

# Formalizing Abstract Algebra in Type Theory with Dependent Records

Xin Yu<sup>1</sup>, Aleksey Nogin<sup>1</sup>, Alexei Kopylov<sup>2</sup>, and Jason Hickey<sup>1</sup>

<sup>1</sup> Department of Computer Science, California Institute of Technology  
M/C 256-80, Pasadena, CA 91125, USA

<sup>2</sup> Department of Computer Science, Cornell University, Ithaca, NY 14853, USA

**Abstract.** Our goal is to develop a general formalization of abstract algebra suitable for a general reasoning. One of the most common ways to formalize abstract algebra is to make use of a module system to specify an algebra as a theory. However, this approach suffers from the fact that modules are usually not first-class objects in the formal system. In this paper, we develop a new approach based on the use of dependent record types. In our account, all algebraic structures are first-class objects, with the natural subtyping properties due to record extension (for example, a group is a subtype of a monoid). Our formalization cleanly separates the axiomatization of the algebra from its typing properties, corresponding more closely to a textbook presentation.

## 1 Introduction

The notions of abstract algebra are central to many areas of mathematics. Abstract algebra has also made many contributions to computer science, including abstract data types and object-oriented programming. Therefore we believe that having an *easily accessible* formalization of the abstract algebra notions in a formal system could be of great value. Formalization of many areas of mathematics could be based on such abstract algebra theory; and formalization of many computer science concepts could be modeled after it.

In this paper we explore a formalization of the abstract algebra concepts in type theory, based on the notion of an extensible record type. This is a part of larger effort to explore different approaches to formalizing basic abstract algebra concepts to find out which approach works the best. In particular, we want to find a formalization that is as close as possible to the standard text-book exposition of abstract algebra and corresponds closely to our intuition.

The group concept is the most central of all algebraic concepts; indeed, the techniques employed in group theory have long served as a pattern for other topics in algebra [4]. Therefore it is not surprising that the axiomatization of the abstract algebra concepts presented in this paper is centered around the axiomatization of the group and axiomatizations of other abstract algebra notions are very similar to the one of the group.

Of course, we are far from being the first ones to work with abstract algebra or group theory in a formal system. For example, Gunter working with HOL [7]

has proved group isomorphism theorems and shown the integers mod  $n$  to be an implementation of abstract groups [8]. Jackson has implemented computational abstract algebra in the NuPRL system [3,13,14]. And in IMPS [5] there is a notion of *little theories* [6] which they use for proving theorems about groups and rings.

One difference is that most, if not all, previous efforts concentrated on proving a specific “big theorem” and their main criterion for picking a certain axiomatization of basic concepts was to simplify the proof of that theorem. In contrast, in this paper we are primarily concerned with creating an axiomatization that would be natural and easily accessible to the widest possible range of theorems and applications.

Kammüller and Paulson [15] have proved Sylow’s theorem in Isabelle-HOL, a large proof that required mechanizing a great deal of group theory. As Kammüller and Paulson state, one of the most common ways to formalize algebra is to make use of a module system, specifying the group as a theory. The main drawback of this approach is that modules are not first-class objects: it is difficult to quantify over them. To address this problem, Kammüller and Paulson proposed the development of a “section” concept, which would be like a module, but would be a first-class object that would capture only the formal parts of that module. In this paper, we propose the use of dependent record types to address this specific problem. In our formalization, records are used for providing a first-class formalization of algebraic objects, while also providing properties like inheritance and subtyping.

## 2 MetaPRL and Dependent Records

### 2.1 MetaPRL

MetaPRL [9,12] is the latest system in the PRL family of theorem provers. The MetaPRL system combines the properties of an interactive LCF-style tactic-based proof assistant, a logical programming environment, and a formal methods programming toolkit. MetaPRL is also a logical framework that allows for reasoning in different logical theories. Its most extensively developed and most frequently used theory is a variation of the NuPRL intuitionistic type theory [3] (which in turn is based on the Martin-Löf type theory [17]).

### 2.2 Dependent Records

Records are tuples of labeled fields. We use the following syntax for records:

$$\{x_1 = a_1; \dots; x_n = a_n\}$$

where  $x_1, \dots, x_n$  are labels of the string type and  $a_1, \dots, a_n$  are corresponding fields. Each field of a record may have its own type. If each  $a_i$  has type  $A_i$ , then the above record has the type

$$\{x_1 : A_1; \dots; x_n : A_n\}.$$

---

*Reduction rules*  $\{r; x = a\} .x \longrightarrow a$                        $\{r; y = a\} .x \longrightarrow r.x$  when  $x \neq y$   
In particular:  $\{x_1 = a_1; \dots; x_n = a_n\} .x_i \longrightarrow a_i$  when all  $x_i$ 's are distinct.

*Type formation*

$$\frac{\Gamma \vdash R_1 \text{ Type} \quad \Gamma; self: R_1 \vdash R_2[self] \text{ Type}}{\Gamma \vdash \{R_1; R_2[self]\} \text{ Type}}$$

*Introduction (membership rules)*

$$\frac{\Gamma \vdash r \in R_1 \quad \Gamma \vdash r \in R_2[r] \quad \Gamma \vdash \{R_1; R_2[self]\} \text{ Type}}{\Gamma \vdash r \in \{R_1; R_2[self]\}}$$

*Elimination (inverse typing rules)*

$$\frac{\Gamma \vdash r \in \{R_1; R_2[self]\}}{\Gamma \vdash r \in R_1 \quad \Gamma \vdash r \in R_2[r]}$$

**Table 1.** Inference rules for dependent records

---

In *dependent records* (or more formally dependently typed records) the type of fields may depend on values of the previous fields. In general a dependent record type has the following form

$$\{x_1 : A_1; x_2 : A_2[x_1]; \dots; x_n : A_n[x_1, \dots, x_{n-1}]\}.$$

Formally speaking, in **MetaPRL** we have two operators for constructing records:

- the constant  $\{\}$  for the empty record;
- the ternary operation  $\{r; x = a\}$  for the record extension.

