Treating Congruences as Equalities Within Proofs

Dale Miller
Inria Saclay & LIX Institut Polytechnique de Paris, France

Abstract

Gentzen's sequent calculus is a foundational tool for describing and investigating provability, yet its fine-grained inference rules generally do not directly support automated proof search. Incorporating synthetic inferences improves automatability, but they do not provide mechanisms for reasoning naturally about the elementary mathematical notions of equivalence and congruence. In this paper, we present a first-order framework for reasoning modulo such relations within the sequent calculus. We provide a setting in which congruences can be treated as actual equalities, mirroring the informal practice of mathematicians and eliminating the need to use the lemmas typically required for formal congruence proofs. We demonstrate that this approach remains strictly first-order, avoids the complexity of higher-order or set-theoretic constructions, and yields proof systems that retain essential meta-theoretic properties.

2012 ACM Subject Classification Theory of computation → Proof theory

Keywords and phrases Proof theory, equality, equivalences, congruences

Digital Object Identifier 10.4230/LIPIcs.FSCD.2026.24

Acknowledgements I thank Kaustuv Chaudhuri, Arunava Gantait, and Dominik Kirst for useful discussions regarding the topic of this paper.

1 Introduction

The necessity of reasoning with equivalences is pervasive in both formal mathematics and program verification. Typically, an equivalence relation $\equiv$ on a structure is used to derive a new "modulo" structure where equivalent elements are treated as identical. Formally, this construction usually requires building either a function mapping elements to unique representatives or a partition of the underlying set into equivalence classes – a process that involves a set-of-sets construction. These approaches are naturally accommodated by higher-order logics, such as Church's Simple Theory of Types [8], or within Zermelo-Fraenkel set theory.

In this paper, we present an alternative framework for reasoning with equivalences and congruences that remains strictly first-order and avoids the overhead of set-theoretic axioms. We demonstrate that, under appropriate conditions, equivalences can be treated as syntactic equality without resorting to the higher-order encodings often found in automated reasoning [28]. Instead, we shall observe that in situations where it makes sense to move from, say, a partition of a group's carrier set to a new group structure based on equivalence classes (i.e., when the lifted group operation is well defined), we identify the equivalence as an actual equality, all without leaving first-order logic and without needing any new constructions. In fact, we shall formalize the common informal mathematical treatment of equivalences as if they really are equalities within our underlying first-order logic.

Our broader objective is to elevate structural proof theory beyond the analysis of logical connectives and consistency (via the cut-elimination theorem) toward the direct support of theories fundamental to elementary mathematics. This work follows the tradition established by Gentzen's study of arithmetic [16], Ketonen's application of sequent calculus to axiomatic geometry [23], and Negri and von Plato's work on orders and algebras [32, 34]. We extend this trajectory by providing a proof-theoretic treatment of equivalence relations, congruences, and compatibility.

2 The Basics: Types, terms, formulas, sequents, theories

Following Church [8], we base the syntax of terms and formulas on the simply typed λ-calculus. We reserve the primitive type o, and assume that there is a fixed set of primitive types (also called *sorts*) $\mathbb{S}$ that do not contain o. When $\sigma \in \mathbb{S}$, we shall also use σ as a predicate of type $\sigma \to o$.

Our first-order, multi-sorted intuitionistic logic is based on the following propositional connectives

$$\mathsf{t}\colon o,\ \wedge\colon o \to o \to o, \mathsf{f}\colon o,\ \vee\colon o \to o \to o, \supset\colon o \to o \to o$$

and the following connectives involve term structure (here, $\sigma \in \mathbb{S}$)

$$=_\sigma\colon \sigma \to \sigma \to o,\ \forall_\sigma\colon (\sigma \to o) \to o,\ \exists_\sigma\colon (\sigma \to o) \to o.$$

(We shall occasionally omit writing a typing subscript on an equality or quantifier when it can be inferred from context.) We work only in a first-order subset of Church's framework by restricting the types assigned to non-logical constants as follows.

- $P : \sigma_1 \to \cdots \to \sigma_n \to o$ is an n-ary predicate symbol and $\{\sigma_1, \ldots, \sigma_n\} \subseteq \mathbb{S}$ $(n \geq 0)$.
- $f : \sigma_1 \to \cdots \to \sigma_n \to \sigma_0$ is an n-ary constructor of type σ_0 and $\{\sigma_0, \ldots, \sigma_n\} \subseteq \mathbb{S}$ $(n \geq 0)$.
- $\sigma : \sigma \to o$ is a unary predicate symbol reflecting the sort σ also as a predicate.

A collection of such declarations is called a *first-order signature*. We usually write Σ_0 to denote the signature of non-logical constants involved in any particular logical specification. A Σ_0-formula is any closed term of type o, all of whose non-logical constants come from Σ_0. Note that if Σ_0 contains declarations of only predicate symbols, then a Σ_0-formula contains no occurrences of constructors, even of primitive arity.

Sequents are two-sided and denoted by $\Sigma :: \Gamma \vdash B$, where the left-hand side Γ is a set of formulas, and the right-hand side B is a single formula. The signature Σ declares the types from $\mathbb{S}$ for all free variables occurring in $\{B\} \cup \Gamma$. Following Gentzen's terminology, Σ is best understood as declaring the *eigenvariables* of the sequent – variables that function as generic or "freshly picked" parameters. Formally, if Σ_0 is the fixed signature of non-logical constants (constructors and predicates), then all formulas in the sequent are $(\Sigma_0 \cup \Sigma)$-formulas. However, since Σ_0 is typically held constant, we usually omit it and simply refer to these as Σ-formulas, implying that their free variables are strictly contained within Σ.

The usual inference rules for LJ are listed in Figure 1. Both the *instan* and *cut* rules are eliminable, in the sense that if a sequent is provable in LJ then it is provable in LJ without these rules.

Theories are finite sets of formulas. We shall often informally say that some formula, say B, holds, while what we generally mean is that B is provable from some theory that might be left implicit.

If C is a formula whose free variables are in the list $\bar{x}$, we will write this formula as $B\bar{x}$ where B is the (closed) *abstraction* $\lambda\bar{x}.C$. Here, λ plays only a notational role and does not need to be seen as part of our first-order logic.

We define a predicate via a *definition*, which takes the form of an equivalence of the form $\forall\bar{x}.[P\ x_1\ \ldots\ x_n \equiv B\ x_1\ \ldots\ x_n]$. Here, P is an n-ary predicate and $B\ x_1\ \ldots\ x_n$ is a formula composed of only $\wedge$, $\vee$, $\exists$, $=$, and atomic formulas (which may include atoms with the predicate P). Furthermore, we assume that the free variables of $B\ x_1\ \ldots\ x_n$ are in the list $x_1, \ldots, x_n$. As a direct consequence of the results in [26], an equally valid treatment of definitions is to omit the equivalence from the theory and instead add the following inference rules:

$$\frac{\Sigma :: \Gamma, B\ t_1 \cdots\ t_n \vdash G}{\Sigma :: \Gamma, P\ t_1 \cdots\ t_n \vdash G} \qquad \frac{\Sigma :: \Gamma \vdash B\ t_1 \cdots\ t_n}{\Sigma :: \Gamma \vdash P\ t_1 \cdots\ t_n}\ .$$

