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

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.

– Some wise dude

ACKNOWLEDGMENTS

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

ABSTRACT

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

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

PUBLIC ABSTRACT

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

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

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

Chapter 1

INTRODUCTION

Initial opener Type Theory as a discipline is a difficult subject to thoroughly introduce because it in essence captures a wide variety of programming languages (if not all programming languages currently defined). To trim the fat this thesis will focus on a particular type theory, System F^{ω}, and the various bits of machinery that are required to describe it. However, even with this focus there are different equivalent methods of presenting the theory: Pure Type Systems, Martin-Löf style presentations, Bidirectional systems, etc. An extrinsic bidirectional presentation will be used with only summary remarks, if any, for the other styles. This introduction is far from complete and is instead focused on providing the reader with enough background to understand the later chapters.

Undergraduate-level description of System F Omega System F^{ω} is a well study type theory that supports basic function types that are in every typed programming language, unconstrained paramterized types (sometimes referred to as generics in other programming languages), and predicates. System F^{ω} was first introduced by Girard in TODO:YEAR as an extension to System F which Girard also invented (in TODO:YEAR) [TODO:CITE]. An extension of System F^{ω} is the core calculus of Haskell [TODO:CITE]. Indeed, System F^{ω} has a privileged role in the design space of functional programming languages.

Describe the syntax including opening/closing/substitution, note that variable bureaucracy is going to be taken for granted in exposition The syntax of System F^{ω} is relatively simple, see Figure TODO:FIG. While the syntax is presented in uniformly there are three distinct sorts: kinds, types, and terms. A function type can exist on that kind or type level and a function definition (i.e. a lambda abstraction) may bind a kind at the type level or a term at the term level. The third constructor is also a function definition, the uppercase lambda abstraction, which binds a type at the term level. Function applica-

tion is represented as jutxaposition (i.e. placing two terms next to eachother with not syntactic marker).

Of course, variables are also present in the syntax but the notion of variable is a surpisingly subtle. There are several methods of formally working with variables in syntax TODO:LIST-AND-CITE. For the exposition of the theory in this work it is assumed that variable bureaucracy is handled when needed. Thus, issues of variable freshness, capture-avoiding substitution, and equivalence of terms up-to renaming are elided.

TODO:Put a tutorial of this stuff in an Appendix?

Describe reduction, multistep reduction (for any predicate), conversion (for any predicate), and how reduction is confluent and transitive, and how reduction is strongly normalizing The syntax itself has a well understood notion of reduction to normal form, if such a form exists. Figure TODO:FIG presents the reduction rules. Most rules are structural, stating that if a reduction occurs in a subterm then the whole term also reduces. Where the reduction does meaningful work is in the β-rule (or the substitution rule). This rule models application of function to argument, replacing copies of the bound variable with the argument term. Thus, reduction on syntax describes only evaluation of functions. As extensions are added, such as pairs or numerals, the reduction rules are augmented to support different evaluations of data.

A syntactic form is said to be a *value* if there is no possible reduction step that can be taken (i.e. all functions are fully evaluated). If there exists some sequence of reduction steps that lead to a value for a given syntax form, then that syntax is said to be *normalizing*. If *all* sequences of reduction steps lead to a value, then the form is *strongly normalizing*. Restricting the set of syntactic forms to a normalizing subset is critical to achieve decidability of the reduction relation, and thus decidability of type checking.

Given any relation, the reflexive-transitive closure may be defined inductively, as shown in Figure TODO:FIG. In particular, if $-R-$ is a relation, then $-R^*-$ is a supserset of that relation such that for any x, y, z we have xR^*x and $xR^*y \to yR^*z \to xR^*z$. The reduction rules defined previously is a relation on terms, and therefore omits a reflexive-transitive closure. Reduction rules will be referred to as the *β-step relation* or simply the *step relation*

and the reflexive-transition closure as the *multistep-relation* because it models reduction taking zero or more individual steps.

Conversion between syntax is defined as two syntactic forms sharing a common reduction, i.e. t_1 is convertible with t_2 (written $t_1 \equiv t_2$) if there exists a z such that $t_1 \rightsquigarrow_\beta^* z$ and $t_2 \rightsquigarrow_\beta^* z$. The conversion relation is used in the typing rules to allow syntax forms to have continued valid types as the syntax reduces. Indeed, it is possible for the evaluation of a function in some given syntax to require simultaneously reductions in the type to maintain the typing relation. It may be tempting to view conversion as the reflexive-symmetric-transitive closure of the step relation, but transitivity is not an obvious property. First, confluence must be proven.