We also have two constructors to form record types:

- the constant  $\{\}$  for the empty record type (that contains *all* records);
- the ternary operation  $\{self : R; x : A[self]\}$ .

Here variable *self* is bounded in *A*. It refers to the record itself as a record of the type *R*. For example we may write something like

$$\{self: \{x: A\}; y: B[self.x]\}.$$

When we use the name “self” for this variable, we can use the shortening  $\{R; A[self]\}$  for this type. Also we omit “self.” in the body of *A*, e.g. instead of  $\{R; A[self.x; self.y]\}$  we write just  $\{R; A[x, y]\}$ . We assume that the ternary operations are left associative. For example, we write  $\{x: A; y: B; z: C\}$  instead of  $\{\{\{\}\}; x: A\}; y: B\}; z: C\}$ .

There is also a binary operation  $r.x$  for field selection, such that

$$\{x_1 = a_1; \dots; x_n = a_n\}.x_i \leftrightarrow a_i.$$

In MetaPRL dependent records are defined using another primitive operation: dependent intersection [16]. The basic rules of our record calculus are shown in Table 1.

### 3 Formalization of Group Theory

The record calculus presented above gives us a natural way of formalizing the basic group theory definitions. It allows us to formalize the type restrictions on group operators (which are often kept implicit in mathematical textbooks) separately from the formalization of the actual axioms.

#### 3.1 Formalization of Group-like Objects

**Groupoid** With dependent record, we formalize groupoid as:

$$Groupoid_i \leftrightarrow \{\text{Car} : \mathbb{U}_i; * : \text{Car} \rightarrow \text{Car} \rightarrow \text{Car}\}.$$

**Semigroup** Since a semigroup is a groupoid in which the binary operation is associative, we use a predicate to describe the constraint:

$$isSemigroup(g) \leftrightarrow \forall x : \text{Car}_g. \forall y : \text{Car}_g. \forall z : \text{Car}_g. (x *_g y) *_g z = x *_g (y *_g z) \in \text{Car}_g.$$

Then the set of semigroups can be defined as

$$Semigroup_i \leftrightarrow \{g : Groupoid_i \mid isSemigroup(g)\}.$$

Note that  $\{x : A \mid P[x]\}$  is a standard type constructor for “set” in the NuPRL type theory, the elements of which are those elements  $x \in A$  where the proposition  $P[x]$  is true.

Obviously we can derive that a semigroup is also a groupoid.

**Monoid** A monoid is a semigroup with an identity element. So we first extend the dependent record  $Groupoid_i$  with an extra field - the identity field,

$$Premonoid_i \leftrightarrow \{Groupoid_i; 1 : \text{Car}\},$$

and then define a predicate describing the constraint for a monoid

$$isMonoid(g) \leftrightarrow isSemigroup(g) \wedge \forall x : \text{Car}_g. (1_g *_g x = x \in \text{Car}_g \wedge x *_g 1_g = x \in \text{Car}_g).$$

Then the set of monoids can be defined as

$$Monoid_i \leftrightarrow \{g : Premonoid_i \mid isMonoid(g)\}.$$

From our definition, a monoid is also a semigroup.

### 3.2 Formalization of Groups

For groups, we extend  $Premonoid_i$  with an “inverse operation” field,

$$Pregroup_i \leftrightarrow \{Premonoid_i; \text{inv}: \text{Car} \rightarrow \text{Car}\},$$

and define the constraints for a group

$$\begin{aligned} isGroup(g) \leftrightarrow & isSemigroup(g) \wedge \forall x: \text{Car}_g. (1_g *_g x = x \in \text{Car}_g) \wedge \\ & \forall x: \text{Car}_g. (x_g^{-1} *_g x = 1_g \in \text{Car}_g) \end{aligned}$$

where  $x_g^{-1}$  is the pretty-printed format of  $\text{inv}_g x$ .

Then similar as the set of semigroups and the set of monoids, the set of groups is defined as

$$Group_i \leftrightarrow \{g: Pregroup_i \mid isGroup(g)\}.$$

The corresponding inference rules for groups are presented in Table 2.

**Theorem 1.** *All rules in Table 2 are derivable from the definitions given above.*

*Proof.* All these rules were derived in the MetaPRL system.

Now we can prove a set-and-binary-operation combination, for example,  $\langle \mathbb{Z}, + \rangle$ , is a group, i.e.,

$$\{\text{Car} = \mathbb{Z}; * = \lambda x. \lambda y. (x + y); 1 = 0; \text{inv} = \lambda x. (-x)\} \in Group_i$$

by applying the *group introduction* rule and then the *Pregroup introduction* and *isGroup introduction* rules.

Many properties of groups are immediate under the elimination rules, like those in Table 3.

We have also proved many useful theorems about groups in MetaPRL. For example, the left inverse/identity is also the right inverse/identity; a group is also a monoid; the identity is unique; the inverse operation is unique; for any  $g \in Group_i$  and  $a, b, c \in \text{Car}_g$ ,  $a *_g b = a *_g c \in \text{Car}_g$  implies  $b = c \in \text{Car}_g$ , and  $(a *_g b)_g^{-1} = b_g^{-1} *_g a_g^{-1} \in \text{Car}_g$ .