$$\frac{}{\Sigma :: \Gamma \vdash \mathsf{t}}\ \mathsf{t}R \qquad \frac{\Sigma :: \Gamma \vdash B, \quad \Sigma :: \Gamma \vdash C,}{\Sigma :: \Gamma \vdash B \wedge C}\ \wedge R \qquad \frac{\Sigma :: \Gamma \vdash B_i}{\Sigma :: \Gamma \vdash B_1 \vee B_2}\ \vee R,\ i \in \{1,2\}$$

$$\frac{\Sigma :: B, \Gamma \vdash C}{\Sigma :: \Gamma \vdash B \supset C}\ \supset R \qquad \frac{y\colon \sigma, \Sigma :: \Gamma \vdash B[y/x]}{\Sigma :: \Gamma \vdash \forall_\sigma x.B}\ \forall R \qquad \frac{\Sigma \vdash t\colon \sigma \qquad \Sigma :: \Gamma \vdash B[t/x]}{\Sigma :: \Gamma \vdash \exists_\sigma x.B}\ \exists R$$

$$\frac{\Sigma :: B_i, \Gamma \vdash C}{\Sigma :: B_1 \wedge B_2, \Gamma \vdash C}\ \wedge L_i,\ i \in \{1,2\} \qquad \frac{\Sigma \vdash t\colon \sigma \quad \Sigma :: B[t/x], \Gamma \vdash C}{\Sigma :: \forall_\sigma x.B, \Gamma \vdash C}\ \forall L$$

$$\frac{\Sigma :: \Gamma_1 \vdash B \quad \Sigma :: C, \Gamma_2 \vdash C}{\Sigma :: B \supset C, \Gamma_1, \Gamma_2 \vdash C}\ \supset L \qquad \frac{\Sigma :: B, \Gamma \vdash C \quad \Sigma :: C, \Gamma \vdash C}{\Sigma :: B \vee C, \Gamma \vdash C}\ \vee L \qquad \frac{}{\Sigma :: \mathsf{f}, \Gamma \vdash C}\ \mathsf{f}L$$

$$\frac{}{\Sigma :: B, \Gamma \vdash B}\ \mathit{init} \qquad \frac{\Sigma :: \Gamma \vdash B \qquad \Sigma :: \Gamma, B \vdash E}{\Sigma :: \Gamma \vdash E}\ \mathit{cut} \qquad \frac{\Sigma \vdash t : \sigma \quad \Sigma, x : \sigma :: \Gamma \vdash B}{\Sigma :: \Gamma[t/x] \vdash B[t/x]}\ \mathit{instan}$$

Figure 1 The inference rules for the LJ proof system.

$$\frac{}{\Sigma :: \Gamma \vdash t = t}\ =_r \qquad \frac{(t \text{ and } s \text{ not unifiable})}{\Sigma :: s = t, \Gamma \vdash C}\ =_l \qquad \frac{(\theta = \mathrm{mgu}(t,s)) \quad \Sigma\theta :: \Gamma\theta \vdash C\theta}{\Sigma :: s = t, \Gamma \vdash C}\ =_l$$

Figure 2 The rules to add to LJ to obtain the $LJ^=$ proof system.

An invariant of the sequents considered in this paper is that, for any sequent $\Sigma :: \Gamma \vdash B$, a variable declaration $x\colon \sigma$ appears in Σ if and only if the atomic formula $\sigma\ x$ appears in Γ. This property – the *typing redundancy invariant* – is preserved provided that the quantifiers $\forall_\sigma x.B$ and $\exists_\sigma x.\ B$ are treated as abbreviations for $\forall x_\sigma.\sigma x \supset B$ and $\exists x_\sigma.\ \sigma x \wedge B$, respectively.

3 Inference rules for equality

An elegant proof-theoretic treatment of equality for first-order terms was given independently by Girard [18] and Schroeder-Heister [35] in the early 1990s. Beyond its elegance, this approach to equality has proved expressive and useful: extended versions of this treatment of equality are part of the Bedwyr model checker [5, 20] and the Abella proof assistant [4, 15]. This treatment of equality is central to this paper.

The sequent calculus proof rules for the introduction of equality are presented in Figure 2. In the two versions of the $=L$ rule, unification is done with respect to the eigenvariables. The signature $\Sigma\theta$ is the result of removing from Σ all variables in the domain of θ and adding in all variables free in a term in the domain. When these rules are added to the LJ proof system, we get the proof system $LJ^=$. It is important to understand that equality is a *logical connective* and that these inference rules fix its meaning. The addition of axioms does not affect this notion of equality. Since we shall reserve the term "atomic formula" for formulas that are not top-level logical connectives, a formula such as $t = s$ is not an atomic formula.

The following are examples of theorems in $LJ^=$, assuming that $i \in \$$.

1. $\forall_i x.\ x = x$
2. $\forall_i x \forall_i y.\ x = y \supset y = x$
3. $\forall_i x \forall_i y \forall_i u.\ x = y \supset y = u \supset x = u$

Thus, the logic establishes that equality is an equivalence. If we assume that z (zero) has type i and s (successor) has type $i \to i$ in Σ_0, then the following are also theorems.

1. $\forall_i x \forall_i y.\ (s\ x) = (s\ y) \supset x = y$.
2. $\forall_i x.\ (s\ x) = z \supset \mathsf{f}$

Thus, $LJ^=$ proves all the Peano axioms except for the one involving induction. Finally, it is immediate that $\forall_\sigma x \forall_\sigma y.x = y \supset Bx \supset By$ is also provable. In the terminology of Section 4, this means that all abstractions B over the sort σ are *compatible* with equality.

In LJ, there is little difference between the constructors of arity 0 and eigenvariables: $B(a)$ can be replaced by $\forall a.B(a)$ and any LJ proof of one of these can be converted directly to an LJ proof of the other. The $LJ^=$ proof system, however, separates these two syntactic classes. For example, $\forall a \forall b.a = b \supset \mathsf{f}$ is not provable in $LJ^=$, while $a = b \supset \mathsf{f}$ is provable.

A formula is *equality-free* if it contains no occurrences of equality.

4 Equivalences and compatibilities

For every primitive type $\sigma \in \mathbb{S}$, we assume that we have a binary predicate eq_σ: that is, a non-logical constant of type $\sigma \to \sigma \to o$. We abbreviate by $\mathbb{E}$ the set $\{eq_\sigma\}_{\sigma \in \mathbb{S}}$ of such binary predicates and by $\mathcal{E}$ the set of formulas that axiomatize the properties of reflexivity, symmetry, and transitivity for all the predicates in $\mathbb{E}$. Given this axiomatization, these binary predicates denote equivalence relations.

Let P be an abstraction with type $\sigma_1 \to \cdots \to \sigma_n \to o$ where $n \geq 0$. We say that P is *compatible* with respect to $\mathbb{E}$ (following terminology in, say, [7]), if the following formula holds.

$$\text{Compatibility:} \quad \forall \bar{x} \forall \bar{y}.(\bigwedge_{i=1}^{n} eq\ x_i\ y_i) \supset P\ \bar{x} \supset P\ \bar{y}$$

The equivalence predicates eq_σ themselves are trivial examples of predicates that are compatible with $\mathbb{E}$.

Let $\mathbb{C}$ be a set of predicates and define $\mathcal{C}$ to be the set of formulas that axiomatize the compatibility of the predicates in $\mathbb{C}$. The following theorem proves that any abstraction over formulas whose only non-logical constants are compatible with $\mathbb{E}$ is also similarly compatible.

