Formalizing Abstract Algebra in Constructive Set Theory

Xin Yu and Jason J. Hickey

California Institute of Technology, Pasadena, CA 91125, USA

Abstract. We present a machine-checked formalization of elementary abstract algebra in constructive set theory. Our formalization uses an approach where we start by specifying the group axioms as a collection of inference rules, defining a logic for groups. Then we can tell whether a given set with a binary operation is a group or not, and derive all properties of groups constructively from these inference rules as well as the axioms of the set theory. The formalization of all other concepts in abstract algebra is based on that of the group. We give an example of a formalization of a concrete group, the Klein 4-group.

1 Introduction

In this paper, we study abstract algebra in a formal, automated system where proofs can be mechanically generated and verified.

Currently most efforts of formalizing algebra using general purpose theorem provers are grounded in type theory. For example, Gunter working with HOL [1] has shown the integers mod n to be an implementation of abstract groups [2]. Jackson has implemented computational abstract algebra in NuPRL [3, 4]. In IMPS [5] there is a notion of *little theories* [6] which they use for proving theorems about groups and rings. In Coq [7], a (constructive) algebraic hierarchy of groups, rings, fields, etc. has been defined and a constructive proof of the Fundamental Theorem of Algebra was formalized under it [8]. In Isabelle [9] HOL, Kammüller and Paulson formalized group theory and proved the Sylow theorem [10]. Bailey has formalized (part of) Galois theory in LEGO [11].

In practice, set theory, as the standard foundation for mathematics, may have an advantage over type theory. Since there is no extensive tradition of presenting mathematics in a type theoretic setting, many techniques for representing mathematical ideas in a set theoretical language have to be reconsidered for a type theoretical language. In addition, there is much less variation among set theories, in which the well known formulations are defined by a small collection of axioms in the predicate calculus, and for practical purpose, are more or less equivalent [12]. In particular, set theory can often present a convenient framework for developing constructive mathematics using ordinary mathematical concepts. Some theorem provers are based on set theory, like Z [13] and VDM [14], or have a set theoretic component, like MetaPRL [15, 16] and Isabelle [17].

In this paper, we present a formalization of group theory, which is taken as a first step in formalizing abstract algebra, in constructive set theory in MetaPRL. Though we have not gone as far as the work in type theory, considering

the advantage of set theory over type theory and the fact that abstract algebra is traditionally defined in the language of set theory, we take this work as a good start of exploring how well set theory can do in formalizing abstract algebra. Though a classical set-theoretic treatment of abstract algebra has been developed in Mizar [18,19] for over a decade [20], the Mizar software only provides tools for checking correctness of mathematical texts, in other words, Mizar helps one to verify a proof, but not to build a proof. In our work, people can not only verify a proof, but also build proofs by themselves.

We first specify the group axioms as a collection of inference rules, defining a logic for groups. Then we can tell whether a given set with a binary operation is a group or not, and derive all properties of groups from these inference rules as well as the axioms of the set theory. The formalization of other abstract algebra concepts, such as subgroups and homomorphisms, is based on that of the group. We have proved many theorems of group theory in an actual formal system (the MetaPRL system). As a verification of the method and a good illustration of constructivity, such a machine-checked formalization plays an important role in our implementation. In the interest of space, we only give an overview of our formalization and sketch some proofs in this paper; more details can be found in [21,22].

The contributions of this paper include the following:

- A formal, mechanically verifiable account of foundations of abstract algebra in set theory.
- The account is constructive. The computational properties are explicit.

Organization. Section 2 introduces our detailed formalization of group theory. Section 3 gives an example of a concrete group, provides a detailed discussion of some properties of our formalization, and suggests some alternative formalization approaches. Section 4 gives conclusions and some outlines for future work.

1.1 Constructive Set Theory and the CZF module in MetaPRL

Constructive set theory, initiated by John Myhill in 1975 [23], is a theory of sets that, among several others, provides a formal framework for the development of constructive mathematics. It is based on the standard first order language of classical axiomatic set theory and makes no use of constructive notions or objects. Therefore the set theoretical development of constructive mathematics can employ the same ideas, conventions and practice as the set theoretical presentation of classical mathematics. To explain the constructive notion of set, Aczel introduced Constructive Zermelo-Fraenkel set theory, CZF [24], as a variant of Myhill's constructive set theory and showed its constructiveness by interpreting it in Martin-Löf's type theory [25], which was considered a precise foundation for the constructive approach to mathematics.

Hickey [15] formalized CZF in the MetaPRL logical framework and interactive proof assistant [16,21]. First, he implemented in MetaPRL a constructive Martin-Löf style type theory called ITT (which stands for intuitionistic type theory) similar to NuPRL's one [3]. Next, he derived the axioms of CZF from ITT. Since

Aczel's CZF theory is described completely explicitly with a collection of axioms, after sets and these axioms are encoded in MetaPRL's CZF module, we can use them directly without referring to the type theory.