In MetaPRL, these theorems are proved in a straightforward way. The basic idea is similar to that done by hand, but since MetaPRL provides some automated reasoning, some proofs might be easier. For illustration, we present a proof of one of the theorems.

Suppose we have already proved that the left inverse is also the right inverse, and now we want to prove the left identity is also the right identity

$$\frac{\Gamma \vdash g \in Group_i \quad \Gamma \vdash a \in \text{Car}_g}{\Gamma \vdash a *_g 1_g = a \in \text{Car}_g}.$$

Our idea for proving it is

$$a *_g 1_g = a *_g (a_g^{-1} *_g a) = (a *_g a_g^{-1}) *_g a = 1_g *_g a = a,$$

---

**Type formation**

<i>Pregroup</i>	$\frac{}{\Gamma \vdash \text{Pregroup}_i \text{ Type}}$
<i>isGroup</i>	$\frac{\Gamma \vdash \text{Car}_g \text{ Type} \quad \Gamma; x: \text{Car}_g; y: \text{Car}_g \vdash x *_g y \in \text{Car}_g \quad \Gamma \vdash 1_g \in \text{Car}_g \quad \Gamma; x: \text{Car}_g \vdash x_g^{-1} \in \text{Car}_g}{\Gamma \vdash \text{isGroup}(g) \text{ Type}}$
<i>Group</i>	$\frac{}{\Gamma \vdash \text{Group}_i \text{ Type}}$

**Introduction**

<i>Pregroup</i>	$\frac{\Gamma \vdash g \in \{\text{Car}: \mathbb{U}_i; *: \text{Car} \rightarrow \text{Car} \rightarrow \text{Car}; 1: \text{Car}; \text{inv}: \text{Car} \rightarrow \text{Car}\}}{\Gamma \vdash g \in \text{Pregroup}_i}$
<i>isGroup</i>	$\frac{\Gamma \vdash \text{Car}_g \text{ Type} \quad \Gamma \vdash \text{isSemigroup}(g) \quad \Gamma; x: \text{Car}_g \vdash 1_g *_g x = x \in \text{Car}_g \quad \Gamma; x: \text{Car}_g \vdash x_g^{-1} *_g x = 1_g \in \text{Car}_g}{\Gamma \vdash \text{isGroup}(g)}$
<i>Group</i>	$\frac{\Gamma \vdash g \in \text{Pregroup}_i \quad \Gamma \vdash \text{isGroup}(g)}{\Gamma \vdash g \in \text{Group}_i}$

**Elimination**

<i>Pregroup</i>	$\frac{\Gamma; g: \{\text{Car}: \mathbb{U}_i; *: \text{Car} \rightarrow \text{Car} \rightarrow \text{Car}; \text{inv}: \text{Car} \rightarrow \text{Car}\}; \Delta[g] \vdash C[g]}{\Gamma; g: \text{Pregroup}_i; \Delta[g] \vdash C[g]}$
<i>isGroup</i>	$\frac{\Gamma; u: \text{isGroup}(g); w: \forall x: \text{Car}_g. (1_g *_g x = x \in \text{Car}_g); \quad v: \forall x: \text{Car}_g. \forall y: \text{Car}_g. \forall z: \text{Car}_g. (x *_g y) *_g z = x *_g (y *_g z) \in \text{Car}_g; \quad y: \forall x: \text{Car}_g. (x_g^{-1} *_g x = 1_g \in \text{Car}_g); \Delta[u] \vdash C[u]}{\Gamma; u: \text{isGroup}(g); \Delta[u] \vdash C[u]}$
<i>Group</i>	$\frac{\Gamma; g: \text{Pregroup}_i; u: \text{isGroup}(g); \Delta[g] \vdash C[g]}{\Gamma; g: \text{Group}_i; \Delta[g] \vdash C[g]}$

**Table 2.** Inference rules for groups

---

---

*Membership*

$$\frac{\Gamma \vdash g \in \text{Group}_i}{\Gamma \vdash \text{Car}_g \in \mathbb{U}_i} \qquad \frac{\Gamma \vdash g \in \text{Group}_i}{\Gamma \vdash *_g \in \text{Car}_g \rightarrow \text{Car}_g \rightarrow \text{Car}_g}$$

$$\frac{\Gamma \vdash g \in \text{Group}_i}{\Gamma \vdash 1_g \in \text{Car}_g} \qquad \frac{\Gamma \vdash g \in \text{Group}_i}{\Gamma \vdash \text{inv}_g \in \text{Car}_g \rightarrow \text{Car}_g}$$

*Associativity*

$$\frac{\Gamma \vdash g \in \text{Group}_i \quad \Gamma \vdash a \in \text{Car}_g \quad \Gamma \vdash b \in \text{Car}_g \quad \Gamma \vdash c \in \text{Car}_g}{\Gamma \vdash (a *_g b) *_g c = a *_g (b *_g c) \in \text{Car}_g}$$

*Left identity and left inverse*

$$\frac{\Gamma \vdash g \in \text{Group}_i \quad \Gamma \vdash a \in \text{Car}_g}{\Gamma \vdash 1_g *_g a = a \in \text{Car}_g} \qquad \frac{\Gamma \vdash g \in \text{Group}_i \quad \Gamma \vdash a \in \text{Car}_g}{\Gamma \vdash x_g^{-1} *_g a = 1_g \in \text{Car}_g}$$

**Table 3.** Some simple group properties

---

where the second equation holds because of the associativity of  $*_g$  and the third holds because the left inverse is also the right inverse.

