Dependently Sorted Theorem Proving For Mathematical Foundations

Yiming Xu
Australian National University, Canberra, Australia

Michael Norrish
Australian National University, Canberra, Australia

Abstract

We describe a new meta-logical system for mechanising foundations of mathematics. Using dependent sorts and first order logic, our system (implemented as an LCF-style theorem-prover) improves on the state-of-the-art by providing efficient type-checking, convenient automatic rewriting and interactive proof support. We assess our implementation by axiomatising Lawvere's Elementary Theory of the Category of Sets (ETCS) [5], and Shulman's Sets, Elements and Relations (SEAR) [17]. We then demonstrate our system's ability to perform some basic mathematical constructions such as quotienting, induction and coinduction by constructing integers, lists and colists. We also compare with some existing work on modal model theory done in HOL4 [20]. Using the analogue of type-quantification, we are able to prove a theorem that this earlier work could not. Finally, we show that SEAR can construct sets that are larger than any finite iteration of the power set operation. This shows that SEAR, unlike HOL, can construct sets beyond $V_{\omega+\omega}$.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases first order logic, sorts, structural set theory, mechanised mathematics, foundation of mathematics, category theory

Digital Object Identifier 10.4230/LIPIcs.ITP.2023.13

Acknowledgements We appreciate all the anonymous reviewers for their time and effort in pointing out typos and suggesting improvements to this paper. We would also like to thank James Borger for the name "DiaToM", which we feel beautifully encapsulates the aims and nature of our project.

1 Introduction

Mathematicians claim to work with set theory all the time, but many do so without really having to, or trying to, grapple with set theory's axioms. Moreover, this attitude is not unreasonable: it is not clear that standard ZF set theory should be mathematicians' foundation of choice. Few people are particularly happy with a foundation insisting that, for example, $1 \in 2$. It is not surprising then that a number of different foundations have been proposed in the literature. Considering variants of set theory, some famous examples are Lawvere's ETCS [5], Shulman's SEAR [17], Quine's New Foundation [14], Tarski-Grothendieck set theory [18] and von Neumann–Bernays–Gödel set theory (see, for example, Mendelson's presentation [11]) . Category theory has also been proposed as a mathematical foundation, in McLarty's CCAF [8] and Lawvere's ETCC [6], with the former having been shown capable of capturing many non-trivial results. And, though ETCC is known to be flawed, people have never lost interest in fixing it, and are continuing to work on similar systems.

Axiomatising a foundation for all of mathematics is a project that must be approached with the utmost care. Our belief is that this care should include mechanical support. That is, we should develop a theorem-proving system to serve as a tool for checking proofs in these foundations.

Our second goal is *expressiveness*: we want our system to be flexible enough to capture a variety of systems. At the same time, it is readily apparent that a significant amount of

work on mathematical foundations concentrates on first-order logic. Certainly, in all of the examples above, first-order logic is enough. We're quite happy to live with this constraint: a richer logic can conceal foundational decisions that we'd prefer to make apparent in our axioms. In the following, we present a first-order system that gains its expressiveness through a simple notion of dependent sort. Despite its simplicity, our system captures three of the foundational systems mentioned above, and is capable of fairly involved constructions in them all.

Contributions

- We develop a logical system that is able to express various first order axiomatic systems, where sorts can depend on terms. We specialise this ambient logical system so as to capture the foundational systems ETCS and SEAR.
- Building on these foundations, we demonstrate that our system can handle common mathematical constructions such as the development of the algebraic and co-algebraic lists.
- In one example, we also demonstrate SEAR's set-theoretic power by extending an existing example in model theory (done in HOL), and prove a theorem impossible to capture in HOL.
- We provide a proof-of-concept implementation that makes logical developments practical through the development of a number of important, though basic tools. For example, in ETCS, where proofs greatly rely on internal logic, we build a tool to automatically construct the internal logic predicates corresponding to "external" predicates. In both ETCS and SEAR, we automate inductive definitions, and provide tools to help with the construction of quotients.

The paper is structured as follows: we first introduce our fundamental logic, which is used for all three foundations. Then we briefly introduce the two structural set theories, ETCS and SEAR. After discussing the automation of comprehension in ETCS, for reasons of space, we present the remaining proofs in SEAR only. We note that with the exception of the modal model theory result (where the *bounded comprehension* schema is not sufficient) and the construction of the large set, all these formalised SEAR results can be formalised in ETCS as well. The proofs of a SEAR statement and its ETCS counterpart are often identical, in the sense that a proof in one system can be cut and pasted into the other. At the end of the paper, we compare our work with some existing logics developed for related purposes.

2 Logical System

We begin with the syntax of our logical system, which is "three-layered", consisting of sorts, terms, and formulas.

2.1 Sorts and Terms

Every sort depends on a (possibly empty) list of terms. The sorts are all of the form $s(t_0, \cdots, t_n)$, where $t_0, \cdots, t_n$ are terms of some pre-existing sorts and s is the name of the sort. A term is either a variable or a function application:

$$t := \mathsf{Var}(n, s) \mid \mathsf{Fun}(f, s, \vec{t})$$

That is, a variable consists of a name and a sort, and a function term consists of the name of the function symbol, the sort, and the arguments, which is a list of terms. A constant

is a nullary function. Each term has a unique sort, carried as a piece of information as an intrinsic property. A sort which does not depend on any term is called a *ground sort*. A term with a ground sort is called a *ground term*.

2.2 Formulas

We are working with a classical logic, and can afford to be minimal with our syntax: a formula Φ is either falsity, a predicate, an implication, a universally quantified formula, or a formula variable.

$$\Phi \ ::= \ \bot \mid \mathsf{Pred}(\mathcal{P}, \vec{t}) \mid \phi_1 \implies \phi_2 \mid \forall n : s.\, \phi \mid \mathsf{fVar}(\mathcal{F}, \vec{s}, \vec{t})$$

In the above, $\mathcal{P}$ is a predicate name, and $\mathcal{F}$ is the name of a formula variable. Boolean operators $\wedge, \vee, \neg$ can hence be built from the implication. We write $\top$ as an abbreviation $\bot \implies \bot$. In the $\forall$ case, the n and s carry the name and sort of the quantified variable. A formula $\mathsf{Pred}(\mathcal{P}, \vec{t})$ is a concrete predicate symbol applied to the argument list $\vec{t}$. Such a predicate symbol is either primitive, which comes together with the axiomatic setting or is defined by the user. A formula variable $\mathsf{fVar}(\mathcal{F}, \vec{s}, \vec{t})$ is analogous to a higher-order lambda expression taking an argument list $\vec{t}$ with sorts $\vec{s}$. We provide an inference rule to instantiate them below in Section 2.3.1. In the following, we will write a predicate formula as $\mathcal{P}(\vec{t})$. For a formula variable with name $\mathcal{F}$ on arguments of sorts $\vec{s}$ applied on $\vec{t}$, we write $\mathcal{F}[\vec{s}](\vec{t})$.

The only primitive predicate embedded in the system is equality between terms of the same sort. However, we need not allow equalities between terms just because they have the same sort. We cannot, for example, write equality between objects in ETCS, or equality between sets in SEAR. Thus, each foundation must record (along with function symbols, predicates symbols and axioms), the list of sorts supporting equality.

2.3 Theorems

A theorem consists of a set of variables Γ, called the context, a finite set A of formulas (the assumptions), and a formula ϕ as the conclusion. A theorem $\Gamma, A \vdash \phi$ reads "for all assignments σ of variables in Γ to terms respecting their sorts, if all the formulas in $\sigma(A)$ hold, then we can conclude $\sigma(\phi)$".

The context is the set of variables we require for the conclusion to be true given the assumptions: it contains at least all the free variables appearing in the assumptions or the conclusion. It can be regarded as a special form of assumption, asserting the existence of terms of certain sorts. We need the context to make sure we cannot use terms before either constructing them or assuming their existence. For instance, there is no arrow from the terminal object 1 to the initial object 0 in either ETCS or SEAR. Using a context, it can be proved that: $\{f : 1 \to 0\} \vdash \exists f : 1 \to 0.\ \top$, but $\vdash \exists f : 1 \to 0.\ \top$ is easily proved to be false.