In CZF, all non-propositional elements of the set theory are sets; the numbers and other structures are coded in the usual manner. Sets use an *extensional* equality; two sets are considered equal if they have the same elements. The following concepts have been formalized in MetaPRL's CZF module: extensional set **equality** $s_1 =_s s_2$, **membership** $s_1 \in_s s_2$, first-order **logic** which includes the restricted quantifiers $\forall x \in_s s, P[x]$ and $\exists x \in_s s, P[x]$, and the unrestricted quantifiers $\forall_s x.P[x]$ and $\exists_s x.P[x]$, **subset** $s_1 \subseteq s_2$, **separation** $\{x \in_s s \mid P[x]\}$, **empty set** $\{\}$, **singleton set** $\{s\}$, **binary union** $s_1 \cup s_2$, **general union** $\cup s$, **unordered pairing** (s_1, s_2), and **infinity** (**the natural numbers**) ω. The subscript s in the representations of $s_1 =_s s_2$, etc., means this is set theoretical compared with those type theoretic implementations in MetaPRL's ITT module.

Our formalization of abstract algebra is built on the basis of MetaPRL's CZF implementation.

2 Formalization of Group Theory

2.1 Formalization of Groups

In mathematics, a group $\langle G, * \rangle$ is defined as a set G together with a binary operation $*$ defined on G that satisfies the following axioms:

G1. $*$ is associative: for any $a, b, c \in G$, $(a * b) * c = a * (b * c)$.
G2. There is a left *identity* element $e \in G$ such that for every $a \in G$, $e * a = a$.
G3. For some left identity element e, there is, for every $a \in G$, at least one left *inverse* element a' such that $a' * a = e$.

A group must satisfy all of the group axioms; and all properties of groups are derived from these axioms. Inspired by this mathematical definition, we use a set theoretic axiomatization to formalize groups in CZF. That is, we first specify the group axioms as a collection of inference rules that any group should satisfy; then all properties of groups are *constructively* derived from the axioms of groups as well as the axioms of CZF.

Before that, we need a representation of groups in CZF. Note that a group has four components: carrier set, binary operation, identity, and inverse operation, which together conform to a collection of axioms. To formalize a group G, we can assign it an arbitrary label[1], say g, and in terms of g represent the four components of group G with car_g, e_g, $*_g$, and $'_g$ respectively[2]. We also need a predicate to specify g is a group, which can be denoted by term group_g.

[1] The label for a group can be entirely arbitrary, which leaves group theory open-ended; it only serves as a representation of the group.

[2] In MetaPRL, input is in ASCII format, while output is pretty-printed so that it can be easily understood by those unfamiliar with the MetaPRL syntax. For example, we use `car`$\{'g\}$ for the input of the carrier set of the group in the actual system. In this paper, we try to avoid the ASCII representations and instead use the pretty-printed forms of terms and definitions for clarity.

G1, G2, and G3 must be included in the collection of axioms since they are the most essential in defining groups (see 5-7 in the list below). In addition, since we are working in set theory, some axioms about the well-formedness of the group terms are needed (as number 1 describes). Furthermore, the properties of binary operation, unary operation, etc. are usually taken for granted when working informally on paper; in a mechanized system, they must be stated explicitly, so axioms 2 through 4 are necessary.

1. In the CZF set theory of MetaPRL, anything that is not a proposition should be a set: car_g and e_g are sets; for any sets a and b, $a *_g b$ and a'^g are sets.

$$\frac{\Gamma \vdash g \text{ is a label}}{\Gamma \vdash \mathrm{car}_g \text{ is a set}}, \qquad \frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash b \text{ is a set}}{\Gamma \vdash a *_g b \text{ is a set}},$$

$$\frac{\Gamma \vdash g \text{ is a label}}{\Gamma \vdash e_g \text{ is a set}}, \qquad \frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set}}{\Gamma \vdash a'^g \text{ is a set}}.$$

2. For $*_g$ to be a binary operation on car_g, car_g has to be closed under $*_g$, and *exactly* one element is assigned to each possible ordered pair of elements of car_g under $*_g$, i.e., for any $a, b, c \in \mathrm{car}_g$, if $a = b$, then $a *_g c = b *_g c$ and $c *_g a = c *_g b$.

$$\frac{\begin{array}{c}\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash b \text{ is a set} \\ \Gamma \vdash \mathrm{group}_g \quad \Gamma \vdash a \in_s \mathrm{car}_g \quad \Gamma \vdash b \in_s \mathrm{car}_g\end{array}}{\Gamma \vdash a *_g b \ \in_s \mathrm{car}_g},$$

$$\frac{\begin{array}{c}\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash b \text{ is a set} \quad \Gamma \vdash c \text{ is a set} \\ \Gamma \vdash \mathrm{group}_g \quad \Gamma \vdash a \in_s \mathrm{car}_g \quad \Gamma \vdash b \in_s \mathrm{car}_g \quad \Gamma \vdash c \in_s \mathrm{car}_g\end{array}}{\Gamma \vdash a =_s b \Rightarrow a *_g c =_s b *_g c},$$

$$\frac{\begin{array}{c}\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash b \text{ is a set} \quad \Gamma \vdash c \text{ is a set} \\ \Gamma \vdash \mathrm{group}_g \quad \Gamma \vdash a \in_s \mathrm{car}_g \quad \Gamma \vdash b \in_s \mathrm{car}_g \quad \Gamma \vdash c \in_s \mathrm{car}_g\end{array}}{\Gamma \vdash a =_s b \Rightarrow c *_g a =_s c *_g b}.$$

3. Similarly, for $'^g$ to be a unary operation on car_g, car_g has to be closed under $'^g$ and exactly one element is assigned to each element of car_g under $'^g$.

$$\frac{\begin{array}{c}\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash \mathrm{group}_g \\ \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash a \in_s \mathrm{car}_g\end{array}}{\Gamma \vdash a'^g \ \in_s \mathrm{car}_g}, \qquad \frac{\begin{array}{c}\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash b \text{ is a set} \\ \Gamma \vdash \mathrm{group}_g \quad \Gamma \vdash a \in_s \mathrm{car}_g \quad \Gamma \vdash b \in_s \mathrm{car}_g\end{array}}{\Gamma \vdash a =_s b \Rightarrow a'^g =_s b'^g}.$$

4. e_g is in car_g.

