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

Thesis Committee:	Aaron Stump, Thesis Supervisor
	Cesare Tinelli
	J. Garrett Morris
	Sriram Pemmaraju
	William J. Bowman

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

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 [72, 73]. Soon after in 1985, the Calculus of Constructions (CC) was introduced by Coquand [36, 37]. Inductive data (e.g. natural numbers, lists, trees) was shown by Guevers to be impossible to derive in CC [49]. Nevertheless, inductive data was added as an extension by Pfenning [80] and the Calculus of Inductive Constructions (CIC) became the basis for the proof assistant Rocq [77].

In the early 1990s Barendregt introduced a generalization to Pure Type Systems (PTS) and studied CC under his now famous λ-cube [16, 15]. 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 [29]. The STLC has only basic functions found in all programming languages. System F, a type theory introduced by Girard [52, 53] and independently by Reynolds [86], 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 [56]. Reynolds and Girard's combined work demonstrated that System F corresponds to second-order intuitionistic propositional logic [52, 86, 87]. 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 [15]. 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

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

$$\mathfrak{a}(\star) = \mathfrak{a}(\square) = 0 \qquad \lambda\, x : t_1.\, t_2 := \mathfrak{b}(\lambda, x : t_1, t_2) \qquad \star := \mathfrak{c}(\star)$$

$$\mathfrak{a}(\text{app}) = 2 \qquad (x : t_1) \to t_2 := \mathfrak{b}(\Pi, x : t_1, t_2) \qquad \square := \mathfrak{c}(\square)$$

$$t_1\ t_2 := \mathfrak{c}(\text{app}, t_1, t_2)$$

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

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 [94, 96]. 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 [74], the dependent intersections of Kopylov [63], 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 [41]. 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 [88]. 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.

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 $(x, y, z, \ldots)$, 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 [43] where $t ::= f(t_1, t_2, \ldots)$ represents a recursive definition defining a category of

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

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

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

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: $\lambda\, x\!:\!A.\, t$. 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) [38], locally-nameless representations [27], nominal sets [82], locally-nameless sets [84], 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 $\lambda\, x\!:\![x := t]A.\, [z := t](y\ x)$.

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.

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

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

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

Figure 1.3: Reduction rules for System F$^\omega$.

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

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

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

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$^\omega$ 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. 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 **??** 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;

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

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

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

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

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

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

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

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

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

The axiom rule has one premise, requiring that the context is well-formed. It concludes that the constant term $\star$ has type $\Box$. 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 $\Box$ 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 [52, 57]. Note that there is no way to determine a type for $\Box$. 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 [65]. 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. □

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 [48], model theory [97], realizability [76], etc. □

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 [30, 31]. 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. □

Recall that induction principles cannot be derived internally for any encoding of data [49]. 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 [13]. 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 [23]. However, the study of proof as a mathematical object dates first to Frege [44] followed soon after by Peano's formalism of arithmetic [78] and Whitehead and Russell's *Principia Mathematica* [104]. For the kinds of logics discussed by the Curry-Howard correspondence, structural proof theories, the originator is Gentzen [46, 47]. 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 [3], and the proof assistant Lean with its notion of casts [75]. Moreover, Lean has been incredibly successful in formalizing research mathematics including the Liquid Tensor Experiment [67] and Tao's formalization of The Polynomial Freiman-Ruzsa Conjecture [98]. 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 Equality

A type theory is *intensional* if its propositional equality is able to observe the operational definition of a function. Within mathematics, functions are defined as a functional relation between their inputs and outputs, but this definition lacks any computational content. In contrast, dependent type theories are constructive in nature, a function is not merely a functional relation but a stronger notion of an operational term encoding the procedure to produce outputs from inputs. This additional structure prevents equality of functions from being defined pointwise in situations where the extra structure is observable. The below variant of Martin-Löf's intensional identity type is a canonical example of an intensional propositional equality.

Note that the variation of the intensional identity type does matter. In dependent type theories with an impredicative universe of types Coquand et al. showed that an identity type with a proof-irrelevant cast refutes strong normalization [1]. This is an instance of propositional equality being sensitive to other features of a type theory.

Function extensionality, the formal statement of which is shown below, is a reasoning principle that is often lost in the intensional setting.

$$\Pi\, f, g : X \to Y.\, (\Pi\, x : X.\, Id_Y(f\ x, g\ x)) \to Id_{X \to Y}(f, g)$$

Postulating function extensionality as an axiom breaks canonicity but as a reasoning principle, particularly when formalizing mathematics, it is significantly more convenient. Absolving the tension between a dependent type theory with good metatheoretical properties, which intensional propositional equality gives, and a theory that admits function extensionality has and is a major goal of current research. The additional constructive information supplied to define a function in dependent type theory gets in the way of strong reasoning principles. Thus, one might say that equality is about what properties of an object one wishes to *ignore* as opposed to some philosophically ordained notion. However, the intensional information of a function is also potentially important. For computer science, the intensional behavior may be critical to security properties or computational complexity (e.g. it may be undesirable to equate merge sort with bubble sort).

Outside of dependent type theory a common definition of equality is Leibniz equality, which equates two objects only if any predicate that holds of one object necessarily holds of the other. Leibniz equality can be viewed as a Church-encoded version of the identity type which means that these different definitions of propositional equality imply one another. However, they are only isomorphic if quantification is parametric and function extensionality holds [2].

Attempting to bridge the gap between intensional and extensional features Streicher proposed his Axiom K in 1993 [93]. Today this axiom is better known as Uniqueness of Identity Proofs (UIP), specified below, but which intuitively means that all proofs of an equality are themselves equal.

$$\Pi\, x, y : X.\, \Pi\, p, q : Id_X(x, y).\, Id_{Id_X(x,y)}(p, q)$$

For many years it was an open question whether or not the intensional identity type always satisfied UIP as it was generally believed there is only one equality proof, the reflexivity proof.

In 1995 Hofmann answered this equation negatively: intensional identity types need not always satisfy UIP, but he did not stop there. Indeed, Hofmann investigated how many extensional notions (function extensionality, quotient types, subset types, UIP, and propositional extensionality) might be added to dependent type theory. To accomplish this goal he modelled identity types in two separate ways. First, as equivalence relations defined inductively on type structure, a precursor to the setoid model of type theory. Second, as groupoids, a precursor to the univalent model of type theory. Unfortunately, his models did not achieve all of his goals where certain metatheoretical properties and desirable definitional equalities are lost depending on the interpretation [54, 55]. Regardless, his contributions broke through a long-standing assumption and opened a world of varying interpretations of intensional propositional equality.

Propositional extensionality means that equivalence of propositions is equivalent to equality of propositions. Stated formally below, equivalence in this context means mutual implication. Propositions are assumed to be distinct from types (i.e. propositions are usually assumed to satisfy some variation of proof-irrelevance). The universe Prop captures this distinction between the more general universe of types and those interpreted as propositions.

$$\forall\, P, Q : \textsc{Prop}.\, (P \leftrightarrow Q) \leftrightarrow (P = Q)$$

Quotient types build a type $A/\!\sim$ from a carrier type A and an equivalence relation on that carrier type $\sim$ such that the propositional equality for elements of $A/\!\sim$ respects the equivalence relation. Mathematical practice utilizes quotients extensively to construct objects from some carrier set and equations on the elements of that set. For example, the simplest mathematical quotient is the set of rational numbers which is the quotient of fractions (i.e. pairs of integers) and the equivalence relation $\forall\,(n_1, d_1), (n_2, d_2) : \mathbb{Z} \times \mathbb{Z}.\, n_1 d_2 = n_2 d_1$. Many algebraic objects are also quotients, such as groups, fields, modules, and tensor products of modules to name a few.

The equivalence relation for a quotient type (or quotient set) induces a partition of the elements of that type into equivalence classes. If a canonical representative of an equivalence class can be effectively computed then quotient types are definable in most dependent type theories. It is the other variant, where a canonical representative can not be computed, that is problematic to encode. The real numbers and multi-sets of elements that are not orderable are two examples of quotient types where a canonical representative is uncomputable [66]. Indeed, the set-theoretic axiom of choice is equivalent to the ability to pick a unique representative from equivalence classes for any arbitrary equivalence relation [71]. Additionally, if a type theory supports an impredicative universe of types, then adding quotients such as the real numbers causes inconsistency [28].

Quotients is an important feature of dependent type theories. Indeed, some theories are designed, at least partially, to explicitly support quotients. The Lean proof assistant in particular incorporates quotients axiomatically in its standard library, a choice that is arguably responsible for mathematicians interest in it as a tool for formalizing theorems (if only because it more closely matches a culture that mathematicians expect) [25]. Quotient inductive types is one technique for adding quotients to type theory which extends inductive definitions with equational variants that the constructors must satisfy. With quotient inductive types the integers can be defined succinctly as the following inductive type.

$$\begin{aligned}\mathbb{Z} := {} & 0 : \mathbb{Z} \\ & \mathit{succ} : \mathbb{Z} \to \mathbb{Z} \\ & \mathit{pred} : \mathbb{Z} \to \mathbb{Z} \\ & \mathit{succ\_pred} : \forall\, i {:} \mathbb{Z}.\, \mathit{succ}\ (\mathit{pred}\ i) = i \\ & \mathit{pred\_succ} : \forall\, i {:} \mathbb{Z}.\, \mathit{pred}\ (\mathit{succ}\ i) = i\end{aligned}$$

Quotient inductive definitions were given a formal foundation in 2007 and demonstrated to be useful for direct internalization of dependent type theories [6, 40].

Extensional Type Theories

Extensional type theory does not have any of the aforementioned extensionality problems, as its namesake might suggest. The distinguishing feature that transforms an intensional theory into an extensional one is the equality reflection rule, listed below. It is difficult to pin an exact year when equality reflection is first introduced but, some of Martin-Löf's early systems are extensional type theories and likewise some of the earliest proof assistant implementations were extensional type theories (e.g. Nuprl). Equality reflection collapses definitional and propositional equality. If propositional equality is undecidable then definitional equality becomes undecidable as a consequence. Thus, even though a proof assistant such as Nuprl supports quotient types, subset types, and other extensionality properties it lacks decidability of conversion checking. To counteract these restrictions users of Nuprl work with derivation trees as opposed to proof terms [35].

This trade-off, between extensional properties and good metatheoretical properties has lead the design of dependent type theories away from full-blown equality reflection. Instead, designers have attempted to leverage other techniques, interpreting types as setoids or groupoids, or adding ad hoc equality reflection via rewriting. However, in 2016 Andrej Bauer demonstrated that equality reflection could be designed around effect handlers. The undecidability of equality reflection is sidestepped as the provided handler resolves the proof obligations generated by definitional equality by what is effectively a try-catch exception handler [17].

More recently, in 2019, it was shown that extensional type theory can be modelled in an intensional type theory that satisfies UIP and function extensionality [105]. This result was later strengthened to modelling extensional type theory in a weak theory [21]. A type theory is *weak* if β and η convertibility rules have to be expressed in the syntax meaning that definitional equality is α-equivalence. These results argue that extensional dependent type theory *is* intensional type theory with UIP and function extensionality. Especially because equality reflection hampers any procedure to automatically decide definitional equalities anyway, making the reduction to a weak intensional type theory moot.

Bishop, in 1967, showed that much of mathematical analysis could be carried out in a constructive setting. His constructive analysis heavily leveraged "Bishop sets" or *setoids* which incorporate a carrier set of elements together with an equivalence relation about those sets [20]. As mentioned previously, setoids were used by Hofmann to construct models of type theory that supported some extensional properties. Although Hofmann's models did not capture all the desired properties for an intensional type theory with extensional features the idea of modelling types as setoids persisted.

Altenkirch in particular has heavily invested in dependent type theories in this style. In 1999, Altenkirch improved Hofmann's setoid model to a type theory that admits large eliminations, enjoys function extensionality, has decidable definitional equality, and satisfies canonicity. Moreover, he stresses that the metatheory used to accomplish this model is itself an *intensional* one, whereas Hofmann's metatheory was extensional. Altenkirch accomplished this task by incorporating a proof irrelevant universe of propositions which will be a common theme for type theories supporting extensional features. However, his 1999 attempt did not contain dependent types or quotients though Altenkirch claimed quotients could be easily added [4].

Observational Type Theory (OTT) was introduced years later (in 2007) by Altenkirch and McBride. The core ideas remained the same: semantically types are interpreted as setoids and universe of propositions is added to an intensional type theory. What OTT introduces, however, is defining propositional equality by recursion on type constructors. This recursive definition grants greater flexibility in how equality is treated for individual type formers. Elimination of propositional equality is additionally defined to act via coercions which are also defined recursively on type structure. Critically, propositional equality for function types (and coercions related to it) can be defined to admit pointwise equality directly. Unfortunately, the definition of OTT makes the connection to the setoid model less clear which prevents an obvious addition of quotient types and a universe of types [7].

In 2022, Pujet et al. introduce an improvement of OTT and move past many of the limitations of the prior work. The reduction of definitional equality is altered to reduce terms to weak head normal form as opposed to a full normal form. Stopping at weak head normal form allows the head constructor to determine the corresponding propositional equality rule. Thus, the recursive definition of propositional equality that was the namesake of OTT is lifted to standard typing rules. In the case of function types, this means that pointwise equality can be directly defined as the typing rule for the identity type. Therefore, function extensionality is baked into the typing judgments and there is no way for the intensional difference of functions to be observed by the identity type. Like prior developments, the techniques of Pujet et al. hinge on a proof-irrelevant universe of propositions. However, the authors note that a critical difference from prior attempts at OTT is that propositions satisfy definitional proof-irrelevance (as opposed to propositional proof-irrelevance) which prevents equational proof obligations muddying goals of type judgments. Moreover, many desirable properties are proven including: propositional extensionality, UIP, strong normalization, consistency, and decidability of type checking. Quotient types are also added provided the equivalence relation is proof-irrelevant (in other words, provided the equivalence relation is expressed in terms of the universe of propositions) [85].

Setoid Type Theory (SeTT), meanwhile, is the evolution of Altenkirch's work on OTT.

Based on the original root idea by Altenkirch of constructing a setoid model directly in an intensional type theory as he did in 1999, the first introduction of SeTT (in 2019) improved on the first attempt by making function extensionality, propositional extensionality, and the elimination principle for the identity type hold definitionally. Indeed, the new model is noted to be a *syntactic translation* from a setoid type theory to an intensional type theory. A critical property of syntactic translations is that definitional equalities are all preserved [9]. Syntactic translations, in the words of Altenkirch, give a way to bootstrap extensionality principles from intensional type theories. The syntactic translation is later improved to internalize a universe of setoids (i.e. a universe of types) [8].

Univalent Type Theories

While setoid type theory was and continues to be explored there is a competing interpretation of types foreshadowed by Hofmann as groupoids. A *groupoid* is a collection of objects and invertible morphisms between those objects that satisfy identity and associativity laws. Setoids are a special case of groupoids. There is a further generalization to n-groupoids that allows invertible 2-morphisms with morphisms as objects, invertible 3-morphisms with 2-morphisms as objects, and so on up-to invertible n-morphisms with $n-1$-morphisms as objects. A limiting process can be applied to this generalization obtaining ω-groupoids that have infinite towers of invertible i-morphisms with $i-1$-morphisms as objects. A groupoid and its generalizations are studied in Category Theory, but for the purposes of modeling dependent type theory the core idea is that propositional equality has a rich proof structure including interesting equality proofs between equality proofs. Of course, as a natural consequence, UIP is refuted in a non-degenerate groupoid model.

In 2006, the mathematician Voevodsky began studying type theory has an alternative foundation for mathematics after expressing doubts about the correctness of results in the mathematical literature. He proposed the Univalence Axiom, shown below, as a desirable feature of dependent type theory because, in his opinion, it accurately modelled the transport of properties between objects that mathematicians take for granted [102]. From this point forward, a *univalent* type theory is any type theory that admits univalence either as an axiom or as a derivable notion. Upon closer inspection of the Univalence Axiom it becomes clear that it is a generalization of propositional extensionality. Where propositional extensionality states an equivalence between equivalent propositions and equal propositions, the Univalence Axiom states that an isomorphism between types is isomorphic to an equality between types.

$$\forall\, A, B\!:\!\star.\, (A \simeq B) \simeq (A = B)$$

Homotopy Type Theory (HoTT) is one of the first univalent theories proposed from these observations. HoTT interprets the intensional identity type as consisting of homotopy equivalences between types and terms. A *homotopy* between two functions $f, g : X \to Y$ is a continuous map $h : \mathbb{I} \to X \to Y$ such that $h(0) = f$ and $h(1) = g$ and where $\mathbb{I} = [0, 1]$ is the unit interval. It is a mathematical device to express the smooth deformation of one function to another. Of course, homotopies are generalizable beyond the case of first-order functions. However, notice that if the identity type consists of homotopies then the metatheory of HoTT necessarily appeals to the existence of a set of real numbers (i.e. $\mathbb{I} \subseteq \mathbb{R}$). Nonetheless, HoTT has been used effectively to build a foundation of mathematics and to work synthetically in the field of Homotopy Theory [99].

The Univalence Axiom was originally just that, an axiom, but this breaks canonicity. Dependent type theories throughout their conception have kept great interest in maintaining a dual purpose: as a programming language *and* as a logic. An initial model construction of univalent type theory in the category of simplicial sets was noted as problematic because of its use of a classical metatheory [62]. Later, Bezem demonstrated that Voevodsky's simplicial sets model is *necessarily* classical [19]. For deciding some metatheoretical properties, such as consistency, a classical model is sufficient, but as a road to providing a constructive derivation of the Univalence Axiom these models are not helpful. Fortunately, Bezem was also able to show that a model with a constructive metatheory is possible using cubical sets which was an important step towards providing a computational basis to the Univalence Axiom [18].

Cubical type theory provides a computational interpretation to the Univalence Axiom by lifting the unit interval from the semantic domain to syntax. The λ-calculus is extended with two constants 0 and 1 and a theory of *names* representing distinguished points inside the interval. In one such development by Cohen et al. various operations are also assumed to hold on elements of the interval including maximums, minimums, and involutions forming a De Morgan algebra which the authors claim simplify semantic justifications [34]. Soon after, more variations of cubical type theories were considered all tweaking the underlying operations allowed on the unit interval. In 2018, Pitts et al. explored the minimal set of axioms in a topos theoretic setting needed to model different kinds of cubical type theories. Propositional equality, called the path type in cubical type theory, is constructed as functions with domain $\mathbb{I}$. Various axioms are postulated including connectedness, distinctness of end-points, a connection algebra on the names of the interval, and a set of "face-formulas" [83].

However, two years later Cavallo et al. observe that a more minimal axiomatic framework works for achieving the goals of Pitts et al. [26]. With their approach there is no need for the diagonals used by Cohen or the connection algebra used by Pitts. Indeed, the minimal set of axioms are:

$$\begin{aligned}
&\mathrm{ax}_1 : \Pi\, P \mathbin{:} \mathbb{I} \to \star.\, (\Pi\, i \mathbin{:} \mathbb{I}.\, P\ i + \neg(P\ i)) \to (\Pi\, i \mathbin{:} \mathbb{I}.\, P\ i) + (\Pi\, i \mathbin{:} \mathbb{I}.\, \neg(P\ i)) \\
&\mathrm{ax}_2 : \neg(0 = 1)
\end{aligned}$$

The minimal *face formulas* needed is a universe of propositions satisfying the following three rules:

$$\begin{aligned}
(- \text{ is } 0) &: \mathbb{I} \to \Phi & \mathrm{ax}_3 &: \Pi\, i \mathbin{:} \mathbb{I}.\, [(i \text{ is } 0)] = (i = 0) \\
(- \text{ is } 1) &: \mathbb{I} \to \Phi & \mathrm{ax}_4 &: \Pi\, i \mathbin{:} \mathbb{I}.\, [(i \text{ is } 1)] = (i = 1) \\
\vee &: \Phi \to \Phi \to \Phi & \mathrm{ax}_5 &: \Pi\, \varphi, \psi \mathbin{:} \Phi.\, [\varphi \vee \phi] = [\varphi] \vee [\psi]
\end{aligned}$$

where $[\phi]$ is a function from syntactic propositional formulas to propositions. With only the minimal set of axioms the model construction only supports weakened rules of propositional equality, but it serves as a consistent basis to explain extensions to different cubical type theory structures. The work of Cavallo et al. arguably presents the essence of cubical type theory:

1. the interval must be connected, which prevents a discretization and maintains an internal continuity for smooth deformations (i.e. homotopy);

2. the two endpoints must be distinct, which prevents collapsing the interval to a unit type and obliterating the internal structure;

3. and a description of face formulas that encode a simple universe of propositions that allow distinguishing endpoints and disjunctive combination.

Like setoid type theories, cubical type theories also rely on a description of a universe of propositions to encode statements about propositional equality. This shared feature hints at the virality of equality. It is not enough to change the rules of equality, one must control the space that equality can act upon as well.

Univalent type theory has been effective in incorporating extensional features into a dependent type theory without hampering metatheoretical properties. For instance, quotient types are representable when an additional rule truncating the higher-order equality structure is also included [64]. Moreover, a new variant of inductive types named *higher inductive types* generalizes quotient inductive types allowing for the definition of homotopy spaces directly as inductive types [11]. Cubical techniques are also useful in constructing setoid type theories that support definitional UIP by adding the appropriate rules to define a degenerate groupoid model [92]. Finally, cubical type theory has recently been implemented as Cubical Agda which extends the development with records, coinductive types, and dependent pattern matching on higher inductive types [100].

Univalent type theories can have some undesirable side effects from a programmatic perspective. For instance, Hofmann noticed early on that in a groupoid model of types it can be the case that $\mathbb{N} = \mathbb{Z}$ because propositional equality is isomorphism. Altenkirch notes that any construction in a univalent type theory is necessarily stable under homotopy equivalence [5]. While a desirable situation for mathematics, programming languages often want even bit-identical representations to not be considered equal because the types capture external notions. Indeed, Voevodsky's early proposals for extension to dependent type theory contain two separate propositional equalities, one to capture isomorphism and support univalence, and one to keep the strict equalities of the original identity type [101]. Today, Voevodsky's proposed type theory would be called a *two-level* type theory (2LTT).

The core idea behind a two-level type theory is to have two universes of types, an "inner" univalent universe satisfying the univalence axiom, and an "outer" strict universe satisfying UIP. This setup can be viewed as an internalization of the "inner" theories metatheory as the "outer" theory. Indeed, some statements in HoTT are metatheoretical in nature and can not be expressed in HoTT alone. For example, the definition of semi-simplicial types (an object used in Homotopy Theory) in a two-level type theory appeals to a natural number in the "outer" theory as a parameter.

There need not be a subsumption requirement between the "inner" universes and the "outer" universes, only an embedding from the "inner" to the "outer" is required. Annenkov et al. note in their formulation of 2LTT that the context of the type theory is shared between both theories though the authors concede that this does not need to be the case. However, critically the dependent function and dependent pair types agree between both theories allowing for intercommunication in their formulation [12]. Capriotti expands upon the foundational aspects of 2LTT showing a conservativity result with respect to HoTT thus confirming that including the "outer" theory does not break any internal results constructed in the "inner" theory [24].

Angiuli adapts 2LTT when introducing Cartesian Cubical Type Theory to incorporate a strict equality and justify a computational semantics in the same spirit as the one used for Nuprl [10]. Indeed, 2LTT even allows a notion of strict proposition (i.e. a universe of propositions that are definitionally irrelevant). However, Gilbert the originator of strict propositions, notes that if a strict equality is included in the universe of strict propositions then univalence is no longer compatible hinting at some limitations [51].

Miscellaneous Type Theories

Not all dependent type theories fit neatly into the above story. Indeed, Cedille itself does not. To conclude the tour of type theories this subsection reflects on these theories.

Quantitative Type Theory augments any dependent type theory with usage annotations for the corresponding variables in context. These usage annotations restrict how often a given variable is used in the term body. Atkey, improving on prior work by McBride, describes a general intensional quantitative type theory parameterized by a semiring of usage annotations, the most familiar variant being $\{0, 1, \omega\}$ or *erased*, *linear*, and *unrestricted* [14]. Idris 2 is an implementation of quantitative type theory that demonstrates how session types can be effectively used in a setting with usage annotations [22].

Rewriting can be added to dependent type theory in a multitude of different ways to achieve ad hoc equality reflection. Cockx investigates a Rewriting Type Theory in detail in his work where rewriting allows the addition of computational axioms. He notes that rewriting can be used to postulate quotients and higher inductive types. In particular, rewriting is safely added to a dependent type theory by restricting the left hand-side to linear patterns, where rewrite rules must be orthogonal (i.e. a term can not be an instance of two or more rewrites on the left-hand side), and each rewrite satisfies a technical triangle property (to guarantee confluence). With these restrictions the addition of rewrites does not break confluence or consistency while also maintaining a modular framework for adding rewrite rules [32, 33]

Zombie, a close cousin to Cedille, is a dependently typed programming language allowing arbitrary recursion. For definitional equality, Zombie does not beta-reduce expressions to normal forms and instead uses congruence closure. Moreover, because Zombie allows non-terminating values, it uses a call by value reduction strategy to prevent non-terminating proof terms from tricking the type system. Like Cedille, Zombie includes irrelevant function spaces and a heterogeneous equality type. Additionally, it refutes function extensionality [89].

Iso-type systems generalize the notion of an iso-recursive type to any type. In these systems definitional equality is α-equivalence while leveraging cast annotations to recover more general conversion rules. In this respect, an iso-type system is a *weak* type theory with limited automation capabilities in its definitional equality [106].

1.4 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.

1.5 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 inference 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 **??** with an explicit algorithm.

2.1 Syntax and Reduction

Syntax for the system is defined generically as before. See Figure 2.1 for a complete description. The intended meaning of the syntax is as follows:

1. tags λ_m, Π_m and $\bullet_m$ (application) represent the function fragment of syntax parameterized by three separates *modes*, ω (free), 0 (erased), and τ (type-level);

2. tags $\cap$, pair, proj_1, and proj_2 represent dependent intersections (i.e. dependent pairs);

3. tags eq, refl, ψ (substitution), ϑ (promotion), δ (separation), and φ (cast) represent equality.

At the moment raw syntax has no essential meaning beyond its intended one. 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. □

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. $\vartheta(\text{refl}(z; Z), a, b; A) \rightsquigarrow \text{refl}(a; A)$) and an elimination form for equality. Note that conversion is different from a traditional PTS. Convertibility with respect to reduction is written: $t \rightleftharpoons s$. A detailed discussion of conversion is delayed until Section 2.3.

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.

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

$$
\begin{aligned}
&\mathfrak{a}(\diamond) = \mathfrak{a}(\star) = \mathfrak{a}(\square) = 0 &\qquad& \mathfrak{a}(\text{pair}) = \mathfrak{a}(\text{eq}) = \mathfrak{a}(\varphi) = 3 \\
&\mathfrak{a}(\text{proj}_1) = \mathfrak{a}(\text{proj}_2) = \mathfrak{a}(\delta) = 1 && \mathfrak{a}(\vartheta) = 4 \\
&\mathfrak{a}(\bullet_m) = \mathfrak{a}(\text{refl}) = 2 && \mathfrak{a}(\psi) = 5
\end{aligned}
$$

$$
\begin{aligned}
\diamond &:= \mathfrak{c}(\diamond) & [t_1, t_2; A] &:= \mathfrak{c}(\text{pair}, t_1, t_2, A) \\
\star &:= \mathfrak{c}(\star) & t.1 &:= \mathfrak{c}(\text{proj}_1, t) \\
\square &:= \mathfrak{c}(\square) & t.2 &:= \mathfrak{c}(\text{proj}_2, t) \\
\lambda_m\, x{:}A.\, t &:= \mathfrak{b}(\lambda_m, x : A, t) & a =_A b &:= \mathfrak{c}(\text{eq}, a, A, b) \\
(x : A) \to_m B &:= \mathfrak{b}(\Pi_m, x : A, B) & \text{refl}(t; A) &:= \mathfrak{c}(\text{refl}, t, A) \\
(x : A) \cap B &:= \mathfrak{b}(\cap, x : A, B) & \vartheta(e, a, b; T) &:= \mathfrak{c}(\vartheta, e, a, b, T) \\
f \bullet_m a &:= \mathfrak{c}(\bullet_m, f, a) & \varphi(a, b, e) &:= \mathfrak{c}(\varphi, a, b, e, A, T) \\
\psi(e, a, b; A, P) &:= \mathfrak{c}(\psi, e, a, b, A, P) & \delta(e) &:= \mathfrak{c}(\delta, e)
\end{aligned}
$$

Figure 2.1: Generic syntax, there are three constructors, variables, a generic binder, and a generic non-binder. Each is 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.

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

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

$$
\begin{aligned}
(\lambda_m\, x{:}A.\, b) \bullet_m t &\rightsquigarrow [x := t]b \\
[t_1, t_2; A].1 &\rightsquigarrow t_1 \\
[t_1, t_2; A].2 &\rightsquigarrow t_2 \\
\psi(\text{refl}(z; Z), a, b; A, P) \bullet_\omega t &\rightsquigarrow t \\
\vartheta(\text{refl}(z; Z), a, b; T) &\rightsquigarrow \text{refl}(a; T)
\end{aligned}
$$

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

Figure 2.2: Reduction and conversion for arbitrary syntax.

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 $a \rightsquigarrow b$. Second projection is the same as first projection case and omitted.

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

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

Case: $\psi(\text{refl}(z; Z), u, v; A, P) \bullet_\omega b \rightsquigarrow b$

$[x := t]\psi(\text{refl}(z; Z), u, v; A, P) \bullet_\omega b = \psi(\text{refl}([x := t]z; [x := t]Z), [x := t]u, [x := t]v; [x := t]A, [x := t]P) \bullet_\omega [x := t]b \rightsquigarrow [x := t]b$

Case: $\vartheta(\text{refl}(z; Z), u, v; A) \rightsquigarrow \text{refl}(u; A)$

$[x := t]\vartheta(\text{refl}(z; Z), u, v; A) = \vartheta(\text{refl}([x := t]z; [x := t]Z), [x := t]u, [x := t]v; [x := t]A) \rightsquigarrow \text{refl}([x := t]u; [x := t]A) = [x := t]\text{refl}(u; A)$

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

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

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

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

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

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

Let $z = t'$. By the IH applied to $\mathcal{D}_2$: $[x := t]z \rightsquigarrow^* [x := t]b$. By Lemma 2.3 applied to $\mathcal{D}_1$: $[x := t]a \rightsquigarrow [x := t]z$. 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. The $\mathfrak{c}$ case is omitted because it is similar to the $\mathfrak{b}$ case.

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.

□

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

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 $[x := s]a \rightsquigarrow^* [x := t]b$.

□

$$\dfrac{}{x_K \Rrightarrow x_K}\ \textsc{ParVar}$$

$$\dfrac{t_i \Rrightarrow t'_i \quad \forall\, i \in \{1, \dots, \mathfrak{a}(\kappa)\}}{\mathfrak{c}(\kappa, t_1, \dots, t_i, \dots, t_{\mathfrak{a}(\kappa)}) \Rrightarrow \mathfrak{c}(\kappa, t'_1, \dots, t'_i, \dots, t'_{\mathfrak{a}(\kappa)})}\ \textsc{ParCtor}$$

$$\dfrac{t_1 \Rrightarrow t'_1 \qquad t_2 \Rrightarrow t'_2}{\mathfrak{b}(\kappa, x : t_1, t_2) \Rrightarrow \mathfrak{b}(\kappa, x : t'_1, t'_2)}\ \textsc{ParBind}$$

$$\dfrac{t_1 \Rrightarrow t'_1 \qquad t_2 \Rrightarrow t'_2 \qquad t_3 \Rrightarrow t'_3}{(\lambda_m\, x\!:\!t_1.\, t_2) \bullet_m t_3 \Rrightarrow [x := t'_3]t'_2}\ \textsc{ParBeta}$$

$$\dfrac{t_1 \Rrightarrow t'_1 \qquad t_2 \Rrightarrow t'_2 \qquad t_3 \Rrightarrow t'_3 \qquad t_4 \Rrightarrow t'_4 \qquad t_5 \Rrightarrow t'_5 \qquad t_6 \Rrightarrow t'_6 \qquad t_7 \Rrightarrow t'_7}{\psi(\text{refl}(t_1; t_2), t_3, t_4; t_5, t_6) \bullet_\omega t_7 \Rrightarrow t'_7}\ \textsc{ParSubst}$$

$$\dfrac{t_1 \Rrightarrow t'_1 \qquad t_2 \Rrightarrow t'_2 \qquad t_3 \Rrightarrow t'_3}{[t_1, t_2; t_3].1 \Rrightarrow t'_1}\ \textsc{ParFst}$$

$$\dfrac{t_1 \Rrightarrow t'_1 \qquad t_2 \Rrightarrow t'_2 \qquad t_3 \Rrightarrow t'_3}{[t_1, t_2; t_3].2 \Rrightarrow t'_2}\ \textsc{ParSnd}$$

$$\dfrac{t_1 \Rrightarrow t'_1 \qquad t_2 \Rrightarrow t'_2 \qquad t_3 \Rrightarrow t'_3 \qquad t_4 \Rrightarrow t'_4 \qquad t_5 \Rrightarrow t'_5}{\vartheta(\text{refl}(t_1; t_2), t_3, t_4; t_5) \Rrightarrow \text{refl}(t'_3; t'_5)}\ \textsc{ParPrm}$$

Figure 2.3: Parallel reduction rules for arbitrary syntax.

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 $[x := t]b \rightsquigarrow^* [x := z_1]z_2$. □

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 reduction rule is $\vartheta(\text{refl}(t.1)) \rightsquigarrow \text{refl}(t)$, but this breaks confluence. To see why, consider $\vartheta(\text{refl}([x, y; T].1))$. One choice leads to $\vartheta(\text{refl}(x))$, and the other leads to $\text{refl}(x)$. However, these terms are not joinable, hence confluence fails.

2.2 Confluence

The proof of confluence follows the PLFA book [103]. This strategy involves the common technique of defining a parallel reduction variant of the one-step reduction described in Figure 2.2. Parallel

$$\begin{aligned}
\llparenthesis(\lambda_m\, x\!:\!t_1.\, t_2) \bullet_m t_3\rrparenthesis &= [x := \llparenthesis t_3\rrparenthesis]\llparenthesis t_2\rrparenthesis \\
\llparenthesis\psi(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5, t_6) \bullet_\omega t_7\rrparenthesis &= \llparenthesis t_7\rrparenthesis \\
\llparenthesis[t_1, t_2; t_3].1\rrparenthesis &= \llparenthesis t_1\rrparenthesis \\
\llparenthesis[t_1, t_2; t_3].2\rrparenthesis &= \llparenthesis t_2\rrparenthesis \\
\llparenthesis\vartheta(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5)\rrparenthesis &= \mathrm{refl}(\llparenthesis t_3\rrparenthesis; \llparenthesis t_5\rrparenthesis) \\
\llparenthesis\mathfrak{c}(\kappa, t_1, \ldots, t_{\mathfrak{a}(\kappa)})\rrparenthesis &= \mathfrak{c}(\kappa, \llparenthesis t_1\rrparenthesis, \ldots, \llparenthesis t_{\mathfrak{a}(\kappa)}\rrparenthesis) \\
\llparenthesis\mathfrak{b}(\kappa, (x : t_1), t_2)\rrparenthesis &= \mathfrak{b}(\kappa, (x : \llparenthesis t_1\rrparenthesis), \llparenthesis t_2\rrparenthesis) \\
\llparenthesis x_K\rrparenthesis &= x_K
\end{aligned}$$

Figure 2.4: 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.

reduction allows reduction steps to occur in any subexpression, but reductions that generate new redexes cannot be reduced in a single step. Figure 2.3 presents the inductive definition of parallel reduction. In fact, it is possible to compute the resulting syntax after all possible redexes are contracted by a single parallel reduction step. This is the *reduction completion* (written $\llparenthesis t\rrparenthesis$) of some syntax t. The definition of reduction completion is shown in Figure 2.4. Reduction completion enables the derivation of a triangle property for parallel reduction of which confluence for parallel reduction is a consequence. With confluence for parallel reduction and logical equivalence then confluence of one-step reduction is immediate.

