Ground Stratification for a Logic of Definitions with Induction

Nathan Guermond
guerm001@umn.edu

Gopalan Nadathur
ngopalan@umn.edu

University of Minnesota Twin-Cities
Minnesota, USA

The logic underlying the Abella proof assistant includes mechanisms for interpreting atomic predicates through fixed point definitions that can additionally be treated inductively or co-inductively. However, the original formulation of the logic includes a strict stratification condition on definitions that is too restrictive for some applications such as those that use a logical relations based approach to semantic equivalence. Tiu has shown how this restriction can be eased by utilizing a weaker notion referred to as ground stratification. Tiu's results were limited to a version of the logic that does not treat inductive definitions. We show here that they can be extended to cover such definitions. While our results are obtained by using techniques that have been previously deployed in related ways in this context, their use is sensitive to the particular way in which we generalize the logic. In particular, although ground stratification may be used with arbitrary fixed-point definitions, we show that weakening stratification to this form for inductive definitions leads to inconsistency. The particular generalization we describe accords well with the way logical relations are used in practice. Our results are also a intermediate step to building a more flexible form for definitions into the full logic underlying Abella, which additionally includes co-induction, generic quantification, and a mechanism referred to as nominal abstraction for analyzing occurrences of objects in terms that are governed by generic quantifiers.

1 Introduction

This paper concerns a family of first-order predicate logics that originate from the work of McDowell and Miller [11] and that have culminated in the logic $\mathcal{G}$ that underlies the proof assistant Abella [7]. These logics endow a conventional predicate logic with the capability of treating predicate constants as defined symbols, to be interpreted via formulas that have been associated with them by a definition. More specifically, the definitions that are associated with predicate constants are given a fixed-point reading that can be further refined to correspond to the least or greatest fixed-point, thereby adding the capability of inductive and co-inductive reasoning to the logic. These logics have been especially useful in encoding and reasoning about object systems that are described in a rule-based and relational fashion: the rules in the object system description can be translated into definitions of predicate constants that represent the relevant relations and the treatment of definitions in the logic provides a transparent means for capturing the informal style of reasoning associated with rule-based specifications.

An important aspect of the logics of interest is that the forms of definitions must be sufficiently constrained to ensure consistency. The definition of a predicate constant can include a use of that constant itself, thereby supporting recursive specifications. However, a definition must not be circular in that it assumes its own existence in its construction. In logics of definitions, this requirement translates into restrictions on the negative occurrences of the predicate constant in the formula defining it. The condition

Chaudhuri, Nantes-Sobrinho (Eds.): International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2025)
EPTCS ??, 2025, pp. 1–17, doi:10.4204/EPTCS.??.??

imposed in the logic described in [11], which has carried over to the logic $\mathcal{G}$, is that the predicate constant must not appear to the left of a top-level implication symbol in that formula. Concretely, this condition takes the form of a *stratification* requirement: there must be an ordering on predicate constants that determines which of them have been already defined and can therefore be used negatively in the definition of another predicate constant. Unfortunately, the stratification restriction based on predicate names is too strong for some applications. A prime example of this is that of logical relations that are often used in reasoning about programming languages properties. The definitions of these relations are typically indexed by types and assume their own definition, albeit for structurally smaller types.

This work is part of the effort to allow for more permissive definitions so as to support the described applications. In past work [16], Tiu has developed the notion of *ground stratification* that effectively allows for the building in of the arguments of an atomic formula into the stratification ordering. The logic considered by Tiu limits definitions to a generic fixed-point variety with inductive reasoning being realized through natural number induction. The results in this paper are to be viewed in the context of Tiu's work and have a twofold character. First, we show that stronger conditions must be satisfied by definitions in the case of predicate constants that are to be treated inductively. In particular, we show that easing the restriction to ground stratification for these constants can lead to an inconsistent logic. Second, we show that if a stronger form of stratification for such constants is coupled with ground quantification for predicate constants that are interpreted via generic fixed-points, then the logic is consistent. Proofs of consistency for logics typically proceed by showing a property called cut-elimination for them. However, such a proof is elusive for our logic. Instead, we reduce consistency for it to consistency for a ground version, for which we show a cut-elimination result. In providing such a proof, we follow the lead of [16], using ideas from [15] in the additional treatment of induction.

The rest of the paper is organized as follows. In the next section we present the logic of study, describing in its context the particular mix of stratification conditions that are needed for consistency. Section 3 exposes the need for the restrictions imposed on the allowable definitions. Section 4 presents some examples that show, amongst other things, that the logic described is capable of encoding a reasoning style that is based on using logical relations. Section 5 sketches the proof of consistency for the logic.[1] We conclude the paper in Section 6 with a discussion of related and future work.

2 A Logic with Definitions and Induction

The logic that we consider in this paper, which we call LD^{μ}, has an intuitionistic, first-order version of Church's Simple Theory of Types [5] as its core. It extends this logic by allowing atomic predicates to be treated as defined symbols, to be interpreted via a collection of clauses that are given a fixed-point interpretation, which can be further refined to correspond to the least fixed-point; the restriction to the least-fixed point amounts to giving the clauses defining the predicate an inductive reading. Qualitatively, LD^{μ} is an extension of Tiu's LD with inductive definitions. It can also be viewed as a fragment of the logic $\mathcal{G}$ [7], which underlies the Abella proof assistant [3], that has been enriched with a more permissive form for definitions. In the subsections below, we outline this logic, focusing mainly on the notions of definitions and, more specifically, on how they extend what is permitted in $\mathcal{G}$.

[1] The complete development of these ideas can be found at the URL https://z.umn.edu/strat-proofs.

2.1 Syntax

The terms of LD$^\mu$ are based on the expressions in Church's Simple Theory of Types, in which we assume a finite set of base types $\iota_1, \ldots, \iota_n$, function types $\alpha \to \beta$, and a distinguished type $\mathbf{o}$ of propositions. We assume we are given a *signature* Σ, which is a set of type annotated constants of the form $\kappa : \tau$. Terms in LD$^\mu$ are then the well-typed terms in the simply typed lambda calculus that are constructed using the constants in the signature Σ and variables from a *variable context* $\mathcal{X}$, which is a finite set of distinct type annotated variables $x : \tau$. Since a term t only exists with respect to a signature Σ and a variable context $\mathcal{X}$, and since Σ is fixed, we will say that t is well formed with respect to $\mathcal{X}$, or that t *lies over* $\mathcal{X}$. In particular this means that $\mathcal{X}$ must contain all free variables in t, but may contain more. A term is *ground* if it lies over the empty variable context $\emptyset$. We denote the set of ground terms of type α by ground(α). We observe that the type of t is uniquely determined by Σ and $\mathcal{X}$, and thus we denote $\mathcal{X} \vdash_\Sigma t : \tau$ whenever t lies over $\mathcal{X}$. Finally, we note that terms are considered to be equal modulo α-, β-, and η-conversion and we shall represent them by their normal forms modulo these rules, which are known to exist.