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash \mathrm{group}_g}{\Gamma \vdash e_g \in_s \mathrm{car}_g}$$

5. $*_g$ is associative.

$$\frac{\begin{array}{c}\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash b \text{ is a set} \quad \Gamma \vdash c \text{ is a set} \\ \Gamma \vdash \mathrm{group}_g \quad \Gamma \vdash a \in_s \mathrm{car}_g \quad \Gamma \vdash b \in_s \mathrm{car}_g \quad \Gamma \vdash c \in_s \mathrm{car}_g\end{array}}{\Gamma \vdash a *_g (b *_g c) =_s (a *_g b) *_g c}$$

6. e_g is the left identity.

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash a \in_s \text{car}_g}{\Gamma \vdash e_g *_g a =_s a}$$

7. $'_g$ is the left inverse operation.

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash a \in_s \text{car}_g}{\Gamma \vdash a'^g *_g a =_s e_g}$$

The above rules define the axioms for groups. For any instance of a group, we will need to verify the axioms. However, for general groups, many properties are immediate, such as the left inverse/identity is also the right inverse/identity, and $a * b = a * c$ implies $b = c$ given $a, b, c \in G$ for any group $\langle G, * \rangle$. We also proved somewhat more complicated theorems, such as the uniqueness of the identity, the uniqueness of the inverse operation, and the unique solutions for linear equations $a * x = b$ and $y * a = b$ in the group $\langle G, * \rangle$ where $a, b \in G$.

In MetaPRL, these properties are proved in a straightforward way. The basic idea is similar to that done by hand, but since MetaPRL is an interactive system and provides some automated reasoning, some proofs tend to be easier. Meanwhile, since CZF in MetaPRL is not yet sufficiently automated, some extra effort might be needed in the proofs. For illustration, we present a proof of one of the theorems below.

Suppose we have already proved, from the axioms of groups and CZF, that the left inverse is also the right inverse and now we want to prove the left identity is also the right identity. First we need to add the statement of this theorem to the `Czf_itt_group` module:

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \in_s \text{car}_g}{\Gamma \vdash a *_g e_g =_s a}.$$

Our idea for proving it is

$$a *_g e_g =_s a *_g (a'^g *_g a) =_s (a *_g a'^g) *_g a =_s e_g *_g a =_s a,$$

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 need to replace e_g with $a'^g *_g a$, which can be done by a tactic `setSubstT` provided by MetaPRL's CZF theory. The usage is `setSubstT` $(s_1 =_s s_2)$ i, which replaces all occurrences of the term s_1 with s_2 in clause i ($i = 0$ implies the conclusion). So we navigate to this rule and apply the `setSubstT` $(e_g =_s a'^g *_g a)$ `0 thenT autoT` tactic.[3]

Two subgoals are generated. The first one,

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \in_s \text{car}_g}{\Gamma \vdash e_g =_s a'^g *_g a},$$

[3] 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: the proof first applies the substitution, and then applies the `autoT` tactic [15].

is trivial since we have the axiom

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \in_s \text{car}_g}{\Gamma \vdash a'^g *_g a =_s e_g}$$

and $=_s$ is symmetric. With the use of the `eqSetSymT` tactic provided by MetaPRL, this subgoal is proved.

As for the second subgoal,

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \in_s \text{car}_g}{\Gamma \vdash a *_g (a'^g *_g a) =_s a},$$

we can utilize the associativity axiom G1 by applying the tactic `setSubstT` $(a *_g (a'^g *_g a) =_s (a *_g a'^g) *_g a)$ `0 thenT autoT`, which generates a new subgoal

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \in_s \text{car}_g}{\Gamma \vdash (a *_g a'^g) *_g a =_s a},$$

where $a *_g a'^g$ can be replaced with e_g thanks to the right inverse property we have proved. After this substitution, we get the goal of proving $e_g *_g a =_s a$, trivial by the left identity axiom G2. This completes the proof of the theorem.

For all the theorems proved as well as their proofs, see [22].

2.2 Formalization of Abelian Groups

With the elementary group concepts formalized, we can go ahead with formalizing the other concepts in group theory, such as the abelian group.

We define the predicate "g is an **abelian** group" as[4]

$$\text{abel}_g \leftrightarrow \text{group}_g \wedge \forall a, b \in_s \text{car}_g.(a *_g b =_s b *_g a).$$

Since abel_g implies group_g, all the properties of groups hold for abel_g.