2.3.1 Proof System

We now introduce the primitive rules. Rules for the propositional connectives are standard, as in Figure 1. The quantifier rules take some extra care of the sort information. When specialising a universal by a term, we need to put all the free variables of such a term into the context. Let $\mathsf{Vars}(t)$ denote the set of variables occurring in the term t, then:

$$\forall\text{-E, } t \text{ is of sort } s \ \frac{\Gamma, A \vdash \forall x : s.\phi(x)}{\Gamma \cup \mathsf{Vars}(t), A \vdash \phi(t)}$$

$$\text{Assume}\ \frac{}{\mathsf{Vars}(\phi), \{\phi\} \vdash \phi} \qquad \text{Ax}\ \frac{}{\mathsf{Vars}(\phi) \vdash \phi}\ \phi \text{ is an axiom}$$

$$\text{CContr}\ \frac{\Gamma, A \cup \{\neg\phi\} \vdash \bot}{\Gamma, A \vdash \phi} \qquad \text{ExF}\ \frac{}{\mathsf{Vars}(A \cup \{\phi\}), A \cup \{\bot\} \vdash \phi}$$

$$\text{Disch}\ \frac{\Gamma, A \vdash \phi}{\Gamma \cup \mathsf{Vars}(\psi), A \setminus \{\psi\} \vdash \psi \implies \phi} \qquad \text{MP}\ \frac{\Gamma_1, A_1 \vdash \phi \implies \psi \qquad \Gamma_2, A_2 \vdash \phi}{\Gamma_1 \cup \Gamma_2, A_1 \cup A_2 \vdash \psi}$$

$$\text{Refl}\ \frac{}{\mathsf{Vars}(a) \vdash a = a} \qquad \text{Sym}\ \frac{\Gamma, A \vdash a = b}{\Gamma, A \vdash b = a} \qquad \text{Trans}\ \frac{\Gamma_1, A_1 \vdash a = b \qquad \Gamma_2, A_2 \vdash b = c}{\Gamma_1 \cup \Gamma_2, A_1 \cup A_2 \vdash a = c}$$

$$\text{InstTM}\ \frac{\Gamma, A \vdash \phi}{\sigma(\Gamma), \sigma(A) \vdash \sigma(\phi)}\ \sigma \text{ is a well-formed map}$$

$$\text{FVCong}\ \frac{\Gamma_1, A_1 \vdash t_1 = t_1', \cdots, \Gamma_n, A_n \vdash t_n = t_n'}{\bigcup_{i=1}^n \Gamma_i, \bigcup_{i=1}^n A_i \vdash \mathcal{F}[\vec{s}](\vec{t}) \Leftrightarrow \mathcal{F}[\vec{s}](\vec{t'})}$$

Figure 1 Natural Deduction style presentation of our sorted FOL

To apply generalisation (∀-I) with a variable $a : s(t_1, \cdots, t_n)$, we require that (i) a does not occur in the assumption set; (ii) there is no term in the context depending on a; (iii) all the variables of sort s must also be in $\Gamma \setminus \{x\}$, and (iv) a does not appear in the sort list of any formula variable appearing in the conclusion. Once all these conditions are satisfied, we have

$$\forall\text{-I}\ \frac{\Gamma, A \vdash \phi(x)}{\Gamma \setminus \{x : s\}, A \vdash \forall x : s.\ \phi(x)}$$

We define $(\exists x.\phi) = \neg(\forall x.\neg\phi)$. The instantiation rule for formula variables is given as:

$$\text{Form-Inst}\ \frac{\Gamma, A(\mathcal{F}[\vec{s}]) \vdash \phi(\mathcal{F}[\vec{s}])}{\Gamma \cup \mathsf{Vars}(\psi), A[\mathcal{F}[\vec{s}] \mapsto \psi] \vdash \phi[\mathcal{F}[\vec{s}] \mapsto \psi]}$$

Instantiating a formula variable $\mathcal{F}[\vec{s}]$ is to replace each occurrence of $\mathcal{F}[\vec{s}]$ into a concrete formula on an argument list with sorts $\vec{s}$, and apply this predicate on $\vec{t}$. This is done by providing a map sending each such formula variable to a formula. This formula may or may not contain more formula variables, and is encoded by a pair consisting of a variable list $v_1, \cdots, v_n$ of sort $\vec{s}$ and a formula ϕ, such that $\forall v_1, \cdots, v_n.\phi$ is a well-formed formula. We rely on the term instantiation rule to make changes to the sort list, and then instantiate the formula variable when required.

When defining a new foundation we assume the existence of a signature recording that foundation's sorts, function symbols and predicate symbols. We extend the signature with new predicate symbols using the predicate specification rule.

$$\text{Pred-spec}\ \frac{}{\mathsf{Vars}(\vec{t}) \vdash \mathcal{P}(\vec{t}) \Leftrightarrow \phi(\vec{t})}\ \mathcal{P} \text{ does not occur in } \phi$$

Applying such a rule will define a new predicate with the name $\mathcal{P}$. The defined predicate will be polymorphic, where each tuple whose sort is matchable with the list $\vec{t}$ can be taken as the arguments. Here the argument of the new predicate symbol is not required to be all of $\mathsf{Vars}(\phi)$, we only require the whole set of free variables involved can be recovered from the arguments. For instance, if $\{a_1 : s_1, a_2 : s_2(a_1)\}$ exhausts the free variables involved, then the predicate can just take the single argument a_2 instead of both a_1 and a_2.

2.3.1.1 Function specification rule

The specification rule for new function symbols is the most complicated. Given a theorem $\Gamma, A \vdash \exists a_1 : s_1, \cdots a_n : s_n.Q(a_1, \cdots, a_n)$, if the existence of the tuple $(a_1, \cdots, a_n)$ is unique up to any sense which is accepted as suitable by the foundation, we define function symbols $f_1, \cdots, f_n$ such that their output tuple satisfies Q. To define a new function symbol, we provide a theorem stating the unique existence of some terms up to some relation, a theorem stating the relation is an equivalence relation, and a theorem guaranteeing non-emptiness of the relevant sorts.

In general, an equivalence must be captured by a predicate on two lists of variables, representing the two entities being related. As the built-in logic does not have a notion of tuples, we cannot define an equivalence relation to be a subset of the set consisting pairs of tuples of a certain form. Instead, we require theorems of the form:

$$\vdash R(\langle a_1 : s_1, ..., a_n : s_n\rangle, \langle a_1 : s_1, ..., a_n : s_n\rangle)$$

$$\vdash R(\langle a_1 : s_1, ..., a_n : s_n\rangle, \langle a'_1 : s'_1, ..., a'_n : s'_n\rangle) \implies R(\langle a'_1 : s'_1, ..., a'_n : s'_n\rangle, \langle a_1 : s_1, ..., a_n : s_n\rangle)$$

$$\vdash R(\langle a^1_1 : s^1_1, ..., a^1_n : s^1_n\rangle, \langle a^2_1 : s^2_1, ..., a^2_n : s^2_n\rangle) \wedge R(\langle a^2_1 : s^2_1, ..., a^2_n : s^2_n\rangle, \langle a^3_1 : s^3_1, ..., a^3_n : s^3_n\rangle) \implies R(\langle a^1_1 : s^1_1, ..., a^1_n : s^1_n\rangle, \langle a^3_1 : s^3_1, ..., a^3_n : s^3_n\rangle)$$

If the three theorems all hold for a concrete property R, then R is an equivalence relation (abbreviated as $\mathsf{eqth}(R)$ in the rest of the discussion). If R is used as the equivalence relation above, the unique existential theorem is required to be of the form:

$$\exists a_i : s_i.\ Q(\langle a_1 : s_1, ..., a_n : s_n\rangle) \wedge \forall a'_i : s'_i.\ Q(\langle a'_1 : s'_1, ..., a'_n : s'_n\rangle) \implies R(\langle a_1 : s_1, ..., a_n : s_n\rangle, \langle a'_1 : s'_1, ..., a'_n : s'_n\rangle)$$

We abbreviate the formula above as $\exists!_R a_i : s_i.\ Q(\langle a_1 : s_1, ..., a_n : s_n\rangle)$. The sorts of the two argument lists are not required to be equal, and they are generally not equal because the sorts of the latter variables often depend on the previous ones. The rule is expressed as:

$$\frac{\Gamma_0, \emptyset \vdash \exists a_i : s_i.\top \qquad \Gamma', A' \vdash \mathsf{eqth}(R) \qquad \Gamma, A \vdash \exists!_R a_i : s_i.Q(\langle a_1 : s_1, ..., a_n : s_n\rangle)}{\Gamma, A \vdash Q(\langle f_1(\Gamma'), ..., f_n(\Gamma')\rangle)}$$

where

- Q and R do not contain any formula variables; and
- $\Gamma_0 \subseteq \Gamma$, $\Gamma' \subseteq \Gamma$, and $A' \subseteq A$.

Our rule's leftmost premise requires the existence of terms of the required (output) sorts, given the existence of variables in the context corresponding to the sorts of the arguments. In this way, the rule guarantees that terms built using the new function symbol will always denote values in the output sort. For the equivalence relation, we can take R to be equality, meaning we are specifying new function symbols according to unique existence. If we take R to be the everywhere-true relation we have imported the Axiom of Choice into our system. The choice of which R's to allow is up to the designer of the object logic.