The theorems in this section consider only formulas that are equality-free: this is necessary since there might be a type $\sigma \in \mathbb{S}$ such that eq_σ is different from $=_\sigma$: this would cause a problem since we intend to lift eq_σ to be equality in a certain sense. It is, however, perfectly possible for eq_σ to be the same as $=_\sigma$. Equality plays a role starting in the next section when we define *determinacy*.

▶ **Theorem 1.** *Let B be an abstraction of type $\sigma_1 \to \cdots \to \sigma_n \to o$ $(n \geq 0)$ of an equality-free formula for which all non-logical symbols are in $\mathbb{E} \cup \mathbb{C}$. The following compatibility-like property is provable in $LJ^=$:*

$$\mathcal{E}, \mathcal{C} \vdash \forall \bar{x} \bar{y}.(\bigwedge_{i} eq\ x_i\ y_i) \supset B\bar{x} \supset B\bar{y}.$$

Proof. Since the displayed sequent above has no occurrences of equality, it is provable in $LJ^=$ if and only if the sequent

$$x_1, y_1, \ldots, x_n, y_n :: \mathcal{E}, \mathcal{C}, eq\ x_1\ y_1, \ldots, eq\ x_n\ y_n, B\bar{x} \vdash B\bar{y}$$

is provable in LJ. We proceed by induction on the structure of $B\bar{x}$ to show that this sequent is provable. Besides being equality-free, $B\bar{x}$ will necessarily contain no term constructors. Our proof is a simple modification of the proof that the general initial rule is admissible when only atomic initial rules are available.

Case. $B\bar{x}$ is an atom. There are three kinds of atomic formulas possible here.

1. If $B\bar{x}$ is of the form $eq_\sigma\ x_i\ x_j$, then the proof is completed using symmetry and transitivity for eq_σ.
2. If $B\bar{x}$ is of the form $\sigma\ x_i$ then we know that y_i is of the same type as x_i and, given the typing redundancy of sequents, we know that $\sigma\ y_i \in \Gamma$.
3. The remaining case is when this atom is of the form $(p\ u_1 \cdots\ u_i)$ for $i \geq 0$, $p \in \mathbb{C}$, and $u_1, \ldots, u_i$ is a list of variables in $\bar{x}$. This case is proved directly by using the compatibility property for p that appears in $\mathcal{C}$.

Case. $B\bar{x}$ is $B_1\bar{x} \supset B_2\bar{x}$. The inductive hypotheses give proofs of two sequents

$$x_1, y_1, \ldots, x_n, y_n :: \mathcal{E}, \mathcal{C}, eq\ x_1\ y_1, \ldots, eq\ x_n\ y_n, B_1\bar{y} \vdash B_1\bar{x}$$
$$x_1, y_1, \ldots, x_n, y_n :: \mathcal{E}, \mathcal{C}, eq\ x_1\ y_1, \ldots, eq\ x_n\ y_n, B_2\bar{x} \vdash B_2\bar{y}$$

The proof of the first of these also requires the symmetry property of the equivalence relations. From these two subproofs, one can immediately build a proof of

$$x_1, y_1, \ldots, x_n, y_n :: \mathcal{E}, \mathcal{C}, eq\ x_1\ y_1, \ldots, eq\ x_n\ y_n, B_1\bar{x} \supset B_2\bar{x} \vdash B_1\bar{x} \supset B_2\bar{x}$$

The cases for the other binary propositional connectives ($\wedge$ and $\vee$) are completely similar. The cases for `t` and `f` are immediate.

Case. $B\bar{x}$ is $\forall u, B_0 u\bar{x}$ where B_0 is an $n+1$ abstraction. The inductive hypothesis yields a proof of

$$x_1, y_1, \ldots, x_n, y_n, u, v :: \mathcal{E}, \mathcal{C}, eq\ x_1\ y_1, \ldots, eq\ x_n\ y_n, eq\ u\ v, B_0 u\bar{x} \vdash B_0 v\bar{y}$$

By substituting u for v into this sequent, cutting with the proof of $eq\ u\ u$, and introduction rules for $\forall$ on the left and right, we have a proof of

$$x_1, y_1, \ldots, x_n, y_n, u, v :: \mathcal{E}, \mathcal{C}, eq\ x_1\ y_1, \ldots, eq\ x_n\ y_n, \forall u.B_0 u\bar{x} \vdash \forall u.B_0 u\bar{y}$$

The case for the existential quantifier is similar. ◀

It is useful at times to replace the formulas in $\mathcal{E}$ and $\mathcal{C}$ with *synthetic rules*. By employing the techniques found in [26, 33], these formulas can also be turned into the following inference rules, encoding, respectively, reflexivity, symmetry, transitivity, and compatibility.

$$\frac{u\colon \sigma, \Sigma :: eq_\sigma\ u\ u, \Gamma \vdash B}{u\colon \sigma, \Sigma :: \Gamma \vdash B} \qquad \frac{\Sigma :: eq_\sigma\ t\ s, eq_\sigma\ s\ t, \Gamma \vdash B}{\Sigma :: eq_\sigma\ t\ s, \Gamma \vdash B} \qquad \frac{\Sigma :: eq_\sigma\ t\ s, eq_\sigma\ s\ r, eq_\sigma\ t\ r, \Gamma \vdash B}{\Sigma :: eq_\sigma\ t\ s, eq_\sigma\ s\ r, \Gamma \vdash B}$$

$$\frac{\Sigma :: eq_{\sigma_1}\ t_1\ s_1, \ldots, eq_{\sigma_1}\ t_1\ s_1, P\ t_1 \cdots t_n, P\ s_1 \cdots s_n, \Gamma \vdash B}{\Sigma :: eq_{\sigma_1}\ t_1\ s_1, \ldots, eq_{\sigma_1}\ t_1\ s_1, P\ t_1 \cdots t_n, \Gamma \vdash B}$$

As shown in [26], the addition of such synthetic rules to LJ results in a proof system that satisfies the cut elimination theorem.

5 Inference rules for equivalences

If we are working only with equality-free formulas over $\mathbb{E} \cup \mathbb{C}$, then we can move to a new proof system in which the (possibly numerous) axioms describing compatibility and equivalence are replaced by inference rules that make the predicates in $\mathbb{E}$ into logical connectives by

giving them their own left and right introduction rules. In particular, those new introduction rules are essentially the rules for equality.

$$\frac{}{\Sigma :: \Gamma \vdash eq_\sigma\ t\ t}\ eq_\sigma R \qquad \frac{u, \Sigma :: \Gamma[u/v] \vdash B[u/v]}{u, v, \Sigma :: eq_\sigma\ u\ v, \Gamma \vdash B}\ eq_\sigma L$$

Given that we will be working with formulas without occurrences of constructors, the rules for introducing equivalences on the left and right are simpler than for general equality. In particular, the left introduction rule lacks the "failure-to-unify" case. Let $LJ_{\mathbb{E}}$ be the proof system that results from adding the above introduction rules (for every $\sigma \in \mathbb{S}$) to LJ, and let $LJ^{=}_{\mathbb{E}}$ be the proof system that results from adding them to $LJ^{=}$.