We call a type a *first-order* type if it does not contain $\mathbf{o}$, and a *predicate* type if it is $\mathbf{o}$ or of the form $\tau \to \omega$ for a first order type τ and a predicate type ω. If $p : \omega$ belongs to Σ for a predicate type ω, then p is said to be a predicate constant or symbol. If ω is $\mathbf{o}$, then p may also be referred to as a propositional symbol. A *formula* is a term of type $\mathbf{o}$. We assume that Σ contains the *logical constants* $\bot$ and $\top$ of type $\mathbf{o}$, $\wedge$, $\vee$, and $\supset$ of type $\mathbf{o} \to \mathbf{o} \to \mathbf{o}$ and, for every first-order type α, $\forall_\alpha$ and $\exists_\alpha$ of type $(\alpha \to \mathbf{o}) \to \mathbf{o}$. *Atomic* formulas are of the form $p\,\vec{t}$ for some predicate symbol $p : \omega$ in Σ where in general, $u\,\vec{t}$ is an abbreviation for the application $(\ldots(u\,t_1)\ \ldots\ t_n)$. We abbreviate $Q_\alpha(\lambda x.C)$, where Q_α is $\forall_\alpha$ or $\exists_\alpha$, by $Qx : \alpha.C$. We will also follow the usual convention of writing $\wedge$, $\vee$, and $\supset$ in infix form in formulas in which they appear as the top-level logical symbol.

Given variable contexts $\mathcal{X}$ and $\mathcal{Y}$, a *substitution* $\theta = [t_1/x_1, \ldots, t_n/x_n]$ of type $\mathcal{Y} \to \mathcal{X}$ is an assignment of terms $t_1 : \tau_1, \ldots, t_n : \tau_n$ lying over $\mathcal{Y}$ to each variable $x_1 : \tau_1, \ldots, x_n : \tau_n = \mathcal{X}$. Such a substitution is said to be *for* $\mathcal{X}$, and its *range*, denoted by range(θ), is the smallest variable context containing all the free variables in $t_1, \ldots, t_n$. Moreover, it is said to be an $\mathcal{X}$*-grounding substitution* if $\mathcal{Y}$ is $\emptyset$. If t is a term lying over $\mathcal{X}$, the *application of* θ *to* t, written as $t\theta$, is the term $(\lambda x_1 \ldots \lambda x_n.t)\,\vec{t}$; this is evidently a term that lies over $\mathcal{Y}$. Given a variable context $\mathcal{X}$, we write $[t/x]_\mathcal{X}$ to denote the simultaneous substitution $[x_1/x_1, \ldots, x_n/x_n, t/x] : \mathcal{X} \to \mathcal{X}, x$, and we denote the trivial substitution by $\varepsilon_\mathcal{X} : \mathcal{X} \to \mathcal{X}$.

We assume a sequent style formulation of derivability for LD$^\mu$. In this context, a *sequent* is a judgment of the form $\mathcal{X}; \Gamma \vdash C$ for a finite multiset of formulas Γ and a formula C all of which lie over a finite variable context $\mathcal{X}$. We will assume all the types in $\mathcal{X}$ are first order. We will omit the type annotations in $\mathcal{X}$ when their specific identity is orthogonal to the discussion. The formulas in Γ are called the *context* or *assumption set* of the sequent, C is called its *conclusion* and the variables in $\mathcal{X}$ constitute its *eigenvariables*. At an intuitive level, the judgment represented by such a sequent is valid if, for every $\mathcal{X}$-grounding substitution θ, the validity of all the formulas in $\Gamma\theta$ implies the validity of $C\theta$.

2.2 Logical Rules

The main content of a sequent style presentation of a logic are the rules for deriving sequents. The rules for LD$^\mu$ are of three varieties: those that explicate the meaning of the logical symbols, those that deal with the structural aspects of sequents, and those that build in the treatment of definitions. We discuss rules of the first two kinds here, leaving the elaboration of definitions to the next two subsections.

The rules corresponding to the logical symbols are identical in content to the ones to be found in usual first-order logics. We assume familiarity with the ones that build in the meanings of the logical

$$\frac{\mathcal{X};\Gamma,C[t/x]_{\mathcal{X}} \vdash D \qquad \mathcal{X} \vdash_{\Sigma} t : \tau}{\mathcal{X};\Gamma,\forall x:\tau.C \vdash D}\,\forall\mathcal{L} \qquad \frac{\mathcal{X},x;\Gamma \vdash C}{\mathcal{X};\Gamma \vdash \forall x:\tau.C}\,\forall\mathcal{R}$$

$$\frac{\mathcal{X},x;\Gamma,C \vdash D}{\mathcal{X};\Gamma,\exists x:\tau.C \vdash D}\,\exists\mathcal{L} \qquad \frac{\mathcal{X};\Gamma \vdash C[t/x]_{\mathcal{X}} \qquad \mathcal{X} \vdash_{\Sigma} t : \tau}{\mathcal{X};\Gamma \vdash \exists x:\tau.C}\,\exists\mathcal{R}$$

Figure 1: Logical rules for quantifiers

$$\frac{\mathcal{X};\Delta_1 \vdash A_1 \qquad \ldots \qquad \mathcal{X};\Delta_n \vdash A_n \qquad \mathcal{X};\Gamma,A_1,\ldots,A_n \vdash C}{\mathcal{X};\Gamma,\Delta_1,\ldots,\Delta_n \vdash C}\,mc \qquad \frac{}{\mathcal{X};A \vdash A}\,\text{Ax} \quad A \text{ atomic}$$

Figure 2: The multicut and axiom rules

connectives. The quantifier rules are shown in Figure 1. Note that we write Γ,F in these rules to denote a multiset that comprises F and the formulas in Γ. Note also that, in the rules in Figure 1, we assume that the variable x bound by the quantifier to be distinct from all the variable in $\mathcal{X}$, a requirement that can always be established by renaming the bound variable. The structural rules include the usual complement: contraction, which builds in the treatment of multisets as sets, weakening, which allows us to add an extraneous assumption, an axiom rule which allows us to match an atomic conclusion with an assumption set comprising just that formula, and the cut rule that underlies the use of lemmas. We use a particular version of the cut rule called *multicut* that is better suited to cut-elimination and consistency arguments. The axiom rule and the multicut rule are shown in Figure 2.

If an inference rule is a right introduction rule, all of its premises are called *major* premises. If it is a left introduction rule or a multicut, then only those premises with the same consequent as its conclusion are called major premises. All other premises are called *minor* premises.

2.3 Treating Predicate Constants as Defined Symbols

LD$^{\mu}$ deviates from a vanilla first-order logic in that it allows atomic predicates to be further analyzed through a *definition* $\mathcal{D}$ that parameterizes the logic. In this context, a definition is a set of clauses of the form $p\,\vec{t} \stackrel{\Delta}{=}_{\mathcal{X}} B$ for which there exists a predicate constant $p : \omega$ in the signature Σ and a formula B, with $\vec{t}$ and B both lying over $\mathcal{X}$. For a given clause $H \stackrel{\Delta}{=}_{\mathcal{X}} B$, we say that H is the *head* of the clause, and B the *body*. Similarly to [16], we require every variable in $\mathcal{X}$ to appear in H, and H to lie in the higher-order pattern fragment that is described, *e.g.*, in [12].