2.3 Formalization of Subgroups

A group can have multiple subgroups. For instance, both $\langle \mathbb{Z}, + \rangle$ and $\langle 2\mathbb{Z}, + \rangle$ are subgroups of $\langle \mathbb{Q}, + \rangle$, where $\mathbb{Z}$ is the integer set, $2\mathbb{Z}$ is the set of even integers, and $\mathbb{Q}$ is the set of rational numbers. To specify a subgroup H of a group G, we need at least two parameters, one specifying the group G and another specifying the subgroup H. Suppose G is represented by the label g; we can use another label, say h, to represent the subgroup H. And then the predicate "h is a **subgroup** of g" can be defined as

$$\text{subgroup}_{h,g} \leftrightarrow \text{group}_h \wedge \text{group}_g \wedge \text{car}_h \subseteq \text{car}_g \wedge \forall a, b \in_s \text{car}_h.(a *_h b =_s a *_g b).$$

The last condition ensures that $*_h$ is the induced operation on car_h from car_g.

We proved that if $\text{subgroup}_{h,g}$, then 1) car_h is closed under $*_g$; 2) $e_h =_s e_g$, and $e_g \in_s \text{car}_h$; 3) for all $a \in_s \text{car}_h$, $a'^h =_s a'^g$ and $a'^g \in_s \text{car}_h$.

[4] $\leftrightarrow$ is definitional equivalence.

2.4 The Power Operation

Before formalizing cyclic subgroups and cyclic groups, let us study the "power" operation which is prerequisite for defining cyclic subgroups and cyclic groups.

Suppose $\langle G, * \rangle$ is a group. For any element $a \in G$, we define

$$a^n = \begin{cases} \underbrace{a * a * ... * a}_{n} & \text{if } n > 0 \\ e & \text{if } n = 0 \\ \underbrace{a' * a' * ... * a'}_{-n} & \text{if } n < 0 \end{cases}$$

as the **power operation** of the group $\langle G, * \rangle$ based on a (a is the **base**).

To formalize it, obviously, we need to use mathematical induction. However, MetaPRL's CZF module does not yet have the integer set or arithmetic on integers defined. Since the MetaPRL definition of CZF is derived from ITT, we can borrow the integers from ITT for use as the induction variable, and also borrow the mathematical induction rules from ITT. This is valid since the induction parameter is n, which means a^n is still a set given a is a set. In other words, under the mathematical induction of ITT, a^0, $a^1, a^2, \ldots$, and $a^{-1}, a^{-2}, \ldots$ are still sets; all set properties and set operations can be applied to them. By doing this we can also utilize the arithmetic part in the MetaPRL type theory, which is currently much more complete than that in the MetaPRL set theory.

Now let us define the power operation in group g as:

$$(a^n)_g =_s \begin{cases} a *_g (a^{n-1})_g & \text{if } n > 0 \\ e_g & \text{if } n = 0 \\ a'^g *_g (a^{n+1})_g & \text{if } n < 0 \end{cases}$$

where n is of the integer type in ITT and the induction is also the one in ITT.

From this definition, we can prove, by induction, that the power operation has the following properties:

1. Well-formedness.

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash n \in \mathbb{Z}}{\Gamma \vdash (a^n)_g \text{ is a set}}$$

2. The membership is preserved.

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash a \in_s \text{car}_g \quad \Gamma \vdash n \in \mathbb{Z}}{\Gamma \vdash (a^n)_g \in_s \text{car}_g}$$

3. The power operation is functional, which means it computes equal set values for equal base arguments.

$$\frac{\begin{array}{c}\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash b \text{ is a set} \\ \Gamma \vdash a \in_s \text{car}_g \quad \Gamma \vdash b \in_s \text{car}_g \quad \Gamma \vdash n \in \mathbb{Z} \quad \Gamma \vdash a =_s b\end{array}}{\Gamma \vdash (a^n)_g =_s (b^n)_g}$$

Also, with the use of arithmetic rules in the ITT type theory, we can prove

$$\frac{\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \text{ is a set} \quad \Gamma \vdash a \in_s \text{car}_g \quad \Gamma \vdash m \in \mathbb{Z} \quad \Gamma \vdash n \in \mathbb{Z}}{\Gamma \vdash (a^m)_g *_g (a^n)_g =_s (a^{m+n})_g}.$$

2.5 Formalization of Cyclic Subgroups

The key to formalizing a cyclic subgroup H of group G generated by a is to build the carrier set $H = \{a^n | n \in \mathbb{Z}\}$ from a where a^n is the power operation of group G. Since it can also be described as the set of all elements in car_g that are equal to a^n for some $n \in \mathbb{Z}$, we use the separation axiom of CZF to define it as

$$\texttt{sep}(x \in_s \text{car}_g | \exists n \in \mathbb{Z}. x =_s (a^n)_g).$$

Note that we are using a type theoretic existential within the construction; the CZF implementation in MetaPRL allows this.

Now we define "h is a **cyclic subgroup** of g generated by a" as

$$\begin{aligned}\text{cyc\_subg}_{h,g,a} \leftrightarrow \quad & \text{group}_h \wedge \text{group}_g \wedge a \in_s \text{car}_g \wedge \forall a, b \in_s \text{car}_h. (a *_h b =_s a *_g b) \\ & \wedge \text{car}_h =_s \text{sep}(x \in_s \text{car}_g | \exists n \in \mathbb{Z}. x =_s (a^n)_g).\end{aligned}$$

Of course, the cyclic subgroup H of G generated by a is a subgroup of G. It can be easily proved here: since $\text{car}_h =_s \text{sep}(x \in_s \text{car}_g | \exists n \in \mathbb{Z}. x =_s (a^n)_g)$, any element in car_h is also in car_g. Thus, car_h is a subset of car_g. All the other requirements for H to be a subgroup of G are satisfied. So, we can conclude $\text{subgroup}_{h,g}$ from $\text{cyc\_subg}_{h,g,a}$.