Lemma 2.8. *For any t, $t \Rrightarrow t$*

Proof. Straightforward by induction on t. □

Lemma 2.9. *If $s \rightsquigarrow t$ then $s \Rrightarrow t$*

Proof. By induction on $s \rightsquigarrow t$. The projection and promotion cases are similar to the substitution and beta case and thus omitted. The second structural binder reduction case is omitted.

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

By Lemma 2.8: $t \Rrightarrow t$ and $b \Rrightarrow b$. Applying the ParBeta rule concludes the case.

Case: $\psi(\mathrm{refl}(z; Z), u, v; A, P) \bullet_\omega b \rightsquigarrow b$

Using Lemma 2.8: $z \Rrightarrow z$, $Z \Rrightarrow Z$, $u \Rrightarrow u$, $v \Rrightarrow v$, $A \Rrightarrow A$, $P \Rrightarrow P$, and $b \Rrightarrow b$. Applying the ParSubst rule concludes the 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 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 2.8. 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 2.8: $t_2 \Rrightarrow t_2$. Using the ParBind rule concludes the case.

□

Lemma 2.10. *If $s \rightsquigarrow^* t$ then $s \Rrightarrow^* t$*

Proof. By induction on $s \rightsquigarrow^* t$ applying Lemma 2.9 in the inductive case. □

Lemma 2.11. *If $s \Rrightarrow t$ then $s \rightsquigarrow^* t$*

Proof. By induction on $s \Rrightarrow t$. The projection, promotion, and substitution cases are similar to the beta case with the only difference being applying the associated rule.

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

By reflexivity of reduction.

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 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.

□

Lemma 2.12. *If $s \Rrightarrow^* t$ then $s \rightsquigarrow^* t$*

Proof. By induction on $s \Rrightarrow^* t$ applying Lemma 2.11 in the inductive case. □

Lemma 2.13. *If $s \Rrightarrow s'$ and $t \Rrightarrow t'$ then $[x := s]t \Rrightarrow [x := s']t'$*

Proof. By induction on $t \Rrightarrow t'$. The second projection case is omitted because it is the same as the first projection case.

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

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

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: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_2}{t_3 \Rrightarrow t_3'} \qquad \overset{\mathcal{D}_2}{t_4 \Rrightarrow t_4'} \qquad \overset{\mathcal{D}_2}{t_5 \Rrightarrow t_5'} \qquad \overset{\mathcal{D}_2}{t_6 \Rrightarrow t_6'} \qquad \overset{\mathcal{D}_2}{t_7 \Rrightarrow t_7'}}{\psi(\text{refl}(t_1; t_2), t_3, t_4; t_5, t_6) \bullet_\omega t_7 \Rrightarrow t_7'}$$

By the IH: $[x := s]t_i \Rrightarrow [x := s']t_i'$ for $i = 1, 2$. The ParSubst rule gives: $[x := s](\psi(\text{refl}(t_1; t_2), t_3, t_4; t_5, t_6) \bullet_\omega t_7) = \psi(\text{refl}([x := s]t_1; [x := s]t_2), [x := s]t_3, [x := s]t_4; [x := s]t_5, [x := s]t_6) \bullet_\omega [x := s]t_7 \Rrightarrow [x := s']t_7'$.

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: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'} \qquad \overset{\mathcal{D}_3}{t_4 \Rrightarrow t_4'} \qquad \overset{\mathcal{D}_3}{t_5 \Rrightarrow t_5'}}{\vartheta(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5) \Rrightarrow \mathrm{refl}(t_3'; t_5')}$$

By the IH: $[x := s]t_i \Rrightarrow [x := s']t_i'$ for $i = 1,2,3$. The ParFst rule gives: $[x := s]\vartheta(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5) = \vartheta(\mathrm{refl}([x := s]t_1; [x := s]t_2), [x := s]t_3, [x := s]t_4; [x := s]t_5) \Rrightarrow \mathrm{refl}([x := s']t_3'; [x := s']t_5') = [x := s']\mathrm{refl}(t_3'; t_5')$.

□

The triangle property of parallel reduction is used to complete the set of possible contractible redexes. Thus, if syntax $s \Rrightarrow t$ where t is only partially reduced then both s and t may be completed to $\llparenthesis s \rrparenthesis$. To the right the situation is visually depicted. Note that the triangle property is "half" of the diamond property. Indeed, if $s \Rrightarrow t'$ then $t' \Rrightarrow \llparenthesis s \rrparenthesis$. Thus, as a consequence of the triangle property, parallel reduction trivially has the diamond property.

$$\begin{tikzcd}
s \arrow[dr, Rrightarrow] \arrow[dd, Rrightarrow] & \\
& t \arrow[dl, Rrightarrow] \\
\llparenthesis s \rrparenthesis &
\end{tikzcd}$$

Lemma 2.14 (Parallel Triangle)**.** *If $s \Rrightarrow t$ then $t \Rrightarrow \llparenthesis s \rrparenthesis$*

Proof. By induction on $s \Rrightarrow t$. The second projection case is omitted.

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

Have $\llparenthesis x_K \rrparenthesis = x_K$. 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 $\mathcal{D}_i$: $t_i' \Rrightarrow \llparenthesis t_i \rrparenthesis$ for all i. Proceed by cases of $\llparenthesis \mathfrak{c}(\kappa, t_1, \ldots t_{\mathfrak{a}(\kappa)}) \rrparenthesis$. The second projection case is omitted because it is the same as the first projection case.

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

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 $(\lambda_m\, x : t_1'.\, t_2') \bullet_m t_3' \Rrightarrow [x := \llparenthesis t_3 \rrparenthesis]\llparenthesis t_2 \rrparenthesis$.

Case: $\llparenthesis \psi(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5, t_6) \bullet_\omega t_7 \rrparenthesis = \llparenthesis t_7 \rrparenthesis$

Note that $\mathfrak{c}(\kappa, t_1', \ldots t_{\mathfrak{a}(\kappa)}') = \psi(\mathrm{refl}(t_1'; t_2'), t_3', t_4'; t_5', t_6') \bullet_\omega t_7'$. Using the ParSubst rule yields $\psi(\mathrm{refl}(t_1'; t_2'), t_3', t_4'; t_5', t_6') \bullet_\omega t_7' \Rrightarrow \llparenthesis t_7 \rrparenthesis$.

Case: $\llparenthesis [t_1, t_2; t_3].1 \rrparenthesis = \llparenthesis t_1 \rrparenthesis$

Note that $\mathfrak{c}(\kappa, t_1', \dots t_{\mathfrak{a}(\kappa)}') = [t_1', t_2'; t_3'].1$. Using the ParFst rule yields $[t_1', t_2'; t_3'].1 \Rrightarrow \llparenthesis t_1 \rrparenthesis$.

Case: $\llparenthesis \vartheta(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5) \rrparenthesis = \mathrm{refl}(\llparenthesis t_3 \rrparenthesis; \llparenthesis t_5 \rrparenthesis)$

Note that $\mathfrak{c}(\kappa, t_1', \dots t_{\mathfrak{a}(\kappa)}') = \vartheta(\mathrm{refl}(t_1'; t_2'), t_3', t_4'; t_5')$. Using the ParPrmFst rule yields $\vartheta(\mathrm{refl}(t_1'; t_2'), t_3', t_4'; t_5') \Rrightarrow \mathrm{refl}(\llparenthesis t_3 \rrparenthesis; \llparenthesis t_5 \rrparenthesis)$.

Case: $\llparenthesis \mathfrak{c}(\kappa, t_1, \dots t_{\mathfrak{a}(\kappa)}) \rrparenthesis = \mathfrak{c}(\kappa, \llparenthesis t_1 \rrparenthesis, \dots \llparenthesis t_{\mathfrak{a}(\kappa)} \rrparenthesis)$

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 $\llparenthesis \mathfrak{b}(\kappa, (x : t_1), t_2) \rrparenthesis = \mathfrak{b}(\kappa, (x : \llparenthesis t_1 \rrparenthesis), \llparenthesis t_2 \rrparenthesis)$. By the IH applied to $\mathcal{D}_i$: $t_i' \Rrightarrow \llparenthesis t_i \rrparenthesis$ for $i = 1, 2$. Thus, by the ParBind rule $\mathfrak{b}(\kappa, (x : t_1'), t_2') \Rrightarrow \mathfrak{b}(\kappa, (x : \llparenthesis t_1 \rrparenthesis), \llparenthesis t_2 \rrparenthesis)$.

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 $\llparenthesis (\lambda_m\, x : t_1 .\, t_2) \bullet_m t_3 \rrparenthesis = [x := \llparenthesis t_3 \rrparenthesis]\llparenthesis t_2 \rrparenthesis$. By the IH applied to $\mathcal{D}_i$: $t_i' \Rrightarrow \llparenthesis t_i \rrparenthesis$ for $i = 1, 2, 3$. Thus, by Lemma 2.13 $[x := t_3']t_2' \Rrightarrow [x := \llparenthesis t_3 \rrparenthesis]\llparenthesis t_2 \rrparenthesis$.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_2}{t_3 \Rrightarrow t_3'} \qquad \overset{\mathcal{D}_2}{t_4 \Rrightarrow t_4'} \qquad \overset{\mathcal{D}_2}{t_5 \Rrightarrow t_5'} \qquad \overset{\mathcal{D}_2}{t_6 \Rrightarrow t_6'} \qquad \overset{\mathcal{D}_2}{t_7 \Rrightarrow t_7'}}{\psi(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5, t_6) \bullet_\omega t_7 \Rrightarrow t_7'}$$

Note that $\llparenthesis \psi(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5, t_6) \bullet_\omega t_7 \rrparenthesis = \llparenthesis t_7 \rrparenthesis$. By the IH applied to $\mathcal{D}_i$: $t_i' \Rrightarrow \llparenthesis t_i \rrparenthesis$ for $i = 1$ through $i = 7$. Applying the ParBind rule yields $t_7' \Rrightarrow \llparenthesis t_7 \rrparenthesis$.

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

Note that $\llparenthesis [t_1, t_2; t_3].1 \rrparenthesis = \llparenthesis t_1 \rrparenthesis$. By the IH applied to $\mathcal{D}_i$: $t_i' \Rrightarrow \llparenthesis t_i \rrparenthesis$ for $i = 1, 2, 3$. Thus, $t_1' \Rrightarrow \llparenthesis t_1 \rrparenthesis$.

Case: $$\frac{\overset{\mathcal{D}_1}{t_1 \Rrightarrow t_1'} \qquad \overset{\mathcal{D}_2}{t_2 \Rrightarrow t_2'} \qquad \overset{\mathcal{D}_3}{t_3 \Rrightarrow t_3'} \qquad \overset{\mathcal{D}_3}{t_4 \Rrightarrow t_4'} \qquad \overset{\mathcal{D}_3}{t_5 \Rrightarrow t_5'}}{\vartheta(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5) \Rrightarrow \mathrm{refl}(t_3'; t_5')}$$

Note that $\llparenthesis \vartheta_1(\mathrm{refl}(t_1; t_2), t_3, t_4; t_5) \rrparenthesis \Rrightarrow \mathrm{refl}(\llparenthesis t_3 \rrparenthesis; \llparenthesis t_5 \rrparenthesis)$. By the IH applied to $\mathcal{D}_i$: $t_i' \Rrightarrow \llparenthesis t_i \rrparenthesis$ for $i = 1$ through $i = 5$. Thus, $\mathrm{refl}(t_3'; t_5') \Rrightarrow \mathrm{refl}(\llparenthesis t_3 \rrparenthesis; \llparenthesis t_5 \rrparenthesis)$ by the ParCtor rule and Lemma 2.8.

□

Lemma 2.15 (Parallel 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 2.14 to $s \Rrightarrow z_1$ yields $z_1 \Rrightarrow \llparenthesis s \rrparenthesis$. By the IH with $z_1 \Rrightarrow \llparenthesis s \rrparenthesis$: $\exists\ z_2$ such that $\llparenthesis s \rrparenthesis \Rrightarrow^* z_2$ and $t_2 \Rrightarrow z_2$. Using Lemma 2.14 again on $s \Rrightarrow t_1$ yields $t_1 \Rrightarrow \llparenthesis s \rrparenthesis$. Now by transitivity $t_1 \Rrightarrow^* z_2$. □

Lemma 2.16 (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 2.15 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 2.17 (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 2.10 applied twice: $s \Rrightarrow^* t_1$ and $s \Rrightarrow^* t_2$. Now by parallel confluence (Lemma 2.16) $\exists\ t$ such that $t_1 \Rrightarrow^* t$ and $t_2 \Rrightarrow^* t$. Finally, two applications of Lemma 2.12 conclude the proof. □

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

Theorem 2.18. *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 of reduction that is occasionally used throughout the rest of this work. That is, like reduction, conversion of subexpressions yields conversion of the entire term.

Lemma 2.19. *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. □

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

Figure 2.5: Erasure of syntax, for type-like and kind-like syntax erasure is homomorphic, for term-like syntax erasure reduces to the untyped lambda calculus.

2.3 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 2.4 the notion of proof will be defined. An object is the erasure of a proof. Erasure is defined in Figure 2.5. With erasure the desired conversion relation is also definable. This definition will enable equating objects in a dependent quantification instead of proofs.

Definition 2.20. $s_1 \equiv s_2$ *iff* $\exists\ t_1, t_2.\ s_1 \rightsquigarrow^* t_1, s_2 \rightsquigarrow^* t_2,$ *and* $|t_1| \rightleftharpoons |t_2|$

Note that the only purpose of the syntactic constructor $\diamond$ is to be a placeholder for erased type annotations of λ_ω 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 syntax. 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 is idempotent and commutes with substitution. Additionally, as a consequence an extension of Lemma 2.7 is possible.

Lemma 2.21. $||t|| = |t|$

Proof. By induction on t. □

Lemma 2.22. $|[x := t]b| = [x := |t|]|b|$

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

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

If $\kappa \in \{\mathrm{refl}\}$ 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.

□

Lemma 2.23. *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 2.22 $[x := |s|]|a| = |[x := s]a|$ and $[x := |t|]|b| = |[x := t]b|$. □

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

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

$$x_K \text{ pseobj}$$

Figure 2.6: Definition of Pseudo Objects.

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 2.6.

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

Proof. By induction on s pseobj.

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

By cases on $s \rightsquigarrow t$, applying the IH and Lemma 2.19.

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 Lemma 2.19.

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

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 2.22 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: $\psi(\text{refl}(z; Z), a, b; A, P) \bullet_\omega t \rightsquigarrow t$

$$|\psi(\text{refl}(z; Z), a, b; A, P) \bullet_\omega t| = |\text{refl}(z; Z)| \bullet_\omega |t| \rightleftharpoons |t|$$

Case: $\vartheta(\text{refl}(z; Z), a, b; A) \rightsquigarrow \text{refl}(a; A)$

$$|\vartheta(\text{refl}(z; Z), a, b; A)| = |\text{refl}(z; Z)| = \lambda_\omega\, x\!:\!\diamond.\, x = |\text{refl}(a; A)|$$

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, $|t_i| \rightleftharpoons |t_i'|$. The goal is achieved by Lemma 2.19

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

By cases on $s \rightsquigarrow t$, applying the IH and Lemma 2.19.

Case: s variable

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

□

Lemma 2.25. *If* s pseobj, $|s| \rightleftharpoons |b|$, *and* $s \rightsquigarrow t$ *then* $|t| \rightleftharpoons |b|$

Proof. By Lemma 2.24 $|s| \rightleftharpoons |t|$ and by Theorem 2.18 $|t| \rightleftharpoons |b|$. □

Lemma 2.26. *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: $$\frac{\overset{\mathcal{D}_1}{t_1 \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t_2 \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{\kappa \neq \lambda_0}}{\mathfrak{b}(\kappa, x : t_1, t_2) \text{ pseobj}}$$

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

By the IH $[x := t]t_i$ pseobj.
Thus, $\mathfrak{c}(\kappa, [x := t]t_1, \dots [x := t]t_{\mathfrak{a}(\kappa)})$ 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 2.27. *If* s pseobj *and* $s \rightsquigarrow t$ *then* t pseobj

Proof. By induction on s pseobj.

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

By cases on $s \rightsquigarrow t$. Suppose w.l.o.g. that $t_2 \rightsquigarrow t_2'$. 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: $$\frac{\overset{\mathcal{D}_1}{A \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{t \text{ pseobj}} \qquad \overset{\mathcal{D}_3}{x \notin FV(|t|)}}{\lambda_0\, x : A.\, t \text{ pseobj}}$$

By cases on $s \rightsquigarrow t$. Suppose w.l.o.g that $t \rightsquigarrow t'$. 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: $$\frac{\overset{\mathcal{D}_1}{\forall\ i \in 1, \dots, \mathfrak{a}(\kappa).\ t_i \text{ pseobj}} \qquad \overset{\mathcal{D}_2}{\kappa \neq \text{pair}}}{\mathfrak{c}(\kappa, t_1, \dots, t_{\mathfrak{a}(\kappa)}) \text{ pseobj}}$$

By cases on $s \rightsquigarrow t$. The first and second projection cases are very similar to the substitution case.

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 2.26 $[x := t]b$ pseobj.

Case: $\psi(\mathrm{refl}(z; Z), a, b; A, P) \bullet_\omega t \rightsquigarrow t$

Immediate by the IH: t pseobj.

Case: $\vartheta_1(\mathrm{refl}(z; Z), a, b; A) \rightsquigarrow \mathrm{refl}(a; A)$

Observe that a pseobj and A pseobj. By application of constructor rule $\mathrm{refl}(a; A)$ pseobj.

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

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

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 2.25 $|t_1'| \rightleftharpoons |t_2|$. 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 2.28. *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 2.25, Lemma 2.27, and applying the IH. □

Lemma 2.29. *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 2.27 and applying the IH. □

Lemma 2.30. *If s pseobj, $|t| \rightleftharpoons |b|$, and $s \rightsquigarrow^* t$ then $|s| \rightleftharpoons |b|$*

Proof. By induction on $s \rightsquigarrow^* t$. Consequence of Lemma 2.24 and Lemma 2.29. □

Lemma 2.31. *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 2.29 z_1 pseobj. Finally, by Lemma 2.28 $|z_1'| \rightleftharpoons |z_2|$. Therefore, $t \equiv b$. □

Unlike with convertibility of reduction, obtaining transitivity of conversion requires the additional assumption that the inner syntax 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, pick $a = [x, y; T].1$, $b = [x, y; T]$, and $c = [y, x; T].2$. Notice that $|a| = |b|$ but $|b| \neq |c|$, however, $c \rightsquigarrow^* x$, thus $b \equiv c$ and $\neg(b \equiv c)$. There is an inconsistency in the definition because b is not a pseudo-object, it is not the case that $|x| \rightleftharpoons |y|$. Really this is more fundamental than just transitivity as it shows that reduction is not consistent with erasure unless the syntax is a pseudo-object.

Lemma 2.32. *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 2.29 u_2 pseobj and v_1 pseobj. Next, by Lemma 2.28 $|u_1| \rightleftharpoons |z|$ and $|z| \rightleftharpoons |v_2|$. Thus, $|u_1| \rightleftharpoons |v_2|$ by Lemma 2.18 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 2.33 means that whenever conversion is checked between terms it is safe to instead check convertibility of 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 when checking $|s| \rightleftharpoons |t|$ instead of $s \equiv t$.

Theorem 2.33. *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 2.30 $|s| \rightleftharpoons |t|$. □

Corollary 2.34. *For s pseobj and t pseobj the relation $s \equiv t$ is an equivalence.*

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

Proof. By Lemma 2.33 $|s| \rightleftharpoons |t|$ and $|a| \rightleftharpoons |b|$. Then, by Lemma 2.23 $|[x := s]a| \rightleftharpoons |[x := t]b|$. Finally, by Lemma 2.33 again, $[x := s]a \equiv [x := t]b$. □

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

Figure 2.7: Domain and codomains for function types. The variable K is either $\star$ or $\square$.

$$\frac{}{\Gamma \vdash \star : \square}\ \textsc{Axiom} \qquad \frac{x \notin FV(\Gamma_1;\Gamma_2) \qquad \Gamma_1 \vdash A : K}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}\ \textsc{Var}$$

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

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

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

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

Figure 2.8: Inference rules for function types, including erased functions. The variable K is either $\star$ or $\square$.

2.4 Inference Judgment

The inference judgment, presented in Figure 2.8; Figure 2.9; and Figure 2.10, 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 $|t|$ its 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.

$\dfrac{}{\Gamma \vdash \star : \square}\ \textsc{Axiom}$ 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.

$\dfrac{x \notin FV(\Gamma_1;\Gamma_2) \qquad \Gamma_1 \vdash A : K}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}\ \textsc{Var}$ 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.

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

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. 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 computationally in the object of a proof.

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

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.

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

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 2.33 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

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

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

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

Figure 2.9: Inference rules for intersection types.

from the original Cedille design.

$$\frac{\Gamma \vdash A : \star \qquad \Gamma \vdash t : A}{\Gamma \vdash \text{refl}(t; A) : t =_A t}\ \textsc{Refl}$$

The reflexivity rule is the only value for equality types. It is the standard inductive formulation of the equality type.

$$\frac{\begin{array}{c}\Gamma \vdash A : \star \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash e : a =_A b \qquad \Gamma \vdash P : (y : A) \to_\tau (p : a =_A y_\star) \to_\tau \star\end{array}}{\Gamma \vdash \psi(e, a, b; A, P) : P \bullet_\tau a \bullet_\tau \text{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e}\ \textsc{Subst}$$

The substitution rule is a dependent variation of the Leibniz's Law. It is also a variation of Martin-Löf's J rule introduced by Pfenning and Paulin-Mohring [81]. 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.

$$\frac{\Gamma \vdash (x : A) \cap B : \star \qquad \Gamma \vdash a : (x : A) \cap B \qquad \Gamma \vdash b : (x : A) \cap B \qquad \Gamma \vdash e : a.1 =_A b.1}{\Gamma \vdash \vartheta(e, a, b; (x : A) \cap B) : a =_{(x:A) \cap B} b}\ \textsc{Prm}$$

The promotion rule enables equational reasoning about intersections. Indeed, because intersections are not inductive it is difficult to reason about them without some auxiliary rule. It states that two elements of an intersection are equal if their first projections are equal. Note that this rule is about dependent intersections. A non-dependent version involving the second projection is internally derivable in the system.

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : (x : A) \cap B \qquad \Gamma \vdash e : a =_A b.1}{\Gamma \vdash \varphi(a, b, e) : (x : A) \cap B}\ \textsc{Cast}$$

The cast rule is a typed version of the original cast rule of Cedille. Note that this means this rule enables non-termination. In Chapter 5 it is shown that this rule is the only source of non-termination and a precise condition for when it may be used in a terminating way is devised. The cast rule is critical to the spirit of Cedille. Thus, simply dropping it to obtain a strongly normalizing system is not really a sufficient choice as it throws too much away in its loss.

$$\frac{\Gamma \vdash e : \text{ctt} =_{\text{cBool}} \text{cff}}{\Gamma \vdash \delta(e) : (X : \star) \to_0 X_\square}\ \textsc{Sep}$$

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

The first critical observation to be made is that the syntax participating in an inference judgment

$$\frac{\Gamma \vdash A : \star \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \star}\ \textsc{Eq} \qquad\qquad \frac{\Gamma \vdash A : \star \qquad \Gamma \vdash t : A}{\Gamma \vdash \text{refl}(t;A) : t =_A t}\ \textsc{Refl}$$

$$\frac{\begin{array}{c}\Gamma \vdash A : \star \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash e : a =_A b \qquad \Gamma \vdash P : (y : A) \to_\tau (p : a =_A y_\star) \to_\tau \star\end{array}}{\Gamma \vdash \psi(e,a,b;A,P) : P \bullet_\tau a \bullet_\tau \text{refl}(a;A) \to_\omega P \bullet_\tau b \bullet_\tau e}\ \textsc{Subst}$$

$$\frac{\Gamma \vdash (x:A)\cap B : \star \qquad \Gamma \vdash a : (x:A)\cap B \qquad \Gamma \vdash b : (x:A)\cap B \qquad \Gamma \vdash e : a.1 =_A b.1}{\Gamma \vdash \vartheta(e,a,b;(x:A)\cap B) : a =_{(x:A)\cap B} b}\ \textsc{Prm}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : (x:A)\cap B \qquad \Gamma \vdash e : a =_A b.1}{\Gamma \vdash \varphi(a,b,e) : (x:A)\cap B}\ \textsc{Cast}$$

$$\frac{\Gamma \vdash e : \text{ctt} =_{\text{cBool}} \text{cff}}{\Gamma \vdash \delta(e) : (X:\star) \to_0 X_\Box}\ \textsc{Sep}$$

Figure 2.10: Inference rules for equality types where cBool $:= (X : \star) \to_0 (x : X_\Box) \to_\omega (y : X_\Box) \to_\omega X_\Box$; ctt $:= \lambda_0\, X\!:\!\star.\, \lambda_\omega\, x\!:\!X_\Box.\, \lambda_\omega\, y\!:\!X_\Box.\, x_\star$; and cff $:= \lambda_0\, X\!:\!\star.\, \lambda_\omega\, x\!:\!X_\Box.\, \lambda_\omega\, y\!:\!X_\Box.\, y_\star$.

are pseudo-objects. Thus, proofs and their types enjoy transitivity of conversion. Next three standard lemmas are proved about the type system: weakening, substitution, and a sort-hierarchy classification.

Lemma 2.36. *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 2.33. □

Lemma 2.37. *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 2.36 to a sub-derivation. The Fst and App rules are omitted because it is similar to the Snd rule. Likewise, the Refl rule is omitted because it is similar to the Prm rule.

Case: $$\frac{\overset{\mathcal{D}_1}{x \notin FV(\Gamma_1;\Gamma_2)} \qquad \overset{\mathcal{D}_2}{\Gamma_1 \vdash A : K}}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

By Lemma 2.36 applied to $\mathcal{D}_2$: 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 2.36 gives t pseobj and thus $t.1$ pseobj. Now by Lemma 2.26: $[x := t.1]B$ pseobj.

Case:
$$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : A} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a =_A b} \qquad \overset{\mathcal{D}_5}{\Gamma \vdash P : (y : A) \to_\tau (p : a =_A y_\star) \to_\tau \star}}{\Gamma \vdash \psi(e, a, b; A, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

By Lemma 2.36: 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:
$$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \cap B : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : (x : A) \cap B} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : (x : A) \cap B} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a.1 =_A b.1}}{\Gamma \vdash \vartheta(e, a, b; (x : A) \cap B) : a =_{(x:A)\cap B} b}$$

Applying the IH to $\mathcal{D}_1$ gives that $(x : A) \cap B$ pseobj. Now, by Lemma 2.36: a, b pseobj. Using the pseudo-object rule for equality concludes the case.

Case:
$$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash b : (x : A) \cap B} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash e : a =_A b.1}}{\Gamma \vdash \varphi(a, b, e) : (x : A) \cap B}$$

By the IH applied to $\mathcal{D}_2$.

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

Immediate by a short sequence of pseudo-object rules.

□

Lemma 2.38 (Weakening)**.** *If* $\Gamma; \Delta \vdash t : A$ *and* $\Gamma \vdash B : K$ *then* $\Gamma; y_m : B; \Delta \vdash t : A$ *for* y *fresh*

Proof. By induction. Most cases are a direct consequence of applying the IH to sub-derivations and applying the associated rule.

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

Trivial by axiom rule.

Case:
$$\dfrac{\overset{\mathcal{D}_1}{x \notin FV(\Gamma_1; \Gamma_2)} \qquad \overset{\mathcal{D}_2}{\Gamma_1 \vdash A : K}}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

Note that y is fresh thus $x \neq y$. If y is placed after x then the case is trivial because Γ_2 is only constrained to carry fresh variables. Thus, suppose y is placed before x. Let $\Gamma_1 = \Delta_1; \Delta_2$. Applying the IH to $\mathcal{D}_2$ gives $\Delta_1; y_m : B; \Delta_2 \vdash A : K$. The Var rule concludes.

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

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

□

Lemma 2.39 (Substitution)**.** *Suppose* $\Gamma \vdash b : B$*. If* $\Gamma, y : B, \Delta \vdash t : A$ *then* $\Gamma, [y := b]\Delta \vdash [y := b]t : [y := b]A$

Proof. By induction on $\Gamma, y : B, \Delta \vdash t : A$. 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, Prm, Cast and Sep rules are proven by applying the IH 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.

Case: $$\frac{\overset{\mathcal{D}_1}{x \notin FV(\Gamma_1;\Gamma_2)} \qquad \overset{\mathcal{D}_2}{\Gamma_1 \vdash A : K}}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

Suppose wlog that $y \in \Gamma_1$. Let $\Gamma_1 = \Delta_1; y : B; \Delta_2$. Applying the IH to $\mathcal{D}_1$ gives $\Delta_1; [y := b]\Delta_2 \vdash [y := b]A : K$. Note that $x \notin FV(\Delta_1; [y := b]\Delta_2; [y := b]\Gamma_2)$. Thus by the Var rule: $\Delta_1; [y := b]\Delta_2; x_m : [y := b]A; [y := b]\Gamma_2 \vdash x_K : [y := b]A$.

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

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

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

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

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

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 *1.* to $\mathcal{D}_1$ and $\mathcal{D}_2$ gives:

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

$\mathcal{D}_2$. $\Gamma, [y := b]\Delta, x_m : [y := b]A \vdash [y := b]a : [y := b]A$

By the App rule $\Gamma, [y := b]\Delta \vdash [y := b]f \bullet_m [y := b]a : [x := a][y := b]B$. Note that x is fresh to Γ, thus $x \notin FV(b)$. By Lemma 2.1 $[x := a][y := b]B = [y := b][x := a]B$.

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

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

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

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

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

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

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

Now by Lemma 2.35: $[y := b]t \equiv [y := b]s$. Thus, by the Pair rule $\Gamma, [y := b]\Delta \vdash [[y := b]t, [y := b]s] : (x : [y := b]A) \cap [y := b]B$.

□

Lemma 2.40. *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 rule is omitted because it is similar to Snd rule. Likewise, the Refl rule is omitted because it is similar to the Prm rule.

Case: $$\frac{\overset{\mathcal{D}_1}{x \notin FV(\Gamma_1; \Gamma_2)} \qquad \overset{\mathcal{D}_2}{\Gamma_1 \vdash A : K}}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

Immediate by $\mathcal{D}_2$ and weakening.

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: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : A} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a =_A b} \qquad \overset{\mathcal{D}_5}{\Gamma \vdash P : (y : A) \to_\tau (p : a =_A y_\star) \to_\tau \star}}{\Gamma \vdash \psi(e, a, b; A, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

By the Refl rule: $\Gamma \vdash \mathrm{refl}(a; A) : a =_A a$. Now by the App rule $\Gamma \vdash P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) : \star$ and $\Gamma \vdash P \bullet_\tau b \bullet_\tau e : \star$. Using weakening gives $\Gamma, x : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \vdash P \bullet_\tau b \bullet_\tau e : \star$. Now the Pi rule concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \cap B : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : (x : A) \cap B} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : (x : A) \cap B} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a.1 =_A b.1}}{\Gamma \vdash \vartheta(e, a, b; (x : A) \cap B) : a =_{(x:A) \cap B} b}$$

Immediate by applying the Eq rule.

Case: $$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash b : (x : A) \cap B} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash e : a =_A b.1}}{\Gamma \vdash \varphi(a, b, e) : (x : A) \cap B}$$

By the IH applied to $\mathcal{D}_2$.

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

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

□

The context of a judgment is, for the moment, unrestrained. Indeed, a variable may bind a type represented by arbitrary syntax and as long as that variable is never used in the body of the term there is no issue. To remove these considerations contexts should instead be well-formed:

Definition 2.41. *A context Γ is **well-formed** (written $\vdash \Gamma$) iff for every possible splitting $\Gamma = \Gamma_1, x : A, \Gamma_2$ the variable $x \notin FV(\Gamma_1; \Gamma_2)$ and $\Gamma_1 \vdash A : K$ for some K*

It is not difficult to see that an inference judgment with a well-formed context is obtained from any general inference judgment. Moving forward it will be assumed that the context is well-formed because an equivalent proof is always obtainable under this assumption and the non-well-founded proofs will not add any value.

Lemma 2.42. *If $\Gamma \vdash t : A$ then $\exists\ \Delta$ such that $\Delta \vdash t : A$ and $\vdash \Delta$*

Proof. By Lemma 2.40: $\Gamma \vdash A : K$. Now, the set of free variable $S = FV(t) \cup FV(A)$ determines Δ. Moreover, every occurrence of $x \in S$ in either t or A must be via a Var rule, hence the associated type is a proof. Delete any variables not found in S from Γ to form Δ. □

2.5 Preservation

Preservation states that the status of a term (i.e. both its classification and status as a well-founded proof) do not change with respect to reduction. Note that Cedille only enjoys a semantic version of preservation and not a syntactic version presented below. While this may not matter from the perspective of the semantics it does indicate that syntax is better behaved. The proof follows by induction on the typing derivation and case analysis on the associated reduction.

Definition 2.43. $\Gamma \rightsquigarrow \Gamma'$ *iff there exists a unique $(x_m : A) \in \Gamma$ such that $A \rightsquigarrow A'$*

Lemma 2.44.

1. *If $\Gamma \vdash t : A$ and $t \rightsquigarrow t'$ then $\Gamma \vdash t' : A$*

2. *If* $\Gamma \vdash t : A$ *and* $\Gamma \rightsquigarrow \Gamma'$ *then* $\Gamma' \vdash t : A$

3. *If* $\vdash \Gamma$ *and* $\Gamma \rightsquigarrow \Gamma'$ *then* $\vdash \Gamma'$

Proof. By mutual recursion.

1. Pattern-matching on $\Gamma \vdash t : A$. The Ax and Var cases are impossible by inversion on $t \rightsquigarrow t'$. Int is very similar to Pi, Fst is very similar to Snd. The Refl, Sep, and Conv rules are trivial.