Definitions provide a natural means for encoding rule-based relational specifications in LD$^{\mu}$. For example consider the specification of the *append* relation over lists that are constructed using the constants *nil*, that represents the empty list, and *cons*, that represents the construction of a new list by adding an element to an already existing list. A typical specification of this relation comprises the following rules:

$$\frac{}{\mathit{append}\ \mathit{nil}\ K\ K} \qquad \frac{\mathit{append}\ L\ K\ M}{\mathit{append}\ (\mathit{cons}\ X\ L)\ K\ (\mathit{cons}\ X\ M)}$$

Such a specification can be rendered into a definition through the following clauses in LD$^{\mu}$:

$$\begin{array}{c} \mathit{append}\ \mathit{nil}\ K\ K \stackrel{\Delta}{=}_{K} \top \\ \mathit{append}\ (\mathit{cons}\ X\ L)\ K\ (\mathit{cons}\ X\ M) \stackrel{\Delta}{=}_{X,L,K,M} \mathit{append}\ L\ K\ M \end{array} \tag{1}$$

Of course, such a rendition is useful only if it is supplemented with a means for reflecting the natural style of reasoning associated with rule-based specifications into LD$^{\mu}$. There are, in general, two forms in which the rules might be used in informal reasoning. First, they may be used to construct derivations for particular relations; for example, in this particular instance, they may be used to show that the relation *append* (*cons a nil*) (*cons b nil*) (*cons a* (*cons b nil*)) holds for particular constants a and b. Second, when we are given as an assumption that a particular relation holds, they can figure in a case analysis style of reasoning. Thus, in this particular instance, they can be used to show that the formula (*append* (*cons a nil*) *nil nil*) $\supset \bot$ must hold because there cannot be a derivation for its antecedent.

$$\frac{\{\mathrm{range}(\theta);\Gamma\theta,B' \vdash C\theta \mid \mathrm{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B,A,\theta,B'),H \stackrel{\Delta}{=}_{\mathcal{X}} B \in \mathcal{D}\}}{\mathcal{Y};\Gamma,A \vdash C}\ \Delta\mathcal{L}$$

$$\frac{\mathcal{Y};\Gamma \vdash B'}{\mathcal{Y};\Gamma \vdash A}\ \Delta\mathcal{R} \qquad \mathrm{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B,A,\varepsilon_{\mathcal{Y}},B') \text{ for } H \stackrel{\Delta}{=}_{\mathcal{X}} B \in \mathcal{D}$$

Figure 3: Definition rules for a predicate p, provided $A = p\,\vec{t}$

These two forms of reasoning are built into LD$^{\mu}$ by rules for introducing atomic predicates into the left and right of a sequent based on the definition $\mathcal{D}$ that parameterizes the logic. These rules are shown in Figure 3. In these rules, we use the notation $\mathrm{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B,A,\theta,B')$ to signify that there exists a substitution $\rho : \mathcal{Z} \to \mathcal{X}$, for some $\mathcal{Z}$, such that $H\rho = A\theta$ and $B' = B\rho$ Conceptually, $\Delta\mathcal{R}$ corresponds to unfolding a definition for a particular instance A of the head H, whereas $\Delta\mathcal{L}$ corresponds to case analysis on all possible instances of A matching with the head H. The reader may confirm that these rules can actually be used to establish the two formulas considered above.

Not all definitions are permissible in LD$^{\mu}$. To explain what definitions are allowed, we must describe a process for assigning an ordinal to each ground formula F that is called its *level* and is designated by $\mathrm{lvl}(F)$. This process assumes an assignment for each ground atomic formula and extends it to arbitrary formulas based on the following rules:

$$\begin{array}{l} \mathrm{lvl}(\bot) := \mathrm{lvl}(\top) := 0 \\ \mathrm{lvl}(A \wedge B) := \mathrm{lvl}(A \vee B) := \max(\mathrm{lvl}(A),\mathrm{lvl}(B)) \\ \mathrm{lvl}(A \supset B) := \max(\mathrm{lvl}(A)+1,\mathrm{lvl}(B)) \\ \mathrm{lvl}(\forall x : \alpha.C) := \mathrm{lvl}(\exists x : \alpha.C) := \sup\{\mathrm{lvl}(C[t/x]_{\emptyset}) \mid t \in \mathrm{ground}(\alpha)\} \end{array}$$

We then say that a definition $\mathcal{D}$ is *ground stratified* if there exists a level assignment to ground atomic formulas such that for every clause $H \stackrel{\Delta}{=}_{\mathcal{X}} B \in \mathcal{D}$, and for every $\mathcal{X}$-grounding substitution ρ, it is the case that $\mathrm{lvl}(H\rho) \geq \mathrm{lvl}(B\rho)$. The requirement of definitions in LD$^{\mu}$, then, is that they must be ground stratified.

There is a stronger version of stratification that is also of interest and that we identify as *strict stratification*. In this version, we require the definition to be ground stratified under a level assignment to

ground atomic formulas that depends only on their predicate head. In other words it must be the case that $\mathrm{lvl}(p\ \vec{t}_1) = \mathrm{lvl}(p\ \vec{t}_2)$ for any sequence of ground terms $\vec{t}_1$ and $\vec{t}_2$. It is actually this less permissive version of stratification that governs definitions in the logic $\mathcal{G}$.

2.4 Inductive Definitions and Fixed-Point Operators

Definitions as we have considered them thus far may contain only one kind of clause: those of the form $H \stackrel{\Delta}{=}_{\mathcal{X}} B$. We now introduce the possibility of including a different kind of clause in definitions, ones that are written as $H \stackrel{\mu}{=}_{\mathcal{X}} B$. We refer to these as *inductive clauses* in contrast to the previously described ones that we distinguish as *fixed-point clauses*. The structural restrictions on inductive clauses parallel those on fixed-point clauses: all the variables in $\mathcal{X}$ must appear in H and H must lie in the higher-order pattern fragment. The predicate symbols in Σ are categorized as inductive or fixed-point predicates. The definition that parameterizes the logic can be a mixture of the two kinds of clauses with the proviso that the clauses defining an inductive predicate must all be of the inductive variety and, similarly, those for the fixed-point predicates must be of the fixed-point variety.

Definitions in this mixed form must once again satisfy a stratification condition. The condition is, in the first instance, similar to that described earlier: the definition must be ground stratified under a level assignment to ground atomic formulas. This initial assignment is permitted to be arbitrary for ground atomic formulas that have a fixed-point predicate as the top-level predicate symbol. However, the requirement is stricter when the top-level predicate is an inductive one. In this case, the level assignment must be independent of the arguments of the predicate, *i.e.*, the conditions for strict stratification must be satisfied in these cases.

The conceptual difference between fixed-point and inductive clauses is that the latter are intended to identify a *least* fixed-point which thereby entails stronger properties for the predicate. At the proof-theoretic level, this more refined view of the predicate definition is realized by a special introduction rule for assumption formulas that have an inductively defined predicate as their top-level predicate symbol. To present this rule we must first render a multi-clause definition of an inductive predicate p into a single clause that has the form $p\ \vec{x} \stackrel{\mu}{=}_{\vec{x}} B\ p\ \vec{x}$, where $\vec{x}$ is a sequence of distinct variables and B is a closed term not containing p; B is referred to as a *fixed-point operator* in such a definition. This translation makes use of a special predicate eq that is assumed to be defined by the sole clause $eq\ X\ X \stackrel{\Delta}{=}_X \top$. More specifically, if the clauses for p are the following

$$p\ \vec{t}_1 \stackrel{\mu}{=}_{\mathcal{X}_1} B_1 \quad \ldots \quad p\ \vec{t}_n \stackrel{\mu}{=}_{\mathcal{X}_n} B_n$$

then the fixed-point operator in the single clause definition of p would be

$$\begin{aligned}\lambda p.\lambda \vec{x}.(\exists \mathcal{X}_1.(eq\ x_1\ t_1^1) \wedge \ldots \wedge (eq\ x_k\ t_1^k) \wedge B_1) \vee \ldots \vee \\ (\exists \mathcal{X}_n.(eq\ x_1\ t_n^1) \wedge \ldots \wedge (eq\ x_k\ t_n^k) \wedge B_n)\end{aligned}$$

where $\vec{t}_i = t_i^1, \ldots, t_i^k$ for each i, with t_i^j lying over $\mathcal{X}_i$.