Equivalently, we can also define $\text{cyc\_subg}_{h,g,a}$ as

$$\text{cyc\_subg}_{h,g,a} \leftrightarrow \text{subgroup}_{h,g} \wedge a \in_s \text{car}_g \wedge \text{car}_h =_s \text{sep}(x \in_s \text{car}_g | \exists n \in \mathbb{Z}. x =_s (a^n)_g).$$

2.6 Formalization of Cyclic Groups

A group G is cyclic if there exists $a \in G$ such that for every $x \in G$ there is an integer n such that $x = a^n$. We define it as

$$\text{cycg}_g \leftrightarrow \text{group}_g \wedge \exists a \in_s \text{car}_g. \forall x \in_s \text{car}_g. \exists n \in \mathbb{Z}. x =_s (a^n)_g.$$

The existential quantifiers in the definition are constructive, so given cycg_g, we know what its generator is and each element is to what power of the generator; and to conclude cycg_g, we need to find its generator first.

Since a cyclic group must be a cyclic subgroup of itself, when its generator is explicitly known, we can define "g is a cyclic group generated by a" as

$$\text{cycg}_{g,a} \leftrightarrow \text{cyc\_subg}_{g,g,a},$$

which is equivalent to (by unfolding $\text{cyc\_subg}_{g,g,a}$)

$$\text{cycg}_{g,a} \leftrightarrow \text{group } g \wedge a \in_s \text{car}_g \wedge \text{car}_g =_s \text{sep}(x \in_s \text{car}_g | \exists n \in \mathbb{Z}. x =_s (a^n)_g).$$

The last condition might look strange at the first glance. What it actually means is the carrier is such a set that any element in it is to some integer power of a.

We proved that cycg_g is equivalent to $\exists a \in_s \text{car}_g.\text{cycg}_{g,a}$.

A cyclic group must be abelian, which is easy to prove formally. Suppose we want to conclude from cycg_g that abel_g. Since group g is cyclic, it has a generator a and for any two elements x and y of car_g, there exist m and n in $\mathbb{Z}$ such that $x =_s (a^m)_g$ and $y =_s (a^n)_g$. g is abelian requires

$$x *_g y =_s y *_g x, \qquad i.e., \qquad (a^m)_g *_g (a^n)_g =_s (a^n)_g *_g (a^m)_g.$$

We already have the result

$$\frac{\begin{array}{c}\Gamma \vdash g \text{ is a label} \quad \Gamma \vdash \text{group}_g \quad \Gamma \vdash a \text{ is a set} \\ \Gamma \vdash a \in_s \text{car}_g \quad \Gamma \vdash m \in \mathbb{Z} \quad \Gamma \vdash n \in \mathbb{Z}\end{array}}{\Gamma \vdash (a^m)_g *_g (a^n)_g =_s (a^{m+n})_g},$$

so it turns out that we need to prove

$$(a^{m+n})_g =_s (a^{n+m})_g,$$

which is trivial by the commutativity of addition.

2.7 Formalization of Cosets and Normal Subgroups

With the separation axiom, we define the left and right cosets as

$$\begin{aligned}\text{lcoset}_{h,g,a} &\leftrightarrow \text{sep}(x \in_s \text{car}_g | \exists y \in_s \text{car}_h.(x =_s a *_g y)) \\ \text{rcoset}_{h,g,a} &\leftrightarrow \text{sep}(x \in_s \text{car}_g | \exists y \in_s \text{car}_h.(x =_s y *_g a)).\end{aligned}$$

We need to specify the following inference rules for them: an element x is in $\text{lcoset}_{h,g,a}$ *iff* it is in car_g and there exists $y \in_s \text{car}_h$ such that $x =_s a *_g y$ where $\text{subgroup}_{h,g}$ and $a \in_s \text{car}_g$; same with $\text{rcoset}_{h,g,a}$ except that $x =_s y *_g a$. Both the left and right cosets are subsets of car_g.

Then we define the predicate "h is a normal subgroup of g" as

$$\text{normal\_subg}_{h,g} \leftrightarrow \text{subgroup}_{h,g} \wedge \forall a \in_s \text{car}_g.(\text{lcoset}_{h,g,a} =_s \text{rcoset}_{s,g,a}).$$

We proved that all subgroups of abelian groups are normal.

2.8 Formalization of Homomorphisms and Isomorphisms

Now let us look at the relationships between groups, which are generally exhibited in terms of a structure-preserving mapping from one group to the other.

For f to be a mapping from H into G, it is required that: 1) $f(a)$ is in G for any a in H; 2) *exactly* one element in G is assigned as $f(a)$ for each a in H.

So, we define "f is a homomorphism from H into G" as

$$\begin{aligned}\text{hom}_{h,g,f} \leftrightarrow \quad & \text{group } h \wedge \text{group } g \wedge \forall a \in_s \text{car}_h.(f(a) \text{ is a set} \wedge f(a) \in_s \text{car}_g) \\ & \wedge \forall a, b \in_s \text{car}_h.(a =_s b \Rightarrow f(a) =_s f(b)) \\ & \wedge \forall a, b \in_s \text{car}_h.(f(a *_h b) =_s f(a) *_g f(b)).\end{aligned}$$

$\text{hom}_{h,g,f}$ is functional in the sense that for any two equal mappings f and f', $\text{hom}_{h,g,f}$ always implies $\text{hom}_{h,g,f'}$.