2.3.2 Semantics *via* Translation to Sorted FOL

In work that is not further described here, we have mechanised the proof that formula variables and their proof rules represent a conservative extension and can be eliminated. Subsequently, the term-instantiation rule (InstTM in Figure 1) can be derived from ∀-I and

∀-E and can also be removed from the list of primitive rules. As a result, our semantics below ignores them (meaning that our formulas come in just four forms: $\bot$, implications, the universal quantifier and predicate symbols). Our logic can be translated into non-dependent sorted FOL, which is equivalent to FOL. Given a list of sorts $s_1, \cdots, s_n$, such that s_k only depends on terms with sorts occurring earlier in the list for each $1 \leq k \leq n$, we create non-dependent sorts $\overline{s}_1, \cdots, \overline{s}_n$. These sorts are thought of as the non-dependent versions of $s_1, \cdots s_n$. We can think of the set of terms of sort $\overline{s_i}$ as the union of all terms of sort $s_i(\vec{t})$ for all possible tuples $\vec{t}$ of terms.

$$\{a \mid a : \overline{s_i}\} = \bigcup_{\vec{t_k}} \{a \mid a : s_i(\vec{t_k})\}$$

For example, the ETCS terms $f : A \to B$ and $g : C \to D$ are of different arrow sorts, but their translation both have sort $\overline{ar}$. For a function symbol f taking a list of terms $[t_1 : s_1, \cdots, t_n : s_n]$, we create a non-dependent sorted function symbol $\overline{f}$, such that its argument term list has the corresponding sort list $\overline{s_1}, \cdots, \overline{s_n}$. We do the same for predicate symbols. Translation from terms of s_k into those of FOL sort $\overline{s}_k$ is done by forgetting sort dependency:

$$\begin{aligned} [\![\mathsf{Var}(x, s_k(t_1, \cdots t_m))]\!]_{\mathsf{t}} &= \mathsf{Var}(x, \overline{s}_k) \\ [\![\mathsf{Fun}(f, s_k(t_1, \cdots, t_m), (a_1, \cdots, a_n))]\!]_{\mathsf{t}} &= \mathsf{Fun}(f, \overline{s}_k, ([\![a_1]\!]_{\mathsf{t}}, \cdots, [\![a_n]\!])) \end{aligned}$$

For sorts s depending on terms $t_1 : s_1, \cdots, t_m : s_m$, we create function constants $d_{s,1}$, $\cdots$, $d_{s,m}$. For $1 \leq i \leq m$, $d_{s,i}$ takes an argument of sort $\overline{s}$ and outputs a term of sort $\overline{s_i}$. If a function symbol f takes arguments $(t_1 : s_1, \cdots, t_n : s_n)$, and outputs a non-ground sort s, where s depends on terms $r_1, \cdots, r_n$, and each s_k depends on terms $q_{k,i}$, then we add an axiom to regulate the dependency information of its sort when translated into non-dependent-sort FOL:

$$\bigwedge_k \bigwedge_i d_{s_k,i}([\![v_k]\!]_{\mathsf{t}}) = [\![q_{k,i}]\!]_{\mathsf{t}} \implies \bigwedge_j d_{s,j}([\![f(v_1, \cdots, v_n)]\!]_{\mathsf{t}}) = [\![r_j]\!]_{\mathsf{t}}$$

As an example, the composition function symbol in ETCS takes $g : B \to C$ and $f : A \to B$, and outputs $g \circ f : A \to C$. The corresponding axiom is:

$$\begin{aligned} &\forall(A : \overline{\mathsf{ob}})\ (B : \overline{\mathsf{ob}})\ (C : \overline{\mathsf{ob}})\ (f : \overline{\mathsf{ar}})\ (g : \overline{\mathsf{ar}}). \\ &\quad d_{\mathsf{ar},1}(f) = A \wedge d_{\mathsf{ar},2}(f) = B \wedge d_{\mathsf{ar},1}(g) = B \wedge d_{\mathsf{ar},2}(g) = C \implies \\ &\qquad d_{\mathsf{ar},1}(g \circ f) = A \wedge d_{\mathsf{ar},2}(g \circ f) = C \end{aligned}$$

For an arbitrary function symbol f, although its arguments can include ground terms, the axiom only needs to state information about the dependently sorted argument, where the functions d_k, as shown above, exist. If the output of a function symbol is a ground sort, we do not need such an axiom for it.

Translation of formulas only makes sense under the translation of some context that contains at least all of its free variables. Defining the translation of a context amounts to translating sort judgments of variables. We translate the sort judgment of any ground sort into $\top$. As for a variable $a : s_k(t_1, \cdots, t_n)$, we write

$$[\![a : s_k(t_1, \cdots, t_n)]\!]_{\mathsf{ts}} = \bigwedge_i d_{k,i}([\![a]\!]_{\mathsf{t}}) = [\![t_n]\!]_{\mathsf{t}}$$

to denote the translation of a context element ($[\![\cdots]\!]_{\mathsf{ts}}$ calculates the denotation of a term's sorting assertion). An entire context Γ is translated into the conjunction of the translation of its elements.

As we do for function symbols, we create for each dependent sorted predicate symbol $\mathcal{P}$ a corresponding non-dependent sorted one, written as $\overline{\mathcal{P}}$. We define the translation of formulas by induction as:

$$\begin{aligned}
[\![\mathcal{P}(t_1 : s_1, \cdots, t_n : s_n)]\!]_{\mathsf{f}} &= \overline{\mathcal{P}}([\![t_1]\!]_{\mathsf{t}}, \cdots, [\![t_n]\!]_{\mathsf{t}}) \\
[\![\phi \implies \psi]\!]_{\mathsf{f}} &= [\![\phi]\!]_{\mathsf{f}} \implies [\![\psi]\!]_{\mathsf{f}} \\
[\![\forall x : s.\ \psi]\!]_{\mathsf{f}} &= \forall x : \overline{s}.\ [\![x]\!]_{\mathsf{ts}} \implies [\![\psi]\!]_{\mathsf{f}}
\end{aligned}$$

Finally, a theorem $\Gamma, A \vdash \psi$ translates into

$$\forall v_1 \ldots v_n. \bigwedge_{(v_i:s_i)\in\Gamma} [\![v_i : s_i]\!]_{\mathsf{ts}} \wedge [\![A]\!]_{\mathsf{f}} \implies [\![\psi]\!]_{\mathsf{f}}$$

It is routine to check that the rules are valid under the translation and hence have the intended sense. As an example, consider ∀-I. Assume $\Gamma, \{a : s(t_1, \cdots, t_n)\}, A \vdash \phi(a)$ and the variable a appears in neither Γ nor A. The theorem translates into

$$[\![\Gamma]\!]_{\mathsf{ts}}, [\![a : s(t_1, \cdots, t_n)]\!]_{\mathsf{ts}}, \bigwedge [\![A]\!]_{\mathsf{f}} \vdash [\![\phi(a)]\!]_{\mathsf{f}}$$

(where we overload $[\![\cdots]\!]_{\mathsf{ts}}$ and $[\![\cdots]\!]_{\mathsf{f}}$ to include the versions mapping sets to conjunctions of translations). The fact that a does not appear in Γ translates into the corresponding variable $a : \overline{s}$ not appearing in $[\![A]\!]_{\mathsf{f}}$, and the requirement that no variable depends on a translates to the requirement that $[\![a : s(\ldots)]\!]_{\mathsf{ts}}$ does not appear in $[\![\Gamma]\!]_{\mathsf{ts}}$ either. Therefore, we can discharge $[\![a]\!]_{\mathsf{ts}}$ from the assumption and deduce from the FOL universal elimination rule that $[\![\Gamma]\!]_{\mathsf{ts}}, [\![A]\!]_{\mathsf{f}} \vdash \forall a : [\![s]\!]_{\mathsf{s}}. [\![a : s]\!]_{\mathsf{ts}} \implies [\![\phi(a)]\!]_{\mathsf{f}}$. This is the translation of $\Gamma, A \vdash \forall a : s(t_1, \cdots, t_n).\phi(a)$, as required.

Implementation

Our implementation is a proof-of-concept written in SML. It provides a simple REPL similar to those provided by HOL4 and HOL Light. The kernel (core syntax and proof rules) is implemented in 2443 lines of code; user-level parsing (including a simple type inference algorithm) and printing is a further 1633 lines of code. Additional core libraries (goal stack package, common tactics including the rewriting tactic) take 4386 lines.

The source code for this implementation is available from `https://github.com/u5943321/DiaToM`

3 ETCS and SEAR

ETCS [5] and SEAR [17] are both structural set theories. With each, we work within a well-pointed boolean topos. In particular, they both have products, coproducts, exponentials, an initial object 0 and a terminal object 1. Whereas the existence of all of these are given as primitive axioms in ETCS, we can construct them in SEAR.

ETCS has two sorts: objects (A, B, ...; a ground sort) and arrows (*e.g.,* $A \to B$), where an arrow sort depends on two object terms. Equality can only hold between arrows. An object is to be considered as a set in the usual sense: an arrow $1 \to X$ is regarded as an element of the set X. As *per* Shulman's original construction, SEAR has three sorts: sets (A, B, ...; a ground sort); members ($\_ \in A$, depending on a set term); and relations ($A \nrightarrow B$, depending on two set terms). SEAR also adds a primitive predicate $\mathsf{Holds}(R : A \nrightarrow B, a \in A, b \in B)$, declaring that the relation R relates a and b.Equality can hold between relations with the same domain and codomain, and elements of the same set.