To provide a concrete example of this translation, we may consider the clauses shown in the display labelled 1 that define the $append$ predicate. Those clauses were originally shown to be of the fixed-point variety, but we will now assume that their annotation and designation has been changed to that of inductive clauses. Letting B be the fixed-point operator

$$\begin{aligned}\lambda p.\lambda \ell.\lambda k.\lambda m.((eq\ \ell\ nil) \wedge (eq\ k\ m)) \vee \\ (\exists x, \ell', m'.(eq\ \ell\ (cons\ x\ \ell')) \wedge (eq\ m\ (cons\ x\ m')) \wedge (p\ \ell'\ k\ m')),\end{aligned}$$

this definition would be transformed into $\mathit{append}\ L\ K\ M \stackrel{\mu}{=}_{L,K,M} ((B\ \mathit{append})\ L\ K\ M)$.

$$\frac{\vec{x};B\ S\ \vec{x} \vdash S\ \vec{x} \qquad \mathcal{X};\Gamma, S\ \vec{t} \vdash C}{\mathcal{X};\Gamma, p\ \vec{t} \vdash C}\ \mu\mathcal{L} \qquad\qquad \frac{\mathcal{X};\Gamma \vdash B\ p\ \vec{t}}{\mathcal{X};\Gamma \vdash p\ \vec{t}}\ \mu\mathcal{R}$$

Figure 4: Rules for introducing $p\ \vec{t}$ after converting the clauses for p into the form $p\ \vec{x} \stackrel{\mu}{=}_{\vec{x}} B\ p\ \vec{x}$

Assuming the translation that we have just described, the rules for introducing atomic formulas that have an inductively defined predicate as their top-level predicate symbol are shown in Figure 4. The symbol S that appears in the left premise sequent in the $\mu\mathcal{L}$ rule is required to be instantiated with a closed term of the same type as the predicate p. The particular term that is chosen for S in a use of this rule is referred to as *inductive invariant.*

It is easily seen that the $\mu\mathcal{R}$ rule, *i.e.*, the rule for introducing an inductively defined atomic formula in the conclusion of a sequent, is no different from the rule for doing so when the formula is defined as simple fixed-point. However, the $\mu\mathcal{L}$ rule adds deductive power to the calculus beyond what is available from just fixed-point definitions. To substantiate this observation, consider showing the following statement, which establishes that append is a functional relation

$$\forall \ell, k, m, m'.(\mathit{append}\ \ell\ k\ m) \wedge (\mathit{append}\ \ell\ k\ m') \supset (\mathit{eq}\ m\ m').$$

This property cannot be proved if append is defined via clauses that are given just a fixed-point interpretation. However, if we view the clauses for append as inductive ones instead, then the property can be proved via the $\mu\mathcal{L}$ rule, using

$$\lambda \ell.\lambda k.\lambda m.\forall m'.(\mathit{append}\ \ell\ k\ m) \wedge (\mathit{append}\ \ell\ k\ m') \supset (\mathit{eq}\ m\ m')$$

as the inductive invariant.

3 Stratification and Consistency

In Section 2, we imposed some conditions on the forms of fixed-point and inductive definitions. In particular, we required the first to be ground stratified and the second to be stratified based on just the names of predicates. In this section, we discuss the purpose of these restrictions.

The need for some kind of stratification for the coherence of definitions should not be difficult to appreciate. As an example of a definition that might be problematic, consider one that has $p \stackrel{\Delta}{=} p \supset \bot$ as the sole clause for a propositional symbol p. A clause of this kind has the intuitive content of assuming a definition of p in the course of constructing one for p. To understand this, consider how we might proceed to prove p. We might unfold this by virtue of the definition into trying to prove $p \supset \bot$, which could be done by proving the sequent $\cdot; p \vdash \bot$. Since p appears as a hypothesis in this sequent, we are effectively assuming that p is already a defined propositional symbol.

The intuitive observation above can be given actual substance by showing that the sequent $\cdot;\cdot \vdash \bot$ can be derived if we allow $p \stackrel{\Delta}{=} p \supset \bot$ to be the sole clause for p in a definition; since there is a derivation for any sequent in which $\bot$ appears in the assumption set, it follows easily from this that a logic that permits such a definition is inconsistent. The crux of the argument is to show that the sequent $\cdot; p \vdash \bot$ would have a derivation in this logic; from this it follows easily that $\cdot;\cdot \vdash p \supset \bot$ and, therefore, $\cdot;\cdot \vdash p$

have derivations and then, using the multicut rule, we may derive $\cdot;\cdot \vdash \bot$. To see that $\cdot; p \vdash \bot$ has a derivation, we observe that it would have one if $\cdot; p, p \vdash \bot$ is derivable. A derivation of the last sequent is easily obtained by using the clause defining p with one of the two occurrences of p in the assumption set of the sequent.

The ground stratification condition that we have described disallows clauses of the kind just discussed in definitions. For such a clause to be allowable, we would need to be able to assign a level to p that would have the property of being greater than or equal to the level of $p \supset \bot$. However, the definition of levels for formulas makes this impossible to do.

In the example considered, ground stratification boils down to a form of stratification that uses only the predicate name; this is because the clause in question is for a predicate with no arguments. When we consider predicates with arguments, the two notions become distinct. In such a case, we may "build" the arguments of the predicate into its name. Of course, this is only possible for ground instances of predicates, but that is sufficient for the coherence of the idea: specifically, we can identify the well-formedness of a definitional clause with the ability to describe a stratification ordering on all its instances. This is the idea that is reflected in ground stratification. As a concrete example, consider a definition that is given by the following clauses:

$$ev\ z \stackrel{\Delta}{=} \top \qquad\qquad ev\ (s\ X) \stackrel{\Delta}{=}_X ev\ X \supset \bot$$

In these clauses, z and s are to be understood to be constants of a designated type *nat* and X should be read as a variable of type *nat*; intuitively, ev, which is a predicate of type $nat \to \mathbf{o}$, identifies the even natural numbers, which are represented by terms constructed using the constant z (representing the numeral 0) and the function symbol s (representing the successor function). Now, if we were to consider an ordering of ground atomic formulas of the form $ev\ t$ that "forgets" the argument, the displayed clauses would not satisfy the stratification requirement. However, if we allow the chosen ordering to also take into account the complexity of the argument, as would be the case if the measure associated with $ev\ t$ is based solely on the complexity of t as a term, then it is easily seen that the definition will satisfy the required condition.

In the above example, the clauses for ev are assumed to provide a fixed-point definition for the predicate. The requirements we have described in Section 2 do not allow them to be treated inductively: for such an interpretation, the relevant definitional clauses must satisfy the stronger condition of being stratified based on an ordering that uses only the predicate name for ground formulas of the form $ev\ t$. A natural question to ask is if this requirement on inductive predicates can be weakened to allow the ordering to depend on the arguments. What we observe below is that such a weakening is not possible because it can render the logic inconsistent.