To illustrate our formalization of the homomorphism, let us study a simple example—the trivial homomorphism, which is a mapping f_e from a group H into a group G such that $f_e(a) = e_G$ for all $a \in H$. Suppose H and G are represented by labels h and g respectively. For any $a, b \in_s \text{car}_h$, $f_e(a) =_s f_e(b) =_s e_g$, so $f_e(a)$ is a set, $f_e(a) \in_s \text{car}_g$, and $a =_s b \Rightarrow f_e(a) =_s f_e(b)$. h is a group implies $a *_h b$ is in car_h, so $f_e(a *_h b) =_s e_g$, which in turn is equal to $e_g *_g e_g =_s f_e(a) *_g f_e(b)$. All the conditions for hom_{h,g,f_e} are satisfied; hom_{h,g,f_e} holds.

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 the identity of H to that of G; it also maps the inverse of an element a in H to the inverse of $f[a]$ in G. And if f is *onto* and H is abelian, then G must also be abelian. In addition, if H_1 is a subgroup of H, then the image $f[H_1]$ of H_1 under f is a subgroup of G; if G_1 is a subgroup of G, then the inverse image $f^{-1}[G_1]$ of G_1 is a subgroup of H. We have proved all these properties of homomorphisms in MetaPRL.

Once homomorphism is formalized, the formalization for isomorphism is trivial since an isomorphism is a bijective homomorphism, i.e., it is a homomorphism that is *one to one* and *onto*. We define "$f : H \to G$ is an isomorphism" as

$$\begin{aligned}\text{iso}_{h,g,f} \leftrightarrow \quad & \text{hom}_{h,g,f} \wedge \forall a, b \in_s \text{car}_h.(f(a) =_s f(b) \Rightarrow a =_s b) \\ & \wedge \forall a \in_s \text{car}_g.\exists b \in_s \text{car}_h.(a =_s f(b)).\end{aligned}$$

2.9 Formalization of Kernels

Given f is a group homomorphism from H into G, the kernel of f is the subgroup of H whose carrier set is $\{x \in H | f(x) = e_G\}$. To describe the homomorphism, three parameters are needed; we also need an extra parameter to specify the kernel itself. We define "k is the kernel of the homomorphism $f : h \to g$" as

$$\text{kernel}_{k,h,g,f} \leftrightarrow \text{hom}_{h,g,f} \wedge \text{subgroup}_{k,h} \wedge \text{car}_k =_s \text{sep}(x \in_s \text{car}_h | f(x) =_s e_g).$$

Noticing that

$$\text{subgroup}_{k,h} \leftrightarrow \text{group}_k \wedge \text{group}_h \wedge \text{car}_k \subseteq \text{car}_h \wedge \forall a, b \in_s \text{car}_k.(a *_k b =_s a *_h b),$$

where group_h is implied in $\text{hom}_{h,g,f}$, and $\text{car}_k \subseteq \text{car}_h$ is implied in $\text{car}_k =_s \text{sep}(x \in_s \text{car}_h | f(x) =_s e_g)$, we can update the kernel formalization to be

$$\begin{aligned}\text{kernel}_{k,h,g,f} \leftrightarrow \quad & \text{hom}_{h,g,f} \wedge \text{group}_k \wedge \text{car}_k =_s \text{sep}(x \in_s \text{car}_h | f(x) =_s e_g) \\ & \wedge \forall a, b \in_s \text{car}_k.(a *_k b =_s a *_h b).\end{aligned}$$

This definition implies that if $\text{kernel}_{k,h,g,f}$ then $\text{subgroup}_{k,h}$.

3 Discussion of the Formalization

3.1 The Formalization of Klein 4-group

We have successfully formalized all the fundamental concepts in group theory. Now the question is: under this formalization, given a set, a binary operation, an identity, and an inverse operation, how can we know whether they form a group or not?

Recall the definition of a group. A group must satisfy all those axioms. So first we assign a label, say h, to such a composition and define car_h, $*_h$, e_h, $'^h$ as the given set, binary operation, identity, and inverse operation separately. Then without making the assumption group_h, check whether all the axioms of groups (number 1-7 in Section 2.1) are satisfied. If not, we can conclude this composition is not a group at all. If yes, we conclude they do form a group and thus all the proven group properties apply to it. The negative case is easy to understand. For the positive case, let us examine a concrete example, the Klein 4-group, to illustrate this method.

	e	a	b	c
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

Fig. 1. Group table of the Klein 4-group

The **Klein 4-group** contains four elements. Figure 1 lists its group table.

