!)](\!|t_2|\!)$

Note that $\mathfrak{c}(\kappa, t_1', \ldots t_{\mathfrak{a}(\kappa)}') = (\lambda_m\, x : t_1'.\, t_2') \bullet_m t_3'$. Using the ParBeta rule yields $(\lambda_m\, x : t_1'.\, t_2') \bullet_m t_3' \Rrightarrow [x := (\!|t_3|\!)](\!|t_2|\!)$.

Case: $(\!|\psi(\text{refl}(t_1), t_2)|\!) = \lambda_\omega\, x : (\!|t_2|\!) \bullet_\tau (\!|t_1|\!).\, x$

Note that $\mathfrak{c}(\kappa, t'_1, \ldots t'_{\mathfrak{a}(\kappa)}) = \psi(\text{refl}(t'_1), t'_2)$. Using the ParSubst rule yields $\psi(\text{refl}(t'_1), t'_2) \Rrightarrow \lambda_\omega\, x : (\!|t_2|\!) \bullet_\tau (\!|t_1|\!) .\, x$.

Case: $(\!|[t_1, t_2; t_3].1|\!) = (\!|t_1|\!)$

Note that $\mathfrak{c}(\kappa, t'_1, \ldots t'_{\mathfrak{a}(\kappa)}) = [t'_1, t'_2; t'_3].1$. Using the ParFst rule yields $[t'_1, t'_2; t'_3].1 \Rrightarrow (\!|t_1|\!)$.

Case: $(\!|[t_1, t_2; t_3].2|\!) = (\!|t_2|\!)$

Similar to previous case.

Case: $(\!|\vartheta_1(\text{refl}(t_1), t_2, t_3)|\!) = (\!|t_2|\!)$

Note that $\mathfrak{c}(\kappa, t'_1, \ldots t'_{\mathfrak{a}(\kappa)}) = \vartheta_1(\text{refl}(t'_1), t'_2, t'_3)$. Using the ParPrmFst rule yields $\vartheta_1(\text{refl}(t'_1), t'_2, t'_3) \Rrightarrow (\!|t_2|\!)$.

Case: $(\!|\vartheta_2(\text{refl}(t_1), t_2, t_3)|\!) = (\!|t_2|\!)$

Similar to previous case.

Case: $(\!|\mathfrak{c}(\kappa, t_1, \ldots t_{\mathfrak{a}(\kappa)})|\!) = \mathfrak{c}(\kappa, (\!|t_1|\!), \ldots (\!|t_{\mathfrak{a}(\kappa)}|\!))$

Using the ParCtor rule concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2}}{\mathfrak{b}(\kappa, x : t_1, t_2) \Rrightarrow \mathfrak{b}(\kappa, x : t'_1, t'_2)}$$

Note that $(\!|\mathfrak{b}(\kappa, (x : t_1), t_2)|\!) = \mathfrak{b}(\kappa, (x : (\!|t_1|\!)), (\!|t_2|\!))$. By the IH applied to $\mathcal{D}_i$: $t'_i \Rrightarrow (\!|t_i|\!)$ for $i = 1, 2$. Thus, by the ParBind rule $\mathfrak{b}(\kappa, (x : t'_1), t'_2) \Rrightarrow \mathfrak{b}(\kappa, (x : (\!|t_1|\!)), (\!|t_2|\!))$.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t'_3}}{(\lambda_m\, x : t_1 .\, t_2) \bullet_m t_3 \Rrightarrow [x := t'_3]t'_2}$$

Note that $(\!|(\lambda_m\, x : t_1.\, t_2) \bullet_m t_3|\!) = [x := (\!|t_3|\!)](\!|t_2|\!)$. By the IH applied to $\mathcal{D}_i$: $t'_i \Rrightarrow (\!|t_i|\!)$ for $i = 1, 2, 3$. Thus, by Lemma A.6 $[x := t'_3]t'_2 \Rrightarrow [x := (\!|t_3|\!)](\!|t_2|\!)$.

Case: $$\dfrac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2}}{\psi(\text{refl}(t_1), t_2) \Rrightarrow \lambda_\omega\, x : t'_2 \bullet_\tau t'_1.\, x}$$

Note that $(\!|\psi(\text{refl}(t_1), t_2)|\!) = \lambda_\omega\, x : (\!|t_2|\!) \bullet_\tau (\!|t_1|\!).\, x$. By the IH applied to $\mathcal{D}_i$: $t'_i \Rrightarrow (\!|t_i|\!)$ for $i = 1, 2$. Applying the ParBind rule yields $\lambda_\omega\, x : t'_2 \bullet_\tau t'_1.\, x \Rrightarrow \lambda_\omega\, x : (\!|t_2|\!) \bullet_\tau (\!|t_1|\!).\, x$.