If we were to weaken the requirement in the way described, it would allow for the definition of the predicate odd through the following clause:

$$odd\ (s\ X) \stackrel{\mu}{=}_X odd\ X \supset \bot$$

Recast into a form based on a fixed-point operator, this definition would correspond to one of the form $odd\ X \stackrel{\mu}{=}_X ((B\ odd)\ X)$ where B is the term $\lambda p.\lambda x.\exists y.(eq\ x\ (s\ y) \wedge (p\ y \supset \bot))$. A crucial observation about this operator is that does not impose the requirement that the predicate it applies to also holds of $s\ z$, *i.e.*, of the representation of the numeral 1. Given the definition of ev, it is therefore not surprising that both $\lambda x.\, ev\ x$ and $\lambda x.\,(ev\ x \supset \bot)$ describe fixed points by virtue of it. Concretely, it is easily seen that if we pick S to be either of these predicates then the sequent $x; B\ S\ x \vdash S\ x$ is derivable. Using the induction rule, it then follows that, for any term t, we can construct derivations for $\mathcal{X}; odd\ t \vdash ev\ t$ and $\mathcal{X}; odd\ t \vdash ev\ t \supset \bot$ and therefore, by the multicut rule, a derivation for $\mathcal{X}; odd\ t \vdash \bot$. As particular

instances of this observation, we see that the sequents $\cdot; odd\ z \vdash \bot$ and $\cdot; odd\ (s\ z) \vdash \bot$ have derivations. The former implies that $\cdot; \cdot \vdash odd\ z \supset \bot$ has a derivation and therefore, by the clause defining odd, that $\cdot; \cdot \vdash odd\ (s\ z)$ has a derivation. Since both $\cdot; \cdot \vdash odd\ (s\ z)$ and $\cdot; odd\ (s\ z) \vdash \bot$ have derivations, it follows by the multicut rule that $\cdot; \cdot \vdash \bot$ must also be derivable.

In the positive direction, the results in this paper show that the restrictions we have described suffice to yield a consistent logic. These results strengthen those in [16] by adding a treatment of inductive definitions to the logic. A question to ask is if the conditions on inductive clauses are not too restrictive in practice. We argue not. In particular, we believe that the weaker form of stratification suffice for general fixed-point definitions and that inductive reasoning can generally be based on definitions that satisfy stronger conditions. In the present context, for example, inductive reasoning can be based on a definition of a predicate that characterizes natural numbers—which is easily seen to be stratifiable under an ordering for atomic formulas that uses only the predicate name—rather than the definitions of ev or odd.

4 Applications for the More Permissive Form for Definitions

The main difference between the definitions permitted in LD^{μ} and in $\mathcal{G}$ is in the use of ground stratification as opposed to strict stratification to determine well-formedness. Several examples have been presented in [16] to illustrate the benefits of the added richness. These examples have included the encoding of the Gödel-Gentzen translation and the Kolmogorov double negation translation of classical logic in intuitionistic logic. Another important class of applications concerns the encoding of logical relations, which are commonly used to reason about properties of programs and programming languages. For example, using the more permissible form of definitions in LD^{μ}, we can encode logical equivalence of programs that asserts that two functions are equivalent if their behavior on equivalent arguments is the same [9]. A more elaborate use of logical relations in this kind of reasoning may be found in [17], where a step-indexed logical equivalence relation is used to prove correctness of compiler transformations for functional programs. Although a different kind of extension to $\mathcal{G}$, one based on adding rewriting and using that to encode definitions [4], was used to validate this application, the use of ground stratification provides an alternative justification that retains the flavor of definitions as originally described by McDowell and Miller.

Another common use of logical relations is in the proof of strong normalizability results in the style of Tait [14]. These arguments are based on the identification of reducibility relations [8], whose encoding in a logic of definitions requires the idea of ground stratification. We will illustrate this application in more detail below. We remark that logical relations arguments are frequently used in conjunction with inductive arguments. We will therefore observe that the pitfall discussed in the previous section can be avoided, since the logical relations in question need not be defined inductively.

We will specifically show how we can encode the reducibility predicate commonly used to prove strong normalization for the simply typed lambda calculus (STLC) using ground stratification. To do this, we introduce a base type ty to encode types in STLC, which may be constructed from the following constant symbols

$$unit : ty \qquad\qquad arrow : ty \to ty \to ty$$

as well as a type tm to encode terms, which may be constructed from the following constant symbols

$$star : tm \qquad\qquad lam : (tm \to tm) \to tm \qquad\qquad app : tm \to tm \to tm$$

where *star* is the unique term of type *unit*. We then assume that we have already defined a predicate $step : tm \to tm \to \mathbf{o}$ specifying a reduction relation on terms, which we use to inductively define the strong normalizability predicate $sn : tm \to \mathbf{o}$ as follows

$$sn\ T \stackrel{\mu}{=} \forall u : tm.(step\ T\ u \supset sn\ u)$$

Using this, we can now define the reducibility predicate $red : ty \to tm \to \mathbf{o}$ as follows

$$red\ unit\ T \stackrel{\Delta}{=} sn\ T$$
$$red\ (arrow\ A\ B)\ T \stackrel{\Delta}{=} \forall u : tm.(red\ A\ u \supset red\ B\ (app\ T\ u))$$

Note that this predicate cannot be strictly stratified because *red* appears negatively in the body of its definition. However, we notice that in the instance *red A u* appearing negatively in the body, the type argument *A* is a subterm of the type argument (*arrow A B*) appearing in the head of the definition. But if we think of *red* as being a definition indexed by its first type argument, then we can think of *red A* as having already been defined when *red* (*arrow A B*) is being defined, suggesting the definition be acceptable. Indeed, this definition is ground stratified because for any ground terms *a* and *b* of type *ty*, the term *a* is strictly smaller than the term (*arrow a b*), and thus our definition is valid.

One may wonder whether this definition suffices for the usual reducibility argument to go through. Indeed, it had been assumed in a previous development that it was safe to do induction directly on the reducibility predicate [1], which violates the strict stratification condition we have now imposed on inductive definitions. The usual argument (see eg. [8]), however, depends on induction on the *type* argument, and not on reducibility itself. We can therefore model this mode of reasoning by defining a predicate $type : ty \to \mathbf{o}$ inductively as follows

$$type\ unit \stackrel{\mu}{=} \top$$
$$type\ (arrow\ A\ B) \stackrel{\mu}{=} (type\ A) \wedge (type\ B)$$

and then proceed by induction on the type, rather than on reducibility. We note that the *type* predicate does not appear negatively in the body of these clauses and hence these clauses satisfy the stratification condition under a level assignment in which the level of an atomic predicate of the form (*type t*) is determined solely by the constant *type*.

5 Proving Consistency for the Logic

In this section, we sketch the proof of consistency for the logic LD^{μ}. The most common method for doing this is to show that there cannot be a derivation for $\bot$ in the logic. In a sequent style formulation, it is usually easy to show this if proofs are limited to those that do not use the cut rule. Thus, a cut-elimination result for the logic comes in handy. However, as with the logic LD described by Tiu, a proof of cut-elimination for LD^{μ} has been elusive. Following Tiu, we therefore use an indirect technique. We first describe a ground logic $\mathrm{LD}^{\mu}_{\infty}$ through a sequent calculus that follows the structure of the one for LD^{μ} except that it is specialized to proving ground sequents. We then show that cut-elimination holds for $\mathrm{LD}^{\mu}_{\infty}$ and use this to conclude that there is no derivation for $\bot$ in this logic. Finally, we produce an interpretation of LD^{μ} into $\mathrm{LD}^{\mu}_{\infty}$, which allows us to conclude that there cannot be a derivation for $\bot$ in LD^{μ} either. The subsections below elaborate on these three steps in the proof.

$$\frac{\{\Gamma \vdash C[t/x]_\emptyset\}_{t \in \text{ground}(\alpha)}}{\Gamma \vdash \forall x : \alpha.C}\ \forall\mathcal{R} \qquad \frac{\{\Gamma, B[t/x]_\emptyset \vdash C\}_{t \in \text{ground}(\alpha)}}{\Gamma, \exists x : \alpha.B \vdash C}\ \exists\mathcal{L}$$