To prove it in the MetaPRL proof editor, we first replace  $1_g$  with  $a_g^{-1} *_g a$ , which can be done by a tactic `substT` provided by MetaPRL. The usage is `substT (a = b ∈ A) i`, which replaces all occurrences of the term  $a$  with  $b$  in clause  $i$  ( $i = 0$  implies the conclusion). So we navigate to this rule and apply the `substT (1g = ag-1 *_g a ∈ Carg) 0 thenT autoT` tactic.<sup>3</sup>

Two subgoals are generated. The first one,

$$\frac{\Gamma \vdash g \in \text{Group}_i \quad \Gamma \vdash a \in \text{Car}_g}{\Gamma \vdash 1_g = a_g^{-1} *_g a \in \text{Car}_g},$$

is trivial since we have

$$\frac{\Gamma \vdash g \in \text{Group}_i \quad \Gamma \vdash a \in \text{Car}_g}{\Gamma \vdash a_g^{-1} *_g a = 1_g \in \text{Car}_g}$$

and  $=$  is symmetric. So, by applying the `equalSymT thenT autoT` tactic, this subgoal is proved.

As for the second subgoal,

$$\frac{\Gamma \vdash g \in \text{Group}_i \quad \Gamma \vdash a \in \text{Car}_g}{\Gamma \vdash a *_g (a_g^{-1} *_g a) = a \in \text{Car}_g},$$

we can utilize the associativity property of groups (listed in Table 3) by applying the tactic `substT (a *_g (ag-1 *_g a) = (ag-1 *_g a) *_g a ∈ Carg) 0 thenT autoT`, which

---

<sup>3</sup> The `autoT` tactic performs “automated” proving based on repeated application of several “basic” tactics; and the infix function `thenT` is a tactical used for sequencing [10]: the proof first applies the substitution, and then applies the `autoT` tactic.

generates two new subgoals

$$\frac{\Gamma \vdash g \in \mathit{Group}_i \quad \Gamma \vdash a \in \mathit{Car}_g}{\Gamma \vdash a *_g (a_g^{-1} *_g a) = (a_g^{-1} *_g a) *_g a \in \mathit{Car}_g}$$

and

$$\frac{\Gamma \vdash g \in \mathit{Group}_i \quad \Gamma \vdash a \in \mathit{Car}_g}{\Gamma \vdash (a *_g a_g^{-1}) *_g a = a \in \mathit{Car}_g}.$$

The first one can be eliminated with the use of `equalSymT`. In the second one  $a *_g a_g^{-1}$  can be replaced with  $1_g$  thanks to the right inverse property we have proved. After this substitution, we get the subgoal of proving  $1 *_g a = a$ , trivial by the left identity property (listed in Table 3). This completes the proof of this theorem.

For all the theorems proved as well as their proofs, see [11].

### 3.3 Formalization of Abelian Groups

If we define a predicate

$$\mathit{isCommut}(g) \leftrightarrow \forall x: \mathit{Car}_g. \forall y: \mathit{Car}_g. (x *_g y = y *_g x \in \mathit{Car}_g),$$

then commutative semigroups, commutative monoids, and abelian groups can be defined respectively as

$$\mathit{Csemigroup}_i \leftrightarrow \{g: \mathit{Semigroup}_i \mid \mathit{isCommut}(g)\};$$

$$\mathit{Cmonoid}_i \leftrightarrow \{g: \mathit{Monoid}_i \mid \mathit{isCommut}(g)\};$$

$$\mathit{Abelg}_i \leftrightarrow \{g: \mathit{Group}_i \mid \mathit{isCommut}(g)\}.$$

Their type formation, introduction, and elimination rules are simple. For example, for  $\mathit{Abelg}_i$ , we have

$$\frac{}{\Gamma \vdash \mathit{Abelg}_i \text{ Type}} \qquad \frac{\Gamma \vdash g \in \mathit{Group}_i \quad \Gamma \vdash \mathit{isCommut}(g)}{\Gamma \vdash g \in \mathit{Abelg}_i}$$

$$\frac{\Gamma; g: \mathit{Group}_i; u: \mathit{isCommut}(g); \Delta[g] \vdash C[g]}{\Gamma; g: \mathit{Abelg}_i; \Delta[g] \vdash C[g]}$$

where

$$\frac{\Gamma \vdash \mathit{Car}_g \text{ Type} \quad \Gamma; x, y: \mathit{Car}_g \vdash x *_g y \in \mathit{Car}_g}{\Gamma \vdash \mathit{isCommut}(g) \text{ Type}} \qquad \frac{\Gamma \vdash \mathit{Car}_g \text{ Type} \quad \Gamma; x, y: \mathit{Car}_g \vdash x *_g y = y *_g x \in \mathit{Car}_g}{\Gamma \vdash \mathit{isCommut}(g)}$$

$$\frac{\Gamma; u: \mathit{isCommut}(g); v: \forall x: \mathit{Car}_g. \forall y: \mathit{Car}_g. (x *_g y = y *_g x \in \mathit{Car}_g); \Delta[u] \vdash C[u]}{\Gamma; u: \mathit{isCommut}(g); \Delta[u] \vdash C[u]}.$$

Apparently, if  $g \in \mathit{Abelg}_i$  then  $g \in \mathit{Group}_i$ .