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

Suppose $A \rightsquigarrow A'$. Applying *1* to $\mathcal{D}_1$ gives $\Gamma \vdash A' : \mathrm{dom}_\Pi(m, K)$. Note that $\Gamma, x_m : A \rightsquigarrow \Gamma, x_m : A'$. Thus, using *2* with $\mathcal{D}_2$ gives $\Gamma, x_m : A' \vdash B : \mathrm{codom}_\Pi(m)$. Using the Pi rule concludes the case.

Suppose $B \rightsquigarrow B'$. Applying *1* to $\mathcal{D}_2$ gives $\Gamma, x_m : A \vdash B' : \mathrm{codom}_\Pi(m)$. The Pi rule concludes the case.

Case: $$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)} \qquad \overset{\mathcal{D}_2}{\Gamma; x_m : A \vdash t : B} \qquad \overset{\mathcal{D}_3}{x \notin FV(|t|) \text{ if } m = 0}}{\Gamma \vdash \lambda_m\, x{:}A.\, t : (x : A) \to_m B}$$

Suppose $A \rightsquigarrow A'$. Then $(x : A) \to_m B \rightsquigarrow (x : A') \to_m B$. Now, using *1* with $\mathcal{D}_1$ gives $\Gamma \vdash (x : A') \to_m B : \mathrm{codom}_\Pi(m)$. Note that $\Gamma, x_m : A \rightsquigarrow \Gamma, x_m : A'$. Using *2* with $\mathcal{D}_2$ yields $\Gamma, x_m : A' \vdash t : B$. Applying the Lam rule concludes the case.

Suppose $t \rightsquigarrow t'$. Using *1* with $\mathcal{D}_2$ gives $\Gamma, x_m : A \vdash t' : B$. Note that reduction does not introduce free variables, thus $x \notin FV(|t'|)$ if $m = 0$. The Lam rule concludes.

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

Suppose $f \rightsquigarrow f'$. Applying *1* with $\mathcal{D}_1$ gives $\Gamma \vdash f' : (x : A) \to_m B$. The App rule concludes.

Suppose $a \rightsquigarrow a'$. Using *1* with $\mathcal{D}_2$ gives $\Gamma \vdash a' : A$. Again, the App rule concludes the case.

Suppose $f = \lambda_m\, x : C.\, t$ and $f \bullet_m a \rightsquigarrow [x := a]t$. There must exist C and D such that $\Gamma \vdash C : \mathrm{dom}_\Pi(m, K)$ and $\Gamma, x_m : C \vdash t : D$ with $A \equiv C$ and $B \equiv D$. By classification (Lemma 2.40) and the Conv rule, $\Gamma \vdash a : C$. Now using the substitution lemma (Lemma 2.39) $\Gamma \vdash [x := a]t : [x := a]D$. Using Lemma 2.35 gives $[x := a]B \equiv [x := a]D$. Classification and Conv again yields $\Gamma \vdash [x := a]t : [x := a]B$.

Suppose $f = \psi(\text{refl}(z;Z), u, v; U, P)$ with $m = \omega$ and $f \bullet_\omega a \rightsquigarrow a$. By inversion on $\mathcal{D}_1$: $A = P \bullet_\tau u \bullet_\tau \text{refl}(u;U)$ and $[x := a]B = P \bullet_\tau v \bullet_\tau \text{refl}(z;Z)$. However, inversion also yields $\Gamma \vdash \text{refl}(z;Z) : u =_U v$ thus $z \equiv u$, $z \equiv v$, and $Z \equiv U$. Thus, $P \bullet_\tau u \bullet_\tau \text{refl}(u;U) \equiv P \bullet_\tau v \bullet_\tau \text{refl}(z;Z)$. The Conv rule concludes the case.

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

Suppose $t \rightsquigarrow t'$. Applying *1* to $\mathcal{D}_2$ gives $\Gamma \vdash t' : A$. Note that $[x := t]B \equiv [x := t']B$ by Lemma 2.35. Moreover, deconstructing $\mathcal{D}_1$ yields $\Gamma, x_\tau : A \vdash B : \star$. By the substitution lemma $\Gamma \vdash [x := t']B : \star$. Thus, by the Conv rule $\Gamma \vdash s : [x := t']B$. Finally, Lemma 2.31 gives $t' \equiv s$ from $\mathcal{D}_4$. The Pair rule concludes the case.

Suppose $s \rightsquigarrow s'$. By *1* applied to $\mathcal{D}_3$: $\Gamma \vdash s' : [x := t]B$. Using Lemma 2.35 with $\mathcal{D}_4$ yields $t \equiv s'$. The Pair rule concludes.

Suppose $A \rightsquigarrow A'$. Then $(x : A) \cap B \rightsquigarrow (x : A') \cap B$. Applying this reduction to *1* with $\mathcal{D}_1$ gives $\Gamma \vdash (x : A') \cap B : \star$. Deconstructing this yields $\Gamma \vdash A' : \star$. Now by the Conv rule $\Gamma \vdash t : A'$. Using the Pair rule concludes.

Suppose $B \rightsquigarrow B'$. Then $(x : A) \cap B \rightsquigarrow (x : A') \cap B$. Applying this reduction to *1* with $\mathcal{D}_1$ gives $\Gamma \vdash (x : A) \cap B' : \star$. Deconstructing this yields $\Gamma, x_m : A' \vdash B' : \star$. Note that $B \rightsquigarrow B'$ implies that $B \equiv B'$. Moreover, using Lemma 2.35 gives $[x := t]B \equiv [x := t]B'$. The substitution lemma gives $\Gamma \vdash [x := t]B' : \star$. Now the Conv rule yields $\Gamma \vdash s[x := t]B'$. The Pair rule concludes the case.

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

Suppose $t \rightsquigarrow t'$. Then applying *1* to $\mathcal{D}_1$ gives $\Gamma \vdash t' : (x : A) \cap B$. Applying the Snd rule concludes the case.

Suppose $t = [t_1, t_2, t_3]$ and $t.2 \rightsquigarrow t_2$. Then we have $\Gamma \vdash [t_1, t_2, t_3] : (x : A) \cap B$. Deconstructing this rule yields $\Gamma \vdash t_1 : A$, $\Gamma, x_\tau : A \vdash B : \star$, and $\Gamma \vdash t_2 : [x := t_1]B$. By the substitution lemma $\Gamma \vdash [x := t.1]B : \star$. Note that $t.1 \rightsquigarrow t_1$ thus $t.1 \equiv t_1$. Now using Lemma 2.35 gives $[x := t.1]B \equiv [x := t_1]B$. Thus, by the Conv rule $\Gamma \vdash t_2 : [x := t.1]B$.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash b : A}}{\Gamma \vdash a =_A b : \star}$$

Suppose $a \rightsquigarrow a'$. Applying *1* to $\mathcal{D}_2$ gives $\Gamma \vdash a' : A$. The Eq rule concludes.

Suppose $b \rightsquigarrow b'$. Applying *1* to $\mathcal{D}_3$ gives $\Gamma \vdash b' : A$. The Eq rule concludes.

Suppose $A \rightsquigarrow A'$. Applying *1* to $\mathcal{D}_1$ gives $\Gamma \vdash A' : \star$. Note that $A \equiv A'$. Thus, by the Conv rule applied twice: $\Gamma \vdash a : A'$ and $\Gamma \vdash b : A'$. Using the Eq rule concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : A} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a =_A b} \qquad \overset{\mathcal{D}_5}{\Gamma \vdash P : (y : A) \to_\tau (p : a =_A y_\star) \to_\tau \star}}{\Gamma \vdash \psi(e, a, b; A, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

Suppose $A \rightsquigarrow A'$. Then $a =_A b \equiv a ='_A b$ and $(y : A) \to_t au(p : a =_A y_\star) \to_\tau \star \equiv (y : A) \to_t au(p : a =_A y_\star) \to_\tau$. Thus, by the Conv rule: $\Gamma \vdash a : A'$, $\Gamma \vdash b : A'$, $\Gamma \vdash e : a =_{A'} b$, and $\Gamma \vdash P : (y : A') \to_t au(p : a =_{A'} y_\star) \to_\tau$. Applying *1* to $\mathcal{D}_1$ gives: $\Gamma \vdash A' : \star$. The Subst rule concludes the case.

Suppose $a \rightsquigarrow a'$. Then $a =_A b \equiv a'_A b$ and $(y : A) \to_t au(p : a =_A y_\star) \to_\tau \star \equiv (y : A) \to_t au(p : a' =_A y_\star) \to_\tau$. Thus, by the Conv rule: $\Gamma \vdash e : a' =_A b$ and $\Gamma \vdash P : (y : A) \to_t au(p : a' =_A y_\star) \to_\tau$. Applying *1* to $\mathcal{D}_2$ gives: $\Gamma \vdash a' : A$. The Subst rule concludes the case.

Suppose $b \rightsquigarrow b'$. Then $a =_A b \equiv a =_A b'$ and by the Conv rule $\Gamma \vdash b' : A$. Applying *1* to $\mathcal{D}_3$ gives: $\Gamma \vdash b' : B$. The Subst rule concludes the case.

Suppose $e \rightsquigarrow e'$. Then by *1* applied to $\mathcal{D}_1$: $\Gamma \vdash e' : a =_A b$. The Subst rule concludes the case.

Suppose $P \rightsquigarrow P'$. By *1* applied to $\mathcal{D}_2$: $\Gamma \vdash P : (y : A) \to_\tau (p : a =_A y) \to \tau\star$. The Subst rule concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \cap B : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : (x : A) \cap B} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : (x : A) \cap B} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a.1 =_A b.1}}{\Gamma \vdash \vartheta(e, a, b; (x : A) \cap B) : a =_{(x:A)\cap B} b}$$

Suppose $e \rightsquigarrow e'$. Then by *1* applied to $\mathcal{D}_4$: $\Gamma \vdash e' a.1 =_A b.1$ and the Prm rule concludes.

Suppose $a \rightsquigarrow a'$. Then $a.1 =_A b.1 \equiv a'.1 =_A b.1$ and the Conv rule yields $\Gamma \vdash e : a'.1 =_A b.1$. Applying *1* to $\mathcal{D}_2$ gives $\Gamma \vdash a' : (x : A) \cap B$. The Prm rule concludes.

Suppose $b \rightsquigarrow b'$. Then $a.1 =_A b.1 \equiv a.1 =_A b'.1$ and the Conv rule yields $\Gamma \vdash e : a.1 =_A b'.1$. Applying *1* to $\mathcal{D}_3$ gives $\Gamma \vdash b' : (x : A) \cap B$. The Prm rule concludes.

Suppose wlog that $B \leadsto B'$, the case when $A \leadsto A'$ is similar. Then $(x : A) \cap B \equiv (x : A) \cap B'$ and the Conv rule yields $\Gamma \vdash a : (x : A) \cap B'$ and $\Gamma \vdash b : (x : A) \cap B'$. Applying *1* to $\mathcal{D}_1$ yields $\Gamma \vdash (x : A) \cap B' : \star$. The Prm rule concludes.

Suppose $e = \mathrm{refl}(z; Z)$ and $\vartheta(e, a, b; (x : A) \cap B) \leadsto \mathrm{refl}(a; (x : A) \cap B)$. By inversion $\Gamma \vdash \mathrm{refl}(z; Z) : a.1 =_A b.1$, hence $z \equiv a.1$, $z \equiv b.1$. Thus, $a \equiv b$ and $\Gamma \vdash \mathrm{refl}(a; (x : A) \cap B) : a =_{(x:A)\cap B} b$.

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

Suppose $a \leadsto a'$. Then $a =_A b.1 \equiv a' =_A b.1$ and by Conv rule $\Gamma \vdash e : a' =_A b.1$. Applying *1* to $\mathcal{D}_1$ yields $\Gamma \vdash a' : A$. The Cast rule concludes.

Suppose $b \leadsto b'$. Then $a =_A b.1 \equiv a =_A b'.1$ and by Conv rule $\Gamma \vdash e : a =_A b'.1$. Applying *1* to $\mathcal{D}_2$ yields $\Gamma \vdash b' : (x : A) \cap B$. The Cast rule concludes.

Suppose $e \leadsto e'$. Applying *1* to $\mathcal{D}_3$ yields $\Gamma \vdash e' : a =_A b.1$ The Cast rule concludes.

2. Pattern-matching on $\Gamma \vdash t : A$. Note that except Ax and Var all the other cases are immediate by applying *2* to all sub-derivations and using the associated rule.

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

Immediate by the Ax rule, the context does not matter.

Case: $$\frac{\overset{\mathcal{D}_1}{x \notin FV(\Gamma_1; \Gamma_2)} \qquad \overset{\mathcal{D}_2}{\Gamma_1 \vdash A : K}}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

Suppose $\Gamma_1 \leadsto \Gamma_1'$. Reduction does not produce free variables, thus $x \notin FV(\Gamma_1'; \Gamma_2)$. Applying *1* to $\mathcal{D}_2$ yields $\Gamma_1' \vdash A : K$. The Var rule concludes.

Suppose $\Gamma_2 \leadsto \Gamma_2'$. As before $x \notin FV(\Gamma_1; \Gamma_2')$. The Var rule concludes.

Suppose $A \leadsto A'$. Applying *1* to $\mathcal{D}_2$ gives $\Gamma_1 \vdash A' : K$. The Var rule concludes.

3. Pattern-matching on Γ. If Γ is empty then $\varepsilon \leadsto \Gamma'$ forces $\Gamma' = \varepsilon$ and $\vdash \varepsilon$. Thus, let $\Gamma = \Delta; x_m : A$.

Suppose $\Delta; x_m : A \leadsto \Delta'; x_m : A$. Then by *3* applied to Δ: $\vdash \Delta'$. Now, because $\vdash \Delta; x_m : A$ it is the case that $\Delta \vdash A : K$. Using *2* gives $\Delta' \vdash A : K$. Therefore, $\vdash \Delta'; x_m : A$.

Suppose $\Delta; x_m : A \rightsquigarrow \Delta; x_m : A'$. Again $\vdash \Delta; x_m : A$ gives $\Delta \vdash A : K$. Using *1* gives $\Delta \vdash A' : K$. Thus, $\vdash \Delta; x_m : A'$. □

Lemma 2.45.

1. *If $\Gamma \vdash t : A$ and $t \rightsquigarrow^* t'$ then $\Gamma \vdash t' : A$*

2. *If $\Gamma \vdash t : A$ and $\Gamma \rightsquigarrow^* \Gamma'$ then $\Gamma' \vdash t : A$*

3. *If $\vdash \Gamma$ and $\Gamma \rightsquigarrow^* \Gamma'$ then $\vdash \Gamma'$*

Proof. For each property the proof proceeds by induction on multistep reduction using Lemma 2.44 and the IH in the inductive case. □

Lemma 2.46. *If $\Gamma \vdash t : A$ and $A \rightsquigarrow^* A'$ then $\Gamma \vdash t : A'$*

Proof. By classification: $\Gamma \vdash A : K$. Using Lemma 2.45 gives $\Gamma \vdash A' : K$. Note that $A \equiv A'$. Thus, by the Conv rule $\Gamma \vdash t : A'$. □

Theorem 2.47 (Preservation)**.** *If $\Gamma \vdash t : A$, $\Gamma \rightsquigarrow^* \Gamma'$, $t \rightsquigarrow^* t'$, and $A \rightsquigarrow^* A'$ then $\Gamma' \vdash t' : A'$*

Proof. Consequence of Lemma 2.45 and Lemma 2.46. □

2.6 Classification

Classification is a critical property of a system like CC with unified syntax. It allows for the syntax to instead be stratified into levels which would enable an intrinsic presentation of the system. For the core theory of Cedille2 there are only two universes like the original CC, thus the stratification places terms into three separate levels. A term is either a *kind* (thus $A = \Box$), a *type* (thus $\Gamma \vdash A : \Box$), or a *term* (thus $\Gamma \vdash A : \star$). Determining the appropriate level for syntax is also decidable with a classification function defined in Figure 2.11. This function is crafted to maintain preservation of classification after both reduction and erasure. Note that because the function is defined on syntax it is possible that there is no valid level because the syntax is not a proof, in these cases the syntax is given the classification *undefined*.

Definition 2.48.

1. t term *iff* $\mathcal{C}(t) = \text{term}$

2. t type *iff* $\mathcal{C}(t) = \text{type}$

3. t kind *iff* $\mathcal{C}(t) = \text{kind}$

$$
\begin{aligned}
\lfloor \text{term} \rfloor &= x_\star & \lfloor \text{kind} \rfloor &= \star \\
\lfloor \text{type} \rfloor &= x_\square & \lfloor \text{undefined} \rfloor &= \delta(\star)
\end{aligned}
$$

$$
\begin{array}{rl}
\mathcal{C}(x_\square) = \text{type} & \mathcal{C}(\star) = \text{kind} \\
\mathcal{C}(x_\star) = \text{term} & \mathcal{C}(\diamond) = \text{type} \\
\mathcal{C}(\lambda_\tau\, x\!:\!A.\, t) = \text{type} & \quad\text{if } (A \text{ kind or } A \text{ type}) \text{ and } t \text{ type} \\
\mathcal{C}(\lambda_0\, x\!:\!A.\, t) = \text{term} & \quad\text{if } (A \text{ kind or } A \text{ type}) \text{ and } t \text{ term} \\
\mathcal{C}(\lambda_\omega\, x\!:\!A.\, t) = \text{term} & \quad\text{if } A \text{ type and } t \text{ term} \\
\mathcal{C}((x:A) \to_\tau B) = \text{kind} & \quad\text{if } (A \text{ kind or } A \text{ type}) \text{ and } B \text{ kind} \\
\mathcal{C}((x:A) \to_0 B) = \text{type} & \quad\text{if } (A \text{ kind or } A \text{ type}) \text{ and } B \text{ type} \\
\mathcal{C}((x:A) \to_\omega B) = \text{type} & \quad\text{if } A \text{ type and } B \text{ type} \\
\mathcal{C}((\lambda_\tau\, x\!:\!A.\, t) \bullet_\tau a) = \text{type} & \quad\text{if } (A \text{ kind and } a \text{ type}) \text{ or } (A \text{ type and } a \text{ term}) \\
 & \qquad\text{and } t \text{ type and } [x := \lfloor \mathcal{C}(a) \rfloor]t \text{ type} \\
\mathcal{C}(f \bullet_\tau a) = \text{type} & \quad\text{if } (a \text{ type or } a \text{ term}) \text{ and } f \text{ type} \\
\mathcal{C}((\lambda_0\, x\!:\!A.\, t) \bullet_0 a) = \text{term} & \quad\text{if } (A \text{ kind and } a \text{ type}) \text{ or } (A \text{ type and } a \text{ term}) \\
 & \qquad\text{and } t \text{ term and } [x := \lfloor \mathcal{C}(a) \rfloor]t \text{ term} \\
\mathcal{C}(f \bullet_0 a) = \text{term} & \quad\text{if } (a \text{ type or } a \text{ term}) \text{ and } f \text{ term} \\
\mathcal{C}((\lambda_\omega\, x\!:\!A.\, t) \bullet_\omega a) = \text{term} & \quad\text{if } A \text{ type and } a, t \text{ term and } [x := \lfloor \mathcal{C}(a) \rfloor]t \text{ term} \\
\mathcal{C}(f \bullet_\omega a) = \text{term} & \quad\text{if } a \text{ term and } f \text{ term} \\
\mathcal{C}((x:A) \cap B) = \text{type} & \quad\text{if } A \text{ type and } B \text{ type} \\
\mathcal{C}([t_1, t_2; A]) = \text{term} & \quad\text{if } t_1, t_2 \text{ term and } A \text{ type} \\
\mathcal{C}(t.1) = \text{term} & \quad\text{if } t \text{ term} \\
\mathcal{C}(t.2) = \text{term} & \quad\text{if } t \text{ term} \\
\mathcal{C}(a =_A b) = \text{type} & \quad\text{if } a, b \text{ term and } A \text{ type} \\
\mathcal{C}(\text{refl}(t; A)) = \text{term} & \quad\text{if } t \text{ term and } A \text{ type} \\
\mathcal{C}(\vartheta(e, a, b; T)) = \text{term} & \quad\text{if } e, a, b \text{ term and } T \text{ type} \\
\mathcal{C}(\psi(e, a, b; A, P)) = \text{term} & \quad\text{if } e, a, b \text{ term and } A, P \text{ type} \\
\mathcal{C}(\varphi(a, b, e)) = \text{term} & \quad\text{if } a, b, e \text{ term} \\
\mathcal{C}(\delta(e)) = \text{term} & \quad\text{if } e \text{ term} \\
\mathcal{C}(t) = \text{undefined} & \quad\text{otherwise}
\end{array}
$$

Figure 2.11: Classification function for sorting raw syntax into three distinct levels: types, kinds, and terms. If the syntactic form does not adhere to the basic structure needed to be correctly sorted then it is assigned undefined and cannot be a proof.

Note that the condition $[x := \lfloor \mathcal{C}(a) \rfloor]t$ type and others like it are necessary. Take for example $\lambda_\tau\, x : \star.\, x_\star$. This is not well-typed and hence not a proof, but it also should not be a kind, type, or term because it will prevent preservation of classification during reduction. If a term then the application will correctly produce a term, but if a type then an application will reduce to a type.

Lemma 2.49. *The definition of $\mathcal{C}(-)$ is terminating*

Proof. The definition is structural except application cases. In particular, application cases require evaluating $\mathcal{C}([x := \lfloor \mathcal{C}(a) \rfloor]t)$ for some subexpressions a and t. Note that computing $\mathcal{C}(-)$ on subxpressions is of course terminating, but moreover $\lfloor - \rfloor$ is a constant function returning a constant syntactic form. Thus, a measure of size can be constructed such that the size of $\lfloor \mathcal{C}(a) \rfloor$ is always zero for any a. Substitution of syntactic forms of zero size do not change the size of the resulting term, therefore $\mathcal{C}([x := \lfloor \mathcal{C}(a) \rfloor]t)$ is a terminating invocation. □

Lemma 2.50. *If $\mathcal{C}(t)$ is defined then $\mathcal{C}(t) = \mathcal{C}(|t|)$*

Proof. By induction on t. Type-like syntax is homomorphic and thus the equation holds by the IH. Term-like syntax eliminates most of the extra structure leaving behind only another term-like syntax. A few cases are presented to illuminate both situations.

Case: $t = a =_A b$

> Have $|a =_A b| = |a| =_{|A|} |b|$, and because $\mathcal{C}(a =_A b)$ is defined it must be the case that a, b term and A type. Applying the IH gives $\mathcal{C}(a) = \mathcal{C}(|a|)$, $\mathcal{C}(b) = \mathcal{C}(|b|)$, and $\mathcal{C}(A) = \mathcal{C}(|A|)$. Thus, $|a| =_{|A|} |b|$ type.

Case: $t = (\lambda_0\, x : A.\, t) \bullet_0 a$

> Have $|(\lambda_0\, x : A.\, t) \bullet_0 a| = |t|$ and t term. Thus, by the IH $|t|$ term.

Case: $t = \text{refl}(t; A)$

> Have $|\text{refl}(t; A)| = \lambda\, x : \diamond.\, x_\star$, and by computation $\lambda_\omega\, x : \diamond.\, x_\star$ term.

□

Lemma 2.51. *If $\mathcal{C}(t)$ and $\mathcal{C}(b)$ are defined then*

$$\mathcal{C}([x := t]b) = \mathcal{C}([x := \lfloor \mathcal{C}(t) \rfloor]b)$$

Proof. If $\mathcal{C}(t)$ is defined then clearly $\mathcal{C}(t) = \mathcal{C}(\lfloor \mathcal{C}(t) \rfloor)$ by definition. The lemma is then shown by induction on b. □

Lemma 2.52. *If $\mathcal{C}(s)$ is defined and $s \rightsquigarrow t$ then $\mathcal{C}(s) = \mathcal{C}(t)$*

Proof. By induction on $s \rightsquigarrow t$, note that $\mathcal{C}(-)$ is structural making the inductive cases trivial. The first projection case is similar to the second projection case and thus omitted.

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

Suppose wlog that $m = \tau$, then $((\lambda_\tau\, x\!:\!A.\, b) \bullet_\tau t)$ type. Note that t type or t term by unraveling the previous definition. Now $[x := \lfloor \mathcal{C}(t) \rfloor]b$ type. By Lemma 2.51 and the above observation: $[x := t]b$ type.

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

Have $[t_1, t_2; A]$ term and by deconstructing the definition t_2 term.

Case: $\psi(\text{refl}(z; Z), a, b; A, P) \bullet_\omega t \rightsquigarrow t$

Have $(\psi(\text{refl}(z; Z), a, b; A, P) \bullet_\omega t)$ term and by deconstruction the definition t term.

Case: $\vartheta(\text{refl}(z; Z), a, b; T) \rightsquigarrow \text{refl}(a; T)$

Have $\vartheta(\text{refl}(z; Z), a, b; T)$ term and by deconstruction the definition a term and T type. Thus, $\text{refl}(a; T)$ term.

□

Lemma 2.53. *If $\mathcal{C}(s)$ is defined and $s \rightsquigarrow^* t$ then $\mathcal{C}(s) = \mathcal{C}(t)$*

Proof. By induction on $s \rightsquigarrow^* t$ and Lemma 2.52. □

Theorem 2.54 (Soundness of $\mathcal{C}(-)$)**.**

1. *If $\Gamma \vdash t : A$ and $A = \square$ then t* kind
2. *If $\Gamma \vdash t : A$ and $\Gamma \vdash A : \square$ then t* type
3. *If $\Gamma \vdash t : A$ and $\Gamma \vdash A : \star$ then t* term

Proof. By induction on $\Gamma \vdash t : A$. The Fst amd PrmFst rules are omitted.

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

Have $\star$ kind and $A = \square$, hence trivial.

Case: $$\frac{x \notin FV(\Gamma_1; \Gamma_2) \qquad \overset{\mathcal{D}_1}{} \qquad \Gamma_1 \vdash \overset{\mathcal{D}_2}{A} : K}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

If $K = \square$ then $x_\square$ type and $\Gamma \vdash A : \square$. Otherwise, $K = \star$ and then $x_\star$ term with $\Gamma \vdash A : \star$.

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

Suppose wlog that $m = \tau$, now by the IH applied to $\mathcal{D}_1$: A kind or A type. Applying the IH to $\mathcal{D}_2$ gives B kind. Thus, $(x : A) \to_\tau B$ kind.

Case:
$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)} \qquad \overset{\mathcal{D}_2}{\Gamma; x_m : A \vdash t : B} \qquad \overset{\mathcal{D}_3}{x \notin FV(|t|) \text{ if } m = 0}}{\Gamma \vdash \lambda_m\, x\!:\!A.\, t : (x : A) \to_m B}$$

Suppose wlog that $m = \tau$. Applying the IH to $\mathcal{D}_1$ gives A kind or A type. Note by $\mathcal{D}_2$ that $\Gamma, x_\tau : A \vdash B : \Box$. Thus, applying the IH to $\mathcal{D}_2$ yields t type. Hence, $\lambda_\tau\, x\!:\!A.\, t$ type.

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

Suppose wlog that $m = \tau$. By classification and inversion with $\mathcal{D}_1$: $\Gamma \vdash (x : A) \to_\tau B : \Box$. Deconstructing this judgment yields $\Gamma \vdash A : K$. Applying the IH to $\mathcal{D}_2$ gives a type or a term. Applying the IH to $\mathcal{D}_1$ yields f type. If f is not an abstraction then the proof is done, thus suppose $f = \lambda\, x\!:\!A.\, t$. Have A kind or A type, but note that $\Gamma \vdash A : K$ thus the classification of a and A must agree. Moreover, t term. Suppose wlog that a type then $\lfloor \mathcal{C}(a) \rfloor = x_\Box$. However, this means that $\Gamma \vdash A : \Box$ and that $\Gamma, x_\tau : A \vdash t : B$. Thus, the occurrences of x in t must be annotated as $x_\Box$ otherwise the Var rule for x would fail. Hence, $[x := x_\Box]t = t$.

Case:
$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma; x_\tau : A \vdash B : \star}}{\Gamma \vdash (x : A) \cap B : \star}$$

Applying the IH to $\mathcal{D}_1$ and $\mathcal{D}_2$ gives A, B type. Hence, $(x : A) \cap B$ type.

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

Deconstructing $\mathcal{D}_1$ gives $\Gamma \vdash A : \star$ and $\Gamma, x : A \vdash B : \star$. Lemma 2.39 gives $\Gamma \vdash [x := t]B : \star$. Using the IH on $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$ yields $(x : A) \cap B$ type and t, s term. Thus, $[t, s; (x : A) \cap B]$ term.

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

By classification and inversion on $\mathcal{D}_1$: $\Gamma \vdash (x : A) \cap B : \star$. Using the IH on $\mathcal{D}_1$ gives t term. Hence, $t.2$ term.

Case:
$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash b : A}}{\Gamma \vdash a =_A b : \star}$$

Applying the IH to $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$ yields A type and a, b term. Hence, $a =_A b$ type.

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

Applying the IH to $\mathcal{D}_1$ and $\mathcal{D}_2$ gives A type and t term. Hence, $\mathrm{refl}(t; A)$ term.

$$\text{Case:}\quad \frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : \star \qquad \Gamma \vdash \overset{\mathcal{D}_2}{a} : A \qquad \Gamma \vdash \overset{\mathcal{D}_3}{b} : A \qquad \Gamma \vdash e : \overset{\mathcal{D}_4}{a} =_A b \qquad \Gamma \vdash P : (y : A) \overset{\mathcal{D}_5}{\to_\tau} (p : a =_A y_\star) \to_\tau \star}{\Gamma \vdash \psi(e, a, b; A, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

Classification and inversion on $\mathcal{D}_4$ gives $\Gamma \vdash a =_A b : \star$. Likewise, $\Gamma \vdash (y : A) \to_\tau (p : a =_A y_\star) \to_\tau \star : \Box$. Applying the IH to all subderivations yields A, P type and a, b, e term. Hence, $\psi(e, a, b; A, P)$ term.

$$\text{Case:}\quad \frac{\Gamma \vdash (x : \overset{\mathcal{D}_1}{A}) \cap B : \star \qquad \Gamma \vdash a : (\overset{\mathcal{D}_2}{x} : A) \cap B \qquad \Gamma \vdash b : (\overset{\mathcal{D}_3}{x} : A) \cap B \qquad \Gamma \vdash e : \overset{\mathcal{D}_4}{a.1} =_A b.1}{\Gamma \vdash \vartheta(e, a, b; (x : A) \cap B) : a =_{(x:A)\cap B} b}$$

By classification, inversion and the IH used with $\mathcal{D}_4$: e term. The IH applied to $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ yields a, b term and $(x : A) \cap B$ type.

$$\text{Case:}\quad \frac{\Gamma \vdash \overset{\mathcal{D}_1}{a} : A \qquad \Gamma \vdash b : (\overset{\mathcal{D}_2}{x} : A) \cap B \qquad \Gamma \vdash e : \overset{\mathcal{D}_3}{a} =_A b.1}{\Gamma \vdash \varphi(a, b, e) : (x : A) \cap B}$$

By Lemma 2.40: $\Gamma \vdash (x : A) \cap B : K$. However, $K = \Box$ is impossible by inversion. Using the IH and inversion applied to the sub-derivations yields: a, b, e term. Thus, $\varphi(a, b, e)$ term.

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

Classification, inversion, and the IH applied to $\mathcal{D}_1$ gives e term. Hence, $\delta(e)$ term.

$$\text{Case:}\quad \frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : K \qquad \Gamma \vdash \overset{\mathcal{D}_2}{t} : B \qquad \overset{\mathcal{D}_3}{A \equiv B}}{\Gamma \vdash t : A}$$

Classification, inversion, $\mathcal{D}_1$ and $\mathcal{D}_3$ yield $\Gamma \vdash B : K$. Suppose wlog that $K = \star$. Applying the IH to $\mathcal{D}_2$ gives t term.

□

2.7 Derivations

$$\begin{aligned}
\mathrm{Cast} &= \lambda_\tau\, A{:}\star.\, \lambda_\tau\, B{:}\star.\, (f : A_\square \to_\tau A_\square \cap B_\square) \cap ((a : A_\square) \to_0 a_\star = (f_\star \bullet_0 a_\star).1)\\
\mathrm{castIrrel} &= \lambda_0\, A{:}\star.\, \lambda_0\, B{:}\star.\, \lambda_0\, k{:}\mathrm{Cast} \bullet_\tau A_\square \bullet_\tau B_\square.\\
&\qquad [\lambda_\omega\, a{:}A_\square.\, \varphi(a_\star, k_\star.1 \bullet_\omega a_\star, k_\star.2 \bullet_0 a_\star), \lambda_0\, a{:}A_\square.\, \mathrm{refl}(a_\star; A_\square); \mathrm{Cast} \bullet_\tau A_\square \bullet_\tau B_\square]
\end{aligned}$$

Theorem 2.55.

1. $\mathrm{Cast} : \star \to_\tau \star \to_\tau \star$

2. $\mathrm{castIrrel} : (A : \star) \to_0 (B : \star) \to_0 \mathrm{Cast} \bullet_\tau A_\square \bullet_\tau B_\square \to_0 \mathrm{Cast} \bullet_\tau A_\square \bullet_\tau B_\square$

$$\begin{aligned}
\mathrm{False} &= (X : \star) \to_0 X_\square\\
\mathrm{Top} &= \mathrm{False} \to_0 \mathrm{False}\\
\mathrm{topInj} &= \lambda_0\, A{:}\star.\, \lambda_\omega\, a{:}A_\square.\\
&\qquad [\lambda_0\, f{:}\mathrm{False}.\, \varphi(a_\star, f_\star \bullet_0 (A_\square \cap \mathrm{False}), f_\star \bullet_0 (a_\star = (f_\star \bullet_0 (A_\square \cap \mathrm{False})).1).2), a_\star; A_\square \cap \mathrm{False}]\\
\mathrm{View} &= \lambda_\tau\, A{:}\star.\, \lambda_\tau\, x{:}\mathrm{Top}.\, (z : (\mathrm{Top} \cap A_\square)) \times (x_\star = z_\star.1)\\
\mathrm{intrView} &= \lambda_0\, A{:}\star.\, \lambda_\omega\, x{:}\mathrm{Top}.\, \lambda_0\, a{:}A.\, \lambda_0\, e{:}(x_\star = (\mathrm{topInj} \bullet_0 A_\square \bullet_\omega a_\star).1).\\
&\qquad \mathrm{sigma} \bullet_\omega \varphi(x_\star, \mathrm{topInj} \bullet_0 A_\square \bullet_\omega a_\star, e_\star) \bullet_\omega \mathrm{refl}(x_\star; \mathrm{Top})\\
\mathrm{elimView} &= \lambda_0\, A{:}\star.\, \lambda_\omega\, b{:}\mathrm{Top}.\, \lambda_0\, v{:}\mathrm{View} \bullet_\tau A_\square \bullet_\tau b_\star.\, \varphi(b_\star, \mathrm{dfst} \bullet_\omega v, \mathrm{dsnd} \bullet_\omega v).2
\end{aligned}$$

Theorem 2.56.

