Formalizing Abstract Algebra in Type Theory with Dependent Records*

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

* $Groupoid_i \equiv \{\mathrm{Car}\colon \mathbb{U}_i; *\colon \mathrm{Car} \to \mathrm{Car} \to \mathrm{Car}\}$

Semigroup:

* $isSemig(g) \equiv \forall x, y, z\colon \mathrm{Car}_g.(x *_g y) *_g z = x *_g (y *_g z) \in \mathrm{Car}_g$
* $Semigroup_i \equiv \{g\colon Groupoid_i \mid isSemig(g)\}$

Monoid:

* $Premonoid_i \equiv \{Groupoid_i; 1\colon \mathrm{Car}\}$
* $isMonoid(g) \equiv isSemig(g)$
  $\wedge\ \forall x\colon \mathrm{Car}_g.(1_g *_g x = x \in \mathrm{Car}_g \wedge x *_g 1_g = x \in \mathrm{Car}_g)$
* $Monoid_i \equiv \{g\colon Premonoid_i \mid isMonoid(g)\}$

Group:

* $Pregroup_i \equiv \{Premonoid_i; \mathrm{inv}\colon \mathrm{Car} \to \mathrm{Car}\}$
* $isGroup(g) \equiv isSemig(g) \wedge \forall x\colon \mathrm{Car}_g.(1_g *_g x = x \in \mathrm{Car}_g)$
  $\wedge\ \forall x\colon \mathrm{Car}_g.(x_g^{-1} *_g x = 1_g \in \mathrm{Car}_g)$
* $Group_i \equiv \{g\colon Pregroup_i \mid isGroup(g)\}$

Groupoid $\supset$ Semigroup $\supset$ Monoid $\supset$ Group

Formalization of Subgroups

Substructure: $h \subseteq_{Str} g \equiv \mathrm{Car}_h \subseteq \mathrm{Car}_g \wedge *_g = *_h \in \mathrm{Car}_h \to \mathrm{Car}_h \to \mathrm{Car}_h$

Submonoid: $h \subseteq_{Monoid_i} g \equiv h \in Monoid_i \wedge g \in Monoid_i \wedge h \subseteq_{Str} g$

Subgroup: $h \subseteq_{Group_i} g \equiv h \in Group_i \wedge g \in Group_i \wedge h \subseteq_{Str} g$

Formalization of Cyclic Groups

Power operation:

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

Cyclic group: $isCyclic(g) \equiv \exists a\colon \mathrm{Car}_g.\forall x\colon \mathrm{Car}_g.\exists n\colon \mathbb{Z}.(x = a_g^n \in \mathrm{Car}_g)$

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.

	previous efforts	our formalization
goal	prove a specific "big theorem"	generality
method	use a module system * modules are not first-class objects	use dependent record type * provide a first-class formalization * provide inheritance and subtyping * close to normal intuition

Records

Records are tuples of labeled fields.

* Syntax: $\{\mathsf{x}_1 = a_1; \ldots; \mathsf{x}_n = a_n\}$, x_i is a label of the string type.
* Type: $\{\mathsf{x}_1\colon A_1; \ldots; \mathsf{x}_n\colon A_n\}$ ($a_i \in A_i$).
* It satisfies the natural *subtyping* relation:
  $\{\mathsf{x}\colon A; \mathsf{y}\colon B; \mathsf{z}\colon C\} \subset \{\mathsf{x}\colon A; \mathsf{z}\colon C\} \subset \{\mathsf{x}\colon A\}$

Example: Let Point $\equiv \{\mathsf{x}\colon \mathbb{Z}; \mathsf{y}\colon \mathbb{Z}\}$, CPoint $\equiv \{\mathsf{x}\colon \mathbb{Z}; \mathsf{y}\colon \mathbb{Z}; \mathsf{color}\colon String\}$

CPoint $\subset$ Point

$\{\mathsf{x}=1; \mathsf{y}=1; \mathsf{color}=red\} = \{\mathsf{x}=1; \mathsf{y}=1; \mathsf{color}=green\} \in$ Point

$\{\mathsf{x}=1; \mathsf{y}=1; \mathsf{color}=red\} \neq \{\mathsf{x}=1; \mathsf{y}=1; \mathsf{color}=green\} \in$ CPoint

Dependent Record Type

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

Example: $Groupoid_i \equiv \{\mathrm{Car}\colon \mathbb{U}_i; *\colon \mathrm{Car} \to \mathrm{Car} \to \mathrm{Car}\}$

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: $\langle \mathbb{Z}, + \rangle$ is a group.

In MetaPRL,

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

Because (let $g = \{\mathrm{Car} = \mathbb{Z}; * = \lambda x.\lambda y.(x + y); 1 = 0; \mathrm{inv} = \lambda x.(-x)\}$)

* $g \in Pregroup_i$
* $isSemig(g)$, i.e., $*_g$ is associative
* $\forall x\colon \mathbb{Z}.(0 + x = x \in \mathbb{Z})$
* $\forall x\colon \mathbb{Z}.((-x) + x = 0 \in \mathbb{Z})$

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.

$$\frac{\Gamma \vdash g \in Group_i \qquad \Gamma \vdash a \in \mathrm{Car}_g}{\Gamma \vdash a *_g 1_g = a \in \mathrm{Car}_g}$$

Ideas for proving: $a *_g 1_g = a *_g (a_g^{-1} *_g a) = (a *_g a_g^{-1}) *_g a = 1_g *_g a = a$.

In MetaPRL:

$\dfrac{\Gamma \vdash g \in Group_i \qquad \Gamma \vdash a \in \mathrm{Car}_g}{\Gamma \vdash a *_g 1_g = a \in \mathrm{Car}_g}$ (left inverse)

– BY[1] `substT` $(1_g = a_g^{-1} *_g a \in \mathrm{Car}_g)$ `0 thenT autoT`

1 $\dfrac{\Gamma \vdash g \in Group_i \qquad \Gamma \vdash a \in \mathrm{Car}_g}{\Gamma \vdash a *_g (a_g^{-1} *_g a) = a \in \mathrm{Car}_g}$ (associativity)

– BY `substT` $(a *_g (a_g^{-1} *_g a) = (a_g^{-1} *_g a) *_g a \in \mathrm{Car}_g)$ `0 thenT`

1.1 $\dfrac{\Gamma \vdash g \in Group_i \qquad \Gamma \vdash a \in \mathrm{Car}_g}{\Gamma \vdash (a *_g a_g^{-1}) *_g a = a \in \mathrm{Car}_g}$ (right inverse assertion)

– BY `substT` $(a *_g a_g^{-1} = 1_g \in \mathrm{Car}_g)$ `0 thenT autoT`

[1] `substT` $(a = b \in T)$ i replaces all occurrences of a with b in clause i.

Formalization of Rings

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

* $Prering_i \equiv \{Groupoid_i; +\colon \mathrm{Car} \to \mathrm{Car} \to \mathrm{Car}; 0\colon \mathrm{Car}; \mathrm{neg}\colon \mathrm{Car} \to \mathrm{Car}\}$
* additive_group$(r) \equiv$ `rename`$(+ \rightsquigarrow *, 0 \rightsquigarrow 1, \mathrm{neg} \rightsquigarrow \mathrm{inv})\{r\}$

  `rename` ensures $\forall r\colon Prering_i.$additive_group$(r) \in Pregroup_i$
* $isRing(r) \equiv isSemigroup(r)$
  $\wedge\ isGroup(\text{additive\_group}(r))$
  $\wedge\ isCommut(\text{additive\_group}(r))$
  $\wedge\ \forall a, b, c\colon \mathrm{Car}_r.((a +_r b) *_r c = a *_r c +_r b *_r c \in \mathrm{Car}_r$
  $\wedge\ a *_r (b +_r c) = a *_r b +_r a *_r c \in \mathrm{Car}_r)$
* $Ring_i \equiv \{g\colon Prering_i \mid isRing(g)\}$

* This work was supported in part by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the Office of Naval Research (ONR) under Grant N00014-01-1-0765, the Defense Advanced Research Projects Agency (DARPA), the United States Air Force, the Lee Center, and by NSF Grant CCR 0204193.