### 3.4 Formalization of Subgroups

First we define the generalized “subStructure” for all groupoids:

$$h \subseteq_{Str} g \leftrightarrow \text{Car}_h \subseteq \text{Car}_g \wedge *_g = *_h \in \text{Car}_h \rightarrow \text{Car}_h \rightarrow \text{Car}_h.$$

Then, for example, the submonoid can be defined as

$$h \subseteq_{Monoid_i} g \leftrightarrow h \in \text{Monoid}_i \wedge g \in \text{Monoid}_i \wedge h \subseteq_{Str} g$$

and the subgroup

$$h \subseteq_{Group_i} g \leftrightarrow h \in \text{Group}_i \wedge g \in \text{Group}_i \wedge h \subseteq_{Str} g.$$

We proved that if  $s \subseteq_{Group_i} g$ , then

1.  $\text{Car}_s$  is closed under  $*_g$ ;
2.  $1_g = 1_s \in \text{Car}_s$ ;
3. for all  $a \in \text{Car}_s$ ,  $a_g^{-1} = a_s^{-1} \in \text{Car}_s$ .

We also proved the intersection of two subgroups is again a subgroup:

$$\frac{\Gamma \vdash s_1 \subseteq_{Group_i} g \quad \Gamma \vdash s_2 \subseteq_{Group_i} g}{\Gamma \vdash \{\text{Car} = \text{Car}_{s_1} \cap \text{Car}_{s_2}; *_g = *_g; 1 = 1_g; \text{inv} = \text{inv}_g\} \subseteq_{Group_i} g}.$$

### 3.5 Formalization of Cyclic Groups

**The group power operation** Suppose  $g$  is a group. For any element  $a \in \text{Car}_g$ , we define

$$a_g^n = \begin{cases} \underbrace{a *_g a *_g \dots *_g a}_n & \text{if } n > 0 \\ e_g & \text{if } n = 0 \\ \underbrace{a_g^{-1} *_g a_g^{-1} *_g \dots *_g a_g^{-1}}_{-n} & \text{if } n < 0 \end{cases}$$

as the power operation of the group  $g$  based on  $a$  ( $a$  is the **base**).

We formalize it with mathematical induction as

$$a_g^n = \begin{cases} a *_g a_g^{n-1} & \text{if } n > 0 \\ e_g & \text{if } n = 0 \\ a_g^{-1} *_g a_g^{n+1} & \text{if } n < 0 \end{cases},$$

where  $n$  is of the integer type. By induction, if  $a$  is in  $\text{Car}_g$ , then  $a_g^n$  is in  $\text{Car}_g$ , and for any  $m, n \in \mathbb{Z}$ ,  $a_g^m *_g a_g^n = a_g^{m+n} \in \text{Car}_g$  and  $(a_g^m)_g^n = a_g^{m \times n} \in \text{Car}_g$ . Also, if  $s \subseteq_{Group_i} g$ , then for all  $a \in \text{Car}_s$  and  $n \in \mathbb{Z}$ ,  $a_g^n = a_s^n \in \text{Car}_s$ .

**The cyclic group** We define a predicate

$$isCyclic(g) \leftrightarrow \exists a : \text{Car}_g. \forall x : \text{Car}_g. \exists n : \mathbb{Z}. (x = a^n \in \text{Car}_g)$$

to describe a group  $g$  is cyclic. Its inference rules are listed below:

$$\begin{array}{l} \text{Type formation} \quad \frac{\Gamma \vdash g \in \text{Group}_i}{\Gamma \vdash isCyclic(g)} \quad \text{Type} \quad \text{Introduction} \quad \frac{\Gamma \vdash g \in \text{Group}_i \quad \Gamma \vdash a \in \text{Car}_g}{\Gamma; x : \text{Car}_g \vdash \exists n : \mathbb{Z}. x = a^n \in \text{Car}_g} \\ \text{Elimination} \quad \frac{\Gamma; x : isCyclic(g); a : \text{Car}_g; \forall x : \text{Car}_g. \exists n : \mathbb{Z}. (x = a^n \in \text{Car}_g); \Delta[x] \vdash C[x]}{\Gamma; x : isCyclic(g); \Delta[x] \vdash C[x]} \end{array}$$

Every cyclic group is abelian. And every non-trivial subgroup of a cyclic group is cyclic.

**The cyclic subgroup** The cyclic subgroup of  $g$  generated by  $a$  is defined as

$$cycSubg(g; a) \leftrightarrow \{ \text{Car} = \{ x : \text{Car}_g \mid \exists n : \mathbb{Z}. x = a^n \in \text{Car}_g \}; * = *_g; 1 = 1_g; \text{inv} = \text{inv}_g \}.$$

We have  $g \in \text{Group}_i$  implies  $cycSubg(g; a) \in \text{Group}_i$  and  $cycSubg(g; a) \subseteq_{\text{Group}_i} g$ .

### 3.6 Formalization of Cosets and Normal Subgroups

We define the left coset and the right coset as

$$Lcoset(s; g; b) \leftrightarrow \{ x : \text{Car}_g \mid \exists a : \text{Car}_s. (x = b *_g a \in \text{Car}_g) \},$$

$$Rcoset(s; g; b) \leftrightarrow \{ x : \text{Car}_g \mid \exists a : \text{Car}_s. (x = a *_g b \in \text{Car}_g) \},$$

and normal subgroup as

$$s \triangleleft_i g \leftrightarrow s \subseteq_{\text{Group}_i} g \wedge \forall x : \text{Car}_g. Lcoset(s; g; x) =_e Rcoset(s; g; x)$$

where  $=_e$  means extensional equality.