Theorem 1 (Confluence)**.** *TODO*

The confluence theorem is non-trivial to prove, involving defining a new parallel reduction relation which is sound and complete relative to the step relation. Confluence for this parallel reduction relation is shown by induction on the relations definition, but proving confluence directly is difficult. Nevertheless, with confluence proven transitivity is a straightforward consequence.

Theorem 2 (Transitivity of Conversion)**.** *TODO*

Proof. TODO □

Describe the typing rules using a bidirectional system, note that type checking is decidable The type system restricts syntax to a set of *proofs*. A proof has significant philosophical connotations, involving the verficiation of some fact. For System F^ω there are *kinds* of which *types* are the proofs, and *types* of which *terms* are the proofs. For this thesis, a proof is some finite syntax tree with enough information to decidably reconstruct the given fact it verifies. The typing rules (i.e. the proof rules) are presented in Figure FIG:TODO. These proof rules are *bidirectional* consisting of four forms:

1. $\Gamma \vdash t \triangleright T$, to be read as T is the inferred fact of the proof t in the context Γ or more simply, t infers T in Γ;

2. $\Gamma \vdash t \mid\triangleright T$, to be read as T is the inferred fact, possibly after some reduction, of the proof t in the context Γ or simply t constrained infers T in Γ;

3. $\Gamma \vdash t \triangleleft T$, to be read as T is checked against the inferred fact of the proof t in the context Γ or simply, 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 are proofs

Each of these relations is called a *judgment*. Judgments are only about proofs and their contexts. The only exception is the constant $\Box$, which is called the *super-kind* and is the only syntactic form that appears only in a fact position and never in a proof position. These judgments are syntax-directed, meaning that if a given proof is pattern matched it is immediately obvious which rule it must be a part of, there is no ambiguity between constructors of a judgment and a given proof. The inference rule is interpretted as the function `infer(t,G)` with output T. The constrained inference rule requires a normalization subproducre `norm(t)` with output the value of t, then it is a pattern match on the value of the inferred T. The checking rule is interpreted as the function `check(t,G,T)` with output a boolean deciding whether or not T checks against the inferred type of t. Finally, well-formed context rule is an iterated application of the above rules to the elements of the context. Because the rules are syntax directed, the implementation of `infer` is straightforward by pattern matching. Likewise, the `check` function is easily defined in terms of `infer` and `norm`. The issue is the `norm` function which may not be terminating. Indeed, `norm` is terminating exactly when the multistep relation is strongly normalizing. With this property type inference and checking are decidable and all syntactic forms are proofs as originally promised.

Describe Church encodings of data in System F Omega note that they cannot be inductive While it is true that System F Omega 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:

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

The data type of lists is one easy generalization away from the definition of natural numbers. Conceptually, data types can be defined by assuming the existence of some abstract type such that it supports a given interface. For example, the natural numbers is conceptually defined using this idea as:

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

where the $\times$ symbol encodes the logical notion of *and*. This defintion is saying that there exists an abstract type that supports two constructors: zero and successor. However, we can encode existentials with function types:

$$\exists X,\ X \times (X \to X) = (X : \star) \to (X \times (X \to X)) \to X$$

Finally, one application of currying yields the original definition of naturals.

Likewise, pairs and sum types are defined:

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

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

For the sum type the final translation to function types is of course still possible, but with pairs already defined it can also be left as is. The logical constants true and false are defined:

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

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

Negation is unique in that there is no constructor, thus it is difficult to give a data type style definition in terms of existentials without already having a false constant defined. Finally, negation is defined thus:

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

While data types like $\mathbb{N}$ are definable they lack induction TODO:CITE. Therefore, if a user wished to reason about properties of the natural numbers they would be forced to postulate an induction principle as an addiitonal argument. This is not only cumbersome but unsatisfactory as the natural

numbers are in their essence the least set satisfying their induction principle. Ultimately, the issue is that these definitions are too general. They omit more possibilities than one would like and those exotic elements break the principle of induction. Note that the syntax and theory of System F^ω is not flexible enough to notice and define these exotic elements, but it is also not strong enough to rule them out.

Carve out the relevant subsystems, connect them to known notions of logic System F^ω is a conservative extension of two other systems of note. An extension is *conservative* if it agrees with the subsystem on all facts expressible in both systems. One subsystem is obvious: System F, but the simply typed lambda calculus (STLC) is also a subsystem of System F^ω. All three of these systems have suprising connections to standard logics. Named the Curry-Howard correspondence TODO:CITE, each system is in fact a reification of a particular logic of note.