Let us assign the Klein 4-group a label $klein_4$ and declare k_0, k_1, k_2, k_3 as its four elements. Its carrier set, binary operation, identity, and inverse operation can be defined as in Figure 2.

$$\text{car}_{klein_4} \leftrightarrow \{k_0\} \cup \{k_1\} \cup \{k_2\} \cup \{k_3\} \qquad e_{klein_4} \leftrightarrow k_0$$

$$\begin{array}{llll}
k_0 *_{klein_4} k_0 \leftrightarrow k_0 & k_1 *_{klein_4} k_0 \leftrightarrow k_1 & k_2 *_{klein_4} k_0 \leftrightarrow k_2 & k_3 *_{klein_4} k_0 \leftrightarrow k_3 \\
k_0 *_{klein_4} k_1 \leftrightarrow k_1 & k_1 *_{klein_4} k_1 \leftrightarrow k_0 & k_2 *_{klein_4} k_1 \leftrightarrow k_3 & k_3 *_{klein_4} k_1 \leftrightarrow k_2 \\
k_0 *_{klein_4} k_2 \leftrightarrow k_2 & k_1 *_{klein_4} k_2 \leftrightarrow k_3 & k_2 *_{klein_4} k_2 \leftrightarrow k_0 & k_3 *_{klein_4} k_2 \leftrightarrow k_1 \\
k_0 *_{klein_4} k_3 \leftrightarrow k_3 & k_1 *_{klein_4} k_3 \leftrightarrow k_2 & k_2 *_{klein_4} k_3 \leftrightarrow k_1 & k_3 *_{klein_4} k_3 \leftrightarrow k_0 \\
k_0'^{klein_4} \leftrightarrow k_0 & k_1'^{klein_4} \leftrightarrow k_1 & k_2'^{klein_4} \leftrightarrow k_2 & k_3'^{klein_4} \leftrightarrow k_3
\end{array}$$

Fig. 2. Definitions for the Klein 4-group

With these definitions, we can verify that all of the group axioms are satisfied for $klein_4$, without assuming group_{klein_4}. For example, we can prove the axiom G2 for $klein_4$

$$\frac{\Gamma \vdash a \text{ is a set} \quad \Gamma \vdash a \in_s \text{car}_{klein_4}}{\Gamma \vdash e_{klein_4} *_{klein_4} a =_s a}.$$

First, since car_{klein_4} is defined as $\{k_0\} \cup \{k_1\} \cup \{k_2\} \cup \{k_3\}$, from the properties of union and singularity, it can be proved that if $a \in_s \text{car}_{klein_4}$, then a must be equal to one of k_0, k_1, k_2, k_3. Then for each of these four cases, by definition,

$$e_{klein_4} *_{klein_4} k_i =_s k_0 *_{klein_4} k_i =_s k_i \quad (i = 0, 1, 2, 3).$$

All the other group axioms can be proved similarly for the $klein_4$ case. Thus we can conclude that this is a group and can make the hypothesis group_{klein_4}. As a consequence, all the group theorems apply for $klein_4$.

3.2 Constructivity

Constructivity sometimes makes things harder, especially for work done with machines. For example, classically, there is a theorem "any subgroup of a cyclic group is cyclic." The proving process for the nontrivial case (i.e., the subgroup is other than $\{e\}$ where e is the identity) is assuming G is a cyclic group generated by a and H is a subgroup of G, then supposing m is the smallest integer in $\mathbb{Z}^+$ such that $a^m \in H$, and then claiming and proving a^m generates H. One of the problems is that in order to assume that m is the smallest natural number such that $a^m \in H$ we need to prove such m exists. In constructive mathematics, the validity of such an existential statement would imply being able to actually compute m. In a straightforward formulation like the one we have implemented, this is not generally possible (since the group membership could be undecidable).

On the other side, constructivity sometimes has advantages. For example, we can extract computational content from the proofs, which allows us to use our formalism for developing guaranteed correct formal abstract algebra algorithms by extracting them from proofs of existentials. However, algorithms extracted naively from proofs are often inefficient. Although Caldwell [26] and Nogin [27] demonstrate methods to address this problem, we have not explored this option in detail in MetaPRL.

3.3 Limitations and Alternatives of the Formalization

As discussed above, our formalization of the foundations of abstract algebra—mainly the group theory—is a success: All the major group concepts are formalized; whether a set-operation combination is a group or not can be decided; most theorems and properties can be proved effectively.

However, it still has some limitations. In the current formalization many concepts are introduced on meta-level instead of being first-class objects in a theory. Groups are implemented as labels rather than as sets, making it impossible to quantify over them and to have sets of groups. Similarly group operations are not sets either because we define them with axioms in meta-theory.

Right now we explicitly give a name for the identity and the inverse operation. Instead, we could have defined a group as being just a pair of a carrier and a binary operation with axioms specifying the existence of an identity element and an existence of an inverse for each group element.

We tried to limit ourselves to pure CZF, although we still ended up using a few elements of type theory when some parts of MetaPRL's CZF theory were not yet implemented. It could be beneficial to try to clean that up and come up with a truly pure-CZF implementation. On the other hand, we may want to try to take advantage of the availability of the embedding of CZF into ITT in MetaPRL

by allowing ourselves to use the type theoretic concepts more freely in our formalization. This way we might be able to come up with some natural "hybrid" formalization where some aspects are formalized using set theoretic concepts and some using type theoretic concepts, picking the most natural approach in every case.

4 Conclusions and Future Work

This paper presents a machine-checked formalization method of abstract algebra in constructive set theory. We use set axiomatization to formalize groups. Every group should agree with all of the group axioms and all properties of groups are derived from the group axioms and set axioms. We further formalize subgroups, cyclic groups, homomorphisms, and other concepts in group theory on the basis of the formalization of groups. Rings, fields and more advanced abstract algebra can be formalized in constructive set theory based on the group formalization.

Though our work is still elementary and has some limitations, since the idea is natural (easy to understand) and the formalization is easy to use (both for proving purposes and for extending purposes), we believe it will have wide applications in the future. We are also considering borrowing this formalization method for formalizing abstract algebra in MetaPRL's ITT theory which is much more complete than its CZF theory.