Figure 5: $\forall\mathcal{R}$ and $\exists\mathcal{L}$ rules in LD^{μ}_{∞}

5.1 The Ground Logic

We now define a ground version of LD^{μ} which we call LD^{μ}_{∞}. The formulas in LD^{μ}_{∞} are the formulas in LD^{μ} which are well formed over the empty variable context. Since all formulas are ground, we restrict the sequent $\mathcal{X}; \Gamma \vdash C$ to have no eigenvariables, which translates to requiring that $\mathcal{X} = \emptyset$. We therefore denote a ground sequent by $\Gamma \vdash C$. The rules of LD^{μ}_{∞} are the same as those of LD^{μ} except for $\forall\mathcal{R}$, $\exists\mathcal{L}$, and $\mu\mathcal{L}$. We replace the $\forall\mathcal{R}$ and $\exists\mathcal{L}$ rules with the ground infinitary rules in Figure 5.

$$\frac{\{\Gamma, B' \vdash C \mid \text{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B, A, \varepsilon_\emptyset, B')\}}{\Gamma, A \vdash C}\ \Delta\mathcal{L} \qquad \frac{\Gamma \vdash B'}{\Gamma \vdash A}\ \Delta\mathcal{R} \quad \text{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B, A, \varepsilon_\emptyset, B')$$

Figure 6: Definition rules in LD^{μ}_{∞} for a predicate p, provided $A = p\,\vec{t}$

The definition $\Delta\mathcal{L}$ and $\Delta\mathcal{R}$ rules in LD^{μ}_{∞} are the same as those of LD^{μ}. The restriction to ground sequents allows us to present these rules as shown in Figure 6. However, we must check that these rules are well-formed, which amounts to checking that the premises are well-formed over the empty variable context. In other words, assuming that the conclusion of the rule is ground, we must check that all the formulas in each of the premises are ground. Suppose we are given a ground context Γ, a ground formula A, and a clause $H \stackrel{\Delta}{=}_{\mathcal{X}} B$ such that $\text{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B, A, \varepsilon_\emptyset, B')$, then there exists $\rho : \mathcal{Z} \to \mathcal{X}$ such that $A = A\varepsilon = H\rho$ and $B\rho = B'$. Since H lies in the higher-order pattern fragment and A is ground, this implies that ρ is unique and furthermore that it must be ground. Since B lies over $\mathcal{X}$, which is the domain of ρ, it follows that B' must be ground, as desired.

$$\frac{\left\{B\,S\,\vec{t} \vdash S\,\vec{t}\right\}_{\vec{t} \in \text{ground}(\vec{\alpha})} \qquad \Gamma, S\,\vec{u} \vdash C}{\Gamma, p\,\vec{u} \vdash C}\ \mu\mathcal{L}$$

Figure 7: $\mu\mathcal{L}$ rule in LD^{μ}_{∞} for $p\,\vec{x} \stackrel{\mu}{=}_{\vec{x}} B\,p\,\vec{x}$ and inductive invariant S

Finally, for any inductive definition with fixed-point form $p\,\vec{x} \stackrel{\mu}{=}_{\vec{x}} B\,p\,\vec{x}$ and inductive invariant S, we introduce a ground infinitary version of the $\mu\mathcal{L}$ rule given in Figure 7.

5.2 Consistency of the Ground Logic

We now show that LD^{μ}_{∞} is consistent by proving the cut-elimination theorem for ground derivations. To do so, we first define a *reduction* relation, which specifies how the multicut rule may be propagated upwards in a derivation tree. Then, we introduce the notion of normalizable derivations, which allows

us to show that the cut reduction relation is well-founded, and from which we can establish the cut-elimination theorem. To prove that every derivation is normalizable, we require the intermediate notion of a reducible derivation. Every derivation is then shown to be reducible via the reducibility technique, inspired by Tait and Martin-Löf (see [10, 14]), and adapted to the current context by McDowell and Miller in [11]. Our proof closely follows that of [16] and [11], together with the treatment of induction from [13, 15]. We outline the essential aspects of these steps below.

5.2.1 Cut reductions

We specify the reduction relation between derivations, following closely the reduction relation in [11]. The *redex*, that is the derivation to be reduced, is always a derivation Ξ ending with the multicut rule

$$\dfrac{\begin{array}{ccccc}\Pi_1 & & \Pi_n & & \Pi \\ \Delta_1 \vdash B_1 & \cdots & \Delta_n \vdash B_n & & B_1,\ldots,B_n,\Gamma \vdash C\end{array}}{\Delta_1,\ldots,\Delta_n,\Gamma \vdash C}\ mc$$

We refer to the formulas $B_1,\ldots,B_n$ in the multicut as *cut formulas*. If a left or right rule introduces a cut formula, we say the rule is *principal*. If Π reduces to Π', we say that Π' is a *reduct* of Π.

The reduction relation relates a derivation Ξ ending with the multicut rule to a new derivation Ξ' which may again end with a multicut but is always of smaller complexity. The reduction relation is specified by case analysis on the last rule of Π. If Π is principal, then we group the reductions by case analysis on the derivation Π_i ending with the cut formula. These are *essential cases* whenever Π_i is also principal (but not $\mu\mathcal{R}$), *left-commutative cases* whenever Π_i is not principal, *inductive cases* whenever Π ends with $\mu\mathcal{L}$, *structural cases* whenever Π ends with a weakening or contraction, *left axiom cases* whenever Π_i is an axiom, and *left multicut case* whenever Π_i ends with a multicut. Otherwise, Π is either an axiom (*right axiom case*), ends with a multicut (*right multicut case*), or ends with a non-principal rule (*right-commutative cases*). We describe the inductive case in more detail below. The full reduction relation is presented in a longer version of this paper.

Inductive case: An inductive case occurs when Π ends with $\mu\mathcal{L}$ introducing the cut formula $p\ \vec{t}$ for an inductive definition in fixed-point form $p\ \vec{x} \stackrel{\mu}{=}_{\vec{x}} B\ p\ \vec{x}$. Suppose the cut formula being introduced is $B_1 = p\ \vec{t}$. In this case, Π_1 is a derivation of the sequent $\Delta_1 \vdash p\ \vec{t}$ and Π is the following derivation for some inductive invariant S

$$\dfrac{\left\{\begin{array}{c}\Pi_S^{\vec{u}} \\ B\ S\ \vec{u} \vdash S\ \vec{u}\end{array}\right\}_{\vec{u}\in\text{ground}(\vec{\alpha})} \qquad \begin{array}{c}\Pi' \\ S\ \vec{t},B_2,\ldots,B_n,\Gamma \vdash C\end{array}}{p\ \vec{t},B_2,\ldots,B_n,\Gamma \vdash C}\ \mu\mathcal{L}$$

The key idea is that since we know that S is an inductive invariant, or pre-fixed-point of B, and p is a least fixed-point of B, then any time $p\ \vec{t}$ is provable in some context, $S\ \vec{t}$ should also be provable in the same context. More specifically, the family of derivations $\{\Pi_S^{\vec{u}}\}_{\vec{u}}$, which we abbreviate by Π_S, may be used to obtain a derivation of $\Delta_1 \vdash S\ \vec{t}$. We call this derivation the *unfolding* of the derivation Π_1 with respect to the family Π_S, which is captured in the following lemma, and which we denote by $\mu(\Pi_1,\Pi_S)$.