1. $\mathrm{False} : \star$

2. $\mathrm{Top} : \star$

3. $\mathrm{topInj} : (A : \star) \to_0 A_\square \to_\omega \mathrm{Top} \cap A_\square$

4. $\mathrm{View} : \star \to_\tau \mathrm{Top} \to_\tau \star$

5. $\mathrm{intrView} : \begin{array}{l}(A : \star) \to_0 (x : \mathrm{Top}) \to_\omega (a : A_\square) \to_0\\ (x_\star = (\mathrm{topInj} \bullet_0 A_\square \bullet_\omega a_\star).1) \to_0 \mathrm{View} \bullet_\tau A_\square \bullet_\tau x_\star\end{array}$

6. $\mathrm{elimView} : (A : \star) \to_0 (b : \mathrm{Top}) \to_\omega \mathrm{View} \bullet_\tau A_\square \bullet_\tau b_\star \to_0 A$

Chapter 3

PROOF NORMALIZATION

There are several techniques for showing strong normalization of a PTS, including saturated sets [48], model theory [97], realizability [76], etc. Geuvers and Nederhof describe yet another technique that models CC inside F^{ω} where term dependencies are all erased at the type level [50]. In this chapter the technique of Geuvers and Nederhof will be adapted to show strong normalization of proof reduction. Note, this will not entail that objects are strongly normalizing. Moreover, proof normalization ends up being a rather weak property, as it will not entail consistency either. Nevertheless, it is an important stepping stone to strong normalization for objects.

3.1 Model Description

Figure 3.1 describes the syntax of System F^{ω} augmented with pairs. The reduction relation for this system is presented in Figure 3.2 and the inference judgment in Figure 3.3. System F^{ω} augmented with pairs is only slightly different from the original PTS description of F^{ω}. Moreover, it is a subsystem of the Calculus of Inductive Constructions and thus enjoys various metatheoretic properties such as substitution and weakening lemmas, preservation, strong normalization, and consistency.

The model follows all the same principles for the CC fragment of Cedille2. For example, consider the Lam rule.

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

$$\begin{aligned}
t &::= x \mid \mathfrak{b}(\kappa_1, x : t_1, t_2) \mid \mathfrak{c}(\kappa_2, t_1, \ldots, t_{\mathfrak{a}(\kappa_2)})\\
\kappa_1 &::= \lambda \mid \Pi\\
\kappa_2 &::= \star \mid \Box \mid \mathrm{app} \mid \mathrm{prod} \mid \mathrm{pair} \mid \mathrm{fst} \mid \mathrm{snd}\\
&\mathfrak{a}(\star) = \mathfrak{a}(\Box) = 0\\
&\mathfrak{a}(\mathrm{fst}) = \mathfrak{a}(\mathrm{snd}) = 1\\
&\mathfrak{a}(\mathrm{app}) = \mathfrak{a}(\mathrm{prod}) = \mathfrak{a}(\mathrm{pair}) = 2
\end{aligned}$$

$$\begin{aligned}
\star &:= \mathfrak{c}(\star)\\
\Box &:= \mathfrak{c}(\Box)\\
\lambda\, x\!:\!t_1.\, t_2 &:= \mathfrak{b}(\lambda, x : t_1, t_2)\\
(x : t_1) \to t_2 &:= \mathfrak{b}(\Pi, x : t_1, t_2)\\
t_1\ t_2 &:= \mathfrak{c}(\mathrm{app}, t_1, t_2)
\end{aligned}
\qquad\qquad
\begin{aligned}
t_1 \times t_2 &= \mathfrak{c}(\mathrm{prod}, t_1, t_2)\\
(t_1, t_2) &= \mathfrak{c}(\mathrm{pair}, t_1, t_2)\\
t.1 &= \mathfrak{c}(\mathrm{fst}, t)\\
t.2 &= \mathfrak{c}(\mathrm{snd}, t)
\end{aligned}$$

Figure 3.1: Syntax for System F^{ω} with pairs.

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

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

$$\begin{gathered}(\lambda\, x\!:\!A.\, b)\ t \rightsquigarrow [x := t]b \\ [t_1, t_2].1 \rightsquigarrow t_1 \\ [t_1, t_2].2 \rightsquigarrow t_2\end{gathered}$$

Figure 3.2: Reduction rules for System F$^\omega$ with pairs.

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

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

$$\frac{\Gamma \vdash A : \Box \qquad \Gamma, x : A \vdash B : \Box}{\Gamma \vdash (x : A) \to B : \Box}\ \textsc{Pi1}$$

$$\frac{\Gamma \vdash (x : A) \to B : K \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda\, x\!:\!A.\, t : (x : A) \to B}\ \textsc{Lam}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t.1 : A}\ \textsc{Fst}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t.2 : B}\ \textsc{Snd}$$

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

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

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

$$\frac{\Gamma \vdash A : \star \qquad \Gamma \vdash B : \star}{\Gamma \vdash A \times B : \star}\ \textsc{Int}$$

$$\frac{\Gamma \vdash A \times B : \star \qquad \Gamma \vdash t : A \qquad \Gamma \vdash s : B}{\Gamma \vdash (t, s) : A \times B}\ \textsc{Pair}$$

Figure 3.3: Typing rules for System F$^\omega$ with pairs. The variable K is a metavariable representing either $\star$ or $\Box$.

The goal is to find three semantic functions: one for kinds ($V(-)$); one for types ($\llbracket - \rrbracket$); and one for terms ($[-]$), such that:

1. $\llbracket \Gamma \rrbracket \vdash_\omega \llbracket (x : A) \to_m B \rrbracket : V(\mathrm{codom}_\Pi(m))$

2. $\llbracket \Gamma; x_m : A \rrbracket \vdash_\omega [t] : \llbracket B \rrbracket$

3. $\llbracket \Gamma \rrbracket \vdash [\lambda_m\, x\!:\!A.\, t] : \llbracket (x : A) \to_m B \rrbracket$

In order for this to work, term dependencies must all be dropped in function types. Moreover, kinds are squished, such that $V(\Box) = V(\star) = \star$. Thus, the judgment $\llbracket \Gamma \rrbracket \vdash_\omega \llbracket (x : A) \to_m B \rrbracket : V(\mathrm{codom}_\Pi(m))$ must form an F$^\omega$ type. The kind and type semantics is allowed to throw away terms and reductions because it only serves the purpose to maintain a well-typed output. Instead,

$$\begin{aligned}
V(\square) &= \star \\
V(\star) &= \star \\
V((x : A) \to_m B) &= V(A) \to V(B) && \text{if } A \text{ kind} \\
V((x : A) \to_m B) &= V(B) && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
[\![\square]\!] &= 0 \\
[\![\star]\!] &= 0 \\
[\![x_\square]\!] &= x \\
[\![(x : A) \to_m B]\!] &= (x : V(A)) \to [\![A]\!] \to [\![B]\!] && \text{if } A \text{ kind} \\
[\![(x : A) \to_m B]\!] &= (x : [\![A]\!]) \to [\![B]\!] && \text{if } A \text{ type} \\
[\![\lambda_\tau\, x \mathbin{:} A.\, t]\!] &= \lambda\, x \mathbin{:} V(A).\, [\![t]\!] && \text{if } A \text{ kind} \\
[\![\lambda_\tau\, x \mathbin{:} A.\, t]\!] &= [\![t]\!] && \text{if } A \text{ type} \\
[\![f \bullet_\tau a]\!] &= [\![f]\!]\ [\![a]\!] && \text{if } a \text{ type} \\
[\![f \bullet_\tau a]\!] &= [\![f]\!] && \text{if } a \text{ term} \\
[\![(x : A) \cap B]\!] &= [\![A]\!] \times [\![B]\!] \\
[\![a =_A b]\!] &= \mathrm{Id}
\end{aligned}$$

$$\begin{aligned}
[\![x_m : A]\!] &= x : V(A), w_x : [\![A]\!] && \text{if } A \text{ kind} \\
[\![x_m : A]\!] &= x : [\![A]\!] && \text{if } A \text{ type} \\
[\![\varepsilon]\!] &= 0 : \star, \bot : (X : \star) \to X \\
[\![\Gamma, x_m : A]\!] &= [\![\Gamma]\!], [\![x_m : A]\!]
\end{aligned}$$

Figure 3.4: Model for kinds and types, not that type dependencies are dropped. Define Id := $(X : \star) \to X \to X$.

it is the term semantics that must take care to preserve all possible reductions such that strong normalization is a consequence of the model.

For dependent intersections, the type semantics is the obvious one: $[\![(x : A) \cap B]\!] = [\![A]\!] \times [\![B]\!]$. Note that because A must be a type, it must be the case that $x \notin FV([\![B]\!])$ otherwise the resulting type is not well-formed in F^ω. This is true already for function types, thus this extension needs no special treatment. For equality the situation is special, the approach taken is to interpret all equalities as the type of the identity function: $[\![a =_A b]\!] = \mathrm{Id}$. There does not appear to be a more sensible choice, as the dependencies a and b must be dropped.

The model interpretation for contexts always introduces two fresh variables, $0 : \star$ which is a canonical type, and $\bot : (X : \star) \to X$ which is used to construct canonical inhabitants for any type or kind. Note that including $\bot$ prevents this model from entailing consistency for the source system. Regardless, F^ω is strongly normalizing in all contexts, thus the addition of $\bot$ does not prevent the model from serving its current purpose. Before exploring more in-depth examples of

$$\begin{aligned}
c^{B} &= \bot\ B && \text{if } B \text{ type}\\
c^{\star} &= 0\\
c^{(x:A)\to B} &= \lambda\, x\!:\!A.\, c^{B}
\end{aligned}$$

$$\begin{aligned}
[*] &= c^{0}\\
[x_{\Box}] &= w_x\\
[x_{\star}] &= x\\
[(x : A) \to_m B] &= c^{0\to 0\to 0}\ [A]\ ([x := c^{V(A)}][w_x := c^{\llbracket A\rrbracket}][B]) && \text{if } A \text{ kind}\\
[(x : A) \to_m B] &= c^{0\to 0\to 0}\ [A]\ ([x := c^{\llbracket A\rrbracket}][B]) && \text{if } A \text{ type}\\
[\lambda_m\, x\!:\!A.\, t] &= (\lambda\, y\!:\!0.\, \lambda\, x\!:\!V(A).\, \lambda\, w_x\!:\!\llbracket A\rrbracket.\, [t])\ [A] && \text{if } A \text{ kind}\\
[\lambda_m\, x\!:\!A.\, t] &= (\lambda\, y\!:\!0.\, \lambda\, x\!:\!\llbracket A\rrbracket.\, [t])\ [A] && \text{if } A \text{ type}\\
[f \bullet_m a] &= [f]\ \llbracket a\rrbracket\ [a] && \text{if } a \text{ type}\\
[f \bullet_m a] &= [f]\ [a] && \text{if } a \text{ term}
\end{aligned}$$

$$\begin{aligned}
[(x : A) \cap B] &= c^{0\to 0\to 0}\ [A]\ ([x := c^{\llbracket A\rrbracket}][B])\\
[[t_1, t_2; A]] &= (\lambda\, y\!:\!0.\, ([t_1], [t_2]))\ [A]\\
[t.1] &= [t].1\\
[t.2] &= [t].2\\
[a =_A b] &= c^{0\to\llbracket A\rrbracket\to\llbracket A\rrbracket\to 0}\ [A]\ [a]\ [b]\\
[\mathrm{refl}(t; A)] &= (\lambda\, y_1\!:\!0.\, \lambda\, y_2\!:\!\llbracket A\rrbracket.\, \mathrm{id})\ [A]\ [t]\\
[\psi(e, a, b; A, P)] &= (\lambda\, y_1\!:\!0.\, \lambda\, y_2\ y_3\!:\!\llbracket A\rrbracket.\, \lambda\, y_2\!:\!\llbracket A\rrbracket \to \mathrm{Id} \to 0.\, [e]\ \llbracket P\rrbracket)\ [A]\ [a]\ [b]\ [P]\\
[\vartheta(e, a, b; T)] &= (\lambda\, y_1\!:\!\llbracket T\rrbracket.\, \lambda\, y_2\!:\!0.\, \lambda\, y_3\!:\!\llbracket T\rrbracket.\, [e])\ [b]\ [T]\ [a]\\
[\varphi(a, b, e)] &= (\lambda\, y\!:\!\mathrm{Id}.\, ([a], [b].2))\ [e]\\
[\delta(e)] &= (\lambda\, y\!:\!\mathrm{Id}.\, \bot)\ [e]
\end{aligned}$$

Figure 3.5: Model for terms, note that critically every subexpression is represented in the model to make sure no reductions are potentially lost. The definition of c is used to construct a canonical element for any kind or type. Define id $:= \lambda\, X\!:\!\star.\, \lambda\, x\!:\!X.\, x$.

the model the reader is invited to skim to the semantic functions in Figure 3.4 and Figure 3.5.

Consider the following examples to garner intuition for the semantic model:

1. Given $\varepsilon \vdash_{\varsigma 2} \lambda_0\, X\!:\!\star.\, \lambda_\omega\, x\!:\!X_\square.\, x_\star : (X : \star) \to_0 X_\square \to_\omega X_\square$ then

$$[\![\varepsilon]\!] = 0 : \star;\ \bot : (X : \star) \to X$$

$$[\lambda_0\, X\!:\!\star.\, \lambda_\omega\, x\!:\!X_\square.\, x_\star] = (\lambda\, y\!:\!0.\, \lambda\, X\!:\!\star.\, \lambda\, w_X\!:\!0.\, (\lambda\, y\!:\!0.\, \lambda\, x\!:\!X.\, x)\ w_X)\ c^0$$

$$[\![(X : \star) \to_0 X_\square \to_\omega X_\square]\!] = (X : \star) \to 0 \to X \to X$$

2. Given $\Gamma \vdash_{\varsigma 2} t : T$ where $\Gamma = A : \star;\ B : \star;\ a : A_\square;\ f : A_\square \to_\omega (x : A_\square) \cap B_\square$, $t = [(f_\star \bullet_\omega a_\star).1, (f_\star \bullet_\omega a_\star).2; (x : A_\square) \cap B_\square]$, and $T = (x : A_\square) \cap B_\square$ then

$$\begin{aligned} &[\![A : \star; B : A \to_\tau \star; a : A; f : A \to_\omega (x : A) \cap B]\!] = \\ &\quad 0 : \star;\ \bot : (X : \star) \to X;\ A : \star;\ w_A : 0;\ B : \star;\ w_B : 0; \\ &\quad a : A;\ f : A \to A \times B \end{aligned}$$

$$[[(f \bullet_\omega a).1, (f \bullet_\omega a).2]] = (\lambda\, y\!:\!0.\, ((f\ a).1, (f\ a).2))\ (c^{0\to 0\to 0}\ w_A\ w_B)$$

$$[\![(x : A) \cap B]\!] = A \times B$$

Notice that from the perspective of the type semantics ($[\![-]\!]$) that term dependencies in predicates must be dropped, but that they are preserved in the term semantics ($[-]$). Thus, extra layers of abstraction are added when interpreting function arguments that are kinds to capture the two different usages of that variable in the separate semantic functions.

3.2 Model Soundness

With the model defined the next step is to prove it is sound. The process begins by showing the interpretation of kinds ($V(-)$) is sound. This is not particularly difficult as the kind interpretation is quite simple. After proving soundness lemmas about substitution and conversion are also shown and follow without much difficulty.

Theorem 3.1 (Soundness of V)**.** *If* $\Gamma \vdash_{\varsigma 2} t : \square$ *then* $\Delta \vdash_\omega V(t) : \square$ *for any* Δ

Proof. By induction on $\Gamma \vdash_{\varsigma 2} t : \square$. The cases: Lam, App, Int, Pair, Fst, Snd, Eq, Refl, Subst, Prm, Cast, Sep, and Conv are impossible by inversion.

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

Trivial by the Ax rule.

Case: $$\frac{x \notin \overset{\mathcal{D}_1}{FV}(\Gamma_1; \Gamma_2) \qquad \Gamma_1 \vdash \overset{\mathcal{D}_2}{A} : K}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

By $\mathcal{D}_2$: $\Gamma \vdash \Box : K$ which is impossible.

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

Suppose A is a kind, then $\text{dom}_\Pi(m, K) = \Box$ and $V((x : A) \to_m B) = V(A) \to V(B)$. Applying the IH to $\mathcal{D}_1$ and $\mathcal{D}_2$ gives $\Delta_1 \vdash_\omega V(A) : \Box$ and $\Delta_2 \vdash_\omega V(B) : \Box$. However, note that there are no variables in any well-defined $V(t)$ which $V(A)$ and $V(B)$ are. Thus, $\Delta \vdash_\omega V(A) : \Box$ and $\Delta, x : V(A) \vdash_\omega V(B) : \Box$ by properties of F^ω. Now by the Pɪ1 rule $\Delta \vdash_\omega V(A) \to V(B) : \Box$ as required.

Suppose A is a type, then $\text{dom}_\Pi(m, K) = \star$ and $V((x : A) \to_m B) = V(B)$. By the IH applied to $\mathcal{D}_2$: $\Delta \vdash_\omega V(B) : \Box$.

□

Lemma 3.2. *If* $\Gamma_1 \vdash A : \Box$, $\Gamma_2 \vdash B : \Box$, *and* $A \equiv B$ *then* $V(A) = V(B)$

Proof. By induction on $\Gamma \vdash A : \Box$. Note that A is either $\star$ or $(x : C) \to_\tau D$. Suppose $A = \star$, then because $\star \equiv B$ it must be that $B = \star$. Thus, $V(A) = \star = V(B)$.

Suppose $A = (x : C_1) \to_\tau D_1$, but this forces $B = (x : C_2) \to_\tau D_2$ where $C_1 \equiv C_2$ and $D_1 \equiv D_2$. Note that $\Gamma \vdash C_1 : K$ and $\Gamma, x : C_1 \vdash D_1 : \Box$. Now by the IH: $V(D_1) = V(D_2)$ (note that the contexts need not agree). If C_1 is a kind, then $V((x : C_1) \to_\tau D_1) = V(C_1) \to V(D_1)$ and by the IH $V(C_1) = V(C_2)$. Instead, if C_1 is a type then $V((x : C_1) \to_\tau D_1) = V(D_1)$, but $V(D_1) = V(D_2)$. Thus, $V(A) = V((x : C_1) \to_\tau D_1) = V((x : C_2) \to_\tau D_2) = V(B)$. □

Lemma 3.3. *If* $\Gamma \vdash V(t) : \Box$ *then* $[x := b]V(t) = V(t) = V([x := b]t)$

Proof. By induction on t and inversion on $\Gamma \vdash V(t) : \Box$. Note that there are only two possibilities:

Case: $t = \star$

Have $[x := b]V(\star) = [x := b]\star = \star = V(\star) = V([x := b]\star)$.

Case: $t = (x : A) \to_m B$

Note that A must be a kind or a type because $\Gamma \vdash V(t) : \Box$. Suppose A is a kind, then $V((x : A) \to_m B) = V(A) \to V(B)$. Destructing the judgment gives $\Gamma \vdash V(A) : \Box$ and $\Gamma, x : V(A) \vdash V(B) : \Box$. Thus, by the IH: $[x := b]V(A) = V(A) = V([x := b]A)$ and $[x := b]V(B) = V(B) = V([x := b]B)$. By computation, $V([x := b](x : A) \to_m B) = V((x : [x := b]A) \to_m [x := b]B) = V([x := b]A) \to V([x := b]B) = V(A) \to V(B) = V((x : A) \to_m B)$. Also, by computation $[x := b]V((x : A) \to_m B) = [x := b](V(A) \to V(B)) = [x := b]V(A) \to [x := b]V(B) = V(A) \to V(B) = V((x : A) \to_m B)$.

Suppose A is a type, then $V((x : A) \to_m B) = V(B)$. By the IH: $[x := b]V(B) = V(B) = V([x := b]B)$.

$\Box$

Next is demonstrating soundness of the type semantics. Note again that type variables cannot appear free in the result of a well-defined interpretation of types. This is codified in the next lemma, and soundness follows from it and soundness of the model for kinds. A standard substitution lemma is proven after.

Lemma 3.4. *Suppose $\Gamma \vdash t : A$, $x_m : B \in \Gamma$, and B type, then $x \notin FV(\llbracket t \rrbracket)$ where $A = \Box$ or $\Gamma \vdash A : \Box$*

Proof. Note that the restrictions on A makes sure that $\llbracket - \rrbracket$ is well-defined. The definition of $\llbracket - \rrbracket$ intentionally throws away any dependence on terms. Thus, if x is a term, because B is a type, the only places where x may appear in t have all been thrown away. Therefore, $x \notin FV(\llbracket t \rrbracket)$. $\Box$

Theorem 3.5 (Soundness of $\llbracket - \rrbracket$)**.** *If $\Gamma \vdash_{\varsigma_2} t : A$ then $\llbracket \Gamma \rrbracket \vdash_\omega \llbracket t \rrbracket : V(A)$ where $A = \Box$ or $\Gamma \vdash A : \Box$*

Proof. By induction on $\Gamma \vdash_{\varsigma_2} t : A$. The cases: Pair, Fst, Snd, Refl, Subst, Prm, Cast, and Sep are impossible by inversion on $A = \Box$ or $\Gamma \vdash A : \Box$.

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

By computation $\llbracket \star \rrbracket = 0$ and $V(\Box) = \star$. Note that $0 : \star \in \llbracket \Gamma \rrbracket$ thus this case is concluded by the Var rule.

Case: $$\frac{\overset{\mathcal{D}_1}{x \notin FV(\Gamma_1; \Gamma_2)} \qquad \overset{\mathcal{D}_2}{\Gamma_1 \vdash A : K}}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

Note that $A \neq \Box$ by $\mathcal{D}_2$, thus $K = \Box$. By computation $\llbracket x_\Box \rrbracket = x$. Moreover, A kind thus $x : V(A) \in \llbracket \Gamma \rrbracket$. Thus, $\llbracket \Gamma \rrbracket \vdash_\omega x : V(A)$

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

By computation $V(\mathrm{codom}_\Pi(m)) = V(\mathrm{dom}_\Pi(m, K)) = \star$. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_\omega \llbracket A \rrbracket : \star$

$\mathcal{D}_2$. $\llbracket \Gamma, x_m : A \rrbracket \vdash_\omega \llbracket B \rrbracket : \star$

Suppose that A is a kind. Then $\llbracket (x : A) \to_m B \rrbracket = (x : V(A)) \to \llbracket A \rrbracket \to \llbracket B \rrbracket$ and $\llbracket \Gamma, x_m : A \rrbracket = \llbracket \Gamma \rrbracket, x : V(A), w_x : \llbracket A \rrbracket$. The Pi2 rule applied with the results of the IH gives

$$\llbracket\Gamma\rrbracket, x : V(A) \vdash_\omega \llbracket A\rrbracket \to \llbracket B\rrbracket : \star$$

Now by Lemma 3.1 applied to $\mathcal{D}_1$: $\llbracket\Gamma\rrbracket \vdash_\omega V(A) : \square$. Using the Pi1 rule gives $\llbracket\Gamma\rrbracket \vdash_\omega V(A) \to \llbracket A\rrbracket \to \llbracket B\rrbracket : \star$.

Suppose that A is a type. Then $\llbracket(x : A) \to_m B\rrbracket = (x : \llbracket A\rrbracket) \to \llbracket B\rrbracket$ and $\llbracket\Gamma, x_m : A\rrbracket = \llbracket\Gamma\rrbracket, x : \llbracket A\rrbracket$. Thus, by the Pi2 rule $\llbracket\Gamma\rrbracket \vdash \llbracket A\rrbracket \to \llbracket B\rrbracket : \star$.

$$\text{Case:}\quad \frac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)} \qquad \overset{\mathcal{D}_2}{\Gamma; x_m : A \vdash t : B} \qquad \overset{\mathcal{D}_3}{x \notin FV(|t|) \text{ if } m = 0}}{\Gamma \vdash \lambda_m\, x{:}A.\, t : (x : A) \to_m B}$$

It must be the case that $\Gamma \vdash (x : A) \to_m B : \square$. Thus, $m = \tau$. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket\Gamma\rrbracket \vdash_\omega \llbracket(x : A) \to_\tau B\rrbracket : \star$

$\mathcal{D}_2$. $\llbracket\Gamma, x_\tau : A\rrbracket \vdash_\omega \llbracket t\rrbracket : V(B)$

Suppose A is a kind. Then $\llbracket(x : A) \to_\tau B\rrbracket = (x : V(A)) \to \llbracket A\rrbracket \to \llbracket B\rrbracket$, $\llbracket\Gamma, x_m : A\rrbracket = \llbracket\Gamma\rrbracket, x : V(A), w_x : \llbracket A\rrbracket$, and $\llbracket\lambda_\tau\, x{:}A.\, t\rrbracket = \lambda\, x{:}V(A).\, \llbracket t\rrbracket$. Note that $\llbracket\Gamma\rrbracket \vdash c^{\llbracket A\rrbracket} : \llbracket A\rrbracket$. Thus, by substitution lemma for F$^\omega$: $\llbracket\Gamma\rrbracket, x : V(A) \vdash_\omega [w_x := c^{\llbracket A\rrbracket}]\llbracket t\rrbracket : [w_x := c^{\llbracket A\rrbracket}]V(B)$. However, because A is kind and by Lemma 3.4: $[w_x := c^{\llbracket A\rrbracket}]\llbracket t\rrbracket = \llbracket t\rrbracket$. Note also that $FV(V(B))$ is empty, thus $[w_x := c^{\llbracket A\rrbracket}]V(B) = V(B)$. Thus, $\llbracket\Gamma\rrbracket, x : V(A) \vdash_\omega \llbracket t\rrbracket : V(B)$. Moreover, by Theorem 3.1 it is the case that $\llbracket\Gamma\rrbracket \vdash V(A) : \square$. Using the Lam rule gives $\llbracket\Gamma\rrbracket \vdash_\omega \lambda\, x{:}V(A).\, \llbracket t\rrbracket : V(A) \to V(B)$.

Suppose A is a type. Then $\llbracket\Gamma, x_m : A\rrbracket = \llbracket\Gamma\rrbracket, x : \llbracket A\rrbracket$ and $\llbracket\lambda_\tau\, x{:}A.\, t\rrbracket = \llbracket t\rrbracket$. Note additionally that $V((x : A) \to_m B) = V(B)$. Note that $\llbracket\Gamma\rrbracket \vdash c^{\llbracket A\rrbracket} : \llbracket A\rrbracket$. By substitution lemma, Lemma 3.4, and as above: $\llbracket\Gamma\rrbracket \vdash_\omega \llbracket t\rrbracket : V(B)$.

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

Note that it cannot be the case that $[x := a]B = \square$ by inversion on $\mathcal{D}_1$, thus $\Gamma \vdash [x := a]B : \square$ which force $m = \tau$. Furthermore, by $\mathcal{D}_1$: $\Gamma \vdash (x : A) \to_\tau B : \square$. Applying the IH to $\mathcal{D}_1$ thus gives $\llbracket\Gamma\rrbracket \vdash_\omega \llbracket f\rrbracket : V((x : A) \to_\tau B)$.

Suppose A is a kind, then a is a type. Thus, $V((x : A) \to_\tau B) = V(A) \to V(B)$ and $\llbracket f \bullet_\tau a\rrbracket = \llbracket f\rrbracket\ \llbracket a\rrbracket$. Applying the IH to $\mathcal{D}_2$ gives $\llbracket\Gamma\rrbracket \vdash_\omega \llbracket a\rrbracket : V(A)$. By the App rule: $\llbracket\Gamma\rrbracket \vdash \llbracket f\rrbracket\ \llbracket a\rrbracket : V(B)$. Now by Lemma 3.3: $V(B) = V([x := a]B)$.

Suppose A is a type, then a is a term. Thus, $V((x : A) \to_\tau B) = V(B)$ and $\llbracket f \bullet_\tau a\rrbracket = \llbracket f\rrbracket$. But, $\llbracket\Gamma\rrbracket \vdash_\omega \llbracket f\rrbracket : V(B)$ already. Now by Lemma 3.3: $V(B) = V([x := a]B)$.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : \star \qquad \Gamma; x_\tau : \overset{\mathcal{D}_2}{A} \vdash B : \star}{\Gamma \vdash (x : A) \cap B : \star}$$

Applying the IH gives:

$\mathcal{D}_1$. $[\![\Gamma]\!] \vdash_\omega [\![A]\!] : \star$

$\mathcal{D}_2$. $[\![\Gamma, x_\tau : A]\!] \vdash_\omega [\![B]\!] : \star$

Note that A is a type thus $[\![\Gamma, x_\tau : A]\!] = [\![\Gamma]\!], x : [\![A]\!]$. Applying the Lam rule twice reduces the goal to $[\![\Gamma]\!], [\![A]\!] : \star, [\![B]\!] : \star \vdash_\omega [\![A]\!] \times [\![B]\!] : \star$. However, the pair case is an otherwise simple F^ω type, thus a short sequence of rules concludes the case.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : \star \qquad \Gamma \vdash \overset{\mathcal{D}_2}{a} : A \qquad \Gamma \vdash \overset{\mathcal{D}_2}{b} : A}{\Gamma \vdash a =_A b : \star}$$

By computation $[\![a =_A b]\!] = \mathrm{Id}$ and $V(\star) = \star$. A short sequence of rules in F^ω yields $[\![\Gamma]\!] \vdash \mathrm{Id} : \star$.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : K \qquad \Gamma \vdash \overset{\mathcal{D}_2}{t} : B \qquad A \overset{\mathcal{D}_3}{\equiv} B}{\Gamma \vdash t : A}$$

Note that $A \neq \Box$ by $\mathcal{D}_1$, and furthermore that $K = \Box$. Now by classification and $\mathcal{D}_3$: $\Gamma \vdash B : \Box$. Applying the IH to $\mathcal{D}_2$ gives $[\![\Gamma]\!] \vdash_\omega [\![t]\!] : V(B)$. Using Lemma 3.2 with $\mathcal{D}_3$ gives $V(A) = V(B)$. Thus, the Conv rule concludes the case.

□

Lemma 3.6. *Suppose* $\Gamma \vdash_\omega [\![t]\!] : T$ *then* $[\![[x := b]t]\!] = [x := [\![b]\!]][\![t]\!]$

Proof. By induction on t and inversion on $\Gamma \vdash_\omega [\![t]\!] : T$. Thus, only the cases where $[\![t]\!]$ is well-defined need to be considered.

Case: $t = \star$ or $t = \Box$

The situation is the same because $[\![\star]\!] = [\![\Box]\!]$. By computation $[\![[x := b]\star]\!] = [\![\star]\!] = 0$ and $[x := [\![b]\!]][\![\star]\!] = [x := [\![b]\!]]0 = 0$.

Case: $t = y_\Box$

Suppose $x \neq y$, then by computation $[\![[x := b]y_\Box]\!] = [\![y_\Box]\!] = y$ and $[x := [\![b]\!]][\![y_\Box]\!] = [x := [\![b]\!]]y = y$. Suppose $x = y$, then $[\![[x := b]y_\Box]\!] = [\![b]\!]$ and $[x := [\![b]\!]][\![y_\Box]\!] = [x := [\![b]\!]]y = [\![b]\!]$.

Case: $t = (y : C) \to_m D$

Suppose A is a kind. Then $[\![[x := b](y : C) \to_m D]\!] = [\![(y : [x := b]C) \to_m ([x := b]D)]\!] = (y : V([x := b]A)) \to [\![[x := b]C]\!] \to [\![[x := b]D]\!]$. By Lemma 3.3 and applying the IH:

$$\begin{aligned}&(y : V([x := b]A)) \to [\![[x := b]C]\!] \to [\![[x := b]D]\!]\\&\quad = (y : [x := [\![b]\!]]V(A)) \to [x := [\![b]\!]][\![C]\!] \to [x := [\![b]\!]][\![D]\!]\\&\quad = [x := [\![b]\!]]((y : V(A)) \to [\![C]\!] \to [\![D]\!])\\&\quad = [x := [\![b]\!]][\![(y : C) \to_m D]\!]\end{aligned}$$

Suppose A is a type. Then $[\![[x := b](y : C) \to_m D]\!] = [\![(y : [x := b]C) \to_m ([x := b]D)]\!] = (y : [\![[x := b]C]\!]) \to [\![[x := b]D]\!]$. Applying the IH and chasing similar computations as above concludes the case.

Case: $t = \lambda_\tau\, C : c$.

Suppose C is a kind. Then $[\![[x := b](\lambda_\tau\, x : C.\, c)]\!] = [\![\lambda_\tau\, x : [x := b]C.\, [x := b]c]\!] = \lambda\, x : V([x := b]C).\, [\![[x := b]c]\!]$. By Lemma 3.3 and the IH:

$$\begin{aligned}&\lambda\, x : V([x := b]C).\, [\![[x := b]c]\!]\\&\qquad\qquad = \lambda\, x : [x := [\![b]\!]]V(C).\, [x := [\![b]\!]][\![c]\!]\\&\qquad\qquad = [x := [\![b]\!]](\lambda\, x : V(C).\, [\![c]\!])\\&\qquad\qquad = [x := [\![b]\!]][\![\lambda\, x : C.\, c]\!]\end{aligned}$$

Suppose C is a type. Then $[\![[x := b](\lambda_\tau\, x : C.\, c)]\!] = [\![\lambda_\tau\, x : [x := b]C.\, [x := b]c]\!] = [\![[x := b]c]\!]$. By the IH: $[\![[x := b]c]\!] = [x := [\![b]\!]][\![c]\!] = [x := [\![b]\!]][\![\lambda_\tau\, x : C.\, c]\!]$.

Case: $t = f \bullet_\tau a$

Suppose a is a type. Then $[\![[x := b](f \bullet_\tau a)]\!] = [\![([x := b]f \bullet_\tau [x := b]a)]\!] = [\![[x := b]f]\!]\ [\![[x := b]a]\!]$. Using the IH gives $[\![[x := b]f]\!]\ [\![[x := b]a]\!] = ([x := [\![b]\!]][\![f]\!])\ ([x := [\![b]\!]][\![a]\!]) = [x := [\![b]\!]]([\![f]\!]\ [\![a]\!]) = [x := [\![b]\!]][\![f \bullet_\tau a]\!]$.

Supppose a is a term. Then $[\![[x := b](f \bullet_\tau a)]\!] = [\![([x := b]f \bullet_\tau [x := b]a)]\!] = [\![[x := b]f]\!]$. Using the IH gives $[\![[x := b]f]\!] = [x := [\![b]\!]][\![f]\!] = [x := [\![b]\!]][\![f \bullet_\tau a]\!]$.

Case: $t = (y : C) \cap D$

By computation $[\![[x := b]((y : C) \cap D)]\!] = [\![(y : [x := b]C) \cap [x := b]D]\!] = [\![[x := b]C]\!] \times [\![[x := b]D]\!]$. Using the IH gives $[\![[x := b]C]\!] \times [\![[x := b]D]\!] = ([x := [\![b]\!]][\![C]\!]) \times ([x := [\![b]\!]][\![D]\!]) = [x := [\![b]\!]]([\![C]\!] \times [\![D]\!]) = [x := [\![b]\!]][\![(x : C) \cap D]\!]$.