Then both cosets are subsets of the carrier of  $g$

$$\frac{\Gamma \vdash s \subseteq_{\text{Group}_i} g \quad \Gamma \vdash b \in \text{Car}_g}{\Gamma \vdash Lcoset(s; g; b) \subseteq \text{Car}_g} \quad \frac{\Gamma \vdash s \subseteq_{\text{Group}_i} g \quad \Gamma \vdash b \in \text{Car}_g}{\Gamma \vdash Rcoset(s; g; b) \subseteq \text{Car}_g}$$

and all subgroups of abelian groups are normal

$$\frac{\Gamma \vdash s \subseteq_{\text{Group}_i} g \quad \Gamma \vdash isCommut(g)}{\Gamma \vdash s \triangleleft_i g}.$$

---

**Type formation**

$$\begin{array}{c}
 \Gamma \vdash f \in \text{Car}_h \rightarrow \text{Car}_g \\
 \Gamma \vdash \text{Car}_h \text{ Type} \quad \Gamma; x: \text{Car}_h; y: \text{Car}_h \vdash x *_h y \in \text{Car}_h \\
 \Gamma \vdash \text{Car}_g \text{ Type} \quad \Gamma; x: \text{Car}_g; y: \text{Car}_g \vdash x *_g y \in \text{Car}_g \\
 \hline
 \Gamma \vdash \text{isGroupHom}(f; h; g) \text{ Type}
 \end{array}$$

$$\begin{array}{c}
 \Gamma \vdash \text{Car}_h \text{ Type} \quad \Gamma; x: \text{Car}_h; y: \text{Car}_h \vdash x *_h y \in \text{Car}_h \\
 \Gamma \vdash \text{Car}_g \text{ Type} \quad \Gamma; x: \text{Car}_g; y: \text{Car}_g \vdash x *_g y \in \text{Car}_g \\
 \hline
 \Gamma \vdash \text{GroupHom}(h; g) \text{ Type}
 \end{array}$$

**Introduction**

$$\begin{array}{c}
 \Gamma \vdash \text{Car}_h \text{ Type} \\
 \Gamma; x: \text{Car}_h; y: \text{Car}_h \vdash f (x *_h y) = f x *_g f y \in \text{Car}_g \\
 \hline
 \Gamma \vdash \text{isGroupHom}(f; h; g)
 \end{array}$$

$$\begin{array}{c}
 \Gamma \vdash h \in \text{Group}_i \quad \Gamma \vdash g \in \text{Group}_i \\
 \Gamma \vdash f \in \text{Car}_h \rightarrow \text{Car}_g \quad \Gamma \vdash \text{isGroupHom}(f; h; g) \\
 \hline
 \Gamma \vdash f \in \text{GroupHom}(h; g)
 \end{array}$$

**Elimination**

$$\begin{array}{c}
 \Gamma; u: \text{isGroupHom}(f; h; g); \Delta[u]; \\
 v: \forall x: \text{Car}_h. \forall y: \text{Car}_h. f (x *_h y) = f x *_g f y \in \text{Car}_g \vdash C[u] \\
 \hline
 \Gamma; u: \text{isGroupHom}(f; h; g); \Delta[u] \vdash C[u]
 \end{array}$$

$$\begin{array}{c}
 \Gamma; f: \text{Car}_h \rightarrow \text{Car}_g; u: \text{isGroupHom}(f; h; g); \Delta[f] \vdash C[f] \\
 \hline
 \Gamma; f: \text{GroupHom}(h; g); \Delta[f] \vdash C[f]
 \end{array}$$

**Table 4.** Inference rules for group homomorphisms.

---

### 3.7 Formalization of Group Mappings

**The group homomorphism** We first define group homomorphisms:

$$\begin{aligned} isGroupHom(f; h; g) &\leftrightarrow \forall x: \text{Car}_h. \forall y: \text{Car}_h. (f(x *_h y) = f x *_g f y \in \text{Car}_g); \\ GroupHom(h; g) &\leftrightarrow \{f: \text{Car}_h \rightarrow \text{Car}_g \mid isGroupHom(f; h; g)\}. \end{aligned}$$

Their inference rules are in Table 4.

Group homomorphisms preserve group structure. Put differently, if  $f$  is a group homomorphism from  $h$  into  $g$ , we might know the structure of  $g$  from that of  $h$ . For example,  $f$  maps  $1_h$  to  $1_g$ , and maps  $a_h^{-1}$  to  $(f a)_g^{-1}$  for  $a \in \text{Car}_h$ ; if  $s$  is a subgroup of  $h$ , then the image of  $s$  under  $f$  is a subgroup of  $g$

$$\frac{\Gamma \vdash g \in Group_i \quad \Gamma \vdash f \in GroupHom(h; g) \quad \Gamma \vdash s \subseteq_{Group_i} h}{\Gamma \vdash \{\text{Car} = \{x: \text{Car}_g \mid \exists y: \text{Car}_h. (x = f y \in \text{Car}_g)\}; * = *_g; 1 = 1_g; \text{inv} = \text{inv}_g\} \subseteq_{Group_i} g};$$

if  $t$  is a subgroup of  $g$ , then the inverse image of  $t$  under  $f$  is a subgroup of  $h$

$$\frac{\Gamma \vdash h \in Group_i \quad \Gamma \vdash f \in GroupHom(h; g) \quad \Gamma \vdash t \subseteq_{Group_i} g}{\Gamma \vdash \{\text{Car} = \{x: \text{Car}_h \mid f x \in \text{Car}_t\}; * = *_h; 1 = 1_h; \text{inv} = \text{inv}_h\} \subseteq_{Group_i} h}.$$

We have proved all these properties of homomorphisms in MetaPRL.