Lemma 1 (Unfolding lemma)**.** *Suppose $p\ \vec{x} \stackrel{\mu}{=} B\ p\ \vec{x}$ is the fixed-point form of an inductive definition, then for any derivation Ψ of $\Delta \vdash D\ p$ where p does not occur in D and occurs only positively in $D\ p$ (i.e., does not occur to the left of an implication), there exists a derivation $\mu(\Psi,\Pi_S)$ of $\Delta \vdash D\ S$.*

The proof of this lemma depends crucially on the inductive definition p to be strictly stratified. Using the unfolding operation, we can now reduce the redex Ξ above to the following

$$\dfrac{\begin{array}{ccc}\mu(\Pi_1,\Pi_S) & & \Pi' \\ \Delta_1 \vdash S\,\vec{t} & \cdots & S\,\vec{t}, B_2, \ldots, B_n, \Gamma \vdash C\end{array}}{\Delta_1, \ldots, \Delta_n, \Gamma \vdash C}\, mc$$

5.2.2 Cut Elimination

Our aim is now to show how the cut reduction relation is well-founded, and can therefore be used to eliminate cuts from a derivation. We start with the following inductive definition, by which we mean the smallest set of derivations closed under the specified operation (see definition 1.1.1 in [2] for a precise definition).

Definition 1 (normalizability)**.** *The set of normalizable derivations in LD^{μ}_{∞} is inductively defined as follows:*

1. *if Π ends with a multicut, then Π is normalizable if every reduct Π' is normalizable*
2. *otherwise, Π is normalizable if each of its premise derivations is normalizable.*

The goal is to show that every derivation in $\mathrm{LD}^{\mu}_{\infty}$ is normalizable. To do this we need the intermediate notion of γ-reducibility, which is defined by transfinite recursion on the level γ of a derivation, which we recall is an ordinal. The *level* of a derivation of a ground sequent $\Gamma \vdash C$ is defined to be $\mathrm{lvl}(C)$. Note therefore that if Π reduces to Π', then $\mathrm{lvl}(\Pi) = \mathrm{lvl}(\Pi')$. Furthermore, note that levels are defined so that every rule has a non-increasing consequent in each of its major premises. In the case of the $\Delta\mathcal{R}$ and $\mu\mathcal{R}$ rules, this condition is guaranteed by the ground stratification condition. In the case of $\Delta\mathcal{L}$, this condition is guaranteed because the consequent is ground. Finally we note that below, γ-reducibility of a derivation ending with the $\supset\mathcal{R}$ rule depends on α-reducibility to already be defined for any $\alpha < \gamma$. Together, these observations guarantee that the following is well defined.

Definition 2 (reducibility)**.** *Define γ-reducibility of a derivation Π inductively as follows:*

1. *if Π ends with a multicut, then Π is γ-reducible if for every reduct Π', Π' is γ-reducible.*
2. *if Π is the derivation*

$$\dfrac{\begin{array}{c}\Pi' \\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash A \supset B}\, \supset\mathcal{R},$$

   then Π is γ-reducible if $\mathrm{lvl}(\Pi) \leq \gamma$, Π' is γ-reducible, and for every α-reducible $\begin{array}{c}\Upsilon \\ \Delta \vdash A\end{array}$ where $\alpha = \mathrm{lvl}(A)$, the derivation $mc(\Upsilon, \Pi')$ is γ-reducible.
3. *if Π ends with any other rule, Π is γ-reducible if each of its major premise derivations is γ-reducible, and each of its minor premise derivations is normalizable.*

We say a derivation Π is *reducible* if there exists a γ such that Π is γ-reducible. The lemma below asserts that reducibility implies normalizability. Since reducibility is a strengthening of normalizability, it is shown by straightforward induction on the γ-reducibility of a derivation.

Lemma 2 (Normalization Lemma)**.** *If Ξ is γ-reducible then Ξ is normalizable.*

The key lemma needed to prove normalizability of derivations in $\mathrm{LD}^{\mu}_{\infty}$ is the following

Lemma 3 (Reducibility Lemma)**.** *Given n derivations* $\Pi_1, \ldots, \Pi_n$ *such that* Π_i *is* γ_i*-reducible, and any derivation* Π *the multicut*

$$\dfrac{\begin{array}{ccccc}\Pi_1 & & \Pi_n & & \Pi \\ \Delta_1 \vdash B_1 & \cdots & \Delta_n \vdash B_n & & \Gamma, B_1, \ldots, B_n \vdash C\end{array}}{\Delta_1, \ldots, \Delta_n, \Gamma \vdash C}\, mc$$

which we denote by Ξ*, is reducible.*

Following [15], the proof is by induction on (1) the number of occurrences of $\mu\mathcal{L}$ in Π, then by subordinate induction on (2) the height of Π, then on (3) the number of cut formulas n, and finally on (4) the γ_i-reducibility of each derivation Π_i.

We now proceed to show how the multicut rule may be eliminated from any $\mathrm{LD}^{\mu}_{\infty}$ derivation.

Corollary 1. *Every derivation* Π *in* LD^{μ}_{∞} *is reducible.*

Proof. Consider the nullary multicut $mc(\Pi)$, which is reducible by Lemma 3. Since $mc(\Pi)$ reduces to Π it follows that Π is reducible. □

Lemma 4 (Normal form lemma)**.** *If a derivation* Π *of a sequent* $\Gamma \vdash C$ *is normalizable, then there exists a cut-free derivation* $\hat{\Pi}$ *of* $\Gamma \vdash C$*, which we call a* normal form *for* Π*.*

Since every derivation in $\mathrm{LD}^{\mu}_{\infty}$ is reducible by Corollary 1, reducibility implies normalizability by Lemma 2, and a normalizable derivation has a cut-free normal form by the previous lemma, we obtain our main theorem

Theorem 1 (Cut admissibility for $\mathrm{LD}^{\mu}_{\infty}$)**.** *Every derivation of a ground sequent* $\Gamma \vdash C$ *in* LD^{μ}_{∞} *admits a cut-free derivation of the same sequent.*

We now obtain the following important consequence of the cut elimination theorem.

Corollary 2 (Consistency of $\mathrm{LD}^{\mu}_{\infty}$)**.** LD^{μ}_{∞} *is consistent.*

Proof. It suffices to notice by case analysis that there are no cut-free derivations of $\vdash \bot$. Since every derivation has a cut-free normal form by Theorem 1, $\vdash \bot$ is not derivable in $\mathrm{LD}^{\mu}_{\infty}$. □

5.3 Consistency of the Full Logic

The goal is now to define an interpretation of LD^{μ} in $\mathrm{LD}^{\mu}_{\infty}$, which will relate the derivability of a sequent in LD^{μ} to the derivability of a ground sequent in $\mathrm{LD}^{\mu}_{\infty}$. This will allow us to reduce the consistency of the former to that of the latter. The interpretation is specified by the following lemma.

Lemma 5 (Grounding Lemma)**.** *If* $\mathcal{Y}; \Gamma \vdash C$ *is derivable in* LD^{μ}*, then for any* $\mathcal{Y}$*-grounding substitution* $\delta : \emptyset \to \mathcal{Y}$*, the ground sequent* $\Gamma\delta \vdash C\delta$ *is derivable in* LD^{μ}_{∞}*.*

Proof. The proof is by induction on the height of the derivation of $\mathcal{Y}; \Gamma \vdash C$, and by case analysis on the last rule of the derivation. We only show the case where the derivation ends with $\Delta\mathcal{L}$ below.