In SEAR, a relation R is called a function if $\forall a.\exists!b.\ \mathsf{Holds}(R, a, b)$. In practice, we want to be able to write $f(a)$ as the result of applying a function to an argument, but we cannot do this if we are restricted to just the relation sort. A first thought might be to create a function symbol Eval, that takes a relation and a member of A, so the term $\mathsf{Eval}(R : A \nrightarrow B, a \in A)$ is a member of B. However, such a function symbol breaks soundness, as the term $\mathsf{Eval}(R, a)$ can be expressed for every a of the correct sort before checking the function condition on R. In particular, we can write a term $\mathsf{Eval}(R : 1 \nrightarrow 0, *)$, nominally producing an element of 0.

Rather, we introduce a function sort which is a "proper subsort" of the relation sort.[1] A function f from A to B is written $f : A \to B$, and we add the following axiom describing terms of function sorts:

$$\begin{array}{l}\mathsf{isFunction}(R) \implies \\ \quad \exists! f : A \to B.\ \forall (a \in A)\ (b \in B).\mathsf{Eval}(f, a) = b \Leftrightarrow \mathsf{Holds}(R, a, b)\end{array}$$

The $\mathsf{isFunction}$ predicate embodies the definition above, and we also have a new Eval function symbol that takes a function term from A to B and an element term of A, and outputs an element term of B.

We will write $\mathsf{Eval}(f, a)$ simply as $f(a)$ in the rest of paper. The Eval symbol is typed so that only functions terms can be its first argument. It is clear that this is a conservative extension, as any theorems involving Eval can be expressed using just Holds and uses of the $\mathsf{isFunction}$ hypothesis if desired.

Subsets are handled differently in ETCS and SEAR. Using the SEAR axioms, it is straightforward to show that for each formula ϕ on members $x \in X$, we can form the subset $\{x \mid \phi(x)\}$. In what follows, $\mathcal{F}$ is an arbitrary formula variable, and we are defining a *comprehension schema*. Our subset is constructed *via* a member of the power set $\mathsf{Pow}(X)$,[2] and ultimately as a term of set sort with an injection to X. This construction is described by the following two theorems (following Shulman [17]). First, we prove the existence of the member of the power-set. Given that A is a set, then IN requires two arguments of sort $\_ \in A$ and $\_ \in \mathsf{Pow}(A)$. Then:

$$\exists! s \in \mathsf{Pow}(A).\ \forall a.\ \mathsf{IN}(a, s) \Leftrightarrow \mathcal{F}[\mathsf{mem}(A)](a)$$

We also have the existence of a set B, and an injection from it into A:

$$\exists B\ (i : B \to A).\ \mathsf{Inj}(i) \wedge \forall (a \in A).\ \mathcal{F}[\mathsf{mem}(A)](a) \Leftrightarrow \exists b \in B.\ a = i(b) \tag{1}$$

The combination of i and B can be seen as identifying the subset of A satisfying predicate P.

The following isset predicate, connecting a member (s) to a set (B, given implicitly in i's sort) is also occasionally useful:

$$\mathsf{isset}(i : B \to A, s \in \mathsf{Pow}(A)) \stackrel{\text{def}}{\Leftrightarrow} \mathsf{Inj}(i) \wedge \mathsf{image}(i, B) = s$$

The "subset story" in ETCS is more restrictive. There we can only form subsets from predicates on elements of X which can be captured by an arrow $p : X \to 2$, where 2 is defined to be the coproduct $1 + 1$. Such arrows are turned into elements of the power object 2^X by taking transposes. We regard 2 as the set of truth values, where $\top_I, \bot_I : 1 \to 2$ denote truth

[1] Shulman (personal communication) agrees that the resulting system is still effectively SEAR as he conceives it.

[2] The existence of the powerset function is easy to establish from the function specification rule: power set of each set is unique up to isomorphism that respects the membership relation.

and falsity respectively. Our arrow p gives rise to a separate object as characterised by the theorem:

$$\forall A\ (p : A \to 2).\ \exists B\ (i : B \to A).\ \mathsf{Inj}(i) \wedge \forall a : 1 \to A.\ p \circ a = \top_I \Leftrightarrow \exists b.\ a = i \circ b$$

The existence of i is witnessed by the pullback of the map $\top_I$ along p. Note that this method only shows the existence of subsets for arrows $p : X \to 2$. We do not achieve the generality of SEAR, where the construction starts with an arbitrary formula variable. ETCS *does* allow for the construction of subsets using something resembling set comprehension, but this requires a detour *via* its internal logic (see Section 4 below).

Another notable difference between the two logics is that ETCS comes with the axiom of choice in the form of the statement that any epimorphism has a section, whereas this is not given in SEAR. In fact, if we change SEAR by adding the axiom of choice, and also requiring that the input formula of our comprehension schema be bounded, then the resulting system has the same strength as ETCS.

For both ETCS and SEAR, the injection we construct from each predicate is unique up to respectful isomorphism. This allows us to use the specification rule to obtain new constants without the full form of choice. In SEAR, for example, we can prove if there are $i : B \to A$ and $i' : B' \to A$, which are both injections, and moreover, we have $\forall a.\ P(a) \Leftrightarrow \exists b \in B.\ a = i(b)$ and $\forall a.\ P(a) \Leftrightarrow \exists b \in B'.\ a = i'(b)$, then the relation between pairs $(B, i : B \to A)$ and $(B', i' : B' \to A)$ defined by

$$\begin{aligned}&\exists (f : B \to B')\ (g : B' \to B).\\ &\quad f \circ g = \mathsf{Id}(B) \wedge g \circ f = \mathsf{Id}(B') \wedge i' \circ f = i \wedge i \circ g = i'\end{aligned}$$

holds. This is clearly an equivalence relation. Moreover, for all sets A, the existence of a set B and a map $B \to A$ is witnessed by the identity isomorphism. Therefore, once we instantiate the P above into a concrete predicate without any formula variables, we have met all of the specification rule's antecedents, and we can use it to define two constants: the subset and its inclusion into the ambient set. In SEAR, the sets of natural numbers, integers, lists and co-lists are all constructed in this way. More generally, given any member $s \in \mathsf{Pow}(A)$, we use the specification rule to turn it into a "real set" via the constant $\mathsf{m2s}(s)$ of set sort. This set is injected into A by the map $\mathsf{minc}(s) : \mathsf{m2s}(s) \to A$.

4 Internal logic in ETCS

As discussed in the last section, an arrow $p : X \to 2$ corresponds to a predicate on X in the sense that if $x : 1 \to X$, then $p \circ x = \top_I$ means p is true for x. An ETCS formula is *bounded* precisely when all quantified variables are elements (*i.e.*, arrows with domain 1). Let us call the formulas of our logic (all formulas seen so far) *external formulas*. If an external formula is bounded with all free variables also elements, we can automatically construct a corresponding *internal formula* as a term of the logic. When the external formula is on variables with sorts $(1 \to X_1), (1 \to X_2), \ldots$, then the internal formula will be an arrow of sort $\Pi X_i \to 2$. For an external formula $\Phi[x_1 : 1 \to X_1, \ldots]$, then let $p : \Pi X_i \to 2$ be the corresponding formula. We require

$$\forall a : 1 \to \Pi X_i.\ p \circ a = \top_I \Leftrightarrow \Phi[(\pi_i \circ a)/x_i]$$

where $\Phi[t/x]$ is the substitution of term t for variable x. This could be regarded as an axiom, one rather like Separation in ZF. However, we can instead prove all results of this form

automatically. This is simply by rewriting with all the theorems with relevant definitions and properties of the internal logic operators as explained below.

We have implemented an automatic translation (a "derived rule") that generates an internal logic formula given a list of variables, considered as the arguments, and the formula. The translation produces an internal logic predicate and proves that it gives the value $\top_I$ if and only if the formula is true when applied to the arguments. We illustrate our algorithm with an example over $\mathbb{N}$, the natural number object, the arrow $\mathsf{SUC} : \mathbb{N} \to \mathbb{N}$, and the function symbol $\_^+$, such that $n^+ \stackrel{\text{def}}{=} \mathsf{SUC} \circ n$. Then, the pair $([n], m^+ - n^+ = m - n)$ encodes a simple unary predicate on n. In this case, the output of our derived rule is an arrow term $p : \mathbb{N} \to 2$ satisfying:

$$\forall n : 1 \to \mathbb{N}.\ \ p \circ n = \top \Leftrightarrow m^+ - n^+ = m - n$$

If the list of arguments is $[m, n]$ instead, the produced arrow $p : \mathbb{N} \times \mathbb{N} \to 2$ will satisfy:

$$\forall m, n : 1 \to \mathbb{N}.\ p \circ \langle m, n\rangle = \top \Leftrightarrow m^+ - n^+ = m - n$$