Case: $$\dfrac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t'_3}}{[t_1, t_2; t_3].1 \Rrightarrow t'_1}$$

Note that $(\!|[t_1, t_2; t_3].1|\!) = (\!|t_1|\!)$. By the IH applied to $\mathcal{D}_i$: $t'_i \Rrightarrow (\!|t_i|\!)$ for $i = 1, 2, 3$. Thus, $t'_1 \Rrightarrow (\!|t_1|\!)$.

Case: $$\dfrac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t'_3}}{[t_1, t_2; t_3].2 \Rrightarrow t'_2}$$

Similar to previous case.

Case: $$\dfrac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t'_3}}{\vartheta_1(\text{refl}(t_1), t_2, t_3) \Rrightarrow \text{refl}(t'_2)}$$

Note that $(\!|\vartheta_1(\text{refl}(t_1), t_2, t_3)|\!) \Rrightarrow (\!|t_2|\!)$. By the IH applied to $\mathcal{D}_i$: $t'_i \Rrightarrow (\!|t_i|\!)$ for $i = 1, 2, 3$. Thus, $t'_2 \Rrightarrow (\!|t_2|\!)$.

Case: $$\dfrac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t'_1} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t'_2} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t'_3}}{\vartheta_2(\text{refl}(t_1), t_2, t_3) \Rrightarrow \text{refl}(t'_2)}$$

Similar to previous case.

$\square$

Lemma A.8 (Paralell Strip). *If $s \Rrightarrow t_1$ and $s \Rrightarrow^* t_2$ then $\exists\, t$ such that $t_1 \Rrightarrow^* t$ and $t_2 \Rrightarrow t$*

Proof. By induction on $s \Rrightarrow^* t_2$, pick $t = t_1$ for the reflexivity case. Consider the transitivity case, $\exists\, z_1$ such that $s \Rrightarrow z_1$ and $z_1 \Rrightarrow^* t_2$. Applying Lemma A.7 to $s \Rrightarrow z_1$ yields $z_1 \Rrightarrow (\!|s|\!)$. By the IH with $z_1 \Rrightarrow (\!|s|\!)$: $\exists\, z_2$ such that $(\!|s|\!) \Rrightarrow^* z_2$ and $t_2 \Rrightarrow z_2$. Using Lemma A.7 again on $s \Rrightarrow t_1$ yields $t_1 \Rrightarrow (\!|s|\!)$. Now by transitivity $t_1 \Rrightarrow^* z_2$. □

Lemma A.9 (Parallel Confluence). *If $s \Rrightarrow^* t_1$ and $s \Rrightarrow^* t_2$ then $\exists\, t$ such that $t_1 \Rrightarrow^* t$ and $t_2 \Rrightarrow^* t$*

Proof. By induction on $s \Rrightarrow^* t_1$, pick $t = t_2$ for the reflexivity case. Consider the transitivity case, $\exists\, z_1$ such that $s \Rrightarrow z_1$ and $z_1 \Rrightarrow^* t_1$. By Lemma A.8 applied with $s \Rrightarrow z_1$ and $s \Rrightarrow^* t_2$ yields $\exists\, z_2$ such that $z_1 \Rrightarrow^* z_2$ and $t_2 \Rrightarrow z_2$. Using the IH with $z_1 \Rrightarrow z_2$ gives $\exists\, z_3$ such that $t_1 \Rrightarrow^* z_3$ and $z_2 \Rrightarrow^* z_3$. By transitivity $t_2 \Rrightarrow^* z_3$. □

Lemma A.10 (Confluence). *If $s \rightsquigarrow^* t_1$ and $s \rightsquigarrow^* t_2$ then $\exists\, t$ such that $t_1 \rightsquigarrow^* t$ and $t_2 \rightsquigarrow^* t$*

Proof. By Lemma A.3 applied twice: $s \Rrightarrow^* t_1$ and $s \Rrightarrow^* t_2$. Now by parallel confluence (Lemma A.9) $\exists\, t$ such that $t_1 \Rrightarrow^* t$ and $t_2 \Rrightarrow^* t$. Finally, two applications of Lemma A.5 conclude the proof. □

BIBLIOGRAPHY

[1] Andreas Abel and Thierry Coquand. “Failure of normalization in impredicative type theory with proof-irrelevant propositional equality”. In: *Logical Methods in Computer Science* 16 (2020).