▶ **Theorem 2.** *Let B be an $\mathbb{E} \cup \mathbb{C}$-formula without equalities. If $\cdot :: \mathcal{E}, \mathcal{C} \vdash B$ has an $LJ^{=}$ proof then $\cdot :: \cdot \vdash B$ has an $LJ^{=}_{\mathbb{E}}$ proof.*

Proof. Assume $\cdot :: \mathcal{E}, \mathcal{C} \vdash B$ has an $LJ^{=}$ proof. Since B contains no equalities, then $\cdot :: \mathcal{E}, \mathcal{C} \vdash B$ has an LJ proof. For the purpose of this proof, we extend the LJ proof system to the LJ^{+} proof system by adding the synthetic rules for all the formulas in $\mathcal{E} \cup \mathcal{C}$: as a result, the assumptions in $\mathcal{E} \cup \mathcal{C}$ can be removed from the sequent. In that extended proof system, every sequent is of the form

$$u_1 \colon \sigma_1, \ldots, u_n \colon \sigma_n :: \mathcal{S}, \mathcal{R}, \Gamma \vdash C, \tag{1}$$

where $\mathcal{S}$ is $\{\sigma_1\ u_1, \ldots, \sigma_n\ u_n\}$, and $\mathcal{R}$ is the set of atomic formulas of the form $eq_\sigma\ u_i\ u_j$ (and we assume that Γ does not contain such atomic formulas). For $\sigma \in \mathbb{S}$, let $\equiv_\sigma$ be the least equivalence relation on the variables in $\{u_1, \ldots, u_n\}$ that contains the relation $\{\langle u, v\rangle \mid eq_\sigma\ u\ v \in \mathcal{R}\}$. We fix some linear order on variables (say, alphabetic) and define the function $\hat{\equiv}_\sigma$ from variables to variables (both of type σ) such that $\hat{\equiv}_\sigma(u) = v$ if v is the representative of the $\equiv_\sigma$-equivalence class containing u. Here, the representative is the member of the equivalence class that is least in the order relation. The function $\hat{\equiv}(C)$ is defined to map the formula C to the formula that results in substituting free variables of C (of type, say, σ) with their image under $\hat{\equiv}_\sigma$. Next, we apply $\hat{\equiv}$ to the formulas in the sequent (1) to get the new sequent

$$v_1, \ldots, v_m :: \hat{\equiv}_\sigma(\mathcal{S}), \hat{\equiv}_\sigma(\mathcal{R}), \hat{\equiv}_\sigma(\Gamma) \vdash \hat{\equiv}_\sigma(C). \tag{2}$$

Here, the v_i's are representatives of equivalence classes and the atoms in $\hat{\equiv}_\sigma(\mathcal{R})$ are all of the form $eq_\sigma\ v_i\ v_i$. We shall say that the sequent (2) is *associated* with the sequent (1).

A simple induction on the structure of LJ^{+} proofs shows that if (1) has an LJ^{+} proof, then its associated sequent has an $LJ_{\mathbb{E}}$ proof. As a result of this induction, we know that since the sequent $\cdot :: \cdot \vdash B$ has an $LJ^{=}$ proof, then the associated sequent $\cdot :: \cdot \vdash B$ has an $LJ^{=}_{\mathbb{E}}$ proof. ◀

▶ **Theorem 3.** *Let B be an $\mathbb{E} \cup \mathbb{C}$-formula without equalities. If $\cdot :: \cdot \vdash B$ has an $LJ^{=}_{\mathbb{E}}$ proof then $\cdot :: \mathcal{E}, \mathcal{C} \vdash B$ has an $LJ^{=}$ proof.*

Proof. We show how to transform a $LJ^{=}_{\mathbb{E}}$ proof of $\Sigma :: \Gamma \vdash B$ into an $LJ^{=}$ proof $\Sigma :: \mathcal{E}, \mathcal{C}, \Gamma \vdash B$. All rules of $LJ^{=}_{\mathbb{E}}$ are rules of $LJ^{=}$ except for the $eq_\sigma L$ and $eq_\sigma R$ rules. The $eq_\sigma R$ rule is easily accounted for using the reflective rules in $\mathcal{E}$. To account for the $eq_\sigma L$ rule, we first prove that the sequents

$$u, v, \Sigma :: \mathcal{E}, \mathcal{C}, eq_\sigma\ u\ v, B \vdash B[u/v] \tag{3}$$

$$u, v, \Sigma :: \mathcal{E}, \mathcal{C}, eq_\sigma\ u\ v, B[u/v] \vdash B \tag{4}$$

have $LJ^=$ proofs. We do this by induction on the structure of B. The case where B is an atomic formula follows immediately from the axioms in $\mathcal{E}, \mathcal{C}$. All the other cases follow directly from the inductive hypothesis.

Now, assume that the last inference rule is the $eq_\sigma L$ rule. We apply the inductive assumption to conclude that the sequent

$$u, \Sigma :: \mathcal{E}, \mathcal{C}, \Gamma[u/v] \vdash B[u/v]$$

has an $LJ^=$ proof. We can now apply the cut rule with (3) for every member of $\Gamma[u/v]$ and with (4) with $B[u/v]$. Cut elimination for $LJ^=$ proofs then yields the desired proof of the sequent $u, v, \Sigma :: \mathcal{E}, \mathcal{C}, eq_\sigma\ u\ v, \Gamma \vdash B$. ◀

▶ **Theorem 4.** *The inference rule $eq_\sigma L$ is invertible.*

Proof. Assume that $u, v, \Sigma :: eq_\sigma\ u\ v, \Gamma \vdash B$ has a proof. It follows (using the *instan* rule) that the sequent $u, \Sigma :: eq_\sigma\ u\ u, \Gamma[u/v] \vdash B[u/v]$ has a proof. If we use cut with the proof of $u, \Sigma :: \mathbb{E}, \mathcal{C} \vdash eq_\sigma\ u\ u$, we can conclude that $u, \Sigma :: \Gamma[u/v] \vdash B[u/v]$ has a proof. ◀

6 Predicates as functions

Let P be a predicate of arity $n+1$ ($n \geq 0$) and assume that we wish to assert that P describes a function from its first n arguments to its last argument. That is, we wish that the predicate P encodes the function f if the (closed) atomic formula $P\ t_1\ \cdots\ t_n\ t_0$ is provable whenever t_0 is the result of $(f\ t_1\ \cdots\ t_n)$. This is possible if we assert the two formulas

$$\begin{aligned} \text{Determinacy:} &\quad \forall \bar{x} \forall u \forall v.\ P\ \bar{x}\ u \supset P\ \bar{x}\ v \supset u = v \\ \text{Totality:} &\quad \forall \bar{x} . \exists y.\ P\ \bar{x}\ y. \end{aligned}$$

In this case, the 1-ary abstraction $\lambda x.P\ t_1 \ldots\ t_n\ x$ (also written simply as $(P\ t_1 \ldots\ t_n)$) denotes a *singleton* set. We shall often interchangeably refer to predicates-as-functions as well as 1-ary abstractions-as-singletons.

The following theorem provides a kind of converse to the *instan* inference rule (Section 2): the *instan* rule allows one to instantiate an eigenvariable in the premise to get the conclusion. The P-det rule allows the following version of the converse: one can instantiate an eigenvariable in the conclusion to get the premise, provided that (i) the eigenvariable that is instantiated is restricted by a 1-ary abstraction known to be a singleton and (ii) the term used in the substitution is known to be an inhabitant of that singleton.

▶ **Theorem 5.** *Let P be a 1-abstraction over σ for which we can prove both the determinacy and totality properties from Σ and Γ. The following rule is admissible in $LJ^=$.*

$$\frac{\Sigma :: \Gamma_1 \vdash P\ t \quad \Sigma :: \Gamma_2[t/x] \vdash B[t/x]}{\Sigma, x :: P\ x, \Gamma_1, \Gamma_2 \vdash B}\ P\text{-det}$$

Furthermore, if $\Sigma :: \Gamma_1 \vdash P\ t$ is provable for some term t and Γ_1 is contained in Γ_2, then if the conclusion is provable, the right premise is provable.