For STLC, it is propositional logic. Each type in STLC encodes a proposition and the proofs encode the derivation tree of a given proposition. For System F, it is second order logic, with propositions being quantifiable, expressed via the function type $(X : \star) \to P$. System F^ω is no exception as it corresponds to higher order logic with a copy of propositional logic at the level of types and kinds. These connections make it clear that type theory is more than just the study of programming languages, but also contains deep connections to logic.

Describe extension to CC

Add an irrelevant equality and demonstrate how it breaks decidability

Describe, briefly, how Cedille enables inductive encodings through a quotient construction

List the goals of Cedille2 and the remaining structure of the thesis

Chapter 2

THEORY DESCRIPTION AND BASIC METATHEORY

The theory described in this chapter is a variation of the core theory of Cedille [3]. It is closely related with the significant differences occurring with the equality type. This variation has two primary goals. First, to have decidable type checking (and thus decidable conversion checking). Second, to retain as many constructions as possible from Cedille. This chapter focuses on the description of the theory and some basic metatheory. By basic, we mean properties that are provable by induction on the various derivations or are otherwise provable using straightforward methods.

Syntax for the theory is described in Figure 2.1. Unlike other presentations a generic syntax tree is used with a tag to indicate different syntactic forms. There are three basic syntactic constructs: variables, binders, and constructors. A generic presentation enables occasional economic benefits in presenting other derivations. However, a more standard syntax is defined in terms of the generic one. The specific syntactic forms and the generic forms are used interchangeably whichever is more convenient.

Formally, syntax is worked with as a locally nameless set following the axioms of Pitts [1]. For the sake of presentation these details are elided. This means that freshness of variables and capture avoiding substitution are largely taken for granted in the exposition of the theory. Moreover, identity of syntactic terms is assumed to be alpha equivalence. Meaning that, again, bureaucracy around variables is taken for granted. Thus, substitution is defined simply:

$$[x := v]y = v \text{ if } x = y$$
$$[x := v]y = y \text{ if } x \neq y$$
$$[x := v]\mathfrak{b}(\kappa_1, x : t_1, t_2) = \mathfrak{b}(\kappa_1, x : [x := v]t_1, [x := v]t_2)$$
$$[x := v]\mathfrak{c}(\kappa_2, t_1, \ldots, t_{\mathfrak{a}(\kappa_2)}) = \mathfrak{c}(\kappa_2, [x := v]t_1, \ldots, [x := v]t_{\mathfrak{a}(\kappa_2)})$$

$$
\begin{aligned}
t &::= x \mid \mathfrak{b}(\kappa_1, x : t_1, t_2) \mid \mathfrak{c}(\kappa_2, t_1, \ldots, t_{\mathfrak{a}(\kappa_2)}) \\
\kappa_1 &::= \lambda_m \mid \Pi_m \mid \cap \\
\kappa_2 &::= \star \mid \Box \mid \bullet_m \mid \mathrm{pair} \mid \mathrm{proj}_1 \mid \mathrm{proj}_2 \mid \mathrm{eq} \mid \mathrm{refl} \mid J \mid \vartheta \mid \delta \mid \phi \\
m &::= \omega \mid 0 \mid \tau \\
&\mathfrak{a}(\star) = \mathfrak{a}(\Box) = 0 \\
&\mathfrak{a}(\mathrm{proj}_1) = \mathfrak{a}(\mathrm{proj}_2) = \mathfrak{a}(\mathrm{refl}) = \mathfrak{a}(\vartheta) = \mathfrak{a}(\delta) = 1 \\
&\mathfrak{a}(\bullet_m) = 2 \\
&\mathfrak{a}(\mathrm{pair}) = \mathfrak{a}(\mathrm{eq}) = \mathfrak{a}(\varphi) = 3 \\
&\mathfrak{a}(J) = 6
\end{aligned}
$$

