Formalizing Abstract Algebra in Type Theory with Dependent Records

Xin Yu, Aleksey Nogin, Alexei Kopylov, Jason Hickey
1California Institute of Technology, 2Cornell University

Basic Idea

The dependent record type presented on the right 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.

Formalization of Groups and Group-like Objects

**Groupoid:**

- \( \text{Groupoid} \equiv \{ \text{Cur} \vdash a : \text{Cur} \to \text{Cur} \to \text{Cur} \} \)

**Semigroup:**

- \( \text{isSemigroup}(g) \equiv \forall x, y : \text{Cur}; \text{Cur}(x \cdot y, y) \equiv x \cdot y \cdot y \equiv x \cdot y \cdot y \subseteq \text{Cur} \)
- \( \text{Semigroup} \equiv \{ g : \text{Groupoid} \mid \text{isSemigroup}(g) \} \)

**Monoid:**

- \( \text{isMonoid}(g) \equiv \text{isSemigroup}(g) \land \forall x : \text{Cur}; \text{Cur}(x, x) \equiv x \land x \equiv x \subseteq \text{Cur} \)
- \( \text{Monoid} \equiv \{ g : \text{Monoid} \mid \text{isMonoid}(g) \} \)

**Group:**

- \( \text{Group} \equiv \{ \text{Groupoid} \mid \text{inV} : \text{Cur} \to \text{Cur} \to \text{Cur} \} \)
- \( \text{isGroup}(g) \equiv \text{isSemigroup}(g) \land \forall x : \text{Cur}; \text{Cur}(x, x) \equiv x \land x \equiv x \subseteq \text{Cur} \)
- \( \text{Group} \equiv \{ g : \text{Group} \mid \text{isGroup}(g) \} \)

**Groupoid ⊆ Semigroup ⊆ Monoid ⊆ Group**

Formalization of Subgroups

- **Substructure:** \( h \triangleleft g \equiv \text{Cur} \subseteq \text{Cur} \land h \subseteq \text{Cur} \land h \cdot h \subseteq h \cdot h \subseteq g \cdot g \)
- **Submonoid:** \( h \subseteq \text{Monoid} \equiv \forall h \subseteq \text{Monoid} \equiv h \triangleleft g \)
- **Subgroup:** \( h \subseteq \text{Group} \equiv \forall h \subseteq \text{Group} \equiv h \triangleleft g \)

Formalization of Cyclic Groups

**Power operation:**

\[ a_0^0 \equiv \left\{ \begin{array}{ll} a_0 & \text{if } n > 0 \\ a_0^0 & \text{if } n = 0 \\ a_0^0 & \text{if } n < 0 \end{array} \right. \]

**Cyclic group:** \( \text{isCyclic}(g) \equiv \exists x : \text{Cur} \vdash \exists y : \text{Cur} \vdash \exists z : \text{Cur} (g = x \cdot y \cdot y) \)

We also defined the Abelian Group, Coset and Normal Subgroup, Group Mappings, Group kernel, Quotient Group, etc.

Motivation

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. 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.

Approach

We explore a formalization of the abstract algebra concepts in type theory, based on the notion of an extensible record type.

<table>
<thead>
<tr>
<th>goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>prove a specific &quot;big theorem&quot;</td>
</tr>
<tr>
<td>method</td>
</tr>
<tr>
<td>use a module system</td>
</tr>
<tr>
<td>previous efforts</td>
</tr>
<tr>
<td>use dependent record type</td>
</tr>
<tr>
<td>our formalization</td>
</tr>
<tr>
<td>generality</td>
</tr>
<tr>
<td>provide a first-class formalization</td>
</tr>
<tr>
<td>provide inheritance and subtyping</td>
</tr>
<tr>
<td>close to normal intuition</td>
</tr>
</tbody>
</table>

**Records**

**Records** are tuples of labeled fields.

* Syntax: \( \{ x_1 \equiv a_1, \ldots, x_n \equiv a_n, y \} \) is a label of the string type.

* Type: \( \{ x_1 : A_1, \ldots, x_n : A_n \} \) (\( a_i \in A \)).