Operator	Sort	Defining Property
$\wedge_I$	$2 \times 2 \to 2$	$\wedge_I \circ \langle p_1, p_2\rangle \Leftrightarrow p_1 = \top_I \wedge p_2 = \top_I$
$\vee_I$	$2 \times 2 \to 2$	$\vee_I \circ \langle p_1, p_2\rangle \Leftrightarrow p_1 = \top_I \vee p_2 = \top_I$
$\Rightarrow_I$	$2 \times 2 \to 2$	$\Rightarrow_I \circ \langle p_1, p_2\rangle \Leftrightarrow p_1 = \top_I \implies p_2 = \top_I$
$\neg_I$	$2 \to 2$	$\neg_I \circ p = \top_I \Leftrightarrow p = \bot_I$
$\forall_X$	$2^X \to 2$	$\forall_X \circ \overline{p} \circ y = \top_I \Leftrightarrow \forall x.p \circ \langle x, y\rangle = \top_I$
$\exists_X$	$2^X \to 2$	$\exists_X \circ \overline{p} \circ y = \top_I \Leftrightarrow \exists x : 1 \to X.p \circ \langle x, y\rangle = \top_I$

Table 1 Operators of the Internal Logic

To convert formulas into internal formulas, we need to first convert terms into 'internal terms'. In particular, function symbols will map into arrows of an appropriate sort. For example, if our 'external formula' is on variables $[x : 1 \to \mathbb{N}, y : 1 \to \mathbb{N}]$, then any 'internal term' built as part of this translation will be from $\mathbb{N} \times \mathbb{N}$. In our $\mathbb{N}$-example, the arrow corresponding to y^+ will be $\mathsf{SUC} \circ \pi_2(\mathbb{N}, \mathbb{N})$. In most circumstances, the connection between the function symbol and the arrow will simply be that symbol's definition. For generality's sake, our implementation stores the external-internal correspondence of function and predicate symbols in a simple dictionary data structure.

Our formula-converting function is recursive on the structure of formula, using the semantics of the various connectives and quantifiers given in Table 1. The only built-in predicate, equality, corresponds to the characteristic map of the diagonal monomorphism. For user-defined predicates, such as $<$ over natural numbers, users can store the correspondences manually. The induction steps for the connectives are straightforward. For quantifiers, for example, consider the formula $\forall a : 1 \to A.\ \ a = a_0$. Begin by converting the body $a = a_0$ into a predicate on $[a, a_0]$; and then transpose the output and post-compose with the internal logic operator $\forall_A$. The existential case is similar.

5 Quotients in ETCS and SEAR

In both ETCS and SEAR, we can make a number of definitions, and prove theorems about quotienting by equivalence relations. Here we present our approach in the terminology of SEAR. We only consider full equivalence relations, since partial equivalences become full by

restricting their domains. Our approach does not require any form of the Axiom of Choice. Given a binary relation R on a set A, we say a map $i : Q \to \mathsf{Pow}(A)$ is a quotient with respect to R if it injects Q into the set of relational images of R (which is the set of equivalence classes if R is an equivalence relation). That is,

$$\begin{aligned}&\mathsf{Quot}(R : A \nrightarrow A, i : Q \to \mathsf{Pow}(A)) \Leftrightarrow \\ &\quad \mathsf{Inj}(i) \ \wedge \ \forall s \in \mathsf{Pow}(A).\ (\exists q \in Q.\ s = i(q)) \Leftrightarrow \exists a \in A.\ s = \{b \mid \mathsf{Holds}(R, a, b)\}\end{aligned}$$

In contrast to HOL, where any injection has an inverse, constructing an inverse of an injection requires an element witnessing that the domain is non-empty. For an injection i from X to Y, and given an element $x \in X$, we define $\mathsf{LINV}(i, x)(y)$ to map $y \in Y$ to x_0 if $i(x_0) = y$, or to x otherwise. This is then a left inverse of i. If $i : Q \to \mathsf{Pow}(A)$ is a quotient of R, then given any $q_0 \in Q$, the composition of the map $a \mapsto \{b \mid \mathsf{Holds}(R, a, b)\}$ and $\mathsf{LINV}(i, q_0)$ is the quotient map $A \to Q$. We denote the output of this map applied to an element $a \in A$ as $\mathsf{abs}(R, i, q_0, a)$. We write $\mathsf{resp1}(f, R)$ if f agrees on elements related by R and $\mathsf{ER}(R)$ for R an equivalence relation. Then we prove:

$$\begin{aligned}&\mathsf{ER}(R) \wedge \mathsf{resp1}(f, R) \wedge \mathsf{Quot}(R, i) \implies \\ &\quad \forall q_0 \in Q.\ \exists! \overline{f} : Q \to B.\ \forall a \in A.\ \overline{f}(\mathsf{abs}(R, i, q_0, a)) = f(a)\end{aligned}$$

This does not only allow us to lift functions at the level of elements related by R, but also supports lifting predicates, which can be regarded as maps to 2. For instance, lifting the definition of evenness of a natural number to that of an integer amounts to lifting a map $\mathbb{N} \to 2$ into $\mathbb{Z} \to 2$.

A function into a quotient can be defined by composing with the inverse of the inclusion map and hence is easy to define. The interesting case is when we want to define a function from a product of quotients. In such cases, we realise the product of quotients as a quotient as well in the following way: Given two relations R_1 on A and R_2 on B, we define their product relation as:

$$\mathsf{Holds}(\mathsf{prrel}(R_1, R_2), (a_1, b_1), (a_2, b_2)) \Leftrightarrow \mathsf{Holds}(R_1, a_1, a_2) \wedge \mathsf{Holds}(R_2, b_1, b_2)$$

And given quotients $i_1 : Q_1 \to \mathsf{Pow}(A), i_2 : Q_2 \to \mathsf{Pow}(B)$ of R_1 and R_2, we define a map $\mathsf{ipow2}(i_1, i_2) : Q_1 \times Q_2 \to \mathsf{Pow}(A \times B)$ such that for every pair $(a, b) \in Q_1 \times Q_2$, we have:

$$\mathsf{IN}((a, b), \mathsf{ipow2}(i_1, i_2)(q_1, q_2)) \Leftrightarrow \mathsf{IN}(a, i_1(q_1)) \wedge \mathsf{IN}(b, i_2(q_2))$$

If R_1 and R_2 are both equivalence relations, we have $\mathsf{Quot}(\mathsf{prrel}(R_1, R_2), \mathsf{ipow2}(i_1, i_2))$ Application of this result allows us to define maps such as integer addition and multiplication, and more generally, the group operation in a quotient group.

6 Group Theory

Many mathematical results look neater in theorem-provers based on dependent type theory (DTT), since instead of assuming complicated predicates, we can internalise those predicates as types, thereby shortening the statement. By formalising some group theory, we demonstrate that we can prove similarly neat versions of statements in our simple logic.

We encode a group with underlying set G as an element of $\mathsf{Grp}(\mathsf{G})$. Such a set is constructed from the comprehension schema which injects to the subset of the product $G^{G \times G} \times G^G \times G$ satisfying the usual group axioms. For a group $g \in \mathsf{Grp}(G)$, also by comprehension, we construct the set of its subgroups $\mathsf{sgrp}(g)$ as injected into $\mathsf{Pow}(G)$, and set of its normal

subgroups $\mathsf{nsgrp}(g)$ that injects to $\mathsf{sgrp}(g)$. As groups are encoded by members of sets, it is possible to compare if two groups are equal, *e.g.,* $g_1 = g_2$, with $g_1, g_2 \in \mathsf{Grp}(G)$. However, if $h_1 \in \mathsf{sgrp}(g_1)$ and $h_2 \in \mathsf{sgrp}(g_2)$, we cannot write $h_1 = h_2$ because such an equality will not type check. We hold this to be appropriate because equality is not the correct way to compare abstract structures such as groups. Even if we wanted to work with equality on groups g_1, g_2, we should compare their representatives or define transferring functions like the ones of sort $\mathsf{sgrp}(g_1) \to \mathsf{sgrp}(g_2)$, which map a subgroup of g_1 to a subgroup of g_2.

For a normal subgroup $N \in \mathsf{nsgrp}(g)$, the underlying set of the quotient group $\mathsf{qgrp}(N)$ has as its underlying set the set of all right cosets of N. The function symbol qgrp only needs to take the group N as argument, since the group being quotiented is contained in the sort information of N. The quotient homomorphism $\mathsf{qhom}(N)$ is a member of $\mathsf{ghom}(g, \mathsf{qgrp}(g))$ of all homomorphisms between the original group and the quotient. Its underlying function $\mathsf{homfun}(\mathsf{qhom}(N))$ sends a group element to its coset.

By construction, each underlying function of a homomorphism respects the relation induced by its kernel. Then the first isomorphism theorem can be obtained by instantiating the quotient mapping theorem as in the last section, giving

$$\begin{array}{l}\forall G_1\ G_2\ \ g_1 \in \mathsf{Grp}(G_1)\ \ g_2 \in \mathsf{Grp}(G_2)\ \ f \in \mathsf{ghom}(g_1, g_2).\\ \quad \exists!\ \overline{f} \in \mathsf{ghom}(\mathsf{qgrp}(\mathsf{ker}(f)), g_2).\\ \qquad \mathsf{Inj}(\mathsf{homfun}(\overline{f}))\ \wedge\ \mathsf{homfun}(\overline{f}) \circ \mathsf{qmap}(\mathsf{ker}(f)) = \mathsf{homfun}(f)\end{array} \tag{2}$$

This is a nice illustration of the strengths of the "DTT style".