$$
\begin{aligned}
\star &:= \mathfrak{c}(\star) & t.1 &:= \mathfrak{c}(\mathrm{proj}_1, t) \\
\Box &:= \mathfrak{c}(\Box) & t.2 &:= \mathfrak{c}(\mathrm{proj}_2, t) \\
\lambda_m\, x{:}t_1.\, t_2 &:= \mathfrak{b}(\lambda_m, x : t_1, t_2) & t_1 =_{t_2} t_3 &:= \mathfrak{c}(\mathrm{eq}, t_1, t_2, t_3) \\
(x : t_1) \to_m t_2 &:= \mathfrak{b}(\Pi_m, x : t_1, t_2) & \mathrm{refl}(t) &:= \mathfrak{c}(\mathrm{refl}, t) \\
(x : t_1) \cap t_2 &:= \mathfrak{b}(\cap, x : t_1, t_2) & \vartheta(t) &:= \mathfrak{c}(\vartheta, t) \\
t_1 \bullet_m t_2 &:= \mathfrak{c}(\bullet_m, t_1, t_2) & \delta(t) &:= \mathfrak{c}(\delta, t) \\
[t_1, t_2, t_3] &:= \mathfrak{c}(\mathrm{pair}, t_1, t_2, t_3) & \varphi(t) &:= \mathfrak{c}(\varphi, t)
\end{aligned}
$$

$$
J(t_1, t_2, t_3, t_4, t_5, t_6) := \mathfrak{c}(J, t_1, t_2, t_3, t_4, t_5, t_6)
$$

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

$$
\begin{aligned}
|x| &= x & |f \bullet_\tau a| &= |f| \bullet_\tau |a| \\
|\star| &= \star & |[t_1, t_2, T]| &= |t_1| \\
|\Box| &= \Box & |t.1| &= |t| \\
|\lambda_0\, x\!:\!A.\, t| &= |t| & |t.2| &= |t| \\
|\lambda_\omega\, x\!:\!A.\, t| &= \lambda_\omega\, x.\, |t| & |x =_A y| &= |x| =_{|A|} |y| \\
|\lambda_\tau\, x\!:\!A.\, t| &= \lambda_\tau\, x\!:\!|A|.\, |t| & |\mathrm{refl}(t)| &= \lambda_\omega\, x.\, x \\
|(x : A) \to_m B| &= (x : |A|) \to_m |B| & |J(A, P, x, y, e, w)| &= |e| \bullet_\omega |w| \\
|(x : A) \cap B| &= (x : |A|) \cap |B| & |\vartheta(e)| &= |e| \\
|f \bullet_0 a| &= |f| & |\delta(e)| &= |e| \\
|f \bullet_\omega a| &= |f| \bullet_\omega |a| & |\varphi(a, f, e)| &= |a|
\end{aligned}
$$

Figure 2.2: Erasure of syntax, for type-like and kind-like syntax erasure is homomorphic, for term-like syntax erasure reduces to the untyped lambda calculus.

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

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

$$
\begin{aligned}
(\lambda_m\, x\!:\!A.\, b) \bullet_m t &\rightsquigarrow_\beta [x := t]b \\
[t_1, t_2, A].1 &\rightsquigarrow_\beta t_1 \\
[t_1, t_2, A].2 &\rightsquigarrow_\beta t_2 \\
J(A, P, x, y, \mathrm{refl}(z), w) &\rightsquigarrow_\beta w \bullet_0 z \\
\vartheta(\mathrm{refl}(t.1)) &\rightsquigarrow_\beta \mathrm{refl}(t) \\
\vartheta(\mathrm{refl}(t.2)) &\rightsquigarrow_\beta \mathrm{refl}(t) \\
\varphi(a, f, e).1 &\rightsquigarrow_\beta a
\end{aligned}
$$

Figure 2.3: Reduction rules for arbitrary syntax.

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

Figure 2.4: Domain and codomains for function types. The variable K is either $\star$ or $\square$.

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

$$\frac{\Gamma \vdash t \rhd A \qquad A \rightsquigarrow^*_\beta B}{\Gamma \vdash t \Rrightarrow B}\ \textsc{HdInf} \qquad \frac{\Gamma \vdash t \rhd A \qquad \Gamma \vdash B \Rrightarrow K \qquad A \equiv B}{\Gamma \vdash t \lhd B}\ \textsc{Chk}$$

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

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

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

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

Figure 2.5: Inference rules for function types, including erased functions. The variable K is either $\star$ or $\square$.

$$\frac{\Gamma \vdash A \Vdash\!\rhd \star \qquad \Gamma, x : A \vdash B \Vdash\!\rhd \star}{\Gamma \vdash (x : A) \cap B \rhd \star}\ \textsc{Int} \qquad \frac{\Gamma \vdash T \Vdash\!\rhd (x : A) \to_\tau B \qquad \Gamma \vdash t \lhd A \qquad \Gamma \vdash s \lhd [x := t]B \qquad |t| =_\beta |s|}{\Gamma \vdash [t, s; T] \rhd (x : A) \cap B}\ \textsc{Pair}$$