Case: $t = c =_C d$

By computation $[\![[x := b](c =_C d)]\!] = [\![([x := b]c) =_{[x:=b]C} ([x := b]d)]\!] = \text{Id}$. Again, by computation $[x := [\![b]\!]][\![c =_C d]\!] = [x := [\![b]\!]]\text{Id} = \text{Id}$.

□

Finally, soundness of the term semantics must be shown. This is not as simple as the original argument for CC modelled in F^ω because conversion happens relative to erasure. Luckily, erasure is homomorphic on type-like structure, and because the type semantics drops any term dependencies it will be the case that erasure has no impact on the semantics of types.

Lemma 3.7. *If* $\Gamma \vdash_\omega V(t) : \square$ *then* $V(t) = V(|t|)$

Proof. By induction on t and inversion on $\Gamma \vdash V(t) : \square$.

Case: $t = \star$ or $t = \square$

By computation $V(|\square|) = V(\square) = V(\star) = V(|\star|)$.

Case: $t = (x : A) \to_m B$

Suppose A is a kind. By Lemma 2.50: $|A|$ kind. Then $V((x : A) \to_m B) = V(A) \to V(B)$. Note that the subexpressions are well-typed, thus by the IH $V(|A|) = V(A)$ and $V(|B|) = V(B)$. Now by computation $V(|(x : A) \to_m B|) = V((x : |A|) \to_m |B|) = V(|A|) \to V(|B|) = V(A) \to V(B)$.

Suppose A is not a kind. Then $V((x : A) \to_m B) = V(B)$. By the IH $V(|B|) = V(B)$. Thus, by computation $V(|(x : A) \to_m B|) = V((x : |A|) \to_m |B|) = V(|B|) = V(B)$.

□

Lemma 3.8. *If* $\Gamma \vdash_\omega [\![t]\!] : T$ *then* $[\![t]\!] = [\![|t|]\!]$

Proof. By induction on t and inversion on $\Gamma \vdash [\![t]\!] : T$. Erasure is again homomorphic on all remaining syntactic forms after inversion, thus only two cases are presented.

Case: $t = \star$ or $t = \square$ or $t = x_\square$

In each case $|t| = t$ thus trivial.

Case: $t = (x : A) \to_m B$

Have $|(x : A) \to_m B| = (x : |A|) \to_m |B|$. Suppose wlog that A is a kind. Then $[\![(x : |A|) \to_m |B|]\!] = (x : V(|A|)) \to [\![|A|]\!] \to [\![|B|]\!]$. By Lemma 3.7 and the IH $(x : V(|A|)) \to [\![|A|]\!] \to [\![|B|]\!] = (x : V(A)) \to [\![A]\!] \to [\![B]\!]$. Likewise, $[\![(x : A) \to_m B]\!] = (x : V(A)) \to [\![A]\!] \to [\![B]\!]$.

□

Now conversion of the kind and type models must be handled relative to erasure. The above lemmas demonstrate that if reduction happens in the erased term it should somehow be mirrored in reduction for the well-typed terms. For kinds this turns out to be simple equality, as any possible dependence involving reduction are always dropped the structure of $V(t)$ for any t is rigid. The type semantics is slightly more complicated, but the same intuition holds: if a reduction where to occur in a term dependency then the resulting type models are equal, otherwise the reduction is exactly mirrored in the model.

Lemma 3.9. *If* $\Gamma \vdash_\omega V(s) : \square$ *and* $|s| \rightsquigarrow t$ *then* $V(s) = V(t)$

Proof. By induction on $|s| \rightsquigarrow t$. Note that only binder reduction is possible by inversion on $\Gamma \vdash V(s) : \square$.

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

Inversion on $\Gamma \vdash V(s) : \square$ forces $s = (x : A) \to_m B$. Note that $|A| \rightsquigarrow A'$. Suppose A kind, then $V((x : A) \to_m B) = V(A) \to V(B)$. Now by the IH $V(A) = V(A')$ and $V((x : A') \to_m |B|) = V(A') \to V(B)$ by Lemma 3.7. Suppose A is not a kind, then $V((x : A) \to_m B) = V(B) = V((x : A') \to_m |B|)$.

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

Inversion on $\Gamma \vdash V(s) : \square$ forces $s = (x : A) \to_m B$. Note that $|B| \rightsquigarrow B'$. Suppose A kind, then $V((x : A) \to_m B) = V(A) \to V(B)$. Now by the IH $V(B) = V(B')$ and $V((x : |A|) \to_m B') = V(A) \to V(B')$ by Lemma 3.7. Suppose A is not a kind, then $V((x : A) \to_m B) = V(B) = V(B') = V((x : |A|) \to_m B')$.

□

Lemma 3.10. *If* $\Gamma \vdash_\omega [\![s]\!] : T$ *and* $|s| \rightsquigarrow t$ *then* $[\![s]\!] \rightsquigarrow [\![t]\!]$ *or* $[\![s]\!] = [\![t]\!]$

Proof. By induction on $|s| \rightsquigarrow t$. Note that only β-reduction is possible, as all other possible reduction steps are erased.

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

By inversion on $\Gamma \vdash [\![s]\!] : T$ it must be the case that $m = \tau$. Thus, $|s| = (\lambda_\tau\, x : |A|.\, |b|) \bullet_\tau |t|$ and $|s| \rightsquigarrow [x := |t|]|b|$. By Lemma 2.22: $[x := |t|]|b| = |[x := t]b|$. Now, Lemma 3.8 yields $[\![|[x := t]b|]\!] = [\![[x := t]b]\!]$ and $[\![|s|]\!] = [\![s]\!]$. Using Lemma 3.6 gives $[\![[x := t]b]\!] = [x := [\![t]\!]][\![b]\!]$. Suppose A is a kind, and thus t is a type. Then $[\![(\lambda_\tau\, x : A.\, b) \bullet_\tau t]\!] = (\lambda\, x : V(A).\, [\![b]\!])\ [\![t]\!] \rightsquigarrow [x := [\![t]\!]][\![b]\!]$. Suppose A is a type, and thus t is a term. Then $[\![(\lambda_\tau\, x : A.\, b) \bullet_\tau t]\!] = [\![b]\!]$, however this also means that $\Gamma \vdash [\![b]\!] : T$. The internally bound variable x is thrown away, so it cannot be the case that $[\![b]\!]$ is well-typed in F^ω while $x \in FV(b)$ (Note that x can be renamed to be disjoint from Γ), hence $x \notin FV(b)$. Thus, $[x := \llbracket t \rrbracket]\llbracket b \rrbracket = \llbracket b \rrbracket$ and the case is concluded.

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 inversion on $\Gamma \vdash \llbracket s \rrbracket : T$ it must be the case that κ is $*$, $\square$, $\bullet_\tau$, or eq. However, the cases $*$ and $\square$ are impossible because they do not reduce. Suppose $|s| = |f| \bullet_\tau |a|$ and assume wlog that $|a| \rightsquigarrow a'$. If a is a term then $\llbracket |f| \bullet_\tau |a| \rrbracket = \llbracket |f| \rrbracket = \llbracket |f| \bullet_\tau |a'| \rrbracket$ and $\llbracket |f| \rrbracket = \llbracket f \rrbracket$ by Lemma 3.8. Suppose a is a type. Then, by the IH $\llbracket a \rrbracket \rightsquigarrow \llbracket a' \rrbracket$ or $\llbracket a \rrbracket = \llbracket a' \rrbracket$. Now $\llbracket |f| \bullet_\tau |a| \rrbracket = \llbracket |f| \rrbracket \; \llbracket |a| \rrbracket$, but by Lemma 3.8: $\llbracket |f| \rrbracket \; \llbracket |a| \rrbracket = \llbracket f \rrbracket \; \llbracket a \rrbracket$. Thus, $\llbracket f \rrbracket \; \llbracket a \rrbracket \rightsquigarrow \llbracket f \rrbracket \; \llbracket a' \rrbracket$ or $\llbracket f \rrbracket \; \llbracket a \rrbracket = \llbracket f \rrbracket \; \llbracket a' \rrbracket$.

Suppose $|s| = |a| =_{|A|} |b|$. Note that $\llbracket u =_U v \rrbracket = \mathrm{Id}$ for any u, v, U. Thus, $\llbracket s \rrbracket = \llbracket |s| \rrbracket = \llbracket t \rrbracket$.

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 inversion on $\Gamma \vdash \llbracket s \rrbracket : T$ it must be the case that κ is Π_m, λ_τ, or $\cap$. The $\cap$ and λ_τ cases are similar to the Π_m case and thus omitted. Have $|s| = (x : |A|) \to_m |B|$ and note that $|A| \rightsquigarrow A'$. Suppose wlog that A kind. Now $\llbracket (x : |A|) \to_m |B| \rrbracket = (x : V(|A|)) \to \llbracket |A| \rrbracket \to \llbracket |B| \rrbracket$. By the IH: $\llbracket A \rrbracket \rightsquigarrow \llbracket A' \rrbracket$ or $\llbracket A \rrbracket = \llbracket A' \rrbracket$. Suppose wlog that $\llbracket A \rrbracket \rightsquigarrow \llbracket A' \rrbracket$, then $(x : V(|A|)) \to \llbracket |A| \rrbracket \to \llbracket |B| \rrbracket \rightsquigarrow (x : V(A')) \to \llbracket A' \rrbracket \to \llbracket |B| \rrbracket$ by Lemma 3.9. Now $\llbracket (x : A') \to_m |B| \rrbracket = (x : V(A')) \to \llbracket A' \rrbracket \to \llbracket |B| \rrbracket$.

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

By inversion on $\Gamma \vdash \llbracket s \rrbracket : T$ it must be the case that κ is Π_m, λ_τ, or $\cap$. The $\cap$ and λ_τ cases are similar to the Π_m case and thus omitted. Have $|s| = (x : |A|) \to_m |B|$ and note that $|B| \rightsquigarrow B'$. Suppose wlog that A kind. Now $\llbracket (x : |A|) \to_m |B| \rrbracket = (x : V(|A|)) \to \llbracket |A| \rrbracket \to \llbracket |B| \rrbracket$. By the IH: $\llbracket B \rrbracket \rightsquigarrow \llbracket B' \rrbracket$ or $\llbracket B \rrbracket = \llbracket B' \rrbracket$. Suppose wlog that $\llbracket B \rrbracket \rightsquigarrow \llbracket B' \rrbracket$, then $(x : V(|A|)) \to \llbracket |A| \rrbracket \to \llbracket |B| \rrbracket \rightsquigarrow (x : V(|A|)) \to \llbracket |A| \rrbracket \to \llbracket B' \rrbracket$. Now $\llbracket (x : |A|) \to_m B' \rrbracket = (x : V(|A|)) \to \llbracket |A| \rrbracket \to \llbracket B' \rrbracket$.

□

Lemma 3.11. *If* $\Gamma \vdash_\omega \llbracket s \rrbracket : T$ *and* $|s| \rightsquigarrow^* t$ *then* $\llbracket s \rrbracket \rightsquigarrow^* \llbracket t \rrbracket$

Proof. By induction on $|s| \rightsquigarrow^* t$. The reflexivity case is trivial by Lemma 3.8. Suppose $|s| \rightsquigarrow z$ and $z \rightsquigarrow^* t$. By Lemma 3.10 either $\llbracket s \rrbracket \rightsquigarrow \llbracket z \rrbracket$ or $\llbracket s \rrbracket = \llbracket z \rrbracket$. If $\llbracket s \rrbracket \rightsquigarrow \llbracket z \rrbracket$ then by preservation

$\Gamma \vdash \llbracket z \rrbracket : T$. Note that $|z| = z$ by Lemma 2.21 and because reduction does not introduce new syntactic forms. Applying the IH to $|z| \rightsquigarrow^* t$ gives $\llbracket z \rrbracket \rightsquigarrow^* \llbracket t \rrbracket$, thus $\llbracket s \rrbracket \rightsquigarrow^* \llbracket t \rrbracket$. If $\llbracket s \rrbracket = \llbracket z \rrbracket$ then obviously $\Gamma \vdash \llbracket z \rrbracket : T$ and the same argument as above works. □

With the reduction lemmas handled the required lemma about conversion is straightforward. Finally, soundness of the term semantics is proven by a straightforward induction on the inference judgment of ç_2.

Lemma 3.12. *If* $\Gamma \vdash_\omega \llbracket A \rrbracket : T$, $\Gamma \vdash_\omega \llbracket B \rrbracket : T$, A, B pseobj, *and* $A \equiv B$ *then* $\llbracket A \rrbracket \rightleftharpoons \llbracket B \rrbracket$

Proof. By Lemma 2.33 $|A| \rightleftharpoons |B|$. Deconstructing this gives $|A| \rightsquigarrow^* z$ and $|B| \rightsquigarrow^* z$. By Lemma 3.11: $\llbracket A \rrbracket \rightsquigarrow^* \llbracket z \rrbracket$ and $\llbracket B \rrbracket \rightsquigarrow^* \llbracket z \rrbracket$. Thus, $\llbracket A \rrbracket \rightleftharpoons \llbracket B \rrbracket$. □

Lemma 3.13. *If* $\Gamma \vdash_\omega t : T$ *and* $\Gamma \vdash_\omega a : A$ *then* $\Gamma \vdash (\lambda\, x\!:\!A.\, t)\ a : T$

Proof. Have $\Gamma \vdash_\omega \lambda\, x\!:\!A.\, t : A \to T$ because x does not appear free in t. Thus, by the App rule $\Gamma \vdash (\lambda\, x\!:\!A.\, t)\ a : T$. □

Lemma 3.14. *If* $\Gamma \vdash_\omega A : T$ *and* $(\bot : (X : \star) \to X) \in \Gamma$ *then* $\Gamma \vdash_\omega c^A : A$

Proof. If A type then the proof is trivial. If A kind then the proof follows by induction on the depth of the function type. □

Theorem 3.15 (Soundness of $[-]$)**.** *If* $\Gamma \vdash_{\text{ç}_2} t : A$ *then* $\llbracket \Gamma \rrbracket \vdash_\omega [t] : \llbracket A \rrbracket$

Proof. By induction on $\Gamma \vdash_{\text{ç}_2} t : A$. The Fst case is omitted because it is very similar to Snd. The cases Ax, Var, Pi, Lam, and App are the same as the translation from CC to F$^\omega$.

$$\text{Case:}\quad \frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : \star \qquad \Gamma; x_\tau : \overset{\mathcal{D}_2}{A} \vdash B : \star}{\Gamma \vdash (x : A) \cap B : \star}$$

Applying the IH to subderivations:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_\omega [A] : 0$

$\mathcal{D}_2$. $\llbracket \Gamma, x_\tau : A \rrbracket \vdash_\omega [B] : 0$

Note that $\llbracket \Gamma \rrbracket \vdash_\omega 0 \to 0 \to 0 : \star$. Thus, $\llbracket \Gamma \rrbracket \vdash_\omega c^{0 \to 0 \to 0} : 0 \to 0 \to 0$. By $\mathcal{D}_1$ it is the case that A type, thus $\llbracket \Gamma, x_\tau : A \rrbracket = \llbracket \Gamma \rrbracket, x : \llbracket A \rrbracket$. Using Lemma 3.5 on $\mathcal{D}_1$ gives $\llbracket \Gamma \rrbracket \vdash_\omega \llbracket A \rrbracket : \star$. The substitution lemma yields $\llbracket \Gamma \rrbracket \vdash_\omega [x := c^{\llbracket A \rrbracket}][B] : 0$. Now applying the App rule two times concludes the case.

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

Applying the IH to subderivations:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_\omega [(x : A) \cap B] : 0$

$\mathcal{D}_2$. $\llbracket \Gamma \rrbracket \vdash_\omega [t] : \llbracket A \rrbracket$

$\mathcal{D}_3$. $\llbracket \Gamma \rrbracket \vdash_\omega [s] : \llbracket [x := t]B \rrbracket$

By Lemma 3.6: $\llbracket [x := t]B \rrbracket = [x := \llbracket t \rrbracket]\llbracket B \rrbracket$. However, A is a type by $\mathcal{D}_1$ and thus $x \notin FV(\llbracket B \rrbracket)$, hence $[x := \llbracket t \rrbracket]\llbracket B \rrbracket = \llbracket B \rrbracket$. Now $\llbracket \Gamma \rrbracket \vdash_\omega ([t_1], [t_2]) : \llbracket A \rrbracket \times \llbracket B \rrbracket$ by the Pair rule. Applying 3.13 concludes the case.

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

Note by $\mathcal{D}_1$ that A is a type, thus $x \notin FV(\llbracket B \rrbracket)$. By Lemma 3.6: $\llbracket [x := t.1]B \rrbracket = [x := \llbracket t.1 \rrbracket]\llbracket B \rrbracket = \llbracket B \rrbracket$. Applying the IH to $\mathcal{D}_1$ gives $\llbracket \Gamma \rrbracket \vdash_\omega [t] : \llbracket A \rrbracket \times \llbracket B \rrbracket$. The Snd rule concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash b : A}}{\Gamma \vdash a =_A b : \star}$$

Applying the IH to subderivations:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_\omega [A] : 0$

$\mathcal{D}_2$. $\llbracket \Gamma \rrbracket \vdash_\omega [a] : \llbracket A \rrbracket$

$\mathcal{D}_3$. $\llbracket \Gamma \rrbracket \vdash_\omega [b] : \llbracket A \rrbracket$

Note that $\llbracket \Gamma \rrbracket \vdash_\omega 0 \to \llbracket A \rrbracket \to \llbracket A \rrbracket \to 0 : \star$. Thus, $\llbracket \Gamma \rrbracket \vdash_\omega c^{0 \to \llbracket A \rrbracket \to \llbracket A \rrbracket \to 0} : 0 \to \llbracket A \rrbracket \to \llbracket A \rrbracket \to 0$. Now applying the App rule three times concludes the case.

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

Applying the IH to subderivations:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_\omega [A] : 0$

$\mathcal{D}_2$. $\llbracket \Gamma \rrbracket \vdash_\omega [t] : \llbracket A \rrbracket$

Of course, $\llbracket \Gamma \rrbracket \vdash_\omega \mathrm{id} : \mathrm{Id}$. Thus, applying Lemma 3.13 twice concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : A} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a =_A b} \qquad \overset{\mathcal{D}_5}{\Gamma \vdash P : (y : A) \to_\tau (p : a =_A y_\star) \to_\tau \star}}{\Gamma \vdash \psi(e, a, b; A, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

Note that by classification and $\mathcal{D}_1$ it is that case that A type. Applying the IH to subderivations:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_\omega [A] : 0$

$\mathcal{D}_2$. $\llbracket \Gamma \rrbracket \vdash_\omega [a] : \llbracket A \rrbracket$

$\mathcal{D}_3$. $[\![\Gamma]\!] \vdash_\omega [b] : [\![A]\!]$

$\mathcal{D}_4$. $[\![\Gamma]\!] \vdash_\omega [e] : \mathrm{Id}$

$\mathcal{D}_5$. $[\![\Gamma]\!] \vdash_\omega [P] : [\![A]\!] \to \mathrm{Id} \to 0$

Now $[\![\Gamma]\!] \vdash_\omega [e]\ [\![P]\!] : [\![P]\!] \to [\![P]\!]$. Note also that $[\![P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e]\!] = [\![P]\!] \to [\![P]\!]$ because $P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A)$ is a type by $\mathcal{D}_3$ and $a, b, e, \mathrm{refl}(a; A)$ are all terms. Applying Lemma 3.13 four times concludes the case.

Case:
$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \cap B : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : (x : A) \cap B} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : (x : A) \cap B} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a.1 =_A b.1}}{\Gamma \vdash \vartheta(e, a, b; (x : A) \cap B) : a =_{(x:A)\cap B} b}$$

Applying the IH to subderivations:

$\mathcal{D}_1$. $[\![\Gamma]\!] \vdash_\omega [(x : A) \cap B] : 0$

$\mathcal{D}_2$. $[\![\Gamma]\!] \vdash_\omega [a] : [\![A]\!] \times [\![B]\!]$

$\mathcal{D}_3$. $[\![\Gamma]\!] \vdash_\omega [b] : [\![A]\!] \times [\![B]\!]$

$\mathcal{D}_4$. $[\![\Gamma]\!] \vdash_\omega [e] : \mathrm{Id}$

Applying Lemma 3.13 three times concludes the case.

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

Note by $\mathcal{D}_1$ it is clear that A is a type. Applying the IH to subderivations:

$\mathcal{D}_1$. $[\![\Gamma]\!] \vdash_\omega [a] : [\![A]\!]$

$\mathcal{D}_2$. $[\![\Gamma]\!] \vdash_\omega [b] : [\![A]\!] \times [\![B]\!]$

$\mathcal{D}_3$. $[\![\Gamma]\!] \vdash_\omega [e] : \mathrm{Id}$

Note that $[\![\Gamma]\!] \vdash_\omega [b].2 : [\![B]\!]$. Thus, $[\![\Gamma]\!] \vdash_\omega ([a], [b].2) : [\![A]\!] \times [\![B]\!]$. Applying Lemma 3.13 concludes the case.

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

By computation $[\delta(e)] = (\lambda\, x : \mathcal{I}([e]).\, \bot)\ [e]$ and $[\![(X : \star) \to_0 X]\!] = (X : \star) \to X$. Note that $[\![\Gamma]\!] \vdash_\omega \bot : (X : \star) \to X$ and by definition $[\![\Gamma]\!] \vdash_\omega [e] : \mathcal{I}([e])$. Thus, by Lemma 3.13: $[\![\Gamma]\!] \vdash [\delta(e)] : [\![(X : \star) \to_0 X]\!]$.

Case:
$$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : K} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash t : B} \qquad \overset{\mathcal{D}_3}{A \equiv B}}{\Gamma \vdash t : A}$$

By classification, $\mathcal{D}_1$ and $\mathcal{D}_3$: $\Gamma \vdash B : K$. Now using Theorem 3.5 gives $\llbracket \Gamma \rrbracket \vdash_\omega \llbracket A \rrbracket : \star$ and $\llbracket \Gamma \rrbracket \vdash_\omega \llbracket B \rrbracket : \star$. Note that A, B pseobj by Lemma 2.37 and $|A| \rightleftharpoons |B|$ by Lemma 2.33. By Lemma 3.12: $\llbracket A \rrbracket \rightleftharpoons \llbracket B \rrbracket$. Applying the IH to $\mathcal{D}_2$ gives $\llbracket \Gamma \rrbracket \vdash_\omega [t] : \llbracket B \rrbracket$. The Conv rule concludes the case.

□

3.3 Normalization

With soundness of the model shown the normalization argument follows in the same way as for CC modelled in F^ω. That is, proof reduction in $ç_2$ is bounded by reduction in F^ω, and thus because F^ω is strongly normalizing it provides a maximum number of reduction steps for which any proof must normalize in $ç_2$. Note that some reduction steps are technical, especially ϑ_i, but they are not conceptually difficult.

Lemma 3.16. $[x := b]c^A = c^{[x:=b]A}$

Proof. Straightforward by unraveling the definition of canonical elements (c) and applying substitution computation rules. □

Lemma 3.17. *If* $\Gamma \vdash t : A$ *and* $(x : B) \in \Gamma$ *then*

1. $[[x := b]a] = [x := \llbracket b \rrbracket][w_x := [b]][a]$ *if* B *kind*

2. $[[x := b]a] = [x := [b]][a]$ *if* B *type*

Proof. By induction on $\Gamma \vdash t : A$. Substitution is structural and with Lemma 3.6, Lemma 3.3, and Lemma 3.16 many cases are straightforward by induction. Thus, only the variable cases and the Int case are presented.

$$\text{Case: } \quad \dfrac{\overset{\mathcal{D}_1}{x \notin FV(\Gamma_1; \Gamma_2)} \qquad \overset{\mathcal{D}_2}{\Gamma_1 \vdash A : K}}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

Rename to y. Suppose $x \neq y$, then $[[x := b]y_\star] = y$, $[x := \llbracket b \rrbracket][w_x := [b]][y_\star] = y$, and $[x := [b]][y_\star] = y$. When $y_\Box$ the situation is the same. Suppose $x = y$ and that B kind. If B is kind, then it must be the case that $y_\Box$. Now $[[x := b]y_\Box] = [b]$ and $[x := \llbracket b \rrbracket][w_x := [b]][y_\Box] = [x := \llbracket b \rrbracket][w_x := [b]]w_y = [b]$. Suppose instead that B type, then $[[x := b]y_\star] = [b]$ and $[x := [b]][y_\star] = [x := [b]]y = [b]$.

$$\text{Case: } \quad \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma; x_\tau : A \vdash B : \star}}{\Gamma \vdash (x : A) \cap B : \star}$$

Suppose wlog that B is a kind. Then $[[x := b](y : A) \cap B] = [(y : [x := b]A) \cap [x := b]B] = c^{0 \to 0 \to 0}[[x := b]A]([y := c^{\llbracket [x:=b]A \rrbracket}][[x := b]B])$. Now by the IH, Lemma 3.6, and the fact that $w_x \notin FV(\llbracket A \rrbracket)$ the right-hand side is equal to $c^{0 \to 0 \to 0}[x := \llbracket b \rrbracket][w_x := [b]][A]([y :=$

$c^{[x := \llbracket b \rrbracket][w_x := [b]]\llbracket A \rrbracket}][x := \llbracket b \rrbracket][w_x := [b]][B])$. Consider $[x := \llbracket b \rrbracket][w_x := [b]][(y : A) \cap B] = [x := \llbracket b \rrbracket][w_x := [b]]c^{0 \to 0 \to 0}[A]([y := c^{\llbracket A \rrbracket}][B])$. Note that $x, w_x \notin FV(0 \to 0 \to 0)$, thus by Lemma 2.1, Lemma 3.16, and computation rules of substitution this matches the previous right-hand side.

□

Lemma 3.18. *If* $\Gamma \vdash s : T$ *and* $s \leadsto t$ *then* $[s] \leadsto^*_{\neq 0} [t]$

Proof. By induction on $s \leadsto t$. The first projection case is very similar to the second projection case. Note by a simple observation that $[-]$ replicates every subexpression on the left-hand side with a matching invocation of $[-]$ on the right-hand side. Thus, if there is a reduction inside a subexpression it will always be tracked in the corresponding $[-]$ invocation via the inductive hypothesis. For this reason the structural reduction cases are omitted.

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

Note by $\Gamma \vdash s : T$ that A is either a kind or a type. Suppose A is a kind and note that makes t a type. Then $[(\lambda_m\, x : A.\, b) \bullet_m t] = (\lambda\, y : 0.\, \lambda\, x : V(A).\, \lambda\, w_x : \llbracket A \rrbracket.\, [b])\ [A]\ \llbracket t \rrbracket\ [t]$. The variable y is fresh thus after one β-reduction $(\lambda\, x : V(A).\, \lambda\, w_x : \llbracket A \rrbracket.\, [b])\ \llbracket t \rrbracket\ [t]$. Now applying two more β-reductions yields $[x := \llbracket t \rrbracket][w_x := [t]][b]$. Note that $[[x := t]b] = [x := \llbracket t \rrbracket][w_x := [t]][b]$ by Lemma 3.17. Thus, $[s] \leadsto^*_{=3} [t]$, i.e. $[s]$ reduces to $[t]$ in three steps.

Suppose A is a type and note that makes t a term. Then $[(\lambda_m\, x : A.\, b) \bullet_m t] = (\lambda\, y : 0.\, \lambda\, w_x : \llbracket A \rrbracket.\, [b])\ [A]\ [t]$. The variable y is fresh thus after one β-reduction $(\lambda\, w_x : \llbracket A \rrbracket.\, [b])\ [t]$. Applying one more β-reduction yields $[x := [t]][b]$. Note that $[[x := t]b] = [x := [t]][b]$ by Lemma 3.17. Thus, $[s] \leadsto^*_{=2} [t]$.

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

Have $[[t_1, t_2; A].2] = ((\lambda\, y : 0.\, ([t_1], [t_2]))\ [A]).2$. Note that the variable y is fresh and thus not in $FV([t_1])$ or $FV([t_2])$. A second projection and one β-reduction yields $[t_2]$. Thus, $[s] \leadsto^*_{=2} [t_2]$.

Case: $\psi(\text{refl}(t; A_1), a, b; A_2, P) \bullet_\omega t \leadsto t$

Note that t is a term by inversion on $\Gamma \vdash s : T$. Have $[\psi(\text{refl}(t; A_1); A_2, P) \bullet_\omega t] = (\lambda\, y_1 : 0.\, \lambda\, y_2\ y_3 : \llbracket A \rrbracket.\, \lambda\, y_4 : \llbracket A_2 \rrbracket \to \text{Id} \to 0.\, [\text{refl}(t; A_1)]\ \llbracket P \rrbracket)\ [A_2]\ [a]\ [b]\ [P]\ [t]$. Applying four β-reductions yields $[\text{refl}(t; A_1)]\ \llbracket P \rrbracket\ [t]$. Now $[\text{refl}(t; A_1)] = (\lambda\, y_1 : 0.\, \lambda\, y_2 : \llbracket A_1 \rrbracket.\, \text{id})\ [A_1]\ [t]$. Applying two more β-reductions gives id $\llbracket P \rrbracket\ [t]$. Finally, applying two remaining β-reductions yields $[t]$. Thus, $[s] \leadsto^*_{=8} [t]$.

Case: $\vartheta(\text{refl}(t; A), a, b; T) \leadsto \text{refl}(a; T)$

Have $[\vartheta(\text{refl}(t; A), a, b; T)] = (\lambda\, y_1 : [\![T]\!].\, \lambda\, y_2 : 0.\, \lambda\, y_3 : [\![T]\!].\, ((\lambda\, y_1 : 0.\, \lambda\, y_2 : [\![A]\!].\, \text{id})\ [A]\ [t]))\ [b]\ [T]\ [a]$. Note that all y_i are fresh and thus not in the free variables of any subexpressions. Performing two β-reductions on the interior (the result of $[\text{refl}(t_1; A)]$) and the outermost β-reduction yields: $(\lambda\, y_2 : 0.\, \lambda\, y_3 : [\![T]\!].\, \text{id})\ [T]\ [a]$. Now $[\text{refl}(a; T)] = (\lambda\, y_2 : 0.\, \lambda\, y_3 : [\![T]\!].\, \text{id})\ [T]\ [a]$. Thus, $[s] \rightsquigarrow^*_{=3} [t]$.

□

Theorem 3.19 (Proof Normalization)**.** *If $\Gamma \vdash t : A$ then t is strongly normalizing and there exists a unique value t_n such that $t \rightsquigarrow^* t_n$*

Proof. Using Lemma 3.5 gives $[\![\Gamma]\!] \vdash_\omega [t] : [\![A]\!]$. Note that F^ω with pairs is strongly normalizing with a unique normal form (because it is also confluent). Thus, *all* possible reduction paths to the normal form are terminating. Let $\partial([t])$ be the *maximum* number of reduction steps $[t]$ could take to reach a normal form. Note that this value is computable by brute force search. Pick any sequence of reductions in t bounded by $\partial([t])$. If this sequence concludes in a value then t is strongly normalizing, because the sequence is arbitrary. If t is not a value then $t \rightsquigarrow^*_{>\partial[t]} t'$, but this is impossible by Lemma 3.18. Now by confluence of reduction, all values reached from any arbitrary reduction path must be joinable at a single value. Thus, $t \rightsquigarrow^* t_n$ where t_n is a unique value. □

Chapter 4

CONSISTENCY AND RELATIONSHIP TO CDLE

The Calculus of Dependent Lambda Eliminations (CDLE) was first introduced in 2017 [94] as the core system for the in progress Cedille tool. At that time, CDLE included complicated machinery for lifting lambda terms to the type-level enabling some large eliminations. Over the years, the core system for the Cedille tool was still referred to as CDLE as it evolved culminating in the current core system used in Cedille version 1.1.2 [96]. The ideas leading to CDLE, of course, grew over time with work on efficient lambda encodings in total theories [95]; self-types for encodings [45]; and experiments involving irrelevance [91, 90]. Ultimately, the modern version of CDLE, as presented in this chapter, is the culmination of these efforts.

CDLE is an affirmative answer to the question: is lambda-encoded data enough for a proof assistant? While there may be other philosophical objections, Mendler-style encodings have been shown to be efficient and enable course-of-values induction [41, 42]. Moreover, the edition of the φ construct, an idea borrowed from the direct computation rule of Nuprl [3], yields efficient data reuse via casts [39]. A non-exhaustive list of the successes of CDLE include: quotient subtypes [69]; coinductive data [61]; zero-cost constructor subtypes [70]; monotonic recursive types [59]; simulated large eliminations [58]; and inductive-inductive data [68].

CDLE commits to impredicative (i.e. parametric in sense of F^{ω}) quantification. With that in mind the well-studied reader may not be surprised at the power and versatility of CDLE. However, taming impredicative quantification without losing logical consistency is a difficult task. Indeed, this is precisely why several proof assistants have discarded impredicative quantification or relegated it into a universe of propositions. A core philosophy behind both Cedille and Cedille2 is to walk a different road and embrace impredicative quantification. To achieve that goal a realizability model was developed for CDLE to demonstrate logical consistency [96]. This chapter will describe a model of ς_2 in CDLE to prove consistency.

4.1 Calculus of Dependent Lambda Eliminations

CDLE is described using an intrinsic style where syntax is presented directly with the typing derivation. However, erasure it still a crucial part of CDLE which gives it an extrinsic philosophy. Whether a system is intrinsic or extrinsic is perhaps not a terribly interesting distinction. Technically, ς_2 is described extrinsically because syntax is defined independently of the typing relation, but there is no essential reason for this choice. Moreover, any intrinsic system necessarily admits a projection of its raw syntax, which would enable an extrinsic presentation. It is better to think about these details via their philosophical import. An intrinsic system wishes to say that raw syntax has no meaning, or at the very least no meaning that anyone should care about. Alternatively,

$$\frac{}{\Gamma \vdash \star} \qquad \frac{\Gamma \vdash A \rhd \star \qquad \Gamma; x : A \vdash \kappa}{\Gamma \vdash \Pi\, x : A.\, \kappa} \qquad \frac{\Gamma \vdash \kappa' \qquad \Gamma; x : \kappa' \vdash \kappa}{\Gamma \vdash \Pi\, x : \kappa'.\, \kappa}$$

Figure 4.1: Judgment for formation of kinds in CDLE.

$$\frac{(x : \kappa) \in \Gamma}{\Gamma \vdash x \rhd \kappa} \qquad \frac{\Gamma \vdash \kappa \qquad \Gamma; x : \kappa \vdash B \rhd \star}{\Gamma \vdash \forall\, x : \kappa.\, B \rhd \star}$$

$$\frac{\Gamma \vdash A \rhd \star \qquad \Gamma; x : A \vdash B \rhd \star}{\Gamma \vdash \forall\, x : A.\, B \rhd \star} \qquad \frac{\Gamma \vdash A \rhd \star \qquad \Gamma; x : A \vdash B \rhd \star}{\Gamma \vdash \Pi\, x : A.\, B \rhd \star}$$

$$\frac{\Gamma \vdash A \rhd \star \qquad \Gamma; x : A \vdash t \rhd \kappa}{\Gamma \vdash \lambda\, x : A.\, t \rhd \Pi\, x : A.\, \kappa} \qquad \frac{\Gamma \vdash \kappa' \qquad \Gamma; x : \kappa' \vdash t \rhd \kappa}{\Gamma \vdash \lambda\, x : \kappa'.\, t \rhd \Pi\, x : \kappa'.\, \kappa}$$

$$\frac{\Gamma \vdash f \rhd \Pi\, x : A.\, \kappa \qquad \Gamma \vdash a \lhd A}{\Gamma \vdash f\ a \rhd [x := \chi\ A - a]\kappa} \qquad \frac{\Gamma \vdash f \rhd \Pi\, x : \kappa_1.\, \kappa_2 \qquad \Gamma \vdash a \rhd \kappa_1' \qquad \kappa_1 \cong \kappa_1'}{\Gamma \vdash f \cdot a \rhd [x := a]\kappa_2}$$

$$\frac{\Gamma \vdash A \rhd \star \qquad \Gamma; x : A \vdash B \rhd \star}{\Gamma \vdash \iota\, x : A.\, B \rhd \star} \qquad \frac{FV(a\ b) \subseteq dom(\Gamma)}{\Gamma \vdash \{a \simeq b\} \rhd \star}$$

Figure 4.2: Inference judgment defining well-formed types and their inferred kind in CDLE.

an extrinsic system wishes to say that types are in some sense only annotations, and it is the raw syntax that is primary.