6.1 Discussion

Our approach to group theory is very different from its counterpart in HOL. Firstly, the HOL type α group is inhabited by values that must record their underlying carrier set. Secondly, the HOL quotient group function takes two α groups and outputs a term of $(\alpha \to \mathsf{bool})$ group, which is proved to satisfy the group axioms if the first term satisfies the group axioms and the second term is a normal subgroup. Further, as HOL types cannot depend on terms, we certainly cannot construct the type of all homomorphisms between two groups.

There is actually a trade-off between choosing the HOL style and the DTT style of stating theorems. Whereas the first isomorphism theorem is clearly better in DTT style (2), the second and third isomorphism theorems in DTT style can look complicated, with a great deal of coercions happening under the covers.[3] Since the HOL quotient group only takes two groups of the same type, we can use exactly the same term for the ambient group and its subgroup, and do not need to construct different terms to regard the same group as subgroups of an ambient group. In this case, the convenience of the HOL style (using assumptions) is evident. We can choose each style in our system, so users can try both approaches and compare them. To find the best form of a statement, we may try combining the two approaches: we do not always have to create a subset once we come up with a new property, but we may use them as assumptions as well.

7 Inductive and Coinductive definitions

We experiment with inductive definitions by mechanising induction on natural numbers, finite sets and lists, and with coinductive definitions by constructing co-lists.

[3] Of course, DTT *systems* offer the ability to write statements in HOL's predicate-heavy style as well.

7.1 Natural numbers, Finite sets and Lists

Our system implements a version of Harrison's [3] inductive relation definition package. To define an inductive subset, we just need to provide the inductive clauses.

For example, there is no primitive natural number object in SEAR. We are given only a set $\mathbb{N}_0$ with an element z_0 and an injection $\mathsf{s}_0 : \mathbb{N}_0 \to \mathbb{N}_0$, where z_0 is not in the range of s_0. To apply our induction tool to cut down $\mathbb{N}_0$ into a natural number object, we firstly define a subset, i.e., a member of the power set $\mathsf{Pow}(\mathbb{N}_0)$, of $\mathbb{N}_0$, by giving two clauses saying that z_0 is in N and if n_0 is in N, then $\mathsf{s}_0(n_0)$ is in N. Using theorem 1 in Section 3 together with the specification rule, we extract the subset of $\mathbb{N}_0$, which consists of elements in N, as a constant term $\mathbb{N}$ of set sort. We call the lifted zero element and successor map 0 and SUC respectively, with SUC obtained by specialising the following lemma with the inclusion from $\mathbb{N}_0$:

$$\begin{array}{l}\forall A\ A_0\ (i : A \to A_0)\ (f_0 : A_0 \to A_0).\\ \quad \mathsf{Inj}(i)\ \wedge\ (\forall a_1.\exists a_2.\ f_0(i(a_1)) = i(a_2))\ \Longrightarrow\\ \qquad \exists! f : A \to A.\ \forall a \in A.\ i(f(a)) = f_0(i(a))\end{array}$$

The constructed $\mathbb{N}$ then can be shown to satisfy the standard induction principle.

$$\begin{array}{l}\mathcal{F}[\mathsf{mem}(\mathbb{N})](0) \wedge (\forall n \in \mathbb{N}.\ \mathcal{F}[\mathsf{mem}(\mathbb{N})](n)\ \Longrightarrow\ \mathcal{F}[\mathsf{mem}(\mathbb{N})](\mathsf{SUC}(n)))\ \Longrightarrow\\ \quad \forall n \in \mathbb{N}.\ \mathcal{F}[\mathsf{mem}(\mathbb{N})](n)\end{array}$$

By instantiating the formula variable $\mathcal{F}$ with concrete properties, we apply the above to perform inductive proofs for ordering and natural number arithmetic. We later use such theorems together with quotient lemmas to construct the set of integers.

Also inductively, we define the predicate $\mathsf{isFinite}$ on members of some set X's power set. The empty subset $\mathsf{Empty}(X)$ is finite, and if $s \in \mathsf{Pow}(X)$ is finite, then the set $\mathsf{Ins}(x, s)$, which inserts x into s, is finite for any $x \in X$. Similar to the counterpart of natural numbers, the principle of induction on the finiteness of a set is proved as:

$$\begin{array}{l}\mathcal{F}[\mathsf{mem}(\mathsf{Pow}(X))](\mathsf{Empty}(X)) \wedge\\ (\forall x\ (xs_0 \in \mathsf{Pow}(X)).\ \mathcal{F}[\mathsf{mem}(\mathsf{Pow}(X))](xs_0)\ \Longrightarrow\ \mathcal{F}[\mathsf{mem}(\mathsf{Pow}(X))](\mathsf{Ins}(x, xs_0)))\ \Longrightarrow\\ \quad \forall xs \in \mathsf{Pow}(X).\ \mathsf{isFinite}(xs)\ \Longrightarrow\ \mathcal{F}[\mathsf{mem}(\mathsf{Pow}(X))](xs)\end{array}$$

We define a relation $\mathsf{Pow}(X) \nrightarrow \mathbb{N}$ relating a subset of X to its cardinality. By induction on finiteness, we prove each subset is related to a unique natural number, which gives us a function $\mathsf{Pow}(X) \to \mathbb{N}$ that sends a finite subset to its cardinality and sends any infinite subset to 0. The output of the function applied on $s \in \mathsf{Pow}(X)$ is denoted as $\mathsf{Card}(s)$. To build lists over a set X as an "inductive type", we firstly define the subset of $\mathsf{Pow}(\mathbb{N} \times X)$ which encodes a list, such sets are finite sets of the form $\{(0, x_1), \cdots, (n, x_n)\}$. The base case of the induction is the empty subset of $\mathsf{Pow}(\mathbb{N} \times X)$, and the step case inserts the set s started with by the pair $(\mathsf{Card}(s), x)$. Using the same approach we constructed $\mathbb{N}$, we form $\mathsf{List}(X)$. It is then straightforward to prove the list induction principle and define the usual list operations like taking the head, tail, n-th element of the list, and map, etc.

7.2 Co-lists

Following the HOL approach, we construct co-lists over sets X, by using maps $\mathbb{N} \to X + 1$ as representatives. The codomain is regarded as X option, whose members either have the form $\mathsf{SOME}(x)$ for $x \in X$, or $\mathsf{NONE}(X)$. First, by dualising the argument we used to define inductive predicates, we define a coinductive predicate on members ($f \in (X+1)^{\mathbb{N}}$) expressing that such a member captures a co-list, and collect the subset where this predicate holds,

defining $\mathsf{list_c}(X)$, just as we did for constructing inductive types. Every term of $\mathsf{list_c}(X)$ has a representative: it is either the constant function mapping to $\mathsf{NONE}(X)$, corresponding to the empty co-list $\mathsf{Nil_c}(X)$, or it is the function obtained by attaching an element $x \in X$ to an existing function encoding a co-list. Almost all the HOL4 definitions can be readily translated into SEAR. The only exception is we cannot write expressions such as $\mathsf{THE}(\mathsf{Hd_c}(l))$. Here $\mathsf{Hd_c}$ is the function that returns $\mathsf{SOME}(x)$ when l is a co-list with element x at its front. If l is the empty co-list, then $\mathsf{Hd_c}(l) = \mathsf{NONE}$. In HOL4, THE is the left-inverse of SOME; in SEAR, our (set) parameter X may be empty, and so there is no general value (even if unspecified) for the head of the co-list. So far, this has not been an obstacle in any of our proofs. The HOL proof of the key co-list principle, which states that two co-lists $l_1, l_2 \in \mathsf{list_c}(X)$ are equal if and only if they are connected by a bisimulation relation R, translates into SEAR, yielding:

$$
\begin{array}{l}
l_1 = l_2 \Leftrightarrow \\
\exists R : \mathsf{list_c}(X) \nrightarrow \mathsf{list_c}(X). \\
\quad \mathsf{Holds}(R, l_1, l_2) \wedge \\
\quad \forall l_3\ l_4 \in \mathsf{list_c}(X).\ \ \mathsf{Holds}(R, l_3, l_4) \implies \\
\qquad\qquad\qquad\qquad (l_3 = \mathsf{Nil_c}(X) \wedge l_4 = \mathsf{Nil_c}(X)) \vee \\
\qquad\qquad\qquad\qquad \exists (h \in X)\ t_1\ t_2.\ \mathsf{Holds}(R, t_1, t_2) \wedge l_3 = \mathsf{Cons_c}(h, t_1) \wedge l_4 = \mathsf{Cons_c}(h, t_2)
\end{array}
$$

where $\mathsf{Nil_c}(X)$ is the empty co-list over X, and $\mathsf{Cons_c}(h, t)$ is the co-list built by putting element $h \in X$ in front of co-list t. We can perform coinductive proofs on co-lists by the theorem above. For instance, the above helps to prove that $\mathsf{Map_c}$ function, with the usual definition, is functorial.