This rule does not satisfy the subformula property: while $P\ t$ is often considered a subformula of $\forall x.P\ x$, it is not generally considered a subformula of $P\ x$. Thus, it is not surprising that the admissibility of this rule relies on the cut rule.

Proof. The P-det inference rule results from combining an instance of cut and the equality left rule with an application of the determinacy axiom, as shown below.

$$\dfrac{\Sigma :: \Gamma_1 \vdash P\ t \qquad \dfrac{\dfrac{\Sigma :: \Gamma_2[t/x] \vdash B[t/x]}{\Sigma :: P\ t, P\ x, x = t, \Gamma_2 \vdash B}\ =_l,\ \text{weakening}}{\Sigma :: P\ t, P\ x, \Gamma_2 \vdash B}\ \text{Determinacy axiom}}{\Sigma, x :: P\ x, \Gamma_1, \Gamma_2 \vdash B}\ cut$$

To prove the other, invertible-like property, assume that t is a Σ-term of type σ, Γ_1 is contained in Γ_2, and both $\Sigma :: \Gamma_1 \vdash P\ t$ and $\Sigma, x :: P\ x, \Gamma_1, \Gamma_2 \vdash B$ are provable. (Note that x is not free in the formulas of Γ_1.) These can be assembled into a proof of the right premise.

$$\dfrac{\Sigma :: \Gamma_1 \vdash P\ t \qquad \dfrac{\Sigma \vdash t : \sigma \quad \Sigma, x :: P\ x, \Gamma_1, \Gamma_2 \vdash B}{\Sigma :: P\ t, \Gamma_1, \Gamma_2[t/x] \vdash B[t/x]}\ instan}{\Sigma :: \Gamma_1, \Gamma_2[t/x] \vdash B[t/x]}\ cut$$

Since Γ_1 is contained in Γ_2 and since contexts are treated as sets, we have a proof of the right premise. ◀

The consequence of this theorem is that if we are searching for a proof of the sequent $\Sigma, x :: P\ x, \Gamma \vdash B$ and we know that P is a singleton, then we can continue the search with the sequent $\Sigma :: \Gamma[t/x] \vdash B[t/x]$ where t is known to be *the* member of the singleton denoted by P. Note that we do not need to use Hilbert's ϵ or Church's ι choice operators here [8, 21].

7 Predicates as congruences

Often, we wish that a particular function is actually a congruence with respect to the equivalence relations $\mathbb{E}$. The following formulas can be used to assert this property.

Determinacy modulo: $\forall \bar{x} \forall \bar{y} \forall u \forall v.(\bigwedge_i eq\ x_i\ y_i) \supset P\ \bar{x}\ u \supset P\ \bar{y}\ v \supset eq\ u\ v$

If the equivalence relations are all trivial (denoting just syntactic equality), then determinacy modulo is simply determinacy.

Congruences can be used to build compatible relations. In particular, let P be a predicate of type $\sigma_1 \to \cdots \to \sigma_n \to \sigma_0 \to o$ ($n \geq 0$) and assume that this predicate satisfies the conditions for being a congruence (i.e., determinacy, totality, and determinacy modulo). Axiomatize the predicate R of the same type as P by the following equivalence.

$$\forall \bar{x}.[R\ x_1 \cdots x_n\ x_0 \equiv \exists y.(P\ x_1 \cdots x_n\ y \wedge eq\ y\ x_0)].$$

▶ **Theorem 6.** *Let Γ be a theory that includes $\mathcal{E}$ and the definition of R above based on the predicates P and eq. If Γ proves that P is a function (determinate and total) as well as a congruence, then Γ also proves that R is compatible for $\mathbb{E}$.*

Proof. Follows as a simple consequence of the assumptions on P and R. ◀

We can now state as a theorem the result of lifting Theorem 5 about functions and $LJ^{=}$ to dealing with congruences and $LJ^{=}_{\mathbb{E}}$.

▶ **Theorem 7.** *Let Γ be a theory composed of $\mathbb{E} \cup \mathbb{C}$-formulas without equalities. Let P be a 1-ary abstraction over σ for which we can prove that it is a congruence from Σ and Γ. The following rule is admissible in $LJ^{=}_{\mathbb{E}}$.*

$$\frac{\Sigma :: \Gamma_1 \vdash P\ t \quad \Sigma :: \Gamma_2[t/x] \vdash B[t/x]}{\Sigma, x :: P\ x, \Gamma_1, \Gamma_2 \vdash B}\ P\text{-}det$$

Furthermore, if $\Sigma :: \Gamma_1 \vdash P\ t$ is provable for some term t and Γ_1 is contained in Γ_2 then if the conclusion is provable, the right premise is provable.

The proof of this theorem follows the proof of Theorem 5.

8 Computing with functions encoded as relations

As was noted in [17], when a predicate P denotes a function, the following equivalence holds for any 1-ary abstraction Q.

$$\forall \bar{x}\,[\,\forall u\,(P\ \bar{x}\ u \supset Q\ u) \equiv \exists u\,(P\ \bar{x}\ u \wedge Q\ u)\,] \tag{5}$$

Although this equivalence is a simple observation to make, it has a striking impact on viewing the operational reading of sequent calculus rules. For example, if one is attempting a proof of $\Gamma \vdash \exists u.(P\ \bar{x}\ u \wedge Q\ u)$ in the Gentzen-style proof systems we are using in this paper, the proof could proceed by first guessing a term t, then checking that it is the result of applying the function encoded by P on the arguments $\bar{x}$, and then checking that $Q\ t$ holds. The equivalence mentioned above allows us to change from this "guess and check" strategy and instead change to an attempt to prove $\Gamma \vdash (\forall u.P\ \bar{x}\ u \supset Q\ u)$. As we now illustrate, the operational reading of proof search for this sequent amounts to computing the value of the function encoded by P.

To present some examples related to natural numbers, we introduce the following primitive sort and two constructors (using the syntax of the Abella prover).

```
Kind nat      type.
Type z        nat.
Type s        nat -> nat.
```

The following definition for the predicate `nat`

```
Define nat : nat -> prop by
  nat z
; nat (s N) := nat N.
```

encodes, within Abella, a least fixed point. (The type of formulas in Abella is written as `prop` instead of as `o`.) In the setting of $LJ^{=}$, we will instead translate this definition as the equivalence

$$\forall x.nat\ x \equiv (x = \mathbf{0} \vee \exists y.\, x = (s\ y) \wedge nat\ y).$$

This equivalence is weaker than the intended least fixed-point definition used in Abella, since it will hold for all fixed points of this recursive definition. This reduction in strength is not surprising since $LJ^{=}$ does not include an induction scheme, whereas Abella is intended as an approach to arithmetic and does include induction.