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

Figure 2.6: Inference rules for intersection types.

$$\frac{\Gamma \vdash A \Vdash\!\rhd \star \qquad \Gamma \vdash a \lhd A \qquad \Gamma \vdash b \lhd A}{\Gamma \vdash a =_A b \rhd \star}\ \textsc{Eq} \qquad \frac{\Gamma \vdash t \rhd A}{\Gamma \vdash \mathrm{refl}(t) \rhd t =_A t}\ \textsc{Refl}$$

$$\frac{\begin{array}{c}\Gamma \vdash A \Vdash\!\rhd \star \\ \Gamma \vdash P \lhd (x\ y : A) \to_\tau (e : x =_A y) \to_\tau \star \qquad \Gamma \vdash x \lhd A \qquad \Gamma \vdash y \lhd A \\ \Gamma \vdash e \lhd x =_A y \qquad \Gamma \vdash w \lhd (a : A) \to_0 P \bullet_\tau a \bullet_\tau a \bullet_\tau \mathrm{refl}(a)\end{array}}{\Gamma \vdash J(A, P, x, y, e, w) \rhd P \bullet_\tau x \bullet_\tau y \bullet_\tau e}\ \textsc{J}$$

$$\frac{\Gamma \vdash e \Vdash\!\rhd a.i =_T b.j \qquad \Gamma \vdash a \Vdash\!\rhd (x : A) \cap B \qquad \Gamma \vdash b \lhd (x : A) \cap B}{\Gamma \vdash \vartheta(e) \rhd a =_{(x:A)\cap B} b}\ \textsc{Prm}$$

$$\frac{\begin{array}{c}\Gamma \vdash f \Vdash\!\rhd (a : A) \to_\omega (x : A) \cap B \\ \Gamma \vdash a \lhd A \qquad \Gamma \vdash e \lhd (a : A) \to_\omega a =_A (f \bullet_\omega a).1 \qquad \mathrm{FV}(|e|) = \varnothing\end{array}}{\Gamma \vdash \varphi(a, f, e) \rhd (x : A) \cap B}\ \textsc{Cast}$$

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

Figure 2.7: Inference rules for equality types where $\mathrm{cBool} := (X : \star) \to_0 (x : X) \to_\omega (y : X) \to_\omega X$; $\mathrm{ctt} := \lambda_0 X : \star.\, \lambda_\omega x : X.\, \lambda_\omega y : X.\, x$; and $\mathrm{cff} := \lambda_0 X : \star.\, \lambda_\omega x : X.\, \lambda_\omega y : X.\, y$. Also, $i, j \in \{1, 2\}$

Chapter 3

PROOF NORMALIZATION AND RELATIONSHIP TO SYSTEM F^{ω}

Chapter 4

CONSISTENCY AND RELATIONSHIP TO CDLE

Chapter 5

OBJECT NORMALIZATION

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

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

Lemma 1 (Strictification Preserves Inference)**.** *Given* $\Gamma \vdash t \rhd A$ *then* $\Gamma \vdash s(t) \rhd A$

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

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

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

□

Lemma 2 (Strict Proofs are Normalizing)**.** *Given* $\Gamma \vdash t \rhd A$ *then* $s(t)$ *is strongly normalizing*

Proof. Direct consequence of strong normalization of proofs □

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

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

Figure 5.1: Strictification of a proof.

Lemma 3 (Strict Objects are Normalizing)**.** *Given* $\Gamma \vdash t \rhd A$ *then* $|s(t)|$ *is strongly normalizing*

Proof. Proof Idea:

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

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

Lemma 4. *A* φ_1*-proof,* p*, is contextually equivalent to its strictification,* $s(p)$

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

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

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

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

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

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

□

Lemma 5. *If t_1 is strongly normalizing and contextually equivalent to t_2 then t_2 is strongly normalizing*

Proof. Immediate by the definition of contextual equivalence. □

Theorem 3. *A φ_i-proof p is strongly normalizing for all i*

Proof. By induction on i.

Case: $i = 0$

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

Inductive Case:

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

□

Chapter 6

CEDILLE2: SYSTEM IMPLEMENTATION

Chapter 7

CEDILLE2: INTERNALLY DERIVABLE CONCEPTS

Chapter 8

CONCLUSION AND FUTURE WORK

Appendix A

PROOFS OF CHAPTER 1

Appendix B

UNDECIDED

Hello!

BIBLIOGRAPHY

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

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

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