8 Modal Model Theory

In recent work, we developed a mechanisation of some basic modal logic theory [20]. While defining the notion of being preserved under simulation, we observed that if a property of a modal formula is defined in terms of the behaviour of the formula on all models, then such a property cannot be faithfully captured by HOL. Such an issue can be resolved by choosing a dependent sorted foundation and doing the proof in our logic. We demonstrate this here by mechanising the proof that characterises formulas preserved under simulation as those are equivalent to a positive existential formula in SEAR.

Using roughly the general method introduced at the end of Harrison [3], we first construct the "type" (actually a set in SEAR) of modal formulas over variables drawn from the set V. We then denote the set of modal formulas over V as $\mathsf{form}(V)$. A Kripke model on a set W of such formulas is an element of $\mathsf{Pow}(W \times W) \times \mathsf{Pow}(V)^W$ (written as $\mathsf{model}(W, V)$ in the following paragraphs). The first component encodes the model's reachability relation, while the second encodes the variable valuation. Satisfaction of modal formulas can then be defined in the standard way, and if ϕ is satisfied at w in the model M, we write $M, w \Vdash \phi$.

The two key definitions of this proof are that of simulation, and of being preserved under simulation (written as PUS below). The former is identical to its counterpart in HOL, and we write $\mathsf{Sim}(R, M_1, M_2)$ to indicate that R is a simulation from M_1 to M_2. The latter is more interesting. Unlike in HOL, where we can only express a formula being preserved under simulation between models of certain HOL types, forcing the definition to take an extra type

parameter, the definition in SEAR is purely a predicate on formulas:

$$\begin{array}{l}\forall V\ (\phi \in \mathsf{form}(V)).\\ \quad \mathsf{PUS}(\phi) \Leftrightarrow\\ \quad \forall W_1\ W_2\ (R : W_1 \nrightarrow W_2)\ (w_1 \in W_1)\ (w_2 \in W_2)\\ \quad\ (M_1 \in \mathsf{model}(W_1, A))\ \ (M_2 \in \mathsf{model}(W_2, A)).\\ \quad\quad \mathsf{Sim}(R, M_1, M_2) \wedge \mathsf{Holds}(R, w_1, w_2)\ \wedge\ M_1, w_1 \Vdash \phi\ \Longrightarrow\ M_2, w_2 \Vdash \phi\end{array}$$

This convenience is brought about by the fact that our logic allows for quantification over sets, whereas HOL does not allow for quantification over types. Thus, the notion of equivalence of modal formulas only takes two modal formulas as arguments and requires no extra 'type parameter'. Under these definitions, the proofs of both directions of theorem 2.78 in [1] can be faithfully translated, yielding the two formal statements:

$$\begin{array}{l}\forall V\ (\phi \in \mathsf{form}(V))\ (\phi_0 \in \mathsf{form}(V)).\ \mathsf{PE}(\phi_0) \wedge \phi \sim \phi_0\ \Longrightarrow\ \mathsf{PUS}(\phi)\\ \forall V\ (\phi \in \mathsf{form}(V)).\ \mathsf{PUS}(\phi)\ \Longrightarrow\ \exists \phi_0 \in \mathsf{form}(V).\ \mathsf{PE}(\phi_0) \wedge \phi \sim \phi_0\end{array}$$

Clearly, the two directions can be put together into an if-and-only-if, hence giving the full form of the characterisation theorem, which cannot even be stated in HOL.

$$\forall V\ (\phi \in \mathsf{form}(V)).\ \mathsf{PUS}(\phi) \Leftrightarrow \exists f_0 \in \mathsf{form}(V).\ \mathsf{PE}(\phi_0) \wedge \phi \sim \phi_0)$$

9 Existence of Large Sets

Whereas iterating the procedure of taking the power set by infinite times is impossible in HOL due to foundational issues, the collection axiom schema in SEAR makes it possible. The statement of the SEAR collection axiom is formalised as:

$$\begin{array}{l}\exists B\ Y\ (p : B \to A)\ (M : B \nrightarrow Y).\\ \quad (\forall S\ (i : S \to Y)\ (b \in B).\\ \quad\quad \mathsf{isset}(i, \{y \mid \mathsf{Holds}(M, b, y)\})\ \Longrightarrow\ \mathcal{F}[\mathsf{mem}(A), \mathsf{set}](p(b), S)) \wedge\\ \quad (\forall (a \in A)\ X.\ \mathcal{F}[\mathsf{mem}(A), \mathsf{set}](a, X)\ \Longrightarrow\ \exists b.\ p(b) = a)\end{array}$$

with $\mathcal{F}[\mathsf{mem}(A), \mathsf{set}]$ a formula variable, to be instantiated to be a predicate on an element of A and a set.

Using this axiom, we will prove:

$$\forall A.\ \exists P.\ \forall n \in \mathbb{N}.\ \exists i : \mathsf{Pow}^n(A) \to P.\ \mathsf{Inj}(i)$$

Here the $\mathsf{Pow}^n(A)$ is "the" n-th power set of A. Note that the induction principle on natural numbers does not allow us to take a set as an argument, and does not allow the output to be a set as well. To create this function symbol, we start by defining a predicate $\mathsf{nPow}(n, A, B)$, which means B is *an* n-th power set of A. We then prove such B is unique up to bijection, hence the specification rule applies. In the following, we write $\mathsf{P}(s) \in \mathsf{Pow}(\mathsf{Pow}(A))$ for the set of subsets of $s \in \mathsf{Pow}(A)$. For $s_1 \in \mathsf{Pow}(A)$ and $s_2 \in \mathsf{Pow}(B)$, we write $|s_1| = |s_2|$ for s_1 and s_2 have the same cardinality. We write $\mathsf{Whole}(A) \in \mathsf{Pow}(A)$ as the subset of A consisting of all members of A.

We define $\mathsf{nPow}(n, A, B)$ if there exists a set X and a function $f : X \to \mathbb{N}$ such that $|f^{-1}(0)| = |\mathsf{Whole}(A)|$, $|f^{-1}(n)| = |\mathsf{Whole}(B)|$, and for each $n_0 < n$, $|f^{-1}(n_0^+)| = |\mathsf{P}(f^{-1}(n_0))|$. Such a function f records a sequence of power set relation, in this case, we write $\mathsf{nPowf}(n, A, B, f)$. By induction on n_0, $\mathsf{nPow}(n, A, B, f)$, implies $\mathsf{nPow}(n_0, A, \mathsf{m2s}(f^{-1}(n_0)), f)$ for each $n_0 \leq n$.

If $\mathsf{nPow}(n, A, B_1)$ and $\mathsf{nPow}(n, A, B_2)$, we can infer B_1 and B_2 have the same cardinality by induction on n. The base case is trivial. Assume $f_1 : X_1 \to \mathbb{N}$ witnesses $\mathsf{nPow}(n^+, A, C_1)$ and $f_2 : X_2 \to \mathbb{N}$ witnesses $\mathsf{nPow}(n^+, A, C_2)$, as $n < n^+$, we have f_1, f_2 witness that their preimage at n is an n-th powerset of A, and hence by inductive hypothesis has the same cardinality. Therefore, the cardinality of C_1 and C_2 are equal as power sets of sets with the same cardinality.

Now we prove the existence of these iterated power sets. Suppose we have $\mathsf{nPowf}(n, A, B, f_0 : X \to \mathbb{N})$, we construct $f' : \mathsf{Pow}(X + 1) \to \mathbb{N}$ such that $\mathsf{nPowf}(n^+, A, \mathsf{Pow}(B), f')$. Define $f : X \to \mathbb{N}$ such that as if $f_0(x) \leq n$ then $f(x) = f_0(x)$, else $f(x) = n^{++}$, then we have $\mathsf{nPowf}(n, A, B, f : X \to \mathbb{N})$, and n^+ is not in the range of f. According to the definition of nPow, there exists an injection $B \to X$, and thus an injection $i : \mathsf{Pow}(B) \to \mathsf{Pow}(X)$. We define the function $f' : \mathsf{Pow}(X + 1) \to \mathbb{N}$ as:

$$f'(s) = \begin{cases} f(x) & \text{if } s = \{\mathsf{SOME}(x)\} \\ n^+ & \text{if } \exists xs \in \mathsf{Pow}(X).\ i(xs) = s_0 \wedge s = \{\mathsf{NONE(X)}\} \cup s_0 \\ n^{++} & \text{else} \end{cases}$$

It follows that $|f'^{-1}(n_0)| = |f^{-1}(n_0)|$ for $n_0 \leq n$, and the preimage of n^+ is a copy of $\mathsf{Pow}(B)$, so f' witnesses $\mathsf{Pow}(B)$ is the n^+-th power set of A.