5 Acknowledgments

The authors would like to thank Aleksey Nogin. His valuable observations have greatly improved the contents and the presentation of the paper. We also want to thank Alexei Kopylov for discussions on the formalization.

References

1. Gordon, M., Melham, T.: Introduction to HOL: A Theorem Proving Environment for Higher-Oder Logic. Cambridge University Press, Cambridge (1993)
2. Gunter, E.: 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 (1989) Distributed with the HOL system in the directory Training/studies/intmod/doingalgpaper.
3. Constable, R.L., Allen, S.F., Bromley, H.M., Cleaveland, W.R., Cremer, J.F., Harper, R.W., Howe, D.J., Knoblock, T.B., Mendler, N.P., Panangaden, P., Sasaki, J.T., Smith, S.F.: Implementing Mathematics with the NuPRL Development System. Prentice-Hall, NJ (1986)
4. Jackson, P.B.: Exploring abstract algebra in constructive type theory. In Bundy, A., ed.: Proceedings of the 12^{th} International Conference on Automated Deduction. Volume 814 of Lecture Notes in Artificial Intelligence., New York, Springer-Verlag (1994) 590–604
5. Farmer, W.M., Guttman, J.D., Thayer, F.J.: IMPS: An interactive mathematical proof system. Journal of Automated Reasoning **11** (1993) 213–248

6. Farmer, W.M., Joshua D. Guttman, F.J.T.: Little theories. In Kapur, D., ed.: Automated-Deduction-CADE-11. Lecture Notes in Artificial Intelligence, New York, Springer-Verlag (1992) 567–581
7. Barras, B., Boutin, S., Cornes, C., Courant, J., Filliâtre, J.C., Giménez, E., Herbelin, H., Gérard-Mohring, Saïbi, A., Werner, B.: The Coq Proof Assistant Reference Manual. INRIA-Rocquencourt, CNRS and ENS Lyon. (1996)
8. Geuvers, H., Wiedijk, F., Zwanenburg, J.: A constructive proof of the fundamental theorem of algebra without using the rationals. In Callaghan, P., Luo, Z., McKinna, J., Pollack, R., eds.: Types for Proofs and Programs, Proceedings of the International Workshop TYPES 2000. Volume 2277 of Lecture Notes in Computer Science., Springer (2001) 96–111
9. Paulson, L., Nipkow, T.: Isabelle tutorial and user's manual. Technical report, University of Cambridge Computing Laboratory (1990)
10. Kammüller, F., Paulson, L.C.: A formal proof of Sylow's first theorem – an experiment in abstract algebra with Isabelle HOL. Journal of Automated Reasoning **23** (1999) 235–264
11. Bailey, A.J.: The Machine-Checked Literate Formalization of Algebra in Type Theory. PhD thesis, University of Manchester (1998)
12. Gordon, M.J.C.: Merging HOL with set theory: preliminary experiments. Technical Report 353, University of Cambridge Computer Laboratory (1994)
13. Spivey, J.M.: The Z Notation – A Reference Manual. 2nd edn. Prentice Hall International Series in Computer Science (1992)
14. Jones, C.B.: Systematic Software Development using VDM. Prentice-Hall, Upper Saddle River, NJ 07458, USA (1990)
15. Hickey, J.J.: The MetaPRL Logical Programming Environment. PhD thesis, Cornell University, Ithaca, NY (2001)
16. Hickey, J.J., Nogin, A., Kopylov, A., et al.: (MetaPRL home page) `http://metaprl.org/`.
17. Paulson, L.C.: Set theory for verification: I from foundations to functions. Journal of Automated Reasoning **11** (1993) 353–389
18. Rudnicki, P.: An overview of the Mizar project. Notes to a talk at the workshop on Types for Proofs and Programs (1992)
19. Trybulec, W.A., et al.: (Mizar home page) `http://www.mizar.org/`.
20. Trybulec, W.A.: Groups. Journal of Formalized Mathematics **2** (1990) `http://mizar.org/JFM/Vol2/group_1.html`.
21. Hickey, J.J., Aydemir, B., Bryukhov, Y., Kopylov, A., Nogin, A., Yu, X.: (A listing of MetaPRL theories) `http://metaprl.org/theories.pdf`.
22. Yu, X.: Formalizing abstract algebra in constructive set theory. Master's thesis, California Institute of Technology (2002)
23. Myhill, J.: Constructive set theory. Journal of Symbolic Logic **40** (1975) 347–382
24. Aczel, P., Rathjen, M.: Notes on constructive set theory. Technical Report 40, Mittag-Leffler (2000/2001)
25. Martin-Löf, P.: Intuitionistic Type Theory. Number 1 in Studies in Proof Theory, Lecture Notes. Bibliopolis, Napoli (1984)
26. Caldwell, J.: Moving proofs-as-programs into practice. In: Proceedings of the 12th IEEE International Conference on Automated Software Engineering, IEEE Computer Society (1997)
27. Nogin, A.: Writing constructive proofs yielding efficient extracted programs. In Galmiche, D., ed.: Proceedings of the Workshop on Type-Theoretic Languages: Proof Search and Semantics. Volume 37 of Electronic Notes in Theoretical Computer Science., Elsevier Science Publishers (2000) `http://www.elsevier.nl/gej-ng/31/29/23/67/22/show/Products/notes/index.htt#005`.