[2] Stuart F Allen et al. “The Nuprl open logical environment”. In: *Automated Deduction-CADE-17: 17th International Conference on Automated Deduction Pittsburgh, PA, USA, June 17-20, 2000. Proceedings 17*. Springer. 2000, pp. 170–176.

[3] Aristotle. *Analytica Priora et Posteriora*. Oxford University Press, 1981. ISBN: 9780198145622.

[4] HENK BARENDREGT. “Introduction to generalized type systems”. In: *Journal of Functional Programming* 1.2 (1991), pp. 125–154.

[5] Henk Barendregt and Kees Hemerik. “Types in lambda calculi and programming languages”. In: *ESOP’90: 3rd European Symposium on Programming Copenhagen, Denmark, May 15–18, 1990 Proceedings 3*. Springer. 1990, pp. 1–35.

[6] Oliver Byrne. *Oliver Byrne’s Elements of Euclid*. Art Meets Science, 2022. ISBN: 978-1528770439.

[7] Arthur Charguéraud. “The locally nameless representation”. In: *Journal of automated reasoning* 49 (2012), pp. 363–408.

[8] Alonzo Church. “A formulation of the simple theory of types”. In: *The journal of symbolic logic* 5.2 (1940), pp. 56–68.

[9] Alonzo Church. “A set of postulates for the foundation of logic”. In: *Annals of mathematics* (1932), pp. 346–366.

[10] Alonzo Church. “A set of postulates for the foundation of logic”. In: *Annals of mathematics* (1933), pp. 839–864.

[11] Thierry Coquand. “Une théorie des constructions”. PhD thesis. Universiteé Paris VII, 1985.

[12] Thierry Coquand and Gérard Huet. *The calculus of constructions*. Tech. rep. RR-0530. INRIA, May 1986. URL: https://hal.inria.fr/inria-00076024.

[13] Nicolaas Govert De Bruijn. “Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem”. In: *Indagationes Mathematicae (Proceedings)*. Vol. 75. 5. Elsevier. 1972, pp. 381–392.

[14] Denis Firsov, Richard Blair, and Aaron Stump. “Efficient Mendler-style lambda-encodings in Cedille”. In: *Interactive Theorem Proving: 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings 9*. Springer. 2018, pp. 235–252.

[15] Robert W Floyd. “A descriptive language for symbol manipulation”. In: *Journal of the ACM (JACM)* 8.4 (1961), pp. 579–584.

[16] Gottlob Frege. “Begriffsschrift, a Formula Language, Modeled upon that of Arithmetic, for Pure Thought [1879]”. In: *From Frege to Gödel: A Source Book in Mathematical Logic* 1931 (1879).

[17] Gerhard Gentzen. “Untersuchungen über das logische schließen. I.” In: *Mathematische zeitschrift* 35 (1935).

[18] Gerhard Gentzen. “Untersuchungen über das logische Schließen. II.” In: *Mathematische zeitschrift* 39 (1935).

[19] Herman Geuvers. “A short and flexible proof of strong normalization for the calculus of constructions”. In: *International Workshop on Types for Proofs and Programs*. Springer. 1994, pp. 14–38.

[20] Herman Geuvers. “Induction is not derivable in second order dependent type theory”. In: *International Conference on Typed Lambda Calculi and Applications*. Springer. 2001, pp. 166–181.

[21] Herman Geuvers and Mark-Jan Nederhof. “Modular proof of strong normalization for the calculus of constructions”. In: *Journal of Functional Programming* 1.2 (1991), pp. 155–189.

[22] Jean-Yves Girard. “Interprétation fonctionnelle et élimination des coupures de l’arithmétique d’ordre supérieur”. PhD thesis. Universiteé Paris VII, 1972.

[23] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Vol. 7. Cambridge university press Cambridge, 1989.

[24] William A Howard. “The formulae-as-types notion of construction”. In: *To HB Curry: essays on combinatory logic, lambda calculus and formalism* 44 (1980), pp. 479–490.

[25] Antonius JC Hurkens. “A simplification of Girard’s paradox”. In: *Typed Lambda Calculi and Applications: Second International Conference on Typed Lambda Calculi and Applications, TLCA’95 Edinburgh, United Kingdom, April 10–12, 1995 Proceedings 2*. Springer. 1995, pp. 266–278.