To prove the existence of the large set. By specialising the axiom of collection, we obtain a set B, a function $p : B \to \mathbb{N}$, a set Y and a relation $M : B \nrightarrow Y$ satisfying:

$$\begin{array}{l} (\forall S\ (i : S \to Y)\ (b \in B).\ \mathsf{isset}(i, \{y \mid \mathsf{Holds}(M, b, y)\}) \implies \mathsf{nPow}(p(b), A, S)) \wedge \\ (\forall n \in \mathbb{N}\ \ X.\ \mathsf{nPow}(n, A, X)) \implies \exists b \in B.\ p(b) = n) \end{array}$$

The set Y is the large set we want to construct. For any $n \in \mathbb{N}$, we have $\mathsf{nPow}(n, A, \mathsf{Pow}^n(A))$, and thus there exists a $b \in B$ with $p(b) = n$. For this b, Let $H(b)$ denotes the set of elements y such that $\mathsf{Holds}(M, b, y)$, then $\mathsf{minc}(H(b))$ gives an injection $\mathsf{m2s}(H(b)) \to Y$. As $\mathsf{nPow}(n, A, \mathsf{m2s}(H(b)))$ and also $\mathsf{nPow}(n, A, \mathsf{Pow}^n(A))$, by uniqueness proved above, there exists a bijection $j : \mathsf{Pow}^n(A) \to \mathsf{m2s}(H(b))$. The composition $\mathsf{minc}(H(b)) \circ j$ is the desired injection.

10 Conclusion

Our work aims to enable the direct encoding of first-order mathematical foundations based on axioms, while keeping the underlying logic as simple as possible.

We have already seen that it is useful to explore various mathematical foundations: by experimenting with SEAR, we overcome two well-known shortcomings of HOL. Firstly, because it allows us to quantify over types, SEAR enables us to prove the full version of our previous theorem in modal model theory. Secondly, using the collection axiom of SEAR, we overcome the cardinality shortcoming of types in HOL. We are unaware of any other work addressing this issue.

10.1 Related Work

Quantification of types in HOL has been addressed in work by Melham [10] and Homeier [4]. Both pieces of work propose to extend the HOL logic, but neither goes so far as to introduce dependencies linking terms to types or sorts.

There is much existing work on logical systems with dependent sorts. All of them are designed with an aim different from ours. For instance, FOLDS (Makkai [7]) is designed to

only be able to capture mathematical theories where truth is invariant under a certain notion of isomorphism, and hence its expressive power is meant to be more restrictive. In particular, in its standard presentation, FOLDS works only with predicate symbols but not function symbols. DFOL (Rabe [15]) does not support expressing axiom schemata at the object level, and is constructed within LF's dependently typed environment. Compared with both, our work is customized for directly embedding axiomatic systems. Our system is simple, and can be easily implemented, not relying on an ambient implementation of dependent types.

When investigating a particular mathematical foundation, one approach is to implement the logic in a domain-specific manner. For instance, Cáccamo and Winskel [2], and New and Licata [12], both present logics addressing formalisation of proofs in category theory by designing particular type theories. In contrast, our system is a generic theorem-prover, making it easier to compare multiple systems, and to reuse proofs.

Isabelle (Paulson [13]) was famously designed as a generic theorem-proving system, and one of the sample object logics distributed with it is `MLTT` (Martin-Löf Type Theory). Nonetheless, as the ambient types of the Isabelle meta-level are those of simple type theory, working with dependent types in Isabelle requires the interesting type structures and typing judgements to appear at the level of terms. Once this compromise has been made, handling equalities, for example, becomes quite tedious; our system's restrictive handling of equality gains us a great deal of pragmatic power: simple rewriting, and a straightforward notion of matching.

10.2 Future Work

In future work, we will publish our formalisation of McLarty's CCAF [8] and our mechanisation of the proof theory of the system.

The existence of large sets is a consequence of the SEAR collection axiom, and is already stronger than what is possible in HOL, but there is still more that is possible in SEAR. In particular, from its collection axiom, we can follow Shulman [16] to derive the replacement schema, and get a minimal set from amongst these large sets. This would enable more transfinite constructions, such as that of Beth cardinals, which we plan to work on next. Moreover, we are interested in implementing a uniform approach of applying an axiomatic foundation as a metatheory, and hence developing the 'two-layered' workspace discussed by McLarty and Rodin [9]. We are also looking forward to mechanising some of the theorems in the list "Formalising 100 Theorems" [19] in either SEAR or ETCS. Finally, it would be interesting to support the usage of different ambient logics, so people might, in particular, choose to do intuitionistic proofs as well.

References

1 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001. doi:10.1017/CBO9781107050884.

2 Mario Cáccamo and Glynn Winskel. A higher-order calculus for categories. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics*, pages 136–153, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

3 John Harrison. Inductive definitions: automation and application. In Phillip J. Windley, Thomas Schubert, and Jim Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications: Proceedings of the 8th International Workshop*, volume 971 of *Lecture Notes in Computer Science*, pages 200–213, Aspen Grove, Utah, 1995. Springer-Verlag.

649 **4** Peter V. Homeier. The HOL-Omega logic. In Stefan Berghofer, Tobias Nipkow, Christian
650 Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics*, pages 244–259,
651 Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
652 **5** F. William Lawvere. An elementary theory of the category of sets. *Proceedings of*
653 *the National Academy of Sciences*, 52(6):1506–1511, 1964. URL: https://www.pnas.org/
654 doi/abs/10.1073/pnas.52.6.1506, arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.
655 52.6.1506, doi:10.1073/pnas.52.6.1506.
656 **6** F. William Lawvere. The category of categories as a foundation for mathematics. In S. Eilenberg,
657 D. K. Harrison, S. MacLane, and H. Röhrl, editors, *Proceedings of the Conference on Categorical*
658 *Algebra*, pages 1–20, Berlin, Heidelberg, 1966. Springer Berlin Heidelberg.
659 **7** Michael Makkai. First order logic with dependent sorts, with applications to category theory.
660 Available from https://www.math.mcgill.ca/makkai/folds/foldsinpdf/FOLDS.pdf, 1995.
661 **8** Colin McLarty. Axiomatizing a category of categories. *The Journal of Symbolic Logic*,
662 56(4):1243–1260, 1991. URL: http://www.jstor.org/stable/2275472.
663 **9** Colin McLarty and Andrei Rodin. A discussion between Colin McLarty and Andrei Rodin about
664 structuralism and categorical foundations of mathematics, 2013. URL: http://philomatica.
665 org/wp-content/uploads/2013/02/colin.pdf.
666 **10** Thomas F. Melham. The HOL logic extended with quantification over type variables. In Luc
667 J. M. Claesen and Michael J. C. Gordon, editors, *Higher Order Logic Theorem Proving and*
668 *its Applications*, IFIP Transactions A: Computer Science and Technology, pages 3–17. North-
669 Holland, Amsterdam, 1993. URL: https://www.sciencedirect.com/science/article/pii/
670 B9780444898807500073, doi:https://doi.org/10.1016/B978-0-444-89880-7.50007-3.
671 **11** Elliott Mendelson. *Introduction to Mathematical Logic*. Princeton: Van Nostrand, 1964.
672 **12** Max S. New and Daniel R. Licata. A formal logic for formal category theory, 2022. URL:
673 https://arxiv.org/abs/2210.08663, doi:10.48550/ARXIV.2210.08663.
674 **13** Lawrence Charles Paulson. Isabelle: The next 700 theorem provers. *ArXiv*, cs.LO/9301106,
675 2000.
676 **14** W. V. Quine. New foundations for mathematical logic. *The American Mathematical Monthly*,
677 44(2):70–80, 1937. URL: http://www.jstor.org/stable/2300564.
678 **15** Florian Rabe. First-order logic with dependent types. In Ulrich Furbach and Natarajan
679 Shankar, editors, *Automated Reasoning*, pages 377–391, Berlin, Heidelberg, 2006. Springer
680 Berlin Heidelberg.
681 **16** Michael Shulman. Comparing material and structural set theories. *Annals of Pure and Applied*
682 *Logic*, 170(4):465–504, 2019. URL: https://www.sciencedirect.com/science/article/pii/
683 S0168007218301295, doi:https://doi.org/10.1016/j.apal.2018.11.002.
684 **17** Michael Shulman. SEAR, 2022. URL: https://ncatlab.org/nlab/show/SEAR.
685 **18** Andrzej Trybulec. Tarski-Grothendieck set theory, 1990. URL: http://mizar.uwb.edu.pl/
686 JFM/Axiomatics/tarski.html.
687 **19** Freek Wiedijk. Formalizing 100 theorems, 2013. URL: https://www.cs.ru.nl/~freek/100/.
688 **20** Yiming Xu and Michael Norrish. Mechanised modal model theory. In Nicolas Peltier
689 and Viorica Sofronie-Stokkermans, editors, *International Joint Conference on Automated*
690 *Reasoning (IJCAR), Paris, France*, volume 12166 of *Lecture Notes in Computer Science*, pages
691 518–533. Springer, 2020. doi:10.1007/978-3-030-51074-9_30.