As one might guess these philosophical positions are not entirely black and white. For example, Pfenning demonstrates how both methods can be combined [79]. Cedille has been historically described as an extrinsic system. With Cedille2 it is more correct to be called a *combined* system, both intrinsic and extrinsic. That is, a *proof* has no meaning as just syntax, but an *object* discards the extra information as mere annotations.

The kind formation rules as presented in Figure 4.1, type formation rules in Figure 4.2, and term annotation rules in Figure 4.3. Lowercase letters are used to refer to metavariables of terms, uppercase letters for metavariables of types, and variations of κ for metavariables of kinds. Call-by-name reduction of the λ-calculus fragment is used in the rules for types and is written $A \rightsquigarrow_n B$. The purpose of this relation is only to reveal a constructor for a type, thus weak-head normal form is sufficient. Conversion for types is presented in Figure 4.4 and kind conversion in Figure 4.5. Note that these conversion relations correspond to β-conversion for types and kinds. Finally, erasure of terms (and only terms) is presented in Figure 4.6. Erasure is only meaningful for terms in CDLE unlike in ς_2 where it is defined for all raw syntax.

The presentation in this work deviates from other descriptions of CDLE by adding a symmetry rule for equality (ς). This rule is admissible using the rewrite rule (ρ), but it is convenient to have available for the model. Otherwise, the presentation is identical to the one by Stump and Jenkins [96].

$$\Gamma \vdash t \blacktriangleright A \text{ iff } \exists\, T.\ (\Gamma \vdash t \triangleright T) \land (T \rightsquigarrow_n^* A)$$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x \triangleright A}$$

$$\frac{T \rightsquigarrow_n^* \Pi\, x{:}A.\, B \qquad \Gamma; x : A \vdash t \triangleleft B}{\Gamma \vdash \lambda\, x.\, t \triangleleft T}$$

$$\frac{T \rightsquigarrow_n^* \forall\, x{:}\kappa.\, B \qquad \Gamma; x : \kappa \vdash t \triangleleft B}{\Gamma \vdash \Lambda\, x.\, t \triangleleft T}$$

$$\frac{\begin{array}{c} T \rightsquigarrow_n^* \forall\, x{:}A.\, B \\ \Gamma; x : A \vdash t \triangleleft B \qquad x \notin FV(|t|) \end{array}}{\Gamma \vdash \Lambda\, x.\, t \triangleleft T}$$

$$\frac{\begin{array}{c} T \rightsquigarrow_n^* \iota\, x{:}A.\, B \qquad \Gamma \vdash t_1 \triangleleft A \\ \Gamma \vdash t_2 \triangleleft [x := \chi\ A - t_1]B \qquad |t_1| \rightleftharpoons_\eta |t_2| \end{array}}{\Gamma \vdash [t_1, t_2] \triangleleft T}$$

$$\frac{\Gamma \vdash t \blacktriangleright \iota\, x{:}A.\, B}{\Gamma \vdash t.2 \triangleright [x := t.1]B}$$

$$\frac{\Gamma \vdash t \triangleright A \qquad A \cong \{\lambda\, x\ y.\, x \simeq \lambda\, x\ y.\, y\}}{\Gamma \vdash \delta - t \triangleleft T}$$

$$\frac{\Gamma \vdash A \triangleright \star \qquad \Gamma \vdash t \triangleleft A}{\Gamma \vdash \chi\ A - t \triangleright A}$$

$$\frac{\Gamma \vdash e \blacktriangleright \{a \simeq b\}}{\Gamma \vdash \text{ç}\ e \triangleright \{b \simeq a\}}$$

$$\frac{\Gamma \vdash t \triangleright A \qquad A \cong B}{\Gamma \vdash t \triangleleft B}$$

$$\frac{\Gamma \vdash f \blacktriangleright \Pi\, x{:}A.\, B \qquad \Gamma \vdash a \triangleleft A}{\Gamma \vdash f\ a \triangleleft [x := \chi\ A - a]B}$$

$$\frac{\Gamma \vdash f \blacktriangleright \Pi\, x{:}\kappa_1.\, B \qquad \Gamma \vdash a \triangleright \kappa_2 \qquad \kappa_1 \cong \kappa_2}{\Gamma \vdash f \cdot a \triangleleft [x := a]B}$$

$$\frac{\Gamma \vdash f \blacktriangleright \forall\, x{:}A.\, B \qquad \Gamma \vdash a \triangleleft A}{\Gamma \vdash f\ \text{-}a \triangleleft [x := \chi\ A - a]B}$$

$$\frac{\Gamma \vdash f \blacktriangleright \iota\, x{:}A.\, B}{\Gamma \vdash t.1 \triangleright A}$$

$$\frac{\begin{array}{c} T \rightsquigarrow_n^* \{a \simeq b\} \\ FV(t) \subseteq dom(\Gamma) \qquad |a| \rightleftharpoons_\eta |b| \end{array}}{\Gamma \vdash \beta\{t\} \triangleleft T}$$

$$\frac{\begin{array}{c} \Gamma \vdash e \blacktriangleright \{a \simeq b'\} \qquad FV(b') \subseteq dom(\Gamma) \\ |b'| \rightleftharpoons_\eta |b| \qquad \Gamma \vdash [x := b]A \triangleright \star \\ \Gamma \vdash t \triangleleft [x := b]A \qquad [x := a]A \cong T \end{array}}{\Gamma \vdash \rho\ e\ @\ x\ \langle b \rangle.\ A - t \triangleleft T}$$

$$\frac{\begin{array}{c} e \triangleleft \{a \simeq b\} \\ \Gamma \vdash a \triangleright A \qquad FV(b) \subseteq dom(\Gamma) \end{array}}{\Gamma \vdash \varphi\ e - a\ \{b\} \triangleright A}$$

$$\frac{\begin{array}{c} e \triangleleft \{a \simeq b\} \\ \Gamma \vdash a \triangleleft A \qquad FV(b) \subseteq dom(\Gamma) \end{array}}{\Gamma \vdash \varphi\ e - a\ \{b\} \triangleleft A}$$

Figure 4.3: Bidirectional annotation judgment for terms defining when an annotated term infers of checks against a type in CDLE.

$$\frac{A \rightsquigarrow_n^* A' \not\rightsquigarrow_n \qquad B \rightsquigarrow_n^* B' \not\rightsquigarrow_n \qquad A' \cong^t B'}{A \cong B}$$

$$\frac{}{x \cong^t x} \qquad \frac{\kappa_1 \cong \kappa_2 \qquad B_1 \cong B_2}{\forall\, x : \kappa_1 .\, B_1 \cong^t \forall\, x : \kappa_2 .\, B_2}$$

$$\frac{A_1 \cong A_2 \qquad B_1 \cong B_2}{\forall\, x : A_1 .\, B_1 \cong^t \forall\, x : A_2 .\, B_2} \qquad \frac{A_1 \cong A_2 \qquad B_1 \cong B_2}{\Pi\, x : A_1 .\, B_1 \cong^t \Pi\, x : A_2 .\, B_2}$$

$$\frac{A_1 \cong A_2 \qquad B_1 \cong B_2}{\lambda\, x : A_1 .\, B_1 \cong^t \lambda\, x : A_2 .\, B_2} \qquad \frac{\kappa_1 \cong \kappa_2 \qquad B_1 \cong B_2}{\lambda\, x : \kappa_1 .\, B_1 \cong^t \lambda\, x : \kappa_2 .\, B_2}$$

$$\frac{A_1 \cong A_2 \qquad B_1 \cong B_2}{\iota\, x : A_1 .\, B_1 \cong^t \iota\, x : A_2 .\, B_2} \qquad \frac{A_1 \cong^t A_2 \qquad |b_1| \rightleftharpoons_\eta |b_2|}{A_1 \ b_1 \cong^t A_2 \ b_2}$$

$$\frac{A_1 \cong^t A_2 \qquad B_1 \cong B_2}{A_1 \cdot B_1 \cong^t A_2 \cdot B_2} \qquad \frac{|a_1| \rightleftharpoons_\eta |a_2| \qquad |b_1| \rightleftharpoons_\eta |b_2|}{\{a_1 \simeq b_1\} \cong^t \{a_2 \simeq b_2\}}$$

Figure 4.4: Definition of conversion for types in CDLE.

$$\frac{}{\star \cong \star}$$

$$\frac{A_1 \cong A_2 \qquad \kappa_1 \cong \kappa_2}{\Pi\, x : A_1 .\, \kappa_1 \cong \Pi\, x : A_2 .\, \kappa_2} \qquad \frac{\kappa_1' \cong \kappa_2' \qquad \kappa_1 \cong \kappa_2}{\Pi\, x : \kappa_1' .\, \kappa_1 \cong \Pi\, x : \kappa_2' .\, \kappa_2}$$

Figure 4.5: Defintion of conversion for kinds in CDLE.

$$\begin{array}{rlrl}
|x| &= x & |\lambda\, x.\, t| &= \lambda\, x.\, |t| \\
|f \ a| &= |f| \ |a| & |f \cdot a| &= |f| \\
|\Lambda\, x.\, t| &= |t| & |f \text{ -}a| &= |f| \\
|[t_1, t_2]| &= |t_1| & |t.1| &= |t| \\
|t.2| &= |t| & |\beta\{t\}| &= |t| \\
|\delta - t| &= \lambda\, x.\, x & |\rho \ e \ @ \ x \ \langle a \rangle .\ A - t| &= |t| \\
|\varphi \ e - a \ \{b\}| &= |b| & |\chi \ A - t| &= |t|
\end{array}$$

Figure 4.6: Erasure of terms in CDLE, note that erasure is not defined for types or kinds.

A few useful facts about CDLE are needed before defining the model. First, some helpful terms are defined below. Note that an annotation rule (χ) is added to some terms in order to guarantee that each definition always infers a type, as opposed to checks against a type. The Bool definition is a standard Church encoded boolean type, with its two associated values (tt and ff). An identity type, Id, is defined as a desired output of the model for the equality of ç$_2$. Indeed, CDLEs equality is very flexible in comparison to ç$_2$. Not only is it untyped, but it allows for any well-scoped term to serve as the erasure (or object) of a reflexivity proof.

$$\begin{aligned}
\mathrm{Bool} &:= \forall\, X\!:\!\star.\, X \to X \to X \\
\mathrm{tt} &:= \chi\ \mathrm{Bool}\ \ -\Lambda\, X.\, \lambda\, x\ y.\, x \\
\mathrm{ff} &:= \chi\ \mathrm{Bool}\ \ -\Lambda\, X.\, \lambda\, x\ y.\, y \\
\mathrm{Id} &:= \lambda\, A\!:\!\star.\, \lambda\, a\ b\!:\!A.\, \iota\, e\!:\!\{a \simeq b\}.\, \iota\, y\!:\!\{(\lambda\, x.\, x) \simeq e\}.\, \forall\, X\!:\!\star.\, X \to X \\
\mathrm{refl} &:= \chi\ \forall\, A\!:\!\star.\, \forall\, a\!:\!A.\, \mathrm{Id} \cdot A\ a\ a\ - \\
&\quad \Lambda\, A\ a.\, [\beta\{\lambda\, x.\, x\}, [\beta\{\lambda\, x.\, x\}, \Lambda\, X.\, \lambda\, x.\, x]] \\
\mathrm{delta} &:= \chi\ \mathrm{Id} \cdot \mathrm{Bool}\ \mathrm{tt}\ \mathrm{ff} \to \forall\, X\!:\!\star.\, X\ - \\
&\quad \lambda\, e.\, (\delta - e.1) \cdot (\mathrm{Id} \cdot \mathrm{Bool}\ \mathrm{tt}\ \mathrm{ff} \to \forall\, X\!:\!\star.\, X)\ e
\end{aligned}$$

Aside from the previous terms it is also useful to have terms representing the target output of the substitution and promotion rules of ç$_2$. All of these terms are constructed to obtain specific erasures.

$$\begin{aligned}
&\mathrm{theta} := \chi\ \forall\, A\!:\!\star.\, \forall\, B\!:\!A \to \star.\, \forall\, a\ b\!:\!(\iota\, x\!:\!A.\, B\ x). \\
&\qquad \mathrm{Id} \cdot A\ a.1\ b.1 \to \mathrm{Id} \cdot (\iota\, x\!:\!A.\, B\ x)\ a\ b\ - \\
&\qquad \Lambda\, A\ B\ a\ b.\, \lambda\, e. \\
&\qquad \varphi\ (\rho\ e.2.1\ @\ x\ \langle e\rangle.\ \{x \cong e\}\ \ -\ \ \beta\{\lambda\, x.\, x\})\ - \\
&\qquad (\rho\ e.1\ @\ x\ \langle b\rangle.\ \mathrm{Id} \cdot (\iota\, x\!:\!A.\, B\ x))\ x\ b\ \ -\ \ \mathrm{refl} \cdot (\iota\, x\!:\!A.\, B\ x)\ \text{-}b) \\
&\qquad \{e\}
\end{aligned}$$

$$\begin{aligned}
&\mathrm{subst} := \chi\ \forall\, A\!:\!\star.\, \forall\, a\ b\!:\!A.\, \forall\, P\!:\!(\Pi\, y\!:\!A.\, \mathrm{Id} \cdot A\ a\ y \to \star). \\
&\qquad \Pi\, e\!:\!\mathrm{Id} \cdot A\ a\ b.\, P\ a\ (\mathrm{refl} \cdot A\ \text{-}a) \to P\ b\ e\ - \\
&\qquad \Lambda\, A\ a\ b\ P.\, \lambda\, e. \\
&\qquad \rho\ e.2.1\ @\ x\ \langle e\rangle.\ P\ a\ x \to P\ b\ e\ - \\
&\qquad \rho\ e.1\ @\ x\ \langle b\rangle.\ P\ x\ e \to P\ b\ e\ - \\
&\qquad e.2.2 \cdot (P\ b\ e)
\end{aligned}$$

The erasure of each term is designed to match with the erasure of the associated construct in ç$_2$. While this might not be strictly necessary to obtain a model of ç$_2$ inside CDLE it makes the process easier. Moreover, carefully crafting terms with specific erasures is a trivial matter in CDLE because of the φ rule.

$$\begin{aligned}
|\mathrm{tt}| &= \lambda\, x\ y.\, x \\
|\mathrm{ff}| &= \lambda\, x\ y.\, y \\
|\mathrm{refl} \cdot A\ \text{-}a| &= \lambda\, x.\, x \\
|\mathrm{delta}\ e| &= |e| \\
|\mathrm{theta} \cdot A \cdot B\ \text{-}a\ \text{-}b\ e| &= |e| \\
|\mathrm{subst} \cdot A\ \text{-}a\ \text{-}b \cdot P\ e| &= |e|
\end{aligned}$$

Finally, each of these terms is shown to infer the desired type. Note that for syntax that is type-like, such as Id and Bool, there is no type-checking rule, only an inference judgment. Moreover, the χ rule only works with term-like syntax. Thus, for these definitions more care is needed to infer the correct kind, but because the definitions are simple there is no real difficulty.

Lemma 4.1.

1. $\vdash_{\varsigma_1} \mathrm{Bool} \rhd \star$

2. $\vdash_{\varsigma_1} \mathrm{tt} \rhd \mathrm{Bool}$

3. $\vdash_{\varsigma_1} \mathrm{ff} \rhd \mathrm{Bool}$

4. $\vdash_{\varsigma_1} \mathrm{Id} \rhd \Pi\, A{:}\star.\, A \to A \to \star$

5. $\vdash_{\varsigma_1} \mathrm{refl} \rhd \forall\, A{:}\star.\, \forall\, a{:}A.\, \mathrm{Id} \cdot A\ a\ a$

6. $\vdash_{\varsigma_1} \mathrm{delta} \rhd \mathrm{Id} \cdot \mathrm{Bool}\ \mathrm{tt}\ \mathrm{ff} \to \forall\, X{:}\star.\, X$

7. $\vdash_{\varsigma_1} \mathrm{theta} \rhd \begin{array}{l} \forall\, A{:}\star.\, \forall\, B{:}A \to \star.\, \forall\, a\ b{:}(\iota\, x{:}A.\, B\ x). \\ \mathrm{Id} \cdot A\ a.1\ b.1 \to \mathrm{Id} \cdot (\iota\, x{:}A.\, B\ x)\ a\ b \end{array}$

8. $\vdash_{\varsigma_1} \mathrm{subst} \rhd \begin{array}{l} \forall\, A{:}\star.\, \forall\, a\ b{:}A.\, \forall\, P{:}(\Pi\, y{:}A.\, \mathrm{Id} \cdot A\ a\ y \to \star). \\ \Pi\, e{:}\mathrm{Id} \cdot A\ a\ b.\, P\ a\ (\mathrm{refl} \cdot A\ \text{-}a) \to P\ b\ e \end{array}$

Proof. Straightforward by applying a short sequence of ς_1 rules in each case. These inferences are trivially formalized in the Cedille tool. □

A small collection of additional lemmas about CDLE is needed to prove soundness of the model and presented next. These lemmas are standard: weakening, symmetry of conversion, and transitivity of conversion. The only real difficulty is the bidirectional presentation which requires stating the desired lemma for each variation of judgment and using mutual recursion in the proof.

Lemma 4.2. *Suppose* $\Gamma \vdash_{\varsigma_1} T \rhd K$ *and* x *fresh*

1. If t is a kind and $\Gamma, \Delta \vdash_{\varsigma_1} t$ then $\Gamma, x : T, \Delta \vdash_{\varsigma_1} t$

2. If t is a type and $\Gamma, \Delta \vdash_{\varsigma_1} t \rhd K$ then $\Gamma, x : T, \Delta \vdash_{\varsigma_1} t \rhd K$

3. If t is a term and $\Gamma, \Delta \vdash_{\varsigma_1} t \rhd A$ then $\Gamma, x : T, \Delta \vdash_{\varsigma_1} t \rhd A$

4. If t is a term and $\Gamma, \Delta \vdash_{\varsigma_1} t \lhd A$ then $\Gamma, x : T, \Delta \vdash_{\varsigma_1} t \lhd A$

Proof. Straightforward by mutual recursion on the associated judgments. □

Lemma 4.3.

1. If a, b are terms and $|a| \rightleftharpoons_\eta |b|$ then $|b| \rightleftharpoons_\eta |a|$

2. If A, B are types and values and $A \cong^t B$ then $B \cong^t A$

3. If A, B are types and $A \cong B$ then $B \cong A$

4. If A, B are kinds and $A \cong B$ then $B \cong A$

Proof. Note that *1.* holds because $|a|$ and $|b|$ are untyped λ-calculus terms. For *2.* through *4.* mutual recursion and pattern match on A is sufficient. □

Lemma 4.4.

1. If a, b, c are terms, $|a| \rightleftharpoons_\eta |b|$, and $|b| \rightleftharpoons_\eta |c|$ then $|a| \rightleftharpoons_\eta |c|$

2. If A, B, C are types and values, $A \cong^t B$, and $B \cong^t C$ then $A \cong^t C$

3. If A, B, C are types, $A \cong B$, and $B \cong C$ then $A \cong C$

4. If A, B, C are kinds, $A \cong B$, and $B \cong C$ then $A \cong C$

Proof. Note that *1.* holds because $|a|$ and $|b|$ are untyped λ-calculus terms and reduction is confluent. The remainder are proved by mutual recursion. Note that in *3.* the types A, B, and C are reduced using call-name to a weak-head normal form. In particular, this reduction strategy is deterministic, thus $B \rightsquigarrow_n^* B'$ for a unique B'. This combined with using *2.* is sufficient for the *3.* case. The other two cases follow by pattern matching on B, inversion on the respective conversions, and applying the IH. □

4.2 Counterexamples to Decidability of Type Checking in CDLE

It is well-known that Cedille does not enjoy decidability of type checking. However, it might not be clear exactly how this property fails. Below is a series of formalized examples in Cedille that will loop when attempting to check using the Cedille tool. Commentary to accompany the formalized Cedille code is also provided to highlight what causes the failure.

Obviously bad because of omega right in the type

```
bad : { (λ x. x x) (λ x. x x) ≃ λ x. x } = β.

omega : { λ x. x ≃ λ x. x } = β{(λ x. x x) (λ x. x x)}.
bad : { omega ≃ λ x. x } = β.
```

Erased delta is bad

```
False : ★ = ∀ X:★. X.
Unit : ★ = ∀ X:★. X ➔ X.
self : Unit ➔ Unit = λ u. u u.
Bool : ★ = ∀ X:★. X ➔ X ➔ X.
tt : Bool = Λ X. λ x. λ y. x.
ff : Bool = Λ X. λ x. λ y. y.
Id : Π A:★. A ➔ A ➔ ★ = λ A:★. λ x:A. λ y:A. {y ≃ x}.
omega : Id·Bool tt ff ➾ False
= Λ e. (δ - e)·((Unit ➔ Unit) ➔ (Unit ➔ Unit) ➔ False) self self.
bad : {omega ≃ λ x. x} = β.
```

Untyped rewrites

```
Id : Π A:★. Π B:★. A ➔ B ➔ ★ = λ A:★. λ B:★. λ x:A. λ y:B. {x ≃ y}.
Unit : ★ = ∀ X:★. X ➔ X.
unit : Unit = Λ X. λ x. x.
self : Unit ➔ Unit = λ u. u u.
False : ★ = ∀ X:★. X.
bad : ∀ P:False ➔ ★. Π f:False. P f
= Λ P. λ f. {e1 = f·(Id·False·(Unit ➔ Unit) f self)}
  - {e2 = f·(Id·False·False f (f·(False ➔ False) f))}
  - ρ e2 - ρ e1 - (f·(P f)).
```

Erased rho with *some* method of talking about equality of True and False

```
False : ★ = ∀ X:★. X.
Not : ★ ➔ ★ = λ A:★. A ➔ False.
True : ★ = Not·False.
self : True = λ f. f·(False ➔ False) f.
Bool : ★ = ∀ X:★. X ➔ X ➔ X.
tt : Bool = Λ X. λ x. λ y. x.
ff : Bool = Λ X. λ x. λ y. y.
Id : Π A:★. A ➔ A ➔ ★ = λ A:★. λ a:A. λ b:A. {b ≃ a}.
subst : ∀ A:★. ∀ a:A. ∀ b:A. ∀ P:A ➔ ★. P a ➔ Id·A a b ➾ P b
= Λ A. Λ a. Λ b. Λ P. λ p. Λ i. ρ i - p.

elim : ★ ➔ ★ ➔ Bool ➔ ★
= λ A:★. λ B:★. λ x:Bool. ι _:{x ≃ tt} ➾ A. {x ≃ ff} ➾ B.
in1 : ∀ A:★. ∀ B:★. A ➔ elim·A·B tt
= Λ A. Λ B. λ a. [Λ e. a, Λ e. {f:False = δ - e} - φ (f·{f ≃ a}) - (f·B) {a}].
```

```
cast : ∀ A:★. ∀ B:★. ∀ a:Bool. ∀ b:Bool. Id·Bool a b ⇒ elim·A·B a ➔ elim·A·B b
= Λ A. Λ B. Λ a. Λ b. Λ e. λ p. subst·Bool -a -b ·(elim·A·B) p -e.
omega : Not·(∀ a:Bool. ∀ b:Bool. Id·Bool a b)
= λ x. (cast·True·False -tt -ff -(x -tt -ff) (in1 self)).2 -β.
Omega : Not·(∀ a:Bool. ∀ b:Bool. Id·Bool a b)
= λ x. self (omega x).
bad : {Omega ≃ λ x. x} = β.
```

By using phi.,

```
False : ★ = ∀ X:★. X.
Unit : ★ = ∀ X:★. X ➔ X.
self : Unit ➔ Unit = λ u. u u.
b : False ➔ ι _:Unit ➔ Unit. Unit = λ f. [f·(Unit ➔ Unit), f·Unit].
e : Π f:False. {b f ≃ self} = λ f. f·{b f ≃ self}.
omega : False ⇒ Unit = Λ f. self (φ (e f) - (b f).2 {self}).
bad : {omega ≃ λ x. x} = β.
```

1. Equality being untyped enables rewriting equations for *any* predicate. In this example the equality `e1` should be an equality between `Id → Id` typed proofs, but it is used with a predicate over `False` proofs.

4.3 Model

Figure 4.7 describes the model of $ç_2$ in CDLE. Note that this model is straightforward: abstractions to abstractions, applications to applications, pairs to pairs, etc. The complicated part is the equality type and its constructs, however all the necessary work to find suitable terms for these constructs was already completed above. There is one hiccup involving the promotion (ϑ) rule. In order to have a fully applied theta it must be the case that the annotation for ϑ is an intersection type. For proofs this will always be the case, but for arbitrary syntax it is not necessarily true. To work around this a catch-all case is also provided where the model only interprets the equality proof e. This choice is largely arbitrary, but it is picked to make sure that one critical property is preserved: erasure.

Lemma 4.5. *If* t term *then* $[\![|t|]\!] = |[\![t]\!]|$

Proof. By induction on t and inversion on t term. The case of first projection and first equality promotion cases are omitted.

Case: $t = x_\star$

Have $[\![|x_\star|]\!] = [\![x_\star]\!] = x$ and $|[\![x_\star]\!]| = |x| = x$, hence trivial.

Case: $t = \lambda_0\, x : A.\, b$

Have $[\![|\lambda_0\, x : A.\, b|]\!] = [\![|b|]\!]$ and $|[\![\lambda_0\, x : A.\, b]\!]| = |\Lambda\, x.\, [\![b]\!]| = |[\![b]\!]|$. Note that b term, hence by the IH $[\![|b|]\!] = |[\![b]\!]|$.

$$
\begin{aligned}
\llbracket (x : A) \to_{\tau} B \rrbracket &= \Pi\, x \!:\! \llbracket A \rrbracket .\, \llbracket B \rrbracket \\
\llbracket (x : A) \to_{\omega} B \rrbracket &= \Pi\, x \!:\! \llbracket A \rrbracket .\, \llbracket B \rrbracket \\
\llbracket (x : A) \to_{0} B \rrbracket &= \forall\, x \!:\! \llbracket A \rrbracket .\, \llbracket B \rrbracket \\
\llbracket \lambda_{\tau}\, x \!:\! A.\, t \rrbracket &= \lambda\, x \!:\! \llbracket A \rrbracket .\, \llbracket t \rrbracket \\
\llbracket \lambda_{\omega}\, x \!:\! A.\, t \rrbracket &= \lambda\, x.\, \llbracket t \rrbracket \\
\llbracket \lambda_{0}\, x \!:\! A.\, t \rrbracket &= \Lambda\, x.\, \llbracket t \rrbracket
\end{aligned}
$$

$$
\begin{aligned}
\llbracket \star \rrbracket &= \star \\
\llbracket x_K \rrbracket &= x \\
\llbracket f \bullet_{\tau} a \rrbracket &= \llbracket f \rrbracket\ \llbracket a \rrbracket && \text{if } a \text{ term} \\
\llbracket f \bullet_{\tau} a \rrbracket &= \llbracket f \rrbracket \cdot \llbracket a \rrbracket && \text{if } a \text{ type} \\
\llbracket f \bullet_{\omega} a \rrbracket &= \llbracket f \rrbracket\ \llbracket a \rrbracket \\
\llbracket f \bullet_{0} a \rrbracket &= \llbracket f \rrbracket\ \text{-}\llbracket a \rrbracket && \text{if } a \text{ term} \\
\llbracket f \bullet_{0} a \rrbracket &= \llbracket f \rrbracket \cdot \llbracket a \rrbracket && \text{if } a \text{ type}
\end{aligned}
$$

$$
\begin{aligned}
\llbracket (x : A) \cap B \rrbracket &= \iota\, x \!:\! \llbracket A \rrbracket .\, \llbracket B \rrbracket \\
\llbracket [t_1, t_2, A] \rrbracket &= [\llbracket t_1 \rrbracket, \llbracket t_2 \rrbracket]
\end{aligned}
\qquad\qquad
\begin{aligned}
\llbracket t.1 \rrbracket &= \llbracket t \rrbracket .1 \\
\llbracket t.2 \rrbracket &= \llbracket t \rrbracket .2
\end{aligned}
$$

$$
\begin{aligned}
\llbracket a =_A b \rrbracket &= \mathrm{Id} \cdot \llbracket A \rrbracket\ \llbracket a \rrbracket\ \llbracket b \rrbracket \\
\llbracket \mathrm{refl}(t; A) \rrbracket &= \mathrm{refl} \cdot \llbracket A \rrbracket\ \text{-}\llbracket t \rrbracket \\
\llbracket \vartheta(e, a, b; (x : A) \cap B) \rrbracket &= \mathrm{theta} \cdot \llbracket A \rrbracket \cdot \llbracket B \rrbracket\ \text{-}\llbracket a \rrbracket\ \text{-}\llbracket b \rrbracket\ \llbracket e \rrbracket \\
\llbracket \vartheta(e, a, b; T) \rrbracket &= \llbracket e \rrbracket \\
\llbracket \psi(e, a, b; A, P) \rrbracket &= \mathrm{subst} \cdot \llbracket A \rrbracket\ \text{-}\llbracket a \rrbracket\ \text{-}\llbracket b \rrbracket \cdot \llbracket P \rrbracket\ \llbracket e \rrbracket \\
\llbracket \varphi(a, b, e) \rrbracket &= \varphi\ \text{ç}\ \llbracket e \rrbracket .1 - \llbracket b \rrbracket\ \{\llbracket a \rrbracket\} \\
\llbracket \delta(e) \rrbracket &= \mathrm{delta}\ \llbracket e \rrbracket
\end{aligned}
$$

$$
\begin{aligned}
\llbracket \varepsilon \rrbracket &= \varepsilon \\
\llbracket \Gamma, x : A \rrbracket &= \llbracket \Gamma \rrbracket, x : \llbracket A \rrbracket
\end{aligned}
$$

Figure 4.7: Model definition interpreting ç$_2$ in CDLE.

Case: $t = \lambda_{\omega}\, x \!:\! A.\, b$

Have $\llbracket |\lambda_{\omega}\, x : A.\, b| \rrbracket = \lambda\, x.\, \llbracket |b| \rrbracket$ and $|\llbracket \lambda_{\omega}\, x : A.\, b \rrbracket| = |\lambda\, x.\, \llbracket b \rrbracket| = \lambda\, x.\, |\llbracket b \rrbracket|$. Note that b term, hence by the IH $\llbracket |b| \rrbracket = |\llbracket b \rrbracket|$.

Case: $t = f \bullet_0 a$

Have $\llbracket |f \bullet_0 a| \rrbracket = \llbracket |f| \rrbracket$ and $|\llbracket f \bullet_0 a \rrbracket| = |\llbracket f \rrbracket\ \text{-}\llbracket a \rrbracket| = |\llbracket f \rrbracket|$. Given $f \bullet_0 a$ term it is always the case that f term. Thus, by the IH $\llbracket |f| \rrbracket = |\llbracket f \rrbracket|$.

Case: $t = f \bullet_{\omega} a$

Have $\llbracket |f \bullet_{\omega} a| \rrbracket = \llbracket |f| \rrbracket\ \llbracket |a| \rrbracket$ and $|\llbracket f \bullet_{\omega} a \rrbracket| = |\llbracket f \rrbracket|\ |\llbracket a \rrbracket|$. Note that f, a term because the mode is ω there is no possibility of a type. Hence, by the IH $\llbracket |f| \rrbracket = |\llbracket f \rrbracket|$ and $\llbracket |a| \rrbracket = |\llbracket a \rrbracket|$.

Case: $t = [t_1, t_2; A]$

Have $\llbracket |[t_1, t_2; A]| \rrbracket = \llbracket |t_1| \rrbracket$ and $|\llbracket [t_1, t_2; A] \rrbracket| = |[\llbracket t_1 \rrbracket, \llbracket t_2 \rrbracket]| = |\llbracket t_1 \rrbracket|$. By the IH applied to t_1 term: $\llbracket |t_1| \rrbracket = |\llbracket t_1 \rrbracket|$.

Case: $t = t.2$

Have $\llbracket |t.2| \rrbracket = \llbracket |t| \rrbracket$ and $|\llbracket t.2 \rrbracket| = |\llbracket t \rrbracket.2| = |\llbracket t \rrbracket|$. By the IH applied to t term: $\llbracket |t| \rrbracket = |\llbracket t \rrbracket|$.

Case: $t = \mathrm{refl}(a; A)$

Have $\llbracket |\mathrm{refl}(a; A)| \rrbracket = \llbracket \lambda\, x : \diamond.\, x_\star \rrbracket = \lambda\, x.\, x$ and $|\llbracket \mathrm{refl}(a; A) \rrbracket| = |\mathrm{refl} \cdot \llbracket A \rrbracket \text{ -}\llbracket a \rrbracket| = \lambda\, x.\, x$.

Case: $t = \vartheta(e, a, b; T)$

Have $\llbracket |\vartheta(e, a, b; T)| \rrbracket = \llbracket |e| \rrbracket$. Suppose $T = (x : A) \cap B$ then $|\llbracket \vartheta(e, a, b; (x : A) \cap B) \rrbracket| = |\mathrm{theta} \cdot \llbracket A \rrbracket \cdot \llbracket B \rrbracket \text{ -}\llbracket a \rrbracket \text{ -}\llbracket b \rrbracket\ \llbracket e \rrbracket| = |\llbracket e \rrbracket|$. Otherwise, $|\llbracket \vartheta(e, a, b; T) \rrbracket| = |\llbracket e \rrbracket|$. By the IH applied to e term: $\llbracket |e| \rrbracket = |\llbracket e \rrbracket|$.