[26] A. Kopylov. “Dependent intersection: a new way of defining records in type theory”. In: *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.* 2003, pp. 86–95. DOI: 10.1109/LICS.2003.1210048.

[27] Meven Lennon-Bertrand. “Complete Bidirectional Typing for the Calculus of Inductive Constructions”. In: *ITP 2021-12th International Conference on Interactive Theorem Proving.* Vol. 193. 24. 2021, pp. 1–19.

[28] *Liquid Tensor Experiment.* https://github.com/leanprover-community/lean-liquid. 2022.

[29] Per Martin-Löf. “An Intuitionistic Theory of Types: Predicative Part”. In: *Logic Colloquium ’73.* Ed. by H.E. Rose and J.C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. Elsevier, 1975, pp. 73–118. DOI: https://doi.org/10.1016/S0049-237X(08)71945-1.

[30] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory.* Vol. 9. Bibliopolis Naples, 1984.

[31] Alexandre Miquel. “The Implicit Calculus of Constructions Extending Pure Type Systems with an Intersection Type Binder and Subtyping”. In: *Typed Lambda Calculi and Applications.* Ed. by Samson Abramsky. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 344–359. ISBN: 978-3-540-45413-7.

[32] Leonardo de Moura and Sebastian Ullrich. “The lean 4 theorem prover and programming language”. In: *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28.* Springer. 2021, pp. 625–635.

[33] C-HL Ong and Eike Ritter. “A generic Strong Normalization argument: application to the Calculus of Constructions”. In: *International Workshop on Computer Science Logic.* Springer. 1993, pp. 261–279.

[34] Christine Paulin-Mohring. “Inductive definitions in the system Coq rules and properties”. In: *Typed Lambda Calculi and Applications.* Ed. by Marc Bezem and Jan Friso Groote. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 328–345. ISBN: 978-3-540-47586-6.

[35] Giuseppe Peano. *Arithmetices principia: Nova methodo exposita.* Fratres Bocca, 1889.

[36] Frank Pfenning and Christine Paulin-Mohring. “Inductively defined types in the Calculus of Constructions”. In: *Mathematical Foundations of Programming Semantics.* Ed. by M. Main et al. New York, NY: Springer-Verlag, 1990, pp. 209–228. ISBN: 978-0-387-34808-7.

[37] Frank Pfenning and Christine Paulin-Mohring. “Inductively defined types in the Calculus of Constructions”. In: *Mathematical Foundations of Programming Semantics: 5th International Conference Tulane University, New Orleans, Louisiana, USA March 29–April 1, 1989 Proceedings 5.* Springer. 1990, pp. 209–228.

[38] Andrew M Pitts. *Nominal sets: Names and symmetry in computer science.* Cambridge University Press, 2013.

[39] Andrew M. Pitts. “Locally Nameless Sets”. In: *Proc. ACM Program. Lang.* 7.POPL (2023). DOI: `10.1145/3571210`. URL: `https://doi.org/10.1145/3571210`.

[40] John C Reynolds. “Towards a theory of type structure”. In: *Programming Symposium: Proceedings, Colloque sur la Programmation Paris, April 9–11, 1974.* Springer. 1974, pp. 408–425.

[41] John C Reynolds. “Types, abstraction and parametric polymorphism”. In: *Information Processing 83, Proceedings of the IFIP 9th World Computer Congres.* 1983, pp. 513–523.

[42] Dana Scott. “Constructive validity”. In: *Symposium on automatic demonstration.* Springer. 1970, pp. 237–275.

[43] Aaron Stump. “The calculus of dependent lambda eliminations”. In: *Journal of Functional Programming* 27 (2017), e14.

[44] Aaron Stump and Christopher Jenkins. *Syntax and Semantics of Cedille.* 2021. arXiv: `1806.04709 [cs.PL]`.

[45] Jan Terlouw. “Strong normalization in type systems: A model theoretical approach”. In: *Annals of Pure and Applied Logic* 73.1 (1995), pp. 53–78.

[46] *The Polynomial Freiman-Ruzsa Conjecture.* `https://github.com/teorth/pfr`. 2024.

[47] Philip Wadler, Wen Kokke, and Jeremy G. Siek. *Programming Language Foundations in Agda.* Aug. 2022. URL: `https://plfa.inf.ed.ac.uk/22.08/`.

[48] Alfred North Whitehead and Bertrand Russell. “Principia Mathematica”. In: (1927).