Using the procedures for transforming such formulas into synthetic inferences given in [26, 33], the two implications given by this equivalence yield the following synthetic inference rules.

$$\frac{}{\Sigma :: \Gamma \vdash nat\ \mathbf{0}} \qquad \frac{\Sigma :: \Gamma \vdash nat\ N}{\Sigma :: \Gamma \vdash nat\ (s\ N)}$$

$$\frac{\Sigma :: n = \mathbf{0}, \Gamma \vdash B \quad \Sigma, y :: n = (s\ y), nat\ y, \Gamma \vdash B}{\Sigma :: \Gamma, nat\ n \vdash B}$$

Similarly, the definition of the predicate that encodes addition of natural numbers

```
Define plus : nat -> nat -> nat -> prop by
  plus z N N := nat N
; plus (s M) N (s K) := plus M N K.
```

yields the following equivalence

$$\forall n \forall m \forall k.\ plus\ n\ m\ k \equiv (n = \mathbf{0} \wedge m = k \wedge nat\ m) \vee (\exists n' \exists k'.n = (s\ n') \wedge k = (s\ k') \wedge plus\ n'\ m\ k')$$

and the following synthetic inference rules.

$$\frac{\Sigma :: \Gamma \vdash nat\ N}{\Sigma :: \Gamma \vdash plus\ \mathbf{0}\ N\ N} \qquad \frac{\Sigma :: \Gamma \vdash plus\ N\ M\ K}{\Sigma :: \Gamma \vdash plus\ (s\ N)\ M\ (s\ K)}$$

$$\frac{\Sigma :: n = \mathbf{0}, k = m, \Gamma \vdash B \quad \Sigma, n', k' :: (s\ n') = n, (s\ k') = k, plus\ n'\ m\ k', \Gamma \vdash B}{\Sigma :: \Gamma, plus\ n\ m\ k \vdash B}$$

Two instances of this last synthetic rule are

$$\frac{\Sigma :: m = k, \Gamma \vdash B}{\Sigma :: \Gamma, plus\ \mathbf{0}\ m\ k \vdash B} \quad \text{and} \quad \frac{\Sigma, k' :: (s\ k') = k, plus\ n'\ m\ k', \Gamma \vdash B}{\Sigma :: \Gamma, plus\ (s\ n)\ m\ k \vdash B}\ .$$

We write the boldface numerals $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, $\mathbf{3}$, and $\mathbf{4}$ to denote the terms z, $(s\ z)$, $(s\ (s\ z))$, $(s\ (s\ (s\ z)))$, and $(s\ (s\ (s\ (s\ z))))$. The following derivation verifies that 4 is the sum of 2 and 2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash nat\ \mathbf{2}}{\Gamma \vdash plus\ \mathbf{0}\ \mathbf{2}\ \mathbf{2}}}{\Gamma \vdash plus\ \mathbf{1}\ \mathbf{2}\ \mathbf{3}}}{\Gamma \vdash plus\ \mathbf{2}\ \mathbf{2}\ \mathbf{4}} \quad \Gamma \vdash nat\ \mathbf{4}}{\Gamma \vdash \exists p.plus\ \mathbf{2}\ \mathbf{2}\ p \wedge nat\ p}\ \exists, \wedge$$

To complete this proof, we must construct the (obvious) proofs of $\vdash nat\ \mathbf{2}$ and $\vdash nat\ \mathbf{4}$. It is most accurate to say that this proof *checks* that 4 is the result of adding 2 and 2. It is possible to use a proof to *compute* by switching this existential formula to its equivalent universal formula.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\cdot :: \Gamma \vdash nat\ (s\ (s\ \mathbf{2}))}{k'' :: \Gamma, \mathbf{2} = k'' \vdash nat\ (s\ (s\ k''))}\ =_l}{k'' :: \Gamma, plus\ \mathbf{0}\ \mathbf{2}\ k'' \vdash nat\ (s\ (s\ k''))}}{k', k'' :: \Gamma, k' = (s\ k''), plus\ \mathbf{0}\ \mathbf{2}\ k'' \vdash nat\ (s\ k')}\ =_l}{k' :: \Gamma, plus\ \mathbf{1}\ \mathbf{2}\ k' \vdash nat\ (s\ k')}}{u, k' :: \Gamma, u = (s\ k'), plus\ \mathbf{1}\ \mathbf{2}\ k' \vdash nat\ u}\ =_l}{u :: \Gamma, plus\ \mathbf{2}\ \mathbf{2}\ u \vdash nat\ u}}{\cdot :: \Gamma \vdash \forall u(plus\ \mathbf{2}\ \mathbf{2}\ u \supset nat\ u)}\ \forall, \supset$$

The conclusion-to-premise construction of this proof involves the systematic computation of the value of 2 plus 2.

This method of computing the value of a function when using its specification as a relation has an important defect: there might be many paths of computation that all arrive at the same value, but all of these paths will be traced out in the proof. To illustrate this issue, consider the following predicate definition for specifying the addition of natural numbers.

```
Define pluss : nat -> nat -> nat -> prop by
  pluss z N N := nat N
; pluss N z N := nat N
; pluss (s N) M (s P) := pluss N M P
; pluss N (s M) (s P) := pluss N M P.
```

It is easy to prove that this predicate is total, determinate, and computes the same function as the `pluss` predicate defined above. As a relation, this relation is highly non-deterministic: for example, it is possible to build 20 different proofs of $\exists N, \texttt{pluss}\ \mathbf{2}\ \mathbf{2}\ N$ by using backchaining on the clauses in this definition: of course, all of these paths will yield the same witness for $\exists N$. However, if we attempt a proof of the sequent $n :: \texttt{pluss}\ \mathbf{2}\ \mathbf{2}\ n \vdash Q\ n$ we will necessarily need to have all of these paths explored on the left (the encoding of the definition of `pluss` involves a disjunction of four cases): of course, all of these cases will yield the sequent $\cdot :: \cdot \vdash Q\ \mathbf{4}$ as their premise, and all of these identical premises must be proved.

The P-det rule can be used to eliminate such highly redundant proof search. For example, although there are 20 proofs (using the associated synthetic rules) of $\cdot \vdash \texttt{pluss}\ \mathbf{2}\ \mathbf{2}\ \mathbf{4}$, we only need the existence of one such path to invoke the P-det rule. Thus, we have the following instance of the P-det rule.

$$\frac{\cdot :: \cdot \vdash \texttt{pluss}\ \mathbf{2}\ \mathbf{2}\ \mathbf{4} \quad \cdot :: \cdot \vdash Q\ \mathbf{4}}{n :: \texttt{pluss}\ \mathbf{2}\ \mathbf{2}\ n \vdash Q\ n}\ \texttt{pluss-det}$$

The resulting proof search is a far more direct way to exploit the functionality of `pluss` via a built-in use of the cut rule. We only needed to develop one trace of computing the `pluss` predicate (the proof of the left premise).

These observations about computing function values using predicate specifications can also be extended to similar observations about congruences. When a predicate encodes also a congruence, we can prove a generalized form of the equivalence in (5), namely,

$$\forall \bar{x} \forall \bar{y} (\bigwedge_i eq\ x_i\ y_i) \supset (\forall v. P\ \bar{x}\ v \supset Q\ v) \equiv (\exists v. P\ \bar{y}\ v \wedge Q\ v), \tag{6}$$

where Q is a 1-ary abstraction that is also compatible. This equivalence can be used to switch from guessing the value of a function on one list of arguments to computing it, but for a different but equivalent list of arguments. The flexibility to choose different arguments can be valuable since one can select values that yield a more direct computation (often via some canonical representative for equivalence classes).

9 Applications to interactive theorem provers

We continue to illustrate the declaration of types, definitions, and theorems using the syntax of the Abella prover, as these declarations are relatively easy to read and closely resemble common logical notions. It is important to say that Abella has a number of specialized features – such as the ∇-quantifier [15, 30] and its own proof script language – but these are not relevant to our discussions here. We are only using Abella as an implementation of a multi-sorted first-order intuitionistic logic.