Case: $t = \psi(e, a, b; A, P)$

Have $\llbracket |\psi(e, a, b; A, P)| \rrbracket = \llbracket |e| \rrbracket$ and $|\llbracket \psi(e, a, b; A, P) \rrbracket| = |\mathrm{subst} \cdot \llbracket A \rrbracket \text{ -}\llbracket a \rrbracket \text{ -}\llbracket b \rrbracket \cdot \llbracket P \rrbracket\ \llbracket e \rrbracket| = |\llbracket e \rrbracket|$. By the IH applied to e term: $\llbracket |e| \rrbracket = |\llbracket e \rrbracket|$.

Case: $t = \varphi(a, b, e)$

Have $\llbracket |\varphi(a, b, e)| \rrbracket = \llbracket |a| \rrbracket$ and $|\llbracket \varphi(a, b, e) \rrbracket| = |\varphi\ \text{ç}\ \llbracket e \rrbracket.1 - \llbracket b \rrbracket\ \{\llbracket a \rrbracket\}| = |\llbracket a \rrbracket|$. By the IH applied to a term: $\llbracket |a| \rrbracket = |\llbracket a \rrbracket|$.

Case: $t = \delta(e)$

Have $\llbracket |\delta(e)| \rrbracket = \llbracket |e| \rrbracket$ and $|\llbracket \delta(e) \rrbracket| = |\mathrm{delta}\ \llbracket e \rrbracket| = |\llbracket e \rrbracket|$. By the IH applied to e term: $\llbracket |e| \rrbracket = |\llbracket e \rrbracket|$.

□

To obtain soundness we first need to know that conversion is preserved for the terms, types, and kinds. Luckily, because ç_2 terms are closely matched with CDLE terms lemmas involving reduction can be precise.

Lemma 4.6. $\llbracket [x := b]t \rrbracket = [x := \llbracket b \rrbracket] \llbracket t \rrbracket$

Proof. Straightforward by induction on t, substitution is structural with the only exception being variables, but $\llbracket x_K \rrbracket = x$. □

Lemma 4.7. *If* t term *and* $|t| \rightsquigarrow t'$ *then* $|\llbracket t \rrbracket| \rightsquigarrow \llbracket t' \rrbracket$

Proof. By induction on t and inversion on t term. The cases: erased lambda, pair, first projection, second projection, promotion (ϑ), substitution (ψ), and separation (δ) all erase to a subexpression that is a term. Hence, these cases are very similar to the erased application case and omitted. The erasure of the variable, reflexivity, and cast cases are values and thus do not reduce.

Case: $t = \lambda_\omega\, x : A.\, b$

> Have $|\lambda_\omega\, x : A.\, b| = \lambda_\omega\, x : \diamond.\, |b|$ which means $\lambda_\omega\, x : \diamond.\, |b| \rightsquigarrow \lambda_\omega\, x : \diamond.\, b'$. Now b term and $|b| \rightsquigarrow b'$, applying the IH gives $|\llbracket b \rrbracket| \rightsquigarrow \llbracket b' \rrbracket$. Note that $|\llbracket \lambda_\omega\, x : A.\, b \rrbracket| = \lambda\, x.\, |\llbracket b \rrbracket| \rightsquigarrow \lambda\, x.\, |\llbracket b' \rrbracket|$. By Lemma 4.6: $|\llbracket b' \rrbracket| = \llbracket |b'| \rrbracket$. However, b' is the result of a contracted redex in an already erased term, hence $|b'| = b'$. Thus, $|\llbracket \lambda_\omega\, x : A.\, b \rrbracket| \rightsquigarrow \llbracket \lambda_\omega\, x : \diamond.\, b' \rrbracket$.

Case: $t = f \bullet_0 a$

> Have $|f \bullet_0 a| = |f|$, thus $|f| \rightsquigarrow t'$. Applying the IH gives $|\llbracket f \rrbracket| \rightsquigarrow \llbracket t' \rrbracket$. Note that $|\llbracket f \bullet_0 a \rrbracket| = |\llbracket f \rrbracket \text{ -}\llbracket a \rrbracket| = |\llbracket f \rrbracket|$. Thus, $|\llbracket f \bullet_0 a \rrbracket| \rightsquigarrow \llbracket t' \rrbracket$.

Case: $t = f \bullet_\omega a$

> Have $|f \bullet_\omega a| = |f| \bullet_\omega |a|$. Suppose $|f| = \lambda_\omega\, x : \diamond.\, b$ and $|f| \bullet_\omega |a| \rightsquigarrow [x := |a|]b$. Now $|\llbracket f \bullet_\omega a \rrbracket| = |\llbracket f \rrbracket|\ |\llbracket a \rrbracket|$. By Lemma 4.5: $|\llbracket f \rrbracket| = \llbracket |f| \rrbracket = \lambda\, x.\, \llbracket b \rrbracket$. Thus, $(\lambda\, x.\, \llbracket b \rrbracket)\ |\llbracket a \rrbracket| \rightsquigarrow [x := |\llbracket a \rrbracket|]\llbracket b \rrbracket$. Using Lemma 4.5 and Lemma 4.6 gives $[x := |\llbracket a \rrbracket|]\llbracket b \rrbracket = \llbracket [x := |a|]b \rrbracket$.
>
> Suppose wlog that $|f| \rightsquigarrow f'$ (the case of $|a| \rightsquigarrow a'$ is very similar). Note that f term, applying the IH gives $|\llbracket f \rrbracket| \rightsquigarrow \llbracket f' \rrbracket$. Now $|\llbracket f \bullet_\omega a \rrbracket| = |\llbracket f \rrbracket|\ |\llbracket a \rrbracket| \rightsquigarrow \llbracket f' \rrbracket\ |\llbracket a \rrbracket| = \llbracket f' \bullet_\omega |a| \rrbracket$. The final equality uses Lemma 4.5.

□

Lemma 4.8. *If t term and $|t| \rightsquigarrow^* t'$ then $|\llbracket t \rrbracket| \rightsquigarrow^* \llbracket t' \rrbracket$*

Proof. By induction on $|t| \rightsquigarrow^* t'$ using Lemma 4.7, Lemma 2.53, and Lemma 2.50. □

Lemma 4.9. *If a, b term and $|a| \rightleftharpoons |b|$ then $|\llbracket a \rrbracket| \rightleftharpoons |\llbracket b \rrbracket|$*

Proof. Deconstructing $|a| \rightleftharpoons |b|$ gives $|a| \rightsquigarrow^* z$ and $|b| \rightsquigarrow^* z$. Applying Lemma 4.8 gives $|\llbracket a \rrbracket| \rightsquigarrow^* \llbracket z \rrbracket$ and $|\llbracket b \rrbracket| \rightsquigarrow^* \llbracket z \rrbracket$. Thus, $|\llbracket a \rrbracket| \rightleftharpoons |\llbracket b \rrbracket|$. □

Lemma 4.10. *If s type and $s \rightsquigarrow_n t$ then $\llbracket s \rrbracket \rightsquigarrow_n \llbracket t \rrbracket$*

Proof. By induction on s and inversion on s type. Note that only the case where s is a redex is important as all other cases are in weak-head normal form. Thus, suppose $s = f \bullet_\tau a$, $f = \lambda_\tau\, x : A.\, b$, and $f \bullet_\tau a \rightsquigarrow_n [x := a]b$. Suppose wlog that a term. Now $\llbracket f \bullet_\tau a \rrbracket = \llbracket f \rrbracket\ \llbracket a \rrbracket = (\lambda\, x : \llbracket A \rrbracket.\, \llbracket b \rrbracket)\ \llbracket a \rrbracket \rightsquigarrow [x := \llbracket a \rrbracket]\llbracket b \rrbracket$. Using Lemma 4.6 gives $[x := \llbracket a \rrbracket]\llbracket b \rrbracket = \llbracket [x := a]b \rrbracket$. □

Lemma 4.11. *If s type and $s \rightsquigarrow_n^* t$ then $\llbracket s \rrbracket \rightsquigarrow_n^* \llbracket t \rrbracket$*

Proof. By induction on $s \rightsquigarrow_n^* t$ using Lemma 4.10 and Lemma 2.53. □

Lemma 4.12.

1. *If A, B* type*, A B are values, and $A \equiv B$ then $\llbracket A \rrbracket \cong^t \llbracket B \rrbracket$*

2. *If A, B* type *and $A \equiv B$ then $\llbracket A \rrbracket \cong \llbracket B \rrbracket$*

3. *If A, B* kind *and $A \equiv B$ then $\llbracket A \rrbracket \cong \llbracket B \rrbracket$*

Proof. By mutual recursion.
1. By induction on A and inversion on A being a value and $A \equiv B$ (hence B must match A). Conversion in ς_1 is structural over weak-head normal forms and in this case A and B must be weak-head normal. Thus, a combination of *1.*, *2.*, *3.*, and Lemma 4.9 on subexpressions in each case is sufficient.
2. By Theorem 3.19, $\exists$ A', B' such that $A \rightsquigarrow^* A'$, $B \rightsquigarrow^* B'$ and A', B' are values. Lemma 2.53 gives that A', B' type. Lemma 2.31 gives that $A' \equiv B'$. Thus, applying *1.* concludes.
3. By induction on A and inversion on $A \equiv B$. Again, conversion of kinds is structural in ς_1. Thus, a combination of *2.* and *3.* on subexpressions in each case is sufficient. □

Theorem 4.13 (Soundness of $\llbracket - \rrbracket$)**.** *Suppose $\Gamma \vdash_{\varsigma_2} t : A$*

1. *if $A = \square$ then $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket t \rrbracket$*

2. *if $\Gamma \vdash_{\varsigma_2} A : \square$ then $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket t \rrbracket \rhd T$ and $T \cong \llbracket A \rrbracket$*

3. *if $\Gamma \vdash_{\varsigma_2} A : \star$ then $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket t \rrbracket \lhd \llbracket A \rrbracket$*

Proof. By induction on $\Gamma \vdash_{\varsigma_2} t : A$. Note that each case is mutually exclusive by classification.

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

Have $A = \square$ and $\Gamma \vdash_{\varsigma_1} \star$, hence trivial.

Case: $$\frac{x \notin \overset{\mathcal{D}_1}{FV}(\Gamma_1; \Gamma_2) \qquad \Gamma_1 \vdash \overset{\mathcal{D}_2}{A} : K}{\Gamma_1; x_m : A; \Gamma_2 \vdash x_K : A}$$

Let $\Gamma = \Gamma_1; x : A; \Gamma_2$. Have $(x : \llbracket A \rrbracket) \in \llbracket \Gamma \rrbracket$. Now $\llbracket \Gamma_1 \rrbracket \vdash_{\varsigma_1} x \rhd \llbracket A \rrbracket$ by the IH and $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} x \rhd \llbracket A \rrbracket$ by Lemma 4.2. Suppose $K = \square$ then $\llbracket A \rrbracket \cong \llbracket A \rrbracket$ and $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} x \rhd \llbracket A \rrbracket$. Suppose $K = \star$ then $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} x \lhd \llbracket A \rrbracket$.

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

Suppose $m = \tau$, then $\mathrm{dom}_\Pi(m, K) = K$ and $\mathrm{codom}_\Pi(m) = \square$. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket A \rrbracket$ if $K = \square$

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket A \rrbracket \rhd \star$ if $K = \star$

$\mathcal{D}_2$. $[\![\Gamma]\!], x : [\![A]\!] \vdash_{\varsigma_1} [\![B]\!]$

The corresponding Π rule for the two possibilities of K concludes the case.

Suppose $m = 0$, then $\text{dom}_\Pi(m, K) = K$ and $\text{codom}_\Pi(m) = \star$. Applying the IH gives:

$\mathcal{D}_1$. $[\![\Gamma]\!] \vdash_{\varsigma_1} [\![A]\!]$ if $K = \square$

$\mathcal{D}_1$. $[\![\Gamma]\!] \vdash_{\varsigma_1} [\![A]\!] \rhd \star$ if $K = \star$

$\mathcal{D}_2$. $[\![\Gamma]\!], x : [\![A]\!] \vdash_{\varsigma_1} [\![B]\!] \rhd \star$

The corresponding $\forall$ rule for the two possibilities of K concludes the case.

Suppose $m = \omega$, then $\text{dom}_\Pi(m, K) = \star$ and $\text{codom}_\Pi(m) = \star$. Applying the IH gives:

$\mathcal{D}_1$. $[\![\Gamma]\!] \vdash_{\varsigma_1} [\![A]\!] \rhd \star$

$\mathcal{D}_2$. $[\![\Gamma]\!], x : [\![A]\!] \vdash_{\varsigma_1} [\![B]\!] \rhd \star$

The corresponding Π rule concludes the case.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \to_m B : \text{codom}_\Pi(m)} \qquad \overset{\mathcal{D}_2}{\Gamma; x_m : A \vdash t : B} \qquad \overset{\mathcal{D}_3}{x \notin FV(|t|) \text{ if } m = 0}}{\Gamma \vdash \lambda_m\, x : A.\, t : (x : A) \to_m B}$$

Suppose $m = \tau$, then $\text{codom}_\Pi(m) = \square$. Note that this means that t type. Applying the IH gives:

$\mathcal{D}_1$. $[\![\Gamma]\!] \vdash_{\varsigma_1} \Pi\, x : [\![A]\!].\, [\![B]\!]$

$\mathcal{D}_2$. $[\![\Gamma]\!], x : [\![A]\!] \vdash_{\varsigma_1} [\![t]\!] \rhd T$ and $T \cong [\![B]\!]$

Suppose $[\![\Gamma]\!] \vdash_{\varsigma_1} [\![A]\!]$, then $[\![\Gamma]\!] \vdash_{\varsigma_1} \lambda\, x : [\![A]\!].\, [\![t]\!] \rhd \Pi\, x : [\![A]\!].\, T$. By rules of conversion for kinds yields $\Pi\, x : [\![A]\!].\, T \cong \Pi\, x : [\![A]\!].\, [\![B]\!]$. The case where $[\![A]\!]$ is a type instead of a kind is similar.

Suppose $m = 0$, then $\text{codom}_\Pi(m) = \star$. Note that this means t term. Applying the IH gives:

$\mathcal{D}_1$. $[\![\Gamma]\!] \vdash_{\varsigma_1} \Pi\, x : [\![A]\!].\, [\![B]\!] \rhd \star$

$\mathcal{D}_2$. $[\![\Gamma]\!], x : [\![A]\!] \vdash_{\varsigma_1} [\![t]\!] \lhd [\![B]\!]$

Note that $FV(|[\![t]\!]|) \subseteq FV(|t|)$, thus $x \notin FV(|[\![t]\!]|)$. Using the corresponding Λ rule based on the classification of $[\![A]\!]$ concludes the case.

Suppose $m = \omega$, then $\text{codom}_\Pi(m) = \star$. This case is omitted because the previous case is a more general version of it.

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

Suppose $m = \tau$. Classification forces f type, but a is either a term or a type. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket \rhd T$ with $T \cong \Pi\, x\!:\!\llbracket A\rrbracket.\, \llbracket B\rrbracket$

$\mathcal{D}_2$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket a\rrbracket \rhd T_2$ with $T_2 \cong \llbracket A\rrbracket$ if a type

$\mathcal{D}_2$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket a\rrbracket \lhd \llbracket A\rrbracket$ if a term

Note that because kinds cannot reduce, it must be the case that $\exists\ C, D$ such that $T = \Pi\, x : C.\, D$. Moreover, $C \cong \llbracket A\rrbracket$ and $D \cong \llbracket B\rrbracket$ by the conversion rules. Suppose a type then using the associated rule yields $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket \cdot \llbracket a\rrbracket \rhd [x := \llbracket a\rrbracket]D$. Now, $[x := \llbracket a\rrbracket]D \cong [x := \llbracket a\rrbracket]\llbracket B\rrbracket$ and the case is concluded. Suppose a term then using the associated rule yields $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket\ \llbracket a\rrbracket \rhd [x := \chi\ C - \llbracket a\rrbracket]D$. Again, $[x := \chi\ C - \llbracket a\rrbracket]D \cong [x := \chi\ C - \llbracket a\rrbracket]\llbracket B\rrbracket$ and the case is concluded. Note that $[x := \chi\ C - \llbracket a\rrbracket]\llbracket B\rrbracket \cong [x := \llbracket a\rrbracket]\llbracket B\rrbracket$ because the χ is only well-typed in term positions, where it is promptly erased during conversion checking.

Suppose $m = 0$. Classification forces f term, but a is either a term or a type. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket \lhd \forall\, x\!:\!\llbracket A\rrbracket.\, \llbracket B\rrbracket$

$\mathcal{D}_2$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket a\rrbracket \rhd T_2$ with $T_2 \cong \llbracket A\rrbracket$ if a type

$\mathcal{D}_2$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket a\rrbracket \lhd \llbracket A\rrbracket$ if a term

Deconstructing the checking judgment for $\llbracket f\rrbracket$ yields $\exists\ C, D$ such that $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket \blacktriangleright \forall\, x\!:\!C.\, D$ and $C \cong \llbracket A\rrbracket$ and $D \cong \llbracket B\rrbracket$. Suppose a type then the associated judgment gives $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket \cdot \llbracket a\rrbracket \rhd [x := \llbracket a\rrbracket]D$. Now, $[x := \llbracket a\rrbracket]D \cong [x := \llbracket a\rrbracket]\llbracket B\rrbracket$ and the case is concluded. Suppose a term then the associated judgment gives $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket\ \text{-}\llbracket a\rrbracket \rhd [x := \chi\, C - \llbracket a\rrbracket]D$. Again, $[x := \chi\, C - \llbracket a\rrbracket]D \cong [x := \chi\, C - \llbracket a\rrbracket]\llbracket B\rrbracket$ and the case is concluded.

Suppose $m = \omega$ Classification forces f, a term. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket \lhd \Pi\, x\!:\!\llbracket A\rrbracket.\, \llbracket B\rrbracket$

$\mathcal{D}_2$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket a\rrbracket \lhd \llbracket A\rrbracket$ if a term

As with the previous case, $\exists\ C, D$ such that $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket \blacktriangleright \Pi\, x\!:\!C.\, D$ and $C \cong \llbracket A\rrbracket$ and $D \cong \llbracket B\rrbracket$. Applying the associated rule yields $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket f\rrbracket\ \llbracket a\rrbracket \rhd [x := \chi\ C - \llbracket a\rrbracket]D$. Now, $[x := \chi\ C - \llbracket a\rrbracket]D \cong [x := \chi\ C - \llbracket a\rrbracket]\llbracket B\rrbracket$ and the case is concluded.

Case: $$\dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma; x_\tau : A \vdash B : \star}}{\Gamma \vdash (x : A) \cap B : \star}$$

Applying the IH gives:

$\mathcal{D}_1$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket A\rrbracket \rhd \star$

$\mathcal{D}_2$. $\llbracket\Gamma\rrbracket, x : \llbracket A\rrbracket \vdash_{\varsigma_1} \llbracket B\rrbracket \rhd \star$

Thus, $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \iota\, x{:}\llbracket A\rrbracket.\, \llbracket B\rrbracket \rhd \star$ as required.

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

Note by classification and $\mathcal{D}_1$: $\Gamma \vdash A : \star$ and $\Gamma, x : A \vdash B : \star$. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \iota\, x{:}\llbracket A\rrbracket.\, \llbracket B\rrbracket \rhd \star$

$\mathcal{D}_2$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket \lhd \llbracket A\rrbracket$

$\mathcal{D}_3$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket s\rrbracket \lhd [x := \llbracket t\rrbracket]\llbracket B\rrbracket$

Note that $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket \lhd \llbracket A\rrbracket$ so clearly $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket s\rrbracket \lhd [x := \chi\ \llbracket A\rrbracket\ -\ \llbracket t\rrbracket]\llbracket B\rrbracket$ as the χ merely adds extra typing information. Lemma 4.9 applied to $\mathcal{D}_4$ and using the fact that t, s term gives $|\llbracket t\rrbracket| \rightleftharpoons |\llbracket s\rrbracket|$. Combining this information yields $\llbracket\Gamma\rrbracket \vdash \llbracket[t, s; A]\rrbracket \lhd \llbracket(x : A) \cap B\rrbracket$.

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

By classification t term. Applying the IH to $\mathcal{D}_1$ gives $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket \lhd \iota\, x : \llbracket A\rrbracket.\, \llbracket B\rrbracket$. Deconstruct this checking rule and notice that either the inferred type is already an intersection or it must reduce to an intersection. Thus, $\exists\ C\ D$ such that $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket \blacktriangleright \iota\, x : \llbracket C\rrbracket.\, \llbracket D\rrbracket$ and $\iota\, x : \llbracket C\rrbracket.\, \llbracket D\rrbracket \cong \iota\, x : \llbracket A\rrbracket.\, \llbracket B\rrbracket$. Deconstructing the congruence yields $\llbracket C\rrbracket \cong \llbracket A\rrbracket$. Thus, $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket.1 \lhd \llbracket A\rrbracket$

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

By classification t term. Applying the IH to $\mathcal{D}_1$ gives $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket \lhd \iota\, x : \llbracket A\rrbracket.\, \llbracket B\rrbracket$. Deconstruct this checking rule and notice that either the inferred type is already an intersection or it must reduce to an intersection. Thus, $\exists\ C\ D$ such that $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket \blacktriangleright \iota\, x : \llbracket C\rrbracket.\, \llbracket D\rrbracket$ and $\iota\, x : \llbracket C\rrbracket.\, \llbracket D\rrbracket \cong \iota\, x : \llbracket A\rrbracket.\, \llbracket B\rrbracket$. Deconstructing the congruence yields $\llbracket D\rrbracket \cong \llbracket B\rrbracket$ and thus $[x := \llbracket t\rrbracket.1]\llbracket D\rrbracket \cong [x := \llbracket t\rrbracket.1]\llbracket B\rrbracket$. Now $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket.2 \rhd [x := \llbracket t\rrbracket.1]\llbracket D\rrbracket$. Thus, $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket t\rrbracket.2 \lhd [x := \llbracket t\rrbracket.1]\llbracket B\rrbracket$.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : \star \qquad \Gamma \vdash \overset{\mathcal{D}_2}{a} : A \qquad \Gamma \vdash \overset{\mathcal{D}_2}{b} : A}{\Gamma \vdash a =_A b : \star}$$

Note that a, b term by $\mathcal{D}_1$. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket A \rrbracket \rhd \star$

$\mathcal{D}_2$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket a \rrbracket \lhd \llbracket A \rrbracket$

$\mathcal{D}_3$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket b \rrbracket \lhd \llbracket A \rrbracket$

By Lemma 4.1, Lemma 4.2, and the application rule for ς_1: $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \mathrm{Id} \cdot \llbracket A \rrbracket\ \llbracket a \rrbracket\ \llbracket b \rrbracket \rhd \star$.

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

Note that t term by $\mathcal{D}_1$. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket A \rrbracket \rhd \star$

$\mathcal{D}_2$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket t \rrbracket \lhd \llbracket A \rrbracket$

By Lemma 4.1, Lemma 4.2, and the application rule for ς_1: $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \mathrm{refl} \cdot \llbracket A \rrbracket$ -$\llbracket t \rrbracket \rhd \mathrm{Id} \cdot \llbracket A \rrbracket\ \llbracket t \rrbracket\ \llbracket t \rrbracket$.

Case: $$\dfrac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : \star \qquad \Gamma \vdash \overset{\mathcal{D}_2}{a} : A \qquad \Gamma \vdash \overset{\mathcal{D}_3}{b} : A \qquad \Gamma \vdash \overset{\mathcal{D}_4}{e : a} =_A b \qquad \Gamma \vdash P : (y : A) \overset{\mathcal{D}_5}{\to_\tau} (p : a =_A y_\star) \to_\tau \star}{\Gamma \vdash \psi(e, a, b; A, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

Note by $\mathcal{D}_1$ that a, b term and by classification e term with A, P type. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket A \rrbracket \rhd \star$

$\mathcal{D}_2$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket a \rrbracket \lhd \llbracket A \rrbracket$

$\mathcal{D}_3$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket b \rrbracket \lhd \llbracket A \rrbracket$

$\mathcal{D}_4$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket e \rrbracket \lhd \mathrm{Id} \cdot \llbracket A \rrbracket\ \llbracket a \rrbracket\ \llbracket b \rrbracket$

$\mathcal{D}_5$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket P \rrbracket \rhd T$ and $T \cong \forall\, y \colon \llbracket A \rrbracket.\, \mathrm{Id} \cdot \llbracket A \rrbracket\ \llbracket a \rrbracket\ \llbracket y \rrbracket \to \star$

By Lemma 4.1, Lemma 4.2, and the application rule for ς_1: $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \mathrm{subst} \cdot \llbracket A \rrbracket$ -$\llbracket a \rrbracket$ -$\llbracket b \rrbracket \cdot \llbracket P \rrbracket\ \llbracket e \rrbracket \rhd \llbracket P \rrbracket\ \llbracket a \rrbracket\ (\mathrm{refl} \cdot \llbracket A \rrbracket$ -$\llbracket a \rrbracket) \to \llbracket P \rrbracket\ \llbracket b \rrbracket\ \llbracket e \rrbracket$.

Case: $$\dfrac{\Gamma \vdash (x : \overset{\mathcal{D}_1}{A}) \cap B : \star \qquad \Gamma \vdash a : \overset{\mathcal{D}_2}{(x : A)} \cap B \qquad \Gamma \vdash b : \overset{\mathcal{D}_3}{(x : A)} \cap B \qquad \Gamma \vdash e : \overset{\mathcal{D}_4}{a.1} =_A b.1}{\Gamma \vdash \vartheta(e, a, b; (x : A) \cap B) : a =_{(x:A) \cap B} b}$$

Note by $\mathcal{D}_1$ that a, b term and by classification e term with $(x : A) \cap B$ type. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \iota\, x \colon \llbracket A \rrbracket.\, \llbracket B \rrbracket \rhd \star$

$\mathcal{D}_2$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket a \rrbracket \lhd \iota\, x \colon \llbracket A \rrbracket.\, \llbracket B \rrbracket$

$\mathcal{D}_3$. $\llbracket \Gamma \rrbracket \vdash_{\varsigma_1} \llbracket b \rrbracket \lhd \iota\, x \colon \llbracket A \rrbracket.\, \llbracket B \rrbracket$

$\mathcal{D}_4$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket e\rrbracket \lhd \mathrm{Id} \cdot \llbracket A\rrbracket\ \llbracket a\rrbracket.1\ \llbracket b\rrbracket.1$

Note that $\llbracket\Gamma\rrbracket, x : \llbracket A\rrbracket \vdash_{\varsigma_1} \llbracket B\rrbracket \rhd \star$ which means $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket B\rrbracket \rhd A \to \star$. By Lemma 4.1, Lemma 4.2, and the application rule for ς_1: $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1}$ theta $\cdot \llbracket A\rrbracket \cdot \llbracket B\rrbracket$ -$\llbracket a\rrbracket$ -$\llbracket b\rrbracket$ $\llbracket e\rrbracket \rhd \mathrm{Id} \cdot (\iota\, x{:}\llbracket A\rrbracket.\, \llbracket B\rrbracket)\ \llbracket a\rrbracket\ \llbracket b\rrbracket$.

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

Note by soundness of classification that a, b, e term. Applying the IH gives:

$\mathcal{D}_1$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket a\rrbracket \lhd \llbracket A\rrbracket$

$\mathcal{D}_2$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket b\rrbracket \lhd \iota\, x{:}\llbracket A\rrbracket.\, \llbracket B\rrbracket$

$\mathcal{D}_3$. $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \llbracket e\rrbracket \lhd \mathrm{Id} \cdot \llbracket A\rrbracket\ \llbracket a\rrbracket\ \llbracket b\rrbracket.1$

By the application and first projection rule and some maneuvering of type conversion: $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1} \varsigma\ \llbracket e\rrbracket.1 \lhd \{\llbracket b\rrbracket \cong \llbracket a\rrbracket\}$. Note that $FV(\llbracket a\rrbracket) \subseteq dom(\Gamma)$ because otherwise $\mathcal{D}_1$ is not a proof. Thus, the goal is obtained by the φ rule of ς_1.

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

Applying the IH to $\mathcal{D}_1$ yields $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1}$ IdBoolttff. By Lemma 4.1, Lemma 4.2, and the application rule for ς_1: $\llbracket\Gamma\rrbracket \vdash_{\varsigma_1}$ delta $\llbracket e\rrbracket \rhd \forall\, X{:}\star.\, X$.

Case: $$\frac{\Gamma \vdash \overset{\mathcal{D}_1}{A} : K \qquad \Gamma \vdash \overset{\mathcal{D}_2}{t} : B \qquad \overset{\mathcal{D}_3}{A \equiv B}}{\Gamma \vdash t : A}$$

Suppose $K = \Box$. Then by classification and $\mathcal{D}_3$: $\Gamma \vdash B : \Box$. Applying the IH to $\mathcal{D}_2$ gives $\llbracket\Gamma\rrbracket \vdash \llbracket t\rrbracket \rhd T$ with $T \cong \llbracket B\rrbracket$. By Lemma 4.12: $\llbracket A\rrbracket \cong \llbracket B\rrbracket$. Now by Lemma 4.4 and Lemma 4.3: $T \cong \llbracket B\rrbracket$.

Suppose $K = \star$. Then by classification and $\mathcal{D}_3$: $\Gamma \vdash B : \star$. Applying the IH to $\mathcal{D}_2$ gives $\llbracket\Gamma\rrbracket \vdash \llbracket t\rrbracket \rhd \llbracket B\rrbracket$. By Lemma 4.12 and Lemma 4.3: $\llbracket B\rrbracket \cong \llbracket A\rrbracket$. Applying the checking rule of ς_1 yields $\llbracket\Gamma\rrbracket \vdash \llbracket t\rrbracket \lhd \llbracket A\rrbracket$.

□

Theorem 4.14 (Logical Consistency)**.** $\neg(\vdash_{c_2} t : (X : \star) \to_0 X_\Box)$

Proof. Proceed using proof by negation. Suppose $\vdash_{c_2} t : (X : \star) \to_0 X_\Box$. By Theorem 4.13: $\vdash_{\varsigma_1} \llbracket t\rrbracket \lhd \forall\, X{:}\star.\, X$. However, this is impossible by consistency of ς_1. □

Corollary 4.15 (Equational Consistency)**.** $\neg(\vdash_{c_2} t : \mathrm{ctt} =_{\mathrm{cBool}} \mathrm{cff})$

Chapter 5

OBJECT NORMALIZATION AND φ THE FOIL

Consistency guarantees that the logic and equational theory of ς_2 is non-trivial. Proof normalization guarantees that, at least, inference for kinds and types is decidable. Neither of these properties are strong enough on their own to guarantee decidability of type checking. To obtain decidability of type checking it must be the case that objects are normalizing. Unfortunately, object normalization does not hold when the Cast rule is used, and it is not clear how the rule may be repaired to acquire object normalization. A proof is **strict** if it does not use the Cast rule in its derivation. While strict proofs do have normalizing objects the technique described below to prove this fact depends on both proof normalization and consistency. This is suggestive of how difficult a property object normalization is to show.

5.1 Normalization for Strict Proofs

The core observation is that proof reduction in strict proofs upper-bounds reduction in their corresponding objects. Thus, if a strict object steps, and note that this must be a β-step, then there is some strict proof such that the original strict proof reduces to it and the erasures match. There could be many more reductions in the strict proof because syntax forms for equality and intersections are all mostly erased. However, none of these forms will block a β-redex because the proof is well-typed. Note that this property hinges on both proof normalization and equational consistency. Proof normalization is used to eliminate any extraneous redexes that would otherwise be erased. Consistency is used to eliminate the δ case as it could theoretically generate a β-redex after erasure if the theory was not equationally consistent. Of course, φ could also generate a β-redex after erasure, but this is impossible because the syntax under consideration is strict.

Definition 5.1. *$\Gamma \vdash_{\bar{\varsigma}_2} t : A$ iff $\mathcal{D} : \Gamma \vdash t : A$ and the* Cast *rule is not used in $\mathcal{D}$*

Lemma 5.2. *If $\Gamma \vdash_{\bar{\varsigma}_2} s : A$ and $|s| \rightsquigarrow t$ then $\exists\ t'$ such that $s \rightsquigarrow^*_{\neq 0} t'$ and $|t'| = t$*

Proof. By induction on $\Gamma \vdash_{\bar{\varsigma}_2} s : A$. The erasure of the Ax, Var, and Refl, cases are values and thus do not reduce. The Cast case is impossible because it is intentionally excluded. First projection is very similar to second projection case. The Int and Eq cases are structural in erasure and are thus very similar to the Pi case.

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

Have $|(x : A) \to_m B| = (x : |A|) \to_m |B|$. Suppose that $|A| \rightsquigarrow t$. By the IH applied to $\mathcal{D}_1$: $\exists\ t'$ such that $A \rightsquigarrow^*_{\neq 0} t'$ and $|t'| = t$. Thus, $(x : A) \to_m B \rightsquigarrow^*_{\neq 0} (x : t') \to_m B$ and $|(x : t') \to_m B| = (x : t) \to_m |B|$. The case where a reduction happens in $|B|$ is similar.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \to_m B : \mathrm{codom}_\Pi(m)} \qquad \overset{\mathcal{D}_2}{\Gamma; x_m : A \vdash t : B} \qquad \overset{\mathcal{D}_3}{x \notin FV(|t|) \text{ if } m = 0}}{\Gamma \vdash \lambda_m\, x\!:\!A.\, t : (x : A) \to_m B}$$

Suppose $m = 0$. Have $|\lambda_0\, x\!:\!A.\, b| = |b|$ with $|b| \rightsquigarrow t$. Applying the IH to $\mathcal{D}_2$ concludes the case.

Suppose that $m = \omega$, note that $m = \tau$ is very similar and thus omitted. Have $|\lambda_\omega\, x : A.\, b| = \lambda_\omega\, x : \diamond.\, |b|$ and $|b| \rightsquigarrow t$. Applying the IH to $\mathcal{D}_2$ yields $\exists\ t'$ such that $b \rightsquigarrow^*_{\neq 0} t'$ and $|t'| = t$. Now $\lambda_\omega\, x\!:\!A.\, b \rightsquigarrow^*_{\neq 0} \lambda_\omega\, x\!:\!A.\, t'$ and $|\lambda_\omega\, x\!:\!A.\, t'| = \lambda_\omega\, x\!:\!\diamond.\, t$.

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

If $m = 0$ then the proof follows by a straightforward application of the IH to $\mathcal{D}_1$.

Suppose that $m = \omega$. Let $|f| = \lambda_\omega\, x\!:\!\diamond.\, v$ and $|f| \bullet_\omega |a| \rightsquigarrow [x := |a|]v$. By Theorem 3.19 f is strongly normalizing in proof reduction. If f contains a projection redex, promotion redex, or erased application redex then produce f_i by contracting that redex. Continue contracting these redexes until none remain, assume k such redexes are contracted, thus $f \rightsquigarrow^* f_k$. Note that none of these redexes affect the erasure of f, thus $|f| = |f_k|$. Now f_k has only three possibilities: $f_k = \lambda_\omega\, x\!:\!A.\, b$, or $f_k = \psi(\mathrm{refl}(z; Z), a, b; A, P)$, or $f_k = \delta(\mathrm{refl}(t; A))$. The φ case is impossible by the restriction of the judgment and by Theorem 4.15 the δ case is impossible.