Case $\Delta\mathcal{L}$: Suppose the derivation ends with $\Delta\mathcal{L}$ on the atom A where $\Gamma = \Gamma', A$. It is clear that if $\mathrm{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B, A\delta, \varepsilon_{\emptyset}, B')$, then $\mathrm{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B, A, \delta, B')$, and furthermore that $\mathrm{range}(\delta) = \emptyset$ since δ is

a ground substitution. Thus a ground derivation of a premise $\Gamma'\delta, B' \vdash C\delta$ below is obtained by applying the induction hypothesis to the ground premise $\cdot; \Gamma'\delta, B' \vdash C\delta$ with the empty grounding substitution.

$$\frac{\{\text{range}(\theta); \Gamma'\theta, B' \vdash C\theta \mid \text{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B, A, \theta, B')\}}{\mathcal{Y}; \Gamma', A \vdash C}\,\Delta\mathcal{L} \quad \rightsquigarrow$$

$$\frac{\{\Gamma'\delta, B' \vdash C\delta \mid \text{defn}(H \stackrel{\Delta}{=}_{\mathcal{X}} B, A\delta, \varepsilon_\emptyset, B')\}}{\Gamma'\delta, A\delta \vdash C\delta}\,\Delta\mathcal{L}$$

The effect of the interpretation on a derivation is therefore to prune branches from the $\Delta\mathcal{L}$ rule. □

As a result of the grounding lemma, we obtain the following

Lemma 6 (Interpretation lemma)**.** *If LD^{μ}_{∞} is consistent, then so is LD^{μ}.*

Proof. From grounding lemma, if $\vdash \bot$ is derivable in LD^{μ}, then it is also derivable in LD^{μ}_{∞}. Thus, the contrapositive also holds: If $\vdash \bot$ is not derivable in LD^{μ}_{∞}, then it is not derivable in LD^{μ}. □

The interpretation lemma in conjunction with the consistency of LD^{μ}_{∞} (Corollary 2) allow us to obtain

Corollary 3 (Consistency of LD^{μ})**.** *LD^{μ} is consistent.*

6 Conclusion

In this paper, we have shown how to accommodate inductive definitions in a logic that permits a more permissive form of fixed-point definitions. We have observed that the provision of inductive definitions must be done with some care in order to ensure consistency. We have also seen how the resulting logic allows us to encode arguments based on logical relations that are often used in formulating and proving properties about programs and programming languages. This work represents an intermediate step towards building a more flexible form for definitions into the logic underlying the Abella proof assistant. That logic additionally includes co-induction, generic quantification, and a notion called *nominal abstraction* which provides a means for analyzing occurrences of objects in expressions that are governed by generic quantifiers. In a longer version of this paper, we have shown that the greater flexibility in fixed-point definitions can be supported even in the presence of generic quantification. The extension of these results to include the remaining features of the logic is the subject of ongoing work.

The work that we have described builds on that of Tiu. Another effort that has a related flavor is that of Baelde and Nadathur [4]. That effort resulted in the development of a natural deduction calculus called μNJ, which extends deduction modulo [6] with inductive and co-inductive definitions. This calculus allows fixed-point definitions to be constructed with a flexibility that overlaps significantly with what is supported by ground stratification, with the difference that such definitions are realized through rewrite rules. An important aspect of μNJ is an equality elimination rule that builds in the ability to generalize equality assumptions, which then allows substitutions into proofs to be defined in a way that maintains their original structure. This feature has been exploited in showing strong normalizability for μNJ, thereby verifying its consistency. While μNJ includes a treatment of co-induction, it does not support generic quantification and nominal abstraction. We believe that the approach underlying this paper may be a preferred way to provide for the richer form of definitions because it does not permit rewrite rules that can modify the meaning of equality. However, there are insights to be gained from the formulation

of equality elimination in μNJ in developing a calculus close to LD$^\mu$ that also includes the additional features of interest and for when we can prove a cut elimination result.

We would like to extend this work in the future in a few ways in addition to including the features mentioned in the logic. First, we would like to build greater flexibility into the treatment of induction. Currently, inductive clauses do not allow the predicate being defined to appear to the left of an implication in the body. However, we believe that this requirement can be weakened to prohibit such occurrences only in truly negative positions. Second, perhaps taking the cue from μNJ, we would like to develop a non-ground calculus close to LD$^\mu$ for which we can prove a cut-elimination result. Finally, in a much more ambitious direction, we would like to add a higher order quantification capability to logics of definition. This would on the one hand allow for greater modularity in definitions, statements of theorems and their proofs, and may, on the other hand, allow for the statement and proofs of stronger results such as strong normalizability for System F [8].

References

[1] *The Abella web-site*. `https://abella-prover.org/index.html`. Accessed: 2025-15-05.

[2] Peter Aczel (1977): *An Introduction to Inductive Definitions*. In Jon Barwise, editor: *Handbook of Mathematical Logic*, *Studies in Logic and the Foundations of Mathematics* 90, Elsevier, pp. 739–782, doi:10.1016/S0049-237X(08)71120-0.

[3] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu & Yuting Wang (2014): *Abella: A System for Reasoning about Relational Specifications*. *Journal of Formalized Reasoning* 7(2), doi:10.6092/issn.1972-5787/4650.

[4] David Baelde & Gopalan Nadathur (2012): *Combining Deduction Modulo and Logics of Fixed-Point Definitions*. In: *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pp. 105–114, doi:10.1109/LICS.2012.22.

[5] Alonzo Church (1940): *A formulation of the simple theory of types*. *Journal of Symbolic Logic* 5(2), p. 56–68, doi:10.2307/2266170.

[6] Gilles Dowek & Benjamin Werner (2003): *Proof Normalization Modulo*. *Journal of Symbolic Logic* 68(4), pp. 1289–1316, doi:10.2178/jsl/1067620188.

[7] Andrew Gacek, Dale Miller & Gopalan Nadathur (2011): *Nominal abstraction*. *Information and Computation* 209(1), p. 48–73, doi:10.1016/j.ic.2010.09.004.

[8] Jean-Yves Girard, Paul Taylor & Yves Lafont (1989): *Proofs and Types*. Cambridge University Press.

[9] Robert Harper (2016): *Practical Foundations for Programming Languages*, 2nd edition. Cambridge University Press, doi:10.1017/CBO9781316576892.

[10] Per Martin-Löf (1971): *Hauptsatz for the intuitionistic theory of iterated inductive definitions*. In: *Proc. of the Second Scandinavian Logic Symposium*, *Studies in Logic and the Foundations of Mathematics* 63, North-Holland, pp. 179–216, doi:10.1016/s0049-237x(08)70847-4.

[11] Raymond McDowell & Dale Miller (2000): *Cut-elimination for a logic with definitions and induction*. *Theoretical Computer Science* 232, pp. 91–119, doi:10.1016/S0304-3975(99)00171-1.

[12] Dale Miller & Gopalan Nadathur (2012): *Programming with Higher-Order Logic*. Cambridge University Press, doi:10.1017/CBO9781139021326.

[13] Alberto Momigliano & Alwen Tiu (2004): *Induction and Co-induction in Sequent Calculus*. In: *TYPES*, *LNCS* 3085, Springer, pp. 293–308, doi:10.1007/978-3-540-24849-1_19.

[14] W. W. Tait (1967): *Intensional Interpretations of Functionals of Finite Type I*. *Journal of Symbolic Logic* 32(2), pp. 198–212, doi:10.2307/2271658.

[15] Alwen Tiu (2004): *A Logical Framework for Reasoning about Logical Specifications*. Ph.D. thesis, Pennsylvania State University. Available at http://etda.libraries.psu.edu/theses/approved/WorldWideIndex/ETD-479/.

[16] Alwen Tiu (2012): *Stratification in Logics of Definitions*. In Bernhard Gramlich, Dale Miller & Uli Sattler, editors: *Automated Reasoning*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 544–558, doi:10.1007/978-3-642-31365-3_43.

[17] Yuting Wang & Gopalan Nadathur (2016): *A Higher-Order Abstract Syntax Approach to Verified Transformations on Functional Programs*. In Peter Thiemann, editor: *Programming Languages and Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 752–779, doi:10.1007/978-3-662-49498-1_29.