9.1 Systematically replacing functions with predicates

A typical approach to providing functions in a relational logic setting is to use a choice operator such as Hilbert's ϵ operator or Church's definite description operator ι. For example, if the predicate P with type $\sigma \to \tau \to o$ is known to describe a function, then that function could be written as $\lambda x_\sigma.(\iota\, y.(P\ x\ y))$ of type $\sigma \to \tau$ (see [1]). As we have mentioned earlier, we intend to remain strictly first-order and not admit such choice operators.

Not only do we not have genuine functions as part of our logic, but some of the theorems in this paper are about formulas built from only the predicates in $\mathbb{E}$ and $\mathcal{C}$: in particular, no term constructors are present.

Thus, we need a strategy for converting informal mathematical statements involving functions and term constructors into statements that admit only predicates. For example, an expression of the form $(x+y) = u$ needs to be translated into the relational expression `plus x y u`. Fortunately, the compiler-based notion of *administrative normal form (ANF)* [12] – where functions are applied only to arguments that are variables can be used to perform this mapping for expressions containing functions to expressions using only predicates. Furthermore, this normal form has been given a proof-theoretic definition (where proof structures are identified with term structures) in [17].

To illustrate a use of ANF, the expression $(f\ (g\ x)\ (f\ x\ y))$ can be written in ANF as

$$name\ u = (g\ x)\ in\ name\ v = (f\ x\ y)\ in\ name\ w = (f\ u\ v)\ in\ w.$$

Although such name-binding expressions (similar to let-binding expressions) are not part of our first-order logic, such expressions can be translated directly into quantified expressions. In particular, if we let P_g denote the predicate that encodes the function g (i.e., $(P_g\ x\ u)$ stands for $g\ x = u$) then the expression $name\ u = (g\ x)\ in\ \cdot$ can be encoded equivalently as either $\forall u.P_g\ x\ u \supset \cdot$ or $\exists u.P_g\ x\ u \wedge \cdot$ (their equivalence follows from the equivalence in (5)).

More generally, assume that we wish to prove the following theorem about the natural numbers (with the usual notions of addition and greater-than).

```
forall N M K, N < M -> (N + K) < (M + K)
```

This can be converted into ANF format, yielding the expression.

```
forall N M K, N < M -> name U = (N + K) in name V = (M + K) in U < V.
```

This expression can be converted into one of the following two theorems

```
forall N M K,
  N < M -> forall U, plus N K U -> forall V,  plus M K V -> U < V.

forall N M K,
  N < M -> exists U, plus N K U /\ exists V, plus M K V /\ U < V.
```

9.2 Separating concrete types from an abstract type

A multiset can be defined as a list modulo permutations. To illustrate this approach, the following items regarding the binary predicate `perm`.

```
Type    perm          list A -> list A -> prop.
Theorem perm_ref    : forall L, list L -> perm L L.
Theorem perm_sym    : forall L K, perm L K -> perm K L.
Theorem perm_trans  : forall L K M, perm L K -> perm K M -> perm L M.
```

These theorems assert that `perm` is an equivalence relation. We intend to manipulate multisets-as-lists using the following predicates.

```
Type empty    list A -> prop.
Type adj      A -> list A -> list A -> prop.
Type merge    list A -> list A -> list A -> prop.
```

Here, `empty L` should be provable exactly when `L` is the empty list; `adj X L K` is provable when adding the item `X` to the multiset `L` yields the multiset `K`; and `merge L K M` is provable when `M` is the result of merging the multisets `L` and `K`. The following are the axioms that assert that these predicates are compatible for permutations.

```
Theorem empty_compat : forall L K, perm L K -> empty L -> empty K.
Theorem adj_compat : forall X, forall L K, forall M N,
   perm L K -> perm M N -> adj X L M -> adj X K N.
Theorem merge_compat : forall L K, forall M N, forall P Q,
   perm L K -> perm M N -> perm P Q ->  merge L M P -> merge K N Q.
```

Defining `empty` to hold only for the empty list yields a relation that is compatible. The `append` operation on lists is a congruence but not compatible. Making it compatible is one way to define the `merge` relation (via Theorem 6).

At this point, we wish to use the proof system $LJ\overline{=}_{\mathbb{E}}$, in which there are inference rules that treat `perm` as if it were an equality on the type `mset A`. One way to achieve this in Abella is to leverage its typing discipline. For example, we can introduce a new (polymorphic) type for multisets and change the typing of the three primitive predicates as follows.

```
Kind mset     type -> type.
Type empty    mset A.
Type adj      A -> mset A -> mset A -> prop.
Type merge    mset A -> mset A -> mset A -> prop.
```

Note that we will not introduce any constructor of type `mset A`. By introducing these type declarations instead of list-based ones above, we can guarantee that all the definitions and formulas are compatible. This allows us to drop the axioms of equivalence and compatibility and to treat lists modulo permutations as multisets with equality. We can also transport any definitions and theorems proved using this restricted set of predicates back to being theorems where `mset A` is replaced by `list A`, where `perm` is, again, a non-trivial relation on lists, and where the compatibility and equivalence relations are again assumed.

10 Related work

The concept of abstract datatypes in programming languages – prominent in the literatures for both functional programming [31] and logic programming [19, 29] – is, of course, strongly related to the topic of this paper. Our approach separates the concrete and abstract views of structures, specifically within a proof environment rather than a traditional programming environment.

The deduction modulo framework [2, 11] is also capable of treating abstractions due to its ability to work modulo equational theories. In that work, equality is enriched with rewriting theories. The notion of equality in this paper remains the simple, syntactic equality, even when it is standing in for equivalences.

Treatments of equivalence and congruence have long appeared in systems based on constructive type theory, with probably the earliest such approach being the quotient types in NuPRL [10]. Setoids [6] are strongly related to the topic of this paper. For example, the pairing of types to equivalences is done using record typing, and compatibilities are captured by the notion of *morphism*. Setoids are generally developed in the context of constructive type theories and have been given mature implementations [9, 37]. In the Rocq

system, a `setoid_rewrite` tactic can be used to mimic the $eq_\sigma L$ and $eq_\sigma R$ inference rules. Although much of the topic of this paper is mirrored in this earlier work on setoids, it is worth examining how a proof-theoretic approach might express these similar concepts. For example, the presentation here remains first-order and can be simply applied to other logics with well-developed sequent calculi, such as classical and linear logics. Finally, the proof-theoretic approach to congruences here should also work seamlessly with the usual unification-based approaches to implementing proof search and with the ∇-quantifier of Abella. However, these possibilities are left for future work.

Foundational inquiries into the nature of equality have long occupied an important role in constructive type theory, spanning from the introduction of Martin-Löf's J-rule [27] to contemporary investigations into the "Uniqueness of Identity Proofs" principle [22]. The results presented in this paper may resonate with these long-standing conceptual concerns.

While we make claims in this paper that various kinds of combinatorial explosions can be averted using specific techniques, those claims are meant to apply entirely within the sequent calculus. Also, no claims are made about fully automating the proof theory results here, although we hope to consider doing so to enhance the automation in Abella. If one leaves the domain of sequent calculus and structural proof theory, one can turn to automated approaches based on resolution, such as those used in the superposition calculus [3]. Such approaches have been given highly effective automation by many systems, including the E Prover [36] and Vampire [24]. Given the divergence between the proof theory in this paper and the resolution-based framework of those systems, it would be interesting to bridge their foundational differences and to see to what extent the sequent calculus could adopt results from the superposition calculus.