- Suppose $f_k = \lambda_\omega\, x\!:\!A.\, b$. Now $f_k \bullet_\omega a \rightsquigarrow [x := a]b$ and $|[x := a]b| = [x := |a|]v$.
- Suppose $f_k = \psi(\mathrm{refl}(z; Z), a, b; A, P)$. Now $\psi(\mathrm{refl}(z; Z), a, b; A, P) \bullet_\omega a \rightsquigarrow a$. Note that $|f_k| = |f|$, but $|\psi(\mathrm{refl}(z; Z), a, b; A, P)| = \lambda_\omega\, x\!:\!\diamond.\, x$ and $|f| = \lambda_\omega\, x\!:\!\diamond.\, v$. Thus, $v = x$ and $|a| = [x := |a|]v$.

Suppose $m = \omega$ and $|f| \rightsquigarrow t$. Note that the case where $|a| \rightsquigarrow t$ is very similar and thus omitted. Applying the IH to $\mathcal{D}_1$ gives $\exists\ t'$ such that $f \rightsquigarrow^*_{\neq 0} t'$ and $|t'| = t$. Now $f \bullet_\omega a \rightsquigarrow^*_{\neq 0} t' \bullet_\omega a$ and $|t' \bullet_\omega a| = t \bullet_\omega |a|$.

Suppose $m = \tau$ then erasure is structural. Thus, a β-redex is tracked exactly and any structural redexes are very similar to the $m = \omega$ case.

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

Have $|[t_1, t_2; A]| = |t_1|$ and $|t_1| \leadsto t$. Applying the IH to $\mathcal{D}_1$ yields $\exists\ t'$ such that $t_1 \leadsto^*_{\neq 0} t'$ and $|t'| = t$. Now $[t_1, t_2; A] \leadsto^*_{\neq 0} [t', t_2; A]$ and $|[t', t_2; A]| = t$.

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

Have $|b.2| = |b|$ and $|b| \leadsto t$. Applying the IH to $\mathcal{D}_1$ gives $\exists\ t'$ such that $b \leadsto^*_{\neq 0} t'$ and $|t'| = t$. Now $b.2 \leadsto^*_{\neq 0} t'.2$ and $|t'.2| = t$.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : A} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : A} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a =_A b} \qquad \overset{\mathcal{D}_5}{\Gamma \vdash P : (y : A) \to_\tau (p : a =_A y_\star) \to_\tau \star}}{\Gamma \vdash \psi(e, a, b; A, P) : P \bullet_\tau a \bullet_\tau \mathrm{refl}(a; A) \to_\omega P \bullet_\tau b \bullet_\tau e}$$

Have $|\psi(e, a, b; A, T)| = |e|$ and $|e| \leadsto t$. Applying the IH to $\mathcal{D}_4$ yields $\exists\ t'$ such that $e \leadsto^*_{\neq 0} t'$ and $|t'| = t$. Now $\psi(e, a, b; A, T) \leadsto^*_{\neq 0} \psi(t', a, b; A, T)$ and $|\psi(t', a, b; A, T)| = t$.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash (x : A) \cap B : \star} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash a : (x : A) \cap B} \qquad \overset{\mathcal{D}_3}{\Gamma \vdash b : (x : A) \cap B} \qquad \overset{\mathcal{D}_4}{\Gamma \vdash e : a.1 =_A b.1}}{\Gamma \vdash \vartheta(e, a, b; (x : A) \cap B) : a =_{(x:A)\cap B} b}$$

Have $|\vartheta(e, a, b; (x : A) \cap B)| = |e|$ and $|e| \leadsto t$. Applying the IH to $\mathcal{D}_4$ gives $\exists\ t'$ where $e \leadsto^*_{\neq 0} t'$ and $|t'| = t$. Now $\vartheta(e, a, b; (x : A) \cap B) \leadsto^*_{\neq 0} \vartheta(t', a, b; (x : A) \cap B)$ and $|\vartheta(t', a, b; (x : A) \cap B)| = t$.

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

Have $|\delta(e)| = |e|$ and $|e| \leadsto t$. Applying the IH to $\mathcal{D}_1$ gives $\exists\ t'$ where $e \leadsto^*_{\neq 0} t'$ and $|t'| = t$. Now $\delta(e) \leadsto^*_{\neq 0} \delta(t')$ and $|\delta(t')| = t$.

Case: $$\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : K} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash t : B} \qquad \overset{\mathcal{D}_3}{A \equiv B}}{\Gamma \vdash t : A}$$

Immediate by the IH applied to $\mathcal{D}_2$.

□

Theorem 5.3 (Strict Object Normalization)**.** *If* $\Gamma \vdash_{\varsigma_2^-} t : A$ *then* $|t|$ *is strongly normalizing*

Proof. By Theorem 3.19: t is strongly normalizing wrt proof reduction. Let ∂ be the maximum length reduction sequence t could take to reach the unique value. Suppose wlog that $|t|$ contains a redex. Contract this redex giving $|t| \rightsquigarrow e_1$. By Lemma 5.2: $\exists\ t_1$ such that $t \rightsquigarrow^*_{\neq 0} t_1$ and $|t_1| = e_1$. Using preservation of proof reduction: $\Gamma \vdash_{\c{c}^{-}_{2}} t_1 : A$. Let the number of contracted redexes by the reduction $t \rightsquigarrow^*_{\neq 0} t_1$ be k, then there is a maximum of $\partial - k$ redexes in t_1. If redexes remain in e_1 than the process can be repeated because t_1 is a strict proof whose erasure is e_1. However, eventually the number of steps taken must run out, because ∂ is a finite value. Thus, the procedure may be repeated as many times as desired, but e_i, the value after i iterations of this process, must eventually run out of redexes by Lemma 5.2. Therefore, $|t|$ is strongly normalizing. □

Strong normalization of strict objects leads to an interesting observation. Recall the definition of conversion: $a \equiv b$ if and only if $\exists\ u, v$ such that $a \rightsquigarrow^* u$, $b \rightsquigarrow^* v$ and $|u| \rightleftharpoons |v|$. An observant reader may wonder why reduction is allowed after two candidates objects, $|u|$ and $|v|$ are obtained. In other words, why not merely compare for equality: $|u| = |v|$. The answer is because φ may generate β-redexes after erasure, and it turns out that it is the only syntax form for which this is possible. Thus, if φ was removed from the system then conversion *could* be defined using equality of objects instead of reduction convertibility of objects. The φ form is unique amongst all the other syntax.

Another question that the reader may have is why not represent the reduction of φ in the proof system. The answer is that there is no obvious way to make the reduction well-typed, thus preservation would be lost. Indeed, the proof witness of a $\varphi(a, b, e)$ form, b, is allowed to be as complicated as required to produce the subtype $(x : A) \cap B$. However, the object, $|a|$, is typed at the super-type A. To make it possible to type this term in the proof system some notion of subtyping would have to be added directly into the rules. It is not immediately clear how to make this move without producing a radically different system. Yet, it does hint that the φ rule is, in some sense, expanding a semantic subtyping relation that is later realized internally via a notion of casts. Indeed, it may be fruitful to view the proof-object distinction as being fundamentally related to subtyping.

5.2 Observational Equivalence of Objects

Unfortunately, proofs involving the φ form do not have normalizing objects. While it is not clear how to augment the proof system to enforce normalization it is possible to describe an external condition on proofs that would guarantee object normalization for any arbitrary proof. The idea is to observe that each $\varphi(a, b, e)$ form has some associated proof witness (b) and some object witness (a). Evidence (e) is also provided that these two witnesses are equal at type A. If e reduces to a value, then that implies $|a| \rightleftharpoons |b|$, but if this holds than whatever usage of φ should be normalizing. However, the evidence produced in a proof need not ever reduce to a value, yet it will still be discarded by the erasure of φ.

Observational (or contextual) equivalence of objects gives a strong enough claim to transfer the normalization property from one object to another. Objects being the concept of interest means that contexts need to be well-typed because an object is only the erasure of a proof. To make contexts the inductive structure of syntax is reused with a unique fresh free variable, labelled h, that represents a hole. The variable is unique meaning it occurs only once in the given syntax, but it can be trivially duplicated by an abstraction. Context structure could be defined inductively, but this methodology allows reuse of erasure and substitution.

Definition 5.4. *A* ***context*** $\gamma : (\Gamma, A) \to (\Delta, B)$ *is a syntactic form with a unique free variable* h *representing a hole such that if* $\Gamma \vdash t : A$ *then* $\Delta \vdash [h := t]\gamma : B$*.*

Observational equivalence is then defined to be logical equivalence of divergence of the associated objects substituted for h in the given context. There are several possible ways to defined observational equivalence including the choice of what counts as an observation. For the purposes of this chapter divergence is the only observation of interest. Note that it is easy to see that observational equivalence forms an equivalence relation relative to the parameters Γ and A.

Definition 5.5. *The syntax* a *and* b *are* ***observationally equivalent*** *at* A *in* Γ *(written:* $\Gamma \vdash a \approx_A b$*) iff for any context* $\gamma : (\Gamma, A) \to (\varepsilon, \text{cUnit})$ *with unique fresh variable* h*:* $|[h := a]\gamma|$ *normalizes iff* $|[h := b]\gamma|$ *normalizes*

Lemma 5.6. $\Gamma \vdash a \approx_A a$

Proof. Immediate by definition. □

Lemma 5.7. *If* $\Gamma \vdash a \approx_A b$ *then* $\Gamma \vdash b \approx_A a$

Proof. By definition the stated condition holds via an if-and-only-if. Hence, observational equivalence is symmetric. □

Lemma 5.8. *If* $\Gamma \vdash a \approx_A b$ *and* $\Gamma \vdash b \approx_A c$ *then* $\Gamma \vdash a \approx_A c$

Proof. Let $\gamma : (\Gamma, A) \to (\varepsilon, \text{cUnit})$ be an arbitrary context with unique fresh variable h. Suppose $|[h := b]\gamma|$ diverges, then by $\Gamma \vdash b \approx_A c$ it must be the case that $|[h := c]\gamma|$ diverges. By Lemma 5.7: $\Gamma \vdash b \approx_A a$ and thus as above $|[h := a]\gamma|$ diverges. Suppose $|[h := b]\gamma|$ normalizes, then by $\Gamma \vdash b \approx_A c$: $|[h := c]\gamma|$ normalizes. Likewise, using symmetry and the same reasoning: $|[h := a]\gamma|$ normalizes. Hence, $|[h := a]\gamma|$ normalizes if and only if $|[h := c]\gamma|$ normalizes. □

Definition 5.9. *A proof is* φ*-****safe*** *iff for every usage of* φ *with* $\Gamma \vdash \varphi(a, b, e; A, (x : A) \cap B) : (x : A) \cap B$ *then* $\Gamma \vdash \varphi(a, b, e; A, (x : A) \cap B) \approx_{(x:A)\cap B} b$

Theorem 5.10. *If* $\Gamma \vdash t : A$ *and* t *is* φ*-safe then* $|t|$ *is strongly normalizing*

Proof. By lexicographic induction on the nesting count of φ in t and the inference judgment $\Gamma \vdash t : A$. If t does not contain any φ subexpressions then it is a strict proof and thus $|t|$ is strongly normalizing by Theorem 5.3. Thus, suppose t has $i + 1$ nested φ expressions. For every case except the App case $|t|$ is strongly normalizing by the IH. The App case is special because the function-part could be a φ and thus generate a β-redex in erasure that is not tracked by proof reduction.

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

Suppose wlog that $f = \varphi(a', b, e).2$ and thus $|f| \bullet_\omega |a| = |a'| \bullet_\omega |a|$. By the IH both $|a'|$ and $|a|$ are strongly normalizing. Note that t is φ-safe thus $\Gamma \vdash \varphi(a', b, e) \approx_{(x:A)\cap B} b$. Thus, it must be the case that $\Gamma \vdash \varphi(a', b, e).2 \bullet_\omega a \approx_{[x:=a]B} b.2 \bullet_\omega a$. Hence, for context γ with hole h: $[h := |a'| \bullet_\omega |a|]\gamma$ is normalizing if and only if $[h := |b| \bullet_\omega |a|]\gamma$ is normalizing. However, $b.2 \bullet_\omega a$ has a smaller nesting level of φ expressions, thus $|b| \bullet_\omega |a|$ is strongly normalizing.

□

Characterizing when φ does not introduce diverging objects is useful because it enables, at the bare minimum, an external validation of each usage. It is not clear how this requirement may be internalized in the system. First, a logical relation capturing observational equivalence would likely need to be developed, but because this relation needs to capture equivalence of objects it is not obvious how to adapt existing approaches. Moreover, that relation would have to be bolted on an as auxiliary proof system in order to prove φ-safety. At least, the evidence required to use a Cast rule is a sanity check. Indeed, if this evidence is "morally" true then contextual equivalence will hold by the Leibniz Law.

Conjecture 5.11. $\Gamma \vdash \varphi(a, b, e) \approx_{(x:A)\cap B} b$ *iff* $\Gamma \vdash a \approx_A b.1$

Note that while the evidence for $\varphi(a, b, e)$ has the type $e : a =_A b.1$ it is easy to use this evidence to construct a proof $e' : \varphi(a, b, e) =_{(x:A)\cap B} b$. Just eliminate e using ψ and the objects will match. Going the opposite direction is just as simple, as b may be substituted with the left-hand side and the objects will again be identical. However, it is not clear that a first projection expressed via observational equivalence is logically equivalent to φ-safety. The primary obstacle is determining if the erasure of every $\gamma : ((x : A) \cap B, \Gamma) \to (\varepsilon, \text{cUnit})$ context can be computed via a first projection operation on contexts to obtain $\gamma.1 : (A, \Gamma) \to (\varepsilon, \text{cUnit})$ with the same erasure. Demonstrating this conjecture holds would be the first important step to defining a logical relation for contextual equivalence, because it would mean that φ terms could be removed entirely from the definition.

5.3 Counterexamples with φ

It does not take much effort to produce an example of divergence using φ. Note, however, that all examples require a context where False is derivable. The first example uses φ to give self a recursive type: self : cUnit and self : cUnit $\to_\omega$ cUnit simultaneously. Divergence is a trivial consequence. In this example, the False premise is completely erased. This is not really a problem as a proof assistant needs to reduce under binders anyway and the erased argument blocks logical issues regardless.

$$\begin{aligned}
\text{False} &= (X : \star) \to_0 X_\square \\
\text{self} &= \lambda_\omega\, x \mathbin{:} \text{cUnit}.\, x \bullet_0 \text{cUnit} \bullet_\omega x \\
|\text{self}| &= \lambda_\omega\, x \mathbin{:} \diamond.\, x \bullet_\omega x \\
b &= \lambda_\omega\, f \mathbin{:} \text{False}.\, [f \bullet_0 (\text{cUnit} \to_\omega \text{cUnit}), f \bullet_0 \text{cUnit}] \\
e &= \lambda_\omega\, f \mathbin{:} \text{False}.\, f \bullet_0 (\text{self} =_{\text{cUnit} \to_\omega \text{cUnit}} (b \bullet_\omega f).1) \\
\mathit{bad} &= \lambda_0\, f \mathbin{:} \text{False}.\, \text{self} \bullet_\omega (\varphi(\text{self}, b \bullet_\omega f, e \bullet_\omega f)).2 \\
|\mathit{bad}| &= |\text{self}| \bullet_\omega |\text{self}|
\end{aligned}$$

What one can learn from the above example is that the hypothetical evidence is problematic for using φ. Restricting the context is one idea to make all usages φ-safe. Unfortunately, the restriction that $FV(|e|)$ is empty is too strong, it prevents all interesting usages because $b.1 \rightsquigarrow^* a$ in all cases as a result. Instead, the reader might imagine that the context is *partially* restricted. For example, suppose $b : (a : A) \to (x : A) \cap B$ and $e : (a : A) \to a_\star =_A (b \bullet_\omega a).1$ with $FV(|e|)$ empty. With this setup, e depends only on the single input and expresses only the fact that b is extensionally an identity function. The object witness term a can then be dropped and the object for the φ term would be: $|\varphi(b, e)| = \lambda_\omega\, x : \diamond.\, x$. Unfortunately, this idea fails as enough of the context may be uncurried into the type of A to construct a divergent term.

$$\begin{aligned}
A &= (\text{cUnit} \to_\omega \text{cUnit}) \times \text{False} \\
T &= (A \to_\omega \text{cUnit} \to_\omega \text{cUnit}) \to_\omega (\text{cUnit} \to_\omega \text{cUnit}) \to_\omega \text{cUnit} \\
b &= \lambda_\omega\, w \mathbin{:} A.\, (\text{csnd} \bullet_\omega w) \bullet_0 (A \cap T) \\
e &= \lambda_\omega\, x \mathbin{:} A.\, (\text{csnd} \bullet_\omega x) \bullet_0 (x =_A (b \bullet_\omega x).1) \\
\mathit{phi} &= \lambda_\omega\, a \mathbin{:} A.\, \varphi(a, b \bullet_\omega a, e \bullet_\omega a) \\
\mathit{p1} &= \lambda_\omega\, f \mathbin{:} \text{False}.\, \text{cpair} \bullet_\omega \text{self} \bullet_\omega f \\
\mathit{p2} &= \lambda_\omega\, x \mathbin{:} A.\, \text{cfst} \bullet_\omega x \\
\mathit{p3} &= \lambda_\omega\, f \mathbin{:} \text{False}.\, (\mathit{phi} \bullet_\omega (\mathit{p1} \bullet_\omega f)).2 \bullet_\omega \mathit{p2} \bullet_\omega \text{self} \\
\mathit{bad} &= \lambda_\omega\, f \mathbin{:} \text{False}.\, (\mathit{p3} \bullet_\omega f \bullet_0 \text{cUnit} \bullet_\omega \text{cunit}) \bullet_0 (\text{cUnit} \to_\omega \text{cUnit}) \bullet_\omega \text{self} \\
|\mathit{bad}| &= \lambda_\omega\, f \mathbin{:} \diamond.\, |\text{self}| \bullet_\omega |\text{self}|
\end{aligned}$$

This counterexample requires a relevant abstraction, but this could probably be avoided by a more sophisticated formulation. Again, it also does not really matter as proof assistants reduce under binders anyway. This example demonstrates that finding a balance between usability and restriction of the context is very difficult, if not simply impossible.

Another option is to remove φ altogether from the system. It is a significant source of complexity because it demands reduction after erasure in the definition of conversion and is the *only* source of divergence in ς_2. To contrast, ς_1 has the following sources of divergence:

1. terms on the left-hand and right-hand side of an equality are untyped-λ-calculus syntax;

2. the Kleene Trick allows for untyped-λ-calculus syntax as witnesses of trivial equalities;

3. the rewrite rule, ρ, is erased and thus enables non-termination by Abel and Coquand [1];

4. the φ rule, for the same reason as ç$_2$.

The first three sources are eliminated by the design of ç$_2$, yet the last remains. Ultimately, the Cast rule is too important to not only the spirit of Cedille but its capability. Losing φ, as far as the current research shows, would prevent almost all existing encodings. The cost to be paid would be too much.

Chapter 6

CONCLUSION

The design of Cedille followed an extrinsic (or curry-style) philosophy that placed programs as primary with types as annotations. Under this philosophy it is only natural to consider an untyped equality because that is the version that is closer to ones semantic understanding. However, these decision led to equality enabling non-terminating derivations originating independently using: untyped indices; untyped witnesses of reflexivity; untyped Leibniz Law; erased ρ (when type equality is present); erased δ; and φ. In this work an alternative road is taken where $ç_2$ is designed around a hybrid philosophy of intrinsic and extrinsic. Proofs are considered primary and the design itself follows proof theoretic principles, but universal (dependent) quantification is with respect to objects. An object is the erasure of a proof, but critically it is not merely an untyped λ-calculus term. Objects do not exist without proofs. This distinction allowed for a description of proof reduction and, separately, object reduction. Moreover, many metatheoretic properties are shown relative to proof reduction and conversion including: syntactic proof preservation and strong proof normalization. Additionally, in the absence of the φ construct, proof reduction upper-bounds object reduction.

A failure of the proof theoretic discipline and victory of the extrinsic philosophy is the φ construct. Unfortunately, the Cast rule does block many desirable properties such as decidability of type checking. However, it does not prevent logical consistency and the apparent strength it adds to the theory is substantial. Indeed, it is not clear how to derive any of the interesting encodings published in existing literature without the φ construct. It is the only source of non-termination in the system $ç_2$.

This trade-off is the best that can be offered at this moment in time as it is not clear how to systematically correct φ without destroying all of its benefits. To that end, the remaining of this chapter will inform the reader on alternative paths taken with equality in other type theories and other pertinent related work. Additionally, a brief section of future work is presented to communicate open problems and, in the opinion of the author, promising ideas.

6.1 Related Work

6.2 Future Work

There are three important open questions that this work does not address that should hopefully be answered in the future. While the first two questions are concerned with the φ rule the third notes of a new possibility obtained by making the equality typed in $ç_2$.

First, there is a question if the φ construct is necessary for efficient inductive data. Without the Cast rule it is possible to derive inductive Church encodings that are *not* efficient because computing an out (e.g. a predecessor of a natural number) is proportional to the size of the data (e.g. linear time for natural numbers). One potential method is using Scott encodings instead which have been shown to be inductive in Cedille [60]. Note that the method for showing this in Cedille does leverage the φ construct. However, it seems unlikely that all ink has dried on the topic of impredicative encodings, especially in Cedille. While an "escape hatch" such as φ would still be present in the system knowing which encodings do not depend on it would be useful to delineate the relative power between the system with and without φ.

Second, modifying the φ rule is another avenue that requires more investigation. While restricting the rule with internal derivations seems to be a dead end because any restriction may be satisfied by an inconsistent context there could be lightweight external additions to enable using the Cast rule without caveats. Another possibility is dropping the φ rule and replacing it with weaker variants that are still capable enough to derive existing encodings. It is, however, not clear just how many special rules would be required. For example, requiring a special rule for each encoding is tantamount to extending the system with that particular construct anyway. Thus, why bother with the special rules.

Finally, now that the equality type is typed it is possible that function extensionality is a consistent axiom. Indeed, because equality is untyped in Cedille the δ rule is powerful enough to separate two extensionally equal functions, refuting the axiom. With a typed equality the separation rule of $ç_2$ may only act on typed terms. Therefore, separation is only capable of differentiation functions by applying test inputs. While this is convincing intuition showing that this axiom is consistent requires an extensional model which, of course, cannot exist for Cedille.

6.3 Closing Remarks

The core system $ç_2$ represents a step forward to a proof theoretic version of Cedille. With this design the equality type is modified to remove all sources of non-termination with the lone exception of the φ construct. Ideally, this construct would also be corrected to disallow creation of non-terminating terms, but there is currently no known method to systematically accomplish this. A system without φ is capable of deriving inductive Church encodings, but many (if not all) of the other encodings published in existing research use φ in an essential way. Nevertheless, there is an external condition that may be checked to ensure a particular usage of φ does not introduce non-termination. Even with this trade-off the system $ç_2$ enjoys a better behaved metatheory than Cedille while enabling derivation of the currently existing encodings.

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] Andreas ABEL et al. “Leibniz equality is isomorphic to Martin-Löf identity, parametrically”. In: *Journal of Functional Programming* 30 (2020), e17. DOI: 10.1017/S0956796820000155.

[3] 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.

[4] Thorsten Altenkirch. “Extensional equality in intensional type theory”. In: *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*. 1999, pp. 412–420. DOI: 10.1109/LICS.1999.782636.

[5] Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. “Extending homotopy type theory with strict equality”. In: *arXiv preprint arXiv:1604.03799* (2016).

[6] Thorsten Altenkirch and Ambrus Kaposi. “Type Theory in Type Theory Using Quotient Inductive Types”. In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’16. St. Petersburg, FL, USA: Association for Computing Machinery, 2016, pp. 18–29. ISBN: 9781450335492. DOI: 10.1145/2837614.2837638.

[7] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. “Observational Equality, Now!” In: *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification*. PLPV ’07. Freiburg, Germany: Association for Computing Machinery, 2007, pp. 57–68. ISBN: 9781595936776. DOI: 10.1145/1292597.1292608.

[8] Thorsten Altenkirch et al. “Constructing a universe for the setoid model”. In: *Foundations of Software Science and Computation Structures*. Ed. by Stefan Kiefer and Christine Tasson. Cham: Springer International Publishing, 2021, pp. 1–21. ISBN: 978-3-030-71995-1. DOI: 10.1007/978-3-030-71995-1_1.

[9] Thorsten Altenkirch et al. “Setoid Type Theory—A Syntactic Translation”. In: *Mathematics of Program Construction*. Ed. by Graham Hutton. Cham: Springer International Publishing, 2019, pp. 155–196. ISBN: 978-3-030-33636-3. DOI: 10.1007/978-3-030-33636-3_7.

[10] Carlo Angiuli. “Computational semantics of Cartesian cubical type theory”. PhD thesis. 2019.

[11] Carlo Angiuli, Robert Harper, and Todd Wilson. “Computational Higher-Dimensional Type Theory”. In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. Paris, France: Association for Computing Machinery, 2017, pp. 680–693. ISBN: 9781450346603. DOI: 10.1145/3009837.30098article61.

[12] Danil Annenkov et al. “Two-level type theory and applications”. In: *arXiv preprint arXiv:1705.03307* (2017).

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

[14] Robert Atkey. “Syntax and Semantics of Quantitative Type Theory”. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS ’18. Oxford, United Kingdom: Association for Computing Machinery, 2018, pp. 56–65. ISBN: 9781450355834. DOI: 10.1145/3209108.3209189.

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

[16] 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.

[17] Andrej Bauer et al. “Design and Implementation of the Andromeda proof assistant, 2016”. In: *TYPES* (2016).

[18] Marc Bezem, Thierry Coquand, and Simon Huber. “A model of type theory in cubical sets”. In: *19th International conference on types for proofs and programs (TYPES 2013)*. Vol. 26. 2014, pp. 107–128.

[19] Marc Bezem, Thierry Coquand, and Erik Parmann. “Non-Constructivity in Kan Simplicial Sets”. In: *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*. Ed. by Thorsten Altenkirch. Vol. 38. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 92–106. ISBN: 978-3-939897-87-3. DOI: 10.4230/LIPIcs.TLCA.2015.92.

[20] Errett Albert Bishop. *Foundations of Constructive Analysis*. New York, NY, USA: Mcgraw-Hill, 1967.

[21] Simon Boulier and Théo Winterhalter. “Weak Type Theory is Rather Strong”. In: *TYPES* (2019).

[22] Edwin Brady. “Idris 2: Quantitative type theory in practice”. In: *arXiv preprint arXiv:2104.00480* (2021).

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

[24] Paolo Capriotti. “Models of type theory with strict equality”. PhD thesis. 2017.

[25] Mario Carneiro. “The Type Theory of Lean”. MA thesis. Carnegie Mellon University, 2019.

[26] Evan Cavallo, Anders Mörtberg, and Andrew W Swan. “Unifying Cubical Models of Univalent Type Theory”. In: *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*. Ed. by Maribel Fernández and Anca Muscholl. Vol. 152. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 14:1–14:17. ISBN: 978-3-95977-132-0. DOI: 10.4230/LIPIcs.CSL.2020.14.

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

[28] Laurent Chicli, Loïc Pottier, and Carlos Simpson. “Mathematical Quotients and Quotient Types in Coq”. In: *Types for Proofs and Programs*. Ed. by Herman Geuvers and Freek Wiedijk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 95–107. ISBN: 978-3-540-39185-2.

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

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

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

[32] Jesper Cockx. “Type Theory Unchained: Extending Agda with User-Defined Rewrite Rules”. In: *25th International Conference on Types for Proofs and Programs (TYPES 2019)*. Ed. by Marc Bezem and Assia Mahboubi. Vol. 175. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 2:1–2:27. ISBN: 978-3-95977-158-0. DOI: 10.4230/LIPIcs.TYPES.2019.2.

[33] Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. “The Taming of the Rew: A Type Theory with Computational Assumptions”. In: *Proc. ACM Program. Lang.* 5.POPL (Jan. 2021). DOI: 10.1145/3434341.

[34] Cyril Cohen et al. “Cubical type theory: a constructive interpretation of the univalence axiom”. In: *arXiv preprint arXiv:1611.02108* (2016).

[35] RL Constable et al. “Implementing Mathematics with the Nuprl Proof Development System”. In: *Prentice-Hall, Inc.* (1986).

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

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

[38] 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.

[39] Larry Diehl, Denis Firsov, and Aaron Stump. “Generic zero-cost reuse for dependent types”. In: *Proceedings of the ACM on Programming Languages* 2.ICFP (2018), pp. 1–30.

[40] Gabe Dijkstra. “Quotient inductive-inductive definitions”. PhD thesis. 2017.

[41] 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.

[42] Denis Firsov et al. “Course-of-Value Induction in Cedille”. In: *arXiv preprint arXiv:1811.11961* (2018).

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

[44] 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).

[45] Peng Fu and Aaron Stump. “Self types for dependently typed lambda encodings”. In: *International Conference on Rewriting Techniques and Applications*. Springer. 2014, pp. 224–239.

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

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

[48] 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.

[49] 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.

[50] 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.

[51] Gaëtan Gilbert et al. “Definitional Proof-Irrelevance without K”. In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019). DOI: 10.1145/3290316.

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

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

[54] Martin Hofmann. “Extensional concepts in intensional type theory”. PhD thesis. 1995.

[55] Martin Hofmann and Thomas Streicher. “The groupoid interpretation of type theory”. In: *Twenty-five years of constructive type theory (Venice, 1995)* 36 (1996), pp. 83–111.

[56] 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.

[57] 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.

[58] Christopher Jenkins, Andrew Marmaduke, and Aaron Stump. “Simulating large eliminations in cedille”. In: *arXiv preprint arXiv:2112.07817* (2021).

[59] Christopher Jenkins and Aaron Stump. “Monotone recursive types and recursive data representations in Cedille”. In: *Mathematical structures in computer science* 31.6 (2021), pp. 682–745.

[60] Christopher Jenkins and Aaron Stump. “Monotone recursive types and recursive data representations in Cedille”. In: *Mathematical structures in computer science* 31.6 (2021), pp. 682–745.

[61] Christopher Jenkins, Aaron Stump, and Larry Diehl. “Efficient lambda encodings for Mendler-style coinductive types in Cedille”. In: *Electronic Proceedings in Theoretical Computer Science, EPTCS* 317 (2020), pp. 72–97.

[62] Chris Kapulkin and Peter LeFanu Lumsdaine. “The simplicial model of univalent foundations (after Voevodsky)”. In: *arXiv preprint arXiv:1211.2851* (2012).

[63] 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.

[64] Nicolai Kraus and Jakob von Raumer. "Coherence via Well-Foundedness: Taming Set-Quotients in Homotopy Type Theory". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. Saarbrücken, Germany: Association for Computing Machinery, 2020, pp. 662–675. ISBN: 9781450371049. DOI: 10.1145/3373718.3394800.

[65] 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.

[66] Nuo Li. "Quotient types in type theory". PhD thesis. 2015.

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

[68] Andrew Marmaduke, Larry Diehl, and Aaron Stump. "Impredicative Encodings of Inductive-Inductive Data in Cedille". In: *International Symposium on Trends in Functional Programming*. Springer. 2023, pp. 1–15.

[69] Andrew Marmaduke, Christopher Jenkins, and Aaron Stump. "Quotients by idempotent functions in cedille". In: *Trends in Functional Programming: 20th International Symposium, TFP 2019, Vancouver, BC, Canada, June 12–14, 2019, Revised Selected Papers 20*. Springer. 2020, pp. 1–20.

[70] Andrew Marmaduke, Christopher Jenkins, and Aaron Stump. "Zero-cost constructor subtyping". In: *Proceedings of the 32nd Symposium on Implementation and Application of Functional Languages*. 2020, pp. 93–103.

[71] Per Martin-Löf. "100 Years of Zermelo's Axiom of Choice: What was the Problem with It?" In: *Logicism, Intuitionism, and Formalism: What has Become of Them?* Ed. by Sten Lindström et al. Dordrecht: Springer Netherlands, 2009, pp. 209–219. ISBN: 978-1-4020-8926-8. DOI: 10.1007/978-1-4020-8926-8_10.

[72] 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.

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

[74] 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.

[75] 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.

[76] 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.

[77] 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.

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

[79] Frank Pfenning. “Church and Curry: Combining intrinsic and extrinsic typing”. In: *Studies in Logic and the Foundations of Mathematics* (2008).

[80] 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.

[81] 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.

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

[83] Andrew M Pitts and Ian Orton. “Axioms for modelling cubical type theory in a topos”. In: *Logical Methods in Computer Science* 14 (2018).

[84] 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.

[85] Loïc Pujet and Nicolas Tabareau. “Observational Equality: Now for Good”. In: *Proc. ACM Program. Lang.* 6.POPL (Jan. 2022). DOI: 10.1145/3498693.

[86] 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.

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

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

[89] Vilhelm Sjöberg. “A dependently typed language with nontermination”. PhD thesis. 2015.

[90] Vilhelm Sjöberg and Aaron Stump. “Equality, quasi-implicit products, and large eliminations”. In: *arXiv preprint arXiv:1101.4430* (2011).

[91] Vilhelm Sjöberg et al. “Irrelevance, heterogeneous equality, and call-by-value dependent type systems”. In: *arXiv preprint arXiv:1202.2923* (2012).

[92] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. “A cubical language for Bishop sets”. In: *arXiv preprint arXiv:2003.01491* (2020).

[93] Thomas Streicher. “Investigations into intensional type theory”. Ludwig Maximilian Universität, 1993.

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

[95] Aaron Stump and Peng Fu. “Efficiency of lambda-encodings in total type theory”. In: *Journal of functional programming* 26 (2016), e3.

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

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

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

[99] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

[100] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. "Cubical Agda: A dependently typed programming language with univalence and higher inductive types". In: *Journal of Functional Programming* 31 (2021), e8. DOI: 10.1017/S0956796821000034.

[101] Vladimir Voevodsky. "A simple type system with two identity types". In: *Unpublished note* (2013).

[102] Vladimir Voevodsky. "A very short note on the homotopy λ-calculus". In: *Unpublished note* (2006), pp. 10–27.

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

[104] Alfred North Whitehead and Bertrand Russell. "Principia Mathematica". In: (1927).

[105] Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. "Eliminating Reflection from Type Theory". In: *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2019. Cascais, Portugal: Association for Computing Machinery, 2019, pp. 91–103. ISBN: 9781450362221. DOI: 10.1145/3293880.3294095.

[106] Yanpeng Yang and Bruno C. D. S. Oliveira. "Pure iso-type systems". In: *Journal of Functional Programming* 29 (2019), e14. DOI: 10.1017/S0956796819000108.