* It satisfies the natural subtyping relation:

\( \{ x_1 : A_1, \ldots, x_n : A_n \} \subseteq \{ x_1 : A_1, \ldots, x_n : A_n \} \subseteq \{ x : A \} \)

**Example:** Let \( \text{Point} = \{ x : \text{yz} \cdot y : \text{z} \} \), \( \text{CPoint} = \{ x : \text{yz} = \text{color} : \text{String} \} \)

- \( \text{CPoint} \subseteq \text{Point} \)
- \( \{ x = \text{y}, y = \text{color} : \text{red} \} \subseteq \{ x = \text{y}, y = \text{color} : \text{green} \} \subseteq \text{Point} \)
- \( \{ x = \text{y}, y = \text{color} : \text{red} \} \nsubseteq \{ x = \text{y}, y = \text{color} : \text{green} \} \subseteq \text{Point} \)

**Dependent Record Type**

The type of a field may depend on values of previous fields.

**Example:** \( \text{Groupoid} \equiv \{ \text{Cur} : \text{Cur} \to \text{Cur} \to \text{Cur} \} \)

In MetaPRL, there is a primitive type constructor, dependent intersection, on which the dependent record type is defined.

* new, simple, expressive, and useful
* similar to a module system, but algebraic objects are first-class
* significantly change the way to formalizing abstract algebra

Forming a Concrete Group

**Example:** \( \langle S_+ \rangle \) is a group.

In MetaPRL,

\[ \langle \text{Cur} = \mathbb{Z} \vdash x : \mathbb{Z}; x + y @ 1 = 0 \vdash \text{inv} : x \to -x @ 1 \in \text{Group} \rangle \]

Because \( g = \langle \text{Cur} = \mathbb{Z} \vdash x : \mathbb{Z}; x + y @ 1 = 0 \vdash \text{inv} : x \to -x @ 1 \in \text{Group} \rangle \)

\* \( g \in \text{PropUnit} \)
\* \( \text{isSemigroup}(g) \), i.e., \( \text{* } \) is associative
\* \( \forall x : \mathbb{Z} (x + 0 = x @ 1 \in \mathbb{Z}) \)
\* \( \forall x : \mathbb{Z} (-x + x = 0 @ 1 \in \mathbb{Z}) \)

**Formal Proof Example**

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.

\[ \Gamma \vdash g \in \text{Group} \]
\[ \Gamma \vdash a \in \text{Cur} \]
\[ \Gamma \vdash s_y \vdash b \in \text{Cur} \]

Idea of proving: \( a 
subseteq b \vdash \text{a} \cdot s \equiv b \cdot s \equiv a \equiv b \cdot a \equiv a \)

In MetaPRL,

\[ \Gamma \vdash g \in \text{Group} \]
\[ \Gamma \vdash a \in \text{Cur} \]
\[ \Gamma \vdash s_y \vdash b \in \text{Cur} \]

\( \text{left inverse} \)

\[ \text{BY substitute} \ (b = \text{a} \cdot s) \equiv a \in \text{Cur} \]

\( \text{associativity} \)

\[ \text{BY substitute} \ (a \cdot s \cdot s) \equiv a \in \text{Cur} \]

\( \text{right inverse} \)

\[ \text{BY substitute} \ (a \cdot s \equiv b) \equiv a \in \text{Cur} \]

\( \text{left substitution} \ (a \equiv b \in T) \equiv \text{replaces all occurrences of } a \text{ with } b \text{ in clause } \Delta \)

Formalization of Rings

A ring is an Abelian group under addition and a semigroup under multiplication.

* \( \text{Proving} \equiv \{ \text{Groupoid} \vdash \text{Cur} \to \text{Cur} \to \text{Cur} : 0 \vdash \text{neg} \vdash \text{inv} \} \)
* \( \text{additive-group}(r) \equiv \text{remains} \equiv \{ a = 0, 0 = a \} \)
* ensures \( a \vdash \text{Proving} \vdash \text{additive-group}(r) \)
* \( \text{isRing}(r) \equiv \{ a \cdot (b + c) = a \cdot b + a \cdot c \} \)
* \( \text{isField}(r) \equiv \{ a \cdot (b + c) = a \cdot b + a \cdot c \} \)

\* This work was supported in part by the NSF-Directed Energy University Research Initiative (DEURI) program administered by the Office of Naval Research (ONR) under Grant N00014-11-1-0079, the Defense Advanced Research Projects Agency (DARPA), the United States Air Force, the Central Coast, and by NSF Grant CCF 1321045.