The treatment of mathematical structures in category theory is likely to be related to our ability here to move from a concrete representation of data structures to a more abstract treatment. Fruitful connections might be made to the multi-sorted, first-order treatment of category theory given in the work by Freyd [13, 14] and Makkai [25].

11 Conclusion

This paper presents a first-order framework based on the sequent calculus that treats congruences and equivalence relations as actual equalities. By elevating structural proof theory to support these basic mathematical relations directly, we are helping bridge the gap between the fine-grained formal inference in proof theory and the informal practices of mathematicians.

The primary contributions of this paper include:

- A strictly first-order framework: We demonstrate that reasoning modulo equivalences can be achieved without resorting to higher-order logic or set-theoretic constructions, such as partitions or equivalence classes, while reasoning on structures modulo congruences.
- The replacement of axioms with inference rules: We show that the numerous axioms typically required for compatibility and equivalence can be replaced by dedicated left and right introduction rules that are essentially familiar rules for treating equality.
- Operational efficiency: By identifying when predicates encode functions, we can shift from a "guess and check" proof search strategy to one that mimics direct computation. This shift also provides flexibility for predicates that encode congruences.
- Articulation with a proof assistant: We illustrate how the various proof-theoretic results can be used within a proof assistant by illustrating how the administrative normal form can be used to map expressions involving functions into expressions involving only the

relations that denote those functions. We also illustrate how a simple change in typing allows one to move from a concrete treatment of structures to an abstracted treatment governed by the assumed equivalences.

Ultimately, this approach to abstraction should make proof search more effective and natural when dealing with the many constructions built from concrete structures modulo various equivalences.

References

1. Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, volume 27 of *Applied Logic Series*. Kluwer Academic Publishers, second edition, 2002. doi:10.1007/978-94-015-9934-4.
2. Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a Logical Framework based on the λΠ-calculus Modulo Theory, September 2016. Working Paper or Preprint. URL: https://hal.science/hal-01368161.
3. Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994. doi:10.1093/logcom/4.3.217.
4. David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. Abella: A system for reasoning about relational specifications. *J. of Formalized Reasoning*, 7(2):1–89, 2014. doi:10.6092/issn.1972-5787/4650.
5. David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. The Bedwyr system for model checking over syntactic expressions. In F. Pfenning, editor, *21th Conf. on Automated Deduction (CADE)*, number 4603 in LNAI, pages 391–397, New York, 2007. Springer. doi:10.1007/978-3-540-73595-3_28.
6. Gilles Barthe, Venanzio Capretta, and Olivier Pons. Setoids in type theory. *Journal of Functional Programming*, 13(2):261–293, 2003. doi:10.1017/S0956796802004501.
7. Stanley Burris and H. P. Sankappanavar. *A Course in Universal Algebra*, volume 78 of *Graduate Texts in Mathematics*. Springer-Verlag, 1981. Millennium Edition available online. doi:10.1007/978-1-4613-8130-3.
8. Alonzo Church. A formulation of the Simple Theory of Types. *J. of Symbolic Logic*, 5:56–68, 1940. doi:10.2307/2266170.
9. Claudio Sacerdoti Coen. A semi-reflexive tactic for (sub-)equational reasoning. In Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner, editors, *Types for Proofs and Programs (TYPES 2004)*, volume 3839 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2006. doi:10.1007/11617990_7.
10. Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, 1986.
11. Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *J. of Automated Reasoning*, 31(1):31–72, 2003.
12. Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. *ACM SIGPLAN Notices*, 28(6):237–247, 1993. doi:10.1145/155090.155113.
13. Peter J. Freyd. Properties invariant within equivalence types of categories. In Alex Heller and Myles Tierney, editors, *Algebra, Topology, and Category Theory: A Collection of Papers in Honor of Samuel Eilenberg*, pages 55–61. Academic Press, New York, 1976. doi:10.1016/B978-0-12-339050-9.50010-0.
14. Peter J. Freyd and Andre Scedrov. *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. Elsevier Science Publishers B.V., Amsterdam, Netherlands, 1990. doi:10.1016/C2009-0-09144-1.

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

16 Gerhard Gentzen. The consistency of elementary number theory. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 132–213. North-Holland, Amsterdam, 1969. Translated from the 1936 German original. doi:10.1016/S0049-237X(08)70823-1.

17 Ulysse Gérard and Dale Miller. Separating functional computation from relations. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *LIPIcs*, pages 23:1–23:17, 2017. doi:10.4230/LIPIcs.CSL.2017.23.

18 Jean-Yves Girard. A fixpoint theorem in linear logic. An email posting to linear@cs.stanford.edu archived at https://www.seas.upenn.edu/~sweirich/types/archive/1992/msg00030.html, February 1992.

19 Joseph A. Goguen and José Meseguer. Equality, types, modules, and (why not?) generics for logic programming. *The Journal of Logic Programming*, 1(2):179–210, 1984. doi:10.1016/0743-1066(84)90011-X.

20 Quentin Heath and Dale Miller. A proof theory for model checking. *J. of Automated Reasoning*, 63(4):857–885, 2019. doi:10.1007/s10817-018-9475-3.

21 David Hilbert. The foundations of mathematics. In Jean van Heijenoort, editor, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, pages 464–479. Harvard University Press, Cambridge, Massachusetts, 1967. doi:10.4159/harvard.9780674324497.c22.

22 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-Five Years of Constructive Type Theory*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford University Press, 1998. doi:10.1093/oso/9780198501275.003.0008.

23 Oiva Ketonen. *Investigations into the Predicate Calculus*. College Publications, 2022. Ed. by S. Negri and J. von Plato.

24 Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013. doi:10.1007/978-3-642-38574-2_1.

25 Michael Makkai. First order logic with dependent sorts, with applications to category theory. Manuscript, McGill University, 1995. URL: https://www.math.mcgill.ca/makkai/folds/foldsinpdf/FOLDS.pdf.

26 Sonia Marin, Dale Miller, Elaine Pimentel, and Marco Volpe. From axioms to synthetic inference rules via focusing. *Annals of Pure and Applied Logic*, 173(5):1–32, 2022. doi:10.1016/j.apal.2022.103091.

27 Per Martin-Löf. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North-Holland.

28 Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008. doi:10.1007/s10817-007-9085-y.

29 Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012. doi:10.1017/CBO9781139021326.

30 Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, October 2005. doi:10.1145/1094622.1094628.

31 John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10(3):470–502, 1988. doi:10.1145/44501.45065.

32 Sara Negri and Jan von Plato. Cut elimination in the presence of axioms. *Bulletin of Symbolic Logic*, 4(4):418–435, 1998. doi:10.2307/420956.

33 Sara Negri and Jan von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.

34 Sara Negri and Jan von Plato. *Proof Analysis: A Contribution to Hilbert's Last Problem*. Cambridge University Press, Cambridge, 2011. doi:10.1017/CBO9780511921629.

35 Peter Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *8th Symp. on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993. doi:10.1109/LICS.1993.287585.

36 Stephan Schulz, Simon Cruanes, and Petar Vukmirović. Faster, higher, stronger: E 2.3. *6th Vampire Workshop*, 71:1–10, 2019. doi:10.29007/sc9t.

37 Matthieu Sozeau. A new look at generalized rewriting in type theory. *Journal of Formalized Reasoning*, 2(1):41–62, 2009. doi:10.6092/issn.1972-5787/1574.