**Other group mappings** Similarly, we can define group monomorphism, group epimorphism, and group isomorphism etc. as

$$\begin{aligned} GroupMono(h; g) &\leftrightarrow \{f: GroupHom(h; g) \mid Injective(f; \text{Car}_h; \text{Car}_g)\}, \\ GroupEpi(h; g) &\leftrightarrow \{f: GroupHom(h; g) \mid Surjective(f; \text{Car}_h; \text{Car}_g)\}, \\ h \cong g &\leftrightarrow \{f: GroupHom(h; g) \mid Bijective(f; \text{Car}_h; \text{Car}_g)\}, \end{aligned}$$

where  $Injective(f; \text{Car}_h; \text{Car}_g)$ ,  $Surjective(f; \text{Car}_h; \text{Car}_g)$ ,  $Bijective(f; \text{Car}_h; \text{Car}_g)$  means  $f$  is injective, surjective, and bijective from  $\text{Car}_h$  to  $\text{Car}_g$  respectively.

If  $f$  is a group epimorphism from  $h$  to  $g$ , then  $h$  is abelian implies  $g$  is abelian. If  $f$  is a group isomorphism, then its reverse mapping is also a group isomorphism.

### 3.8 Formalization of Group Kernels

We define the group kernel of a homomorphism  $f$  from  $h$  to  $g$  as

$$\begin{aligned} GroupKer(f; h; g) &\leftrightarrow \\ &\{\text{Car} = \{x: \text{Car}_h \mid f x = 1_g \in \text{Car}_g\}; * = *_h; 1 = 1_h; \text{inv} = \text{inv}_h\}. \end{aligned}$$

Its introduction rule is

$$\frac{\Gamma \vdash h \in Group_i \quad \Gamma \vdash g \in Group_i \quad \Gamma \vdash f \in GroupHom(h; g)}{\Gamma \vdash GroupKer(f; h; g) \in Group_i}.$$

We proved if  $h \in Group_i$ ,  $g \in Group_i$ , and  $f \in GroupHom(f; h; g)$ , then  $GroupKer(f; h; g) \subseteq_{Group_i} h$ ; what's more,  $GroupKer(f; h; g) \triangleleft_i h$  (because both cosets of  $GroupKer(f; h; g)$  containing any  $x \in \text{Car}_h$  are equal to the set whose element has the same image under  $f$  as  $x$ ). We also proved that  $f$  is a group monomorphism from  $h$  to  $g$  if and only if the kernel of  $f$  contains  $1_h$  alone.

### 3.9 Formalization of Quotient Group

Traditionally, if  $G$  is a group and  $H$  is a normal subgroup of  $G$ , then the collection of all cosets of  $H$  in  $G$  together with the multiplication of cosets in the form

$$(aH)(bH) = abH \quad a, b \in G$$

forms a group called *quotient group*, or *factor group*.

Equivalently the quotient group of  $G$  by  $H$  can also be defined as the set of collections of elements in  $G$  together with  $G$ 's binary operation  $*$ , where two elements  $a, b \in G$  are in the same collection if and only if  $a * b^{-1} \in H$ .

So we define the quotient group of  $g$  by  $h$  as

$$g/h \leftrightarrow \{ \text{Car} = \text{quot } x, y : \text{Car}_g // x *_g y_g^{-1} \in \text{Car}_h; * = *_g; 1 = 1_g; \text{inv} = \text{inv}_g \},$$

where the elements of  $\text{quot } x, y : \text{Car}_g // x *_g y_g^{-1} \in \text{Car}_h$  are the elements of  $\text{Car}_g$ , but the equality is determined by the equivalence relation  $x *_g y_g^{-1} \in \text{Car}_h$ .

Compared with specifying the quotient group as a set of cosets, our formalization using the quotient type has many advantages. If we use the former method, then first  $g/h$  will be in  $\text{Group}_{i+1}$ , not  $\text{Group}_i$ ; second, it is difficult to define operations on cosets, but for quotient type, we have them for free (we only need to prove that they are well-formed); third, we will not be able to prove the *Fundamental Homomorphism Theorem* because for a given coset we will not be able to construct an element of this coset since there is no axiom of choice in intuitionistic type theory.

With our formalization, assuming  $h \triangleleft_i g$ , we have proved  $g/h \in \text{Group}_i$ , and if  $g$  is abelian, then  $g/h$  is abelian. Also,  $\lambda x.x$  is an epimorphism of  $g$  to  $g/h$  with kernel  $h$ . We also proved the *First Isomorphism Theorem for Groups*, which states that if  $k$  is the kernel of a group epimorphism from  $g$  to  $h$ ,  $h$  is isomorphism to the quotient group  $g/k$ .

## 4 Discussion, Conclusions, Future Work

We have shown an initial part of the formalization of abstract algebra that provides a general-purpose account using records to specify first-class algebraic objects. This formalization has shown itself to be quite natural compared with many other methods, and reasoning often corresponds closely to textbook accounts. We believe this methodology can be extended easily to more advanced algebraic objects.

### 4.1 Ring/Field Theory

We have not yet implemented any parts of ring and field theory; however we do believe that we already have a good understanding on how to proceed. The objective is to use a `rename` operation in the record theory that would allow renaming fields of record objects (but not record types). Then we could define

the ring type by simply extending the existing group theory types with extra fields. We could define, for example,

$$\begin{aligned} Prering_i &\leftrightarrow \{Groupoid_i; +: Car \rightarrow Car \rightarrow Car; 0: Car; neg: Car \rightarrow Car\} \\ additive\_group(r) &\leftrightarrow \mathbf{rename}(+ \rightsquigarrow *, 0 \rightsquigarrow 1, neg \rightsquigarrow inv)\{r\} \end{aligned}$$

The proof automation for `rename` should be capable of easily establishing  $\forall r : Prering_i.additive\_group(r) \in Pregroup_i$ . In turn, this would allow us to define

$$\begin{aligned} isRing(g) &\leftrightarrow isSemigroup(g) \\ &\quad \wedge isGroup(additive\_group(r)) \wedge isCommut(additive\_group(r)) \\ &\quad \wedge \forall a, b, c: Car_r. ((a +_r b) *_r c = a *_r c +_r b *_r c \in Car_r \wedge \\ &\quad \quad \quad a *_r (b +_r c) = a *_r b +_r a *_r c \in Car_r) \end{aligned}$$

and allow reusing all the existing group theorems for the additive group of a ring.

## 4.2 Comparison with the Formalization in CZF

We have also formalized group theory in MetaPRL CZF, which is based on Aczel's Constructive Zermelo-Fraenkel set theory [1,2]. In the CZF formalization, groups and other objects are specified as a set of rules [18,19], where the parts of an object are labeled constants. This is adequate for formalizing many objects in abstract algebra; we can derive theorems about them, and we can also formalize specific objects. However, with this method, the formalized objects are not first-class, which makes it impossible to quantify over them. We would need first-class modules to make it work effectively, a feature not yet supported in MetaPRL. In addition, the formalization is awkward because typing axioms are not cleanly separated from the principal algebra axioms. In our type theory formalization, all objects are first-class and the type information is cleanly separated.

On the other hand, set theory is more natural in some cases. For example, types use intensional equality, but we often care more about extensional properties of algebraic objects. We are currently investigating extensional type-theoretic interpretations.

## References

1. Peter Aczel. The type theoretic interpretation of constructive set theory: Inductive definition. In *Logic, Methodology and Philosophy of Science VII*, pages 17–49. Elsevier Science Publishers, 1986.
2. Peter Aczel and Michael Rathjen. Notes on constructive set theory. Technical Report 40, Mittag-Leffler, 2000/2001.
3. Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cramer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the NuPRL Development System*. Prentice-Hall, NJ, 1986.

4. Richard A. Dean. *Elements of Abstract Algebra*. Wiley, New York, 2nd edition, 1966.
5. William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An interactive mathematical proof system. *Journal of Automated Reasoning*, 11:213–248, 1993.
6. William M. Farmer and F. Javier Thayer Joshua D. Guttman. Little theories. In D. Kapur, editor, *Automated-Deduction-CADE-11*, Lecture Notes in Artificial Intelligence, pages 567–581, New York, June 1992. Springer-Verlag.
7. Michael Gordon and Tom Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, Cambridge, 1993.
8. Elsa Gunter. Doing algebra in simple type theory. Technical Report MS-CIS-89-38, Logic & Computation 09, Department of Computer and Information Science, Moore School of Engineering, University of Pennsylvania, Jun 1989. Distributed with the HOL system in the directory Training/studies/intmod/doingalgpaper.
9. Jason Hickey, Aleksey Nogin, Robert L. Constable, Brian E. Aydemir, Eli Barzilay, Yegor Bryukhov, Richard Eaton, Adam Granicz, Alexei Kopylov, Christoph Kreitz, Vladimir N. Krupski, Lori Lorigo, Stephan Schmitt, Carl Witty, and Xin Yu. *MetaPRL — A modular logical environment*. Accepted to the TPHOLs 2003 Conference, 2003.
10. Jason J. Hickey. *The MetaPRL Logical Programming Environment*. PhD thesis, Cornell University, Ithaca, NY, January 2001.
11. Jason J. Hickey, Brian Aydemir, Yegor Bryukhov, Alexei Kopylov, Aleksey Nogin, and Xin Yu. A listing of MetaPRL theories. <http://metapr1.org/theories.pdf>.
12. Jason J. Hickey, Aleksey Nogin, Alexei Kopylov, et al. *MetaPRL home page*. <http://metapr1.org/>.
13. Paul B. Jackson. Exploring abstract algebra in constructive type theory. In A. Bundy, editor, *Proceedings of the 12<sup>th</sup> International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 590–604, New York, June 1994. Springer-Verlag.
14. Paul B. Jackson. *Enhancing the NuPRL Proof Development System and Applying it to Computational Abstract Algebra*. PhD thesis, Cornell University, Ithaca, NY, January 1995.
15. F. Kammüller and L. C. Paulson. A formal proof of Sylow’s first theorem – an experiment in abstract algebra with Isabelle HOL. *Journal of Automated Reasoning*, 23(3-4):235–264, 1999.
16. Alexei Kopylov. Dependent intersection: A new way of defining records in type theory. In *Proceedings of 18<sup>th</sup> IEEE Symposium on Logic in Computer Science*, 2003. To appear.
17. Per Martin-Löf. Constructive mathematics and computer programming. In *Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North Holland.
18. Xin Yu. Formalizing abstract algebra in constructive set theory. Master’s thesis, California Institute of Technology, 2002.
19. Xin Yu and Jason J. Hickey. Formalizing abstract algebra in constructive set theory. Technical Report caltechCSTR2003.004, California Institute of Technology, Caltech, CA 91125, June 2003.