Formal Abstract Algebra in MetaPRL

Jason Hickey
Caltech Computer Science

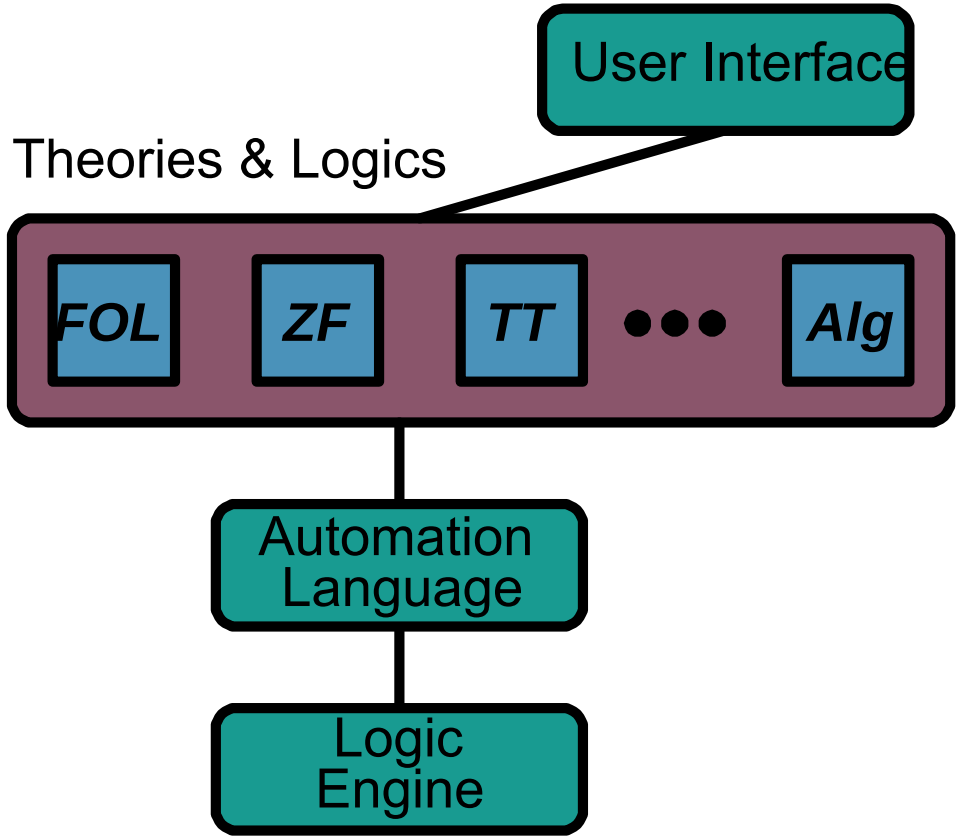
Xin Yu, Aleksey Nogin, Alexei Kopylov

Objectives and questions

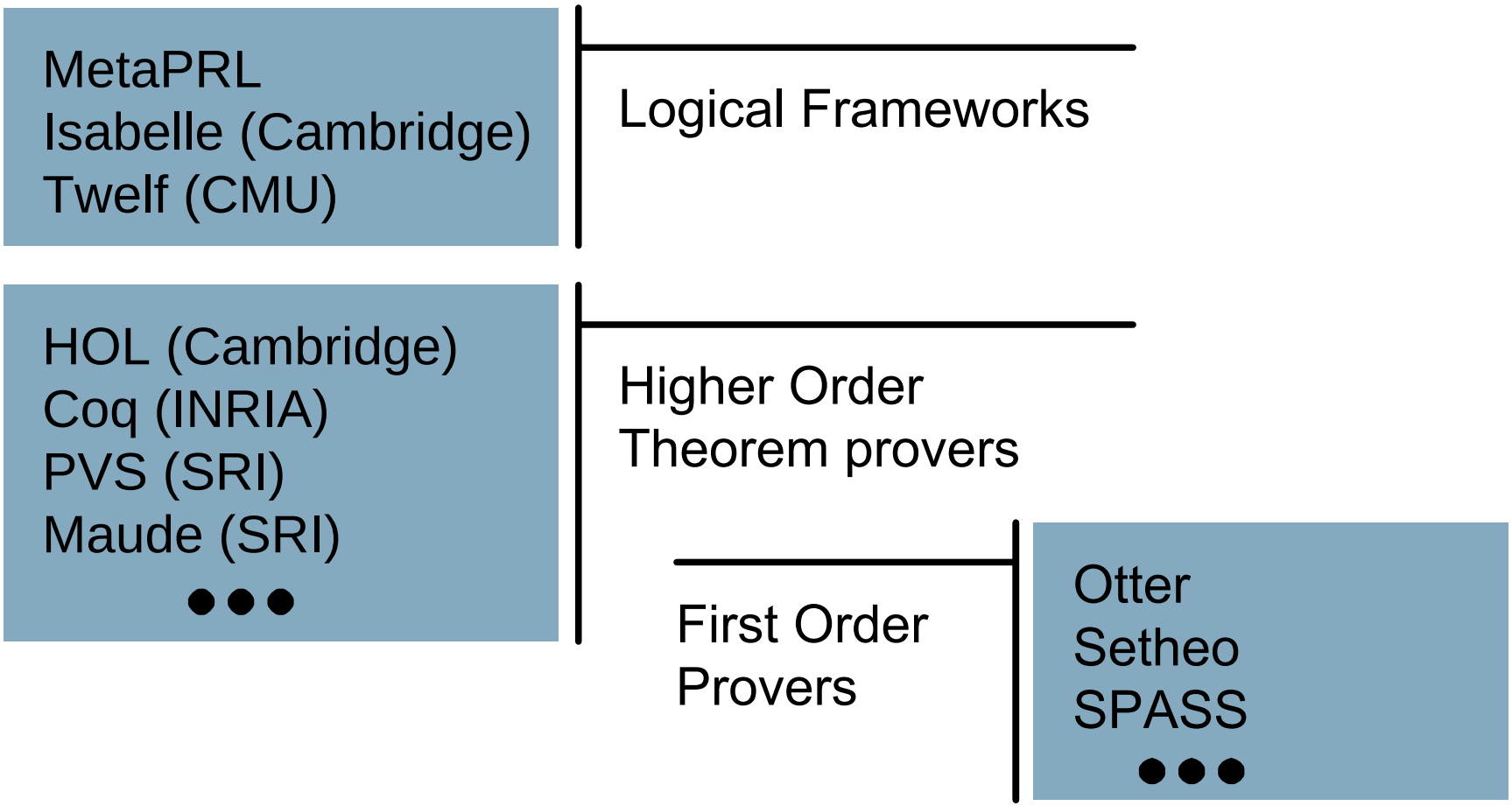
- What is it like to do formal, automated theorem proving?
- How can reasoning tools help the mathematician?
- We'll explore this in three parts:
  - *Intro to reasoning in MetaPRL (a logical framework)*
  - *Establishing a formal foundation*
  - *Application to abstract algebra*

Higher order, automated theorem proving

- An automated theorem prover has four major parts
  - *A logic engine*
  - *A language for automation*
  - *Constructed knowledge*
  - *A user interface*



Systems


MetaPRL
Isabelle (Cambridge)
Twelf (CMU)
Logical Frameworks
HOL (Cambridge)
Coq (INRIA)
PVS (SRI)
Maude (SRI)
●●●
Higher Order
Theorem provers
First Order
Provers
Otter
Setheo
SPASS
●●●

Goals of automated reasoning

- *Ease* the task of proving
  - *Automate the simple parts of a proof*
  - *Suggest approaches in more difficult cases*
- *Validate* the proof
  - *(against a set of axioms)*

The MetaPRL Logical Environment

- Based on over 20 years of research in a series of PRL provers at Cornell
- A logical framework
  - *Logics can be defined and related*
  - *Fast, distributed*
  - *Has been used in validating large software systems*
- Logics:
  - *Intuitionistic type theory (well developed)*
  - *Constructive ZF set theory (Aczel)*
  - *Other: FOL, programming language logics, …*

Outline

- Formalizing, and reasoning about, abstract algebra
  - *First, we need to set the logical foundations*
  - *Second, describe how to formalize algebraic objects*
  - *Finally, reason about these objects*

Logical Foundations

- A logic has three parts
  - *Terms: represent syntax and formulas*
  - *Computation: describe and equivalence relation on terms*
  - *Judgments: describe what is true*
- We *could* do this in set theory
- Type theory is more developed
  - *Usually Intuitionistic, but we'll consider this optional*
  - *A standard foundation for reasoning about programs*
  - *Broad support for automation in MetaPRL*

Formulas in type theory

- There are *types* and there are *values*
- Types are also values

Type	Canonical Value	Description
$Void$		Empty type
$Unit$	$\bullet$	Singleton type
$\mathbb{B}$	$\top, \bot$	Boolean values
$\mathbb{N}$	$0, 1, 2, \ldots$	Natural numbers
$\mathbb{Z}$	$\ldots, -2, -1, 0, 1, 2, \ldots$	Integers
$A + B$	$inl(a), inr(b)$	Disjoint union
$A \times B$	(a, b)	Product space
$A \rightarrow B$	$\lambda x.b$	Function space

Computation

- Computation is defined as a relation on terms

$$t_1 \leftrightarrow t_2$$

- Function application:

$$(\lambda x.b[x])(a) \leftrightarrow b[a]$$

- $b[x]$ is a meta-variable denoting a term that may have a free variable x
- $b[t]$ *substitutes* t for x in b

Types

- A *type* defines a *partial equivalence relation*
  - *Equality is a ternary relation*
  - *Computation coarsens the relation*

The relation $e_1 = e_2 \in T$ means:

- T is a type
- $e_1 \twoheadrightarrow v_1, e_2 \twoheadrightarrow v_2$
- $v_1 \in T, v_2 \in T$
- v_1 and v_2 are equal, when considered as elements of T

Some examples of equality

Valid

$-5 = -5 \in \mathbb{Z}$

$3 + 18 \textbf{ mod } 2 = 1 \in \mathbb{N}$

Invalid

$0 = 1 \in \mathbb{Z}$

$-1 = -1 \in \mathbb{N}$

Boolean Equivalence classes	
$\top$	$\bot$
$\bot \Rightarrow_b \top$	$\top \Rightarrow_b \bot$
$\lambda(x.\top)\ 1$	$\lambda(x.x)\ \bot$
$\vdots$	$\vdots$

Judgments, sequents

- A *judgment* is specified as a sequent:

$$x_1 : T_1, x_2 : T_2[x_1], \ldots, x_n : T_n[x_1, \ldots, x_{n-1}] \vdash T$$

- Meaning:
- If all of the following are true for $i \in \{1, \ldots, n\}$,
  - $T_i[x_1, \ldots, x_{i-1}]$ is a type
  - $x_i \in T_i[x_1, \ldots, x_{i-1}]$
- Then T is a non-empty type.

Dependent types

- Increasing functions

$$i\colon \mathbb{Z} \rightarrow \{j\colon \mathbb{Z} \mid j > i\}$$

- An abstract type, with one operator

$$T\colon \mathbb{U}_i * ((T * T) \rightarrow T)$$

Propositions-as-types

- (Curry-Howard Isomorphism)

Proposition	Type
$\bot$	*Void*
$A \vee B$	$A + B$
$A \wedge B$	$A * B$
$A \Rightarrow B$	$A \rightarrow B$
$\exists x\colon A.B[x]$	$x\colon A * B[x]$
$\forall x\colon A.B[x]$	$x\colon A \rightarrow B[x]$

MetaPRL

Summary

- Propositions-as-types (in this type theory)
  - *Ordinary types are used for propositional operators*
  - *Dependent types are used for quantifiers*
  - *This is all transparent, but bay be exposed*
- Two kinds of reasoning
  - *Rewrites are used*
    - *to perform computation*
    - *To unfold definitions*
  - *Tactics are used to perform reasoning*
  - *autoT performs default automated reasoning*

Question:

- What is 2?

Answer:

- It depends

An example: quotient types

- The quotient type $T//E$, where T is a type and E is a equivalence relation on T
- The equality rule:

$$\frac{\Gamma \vdash (T//E)\ \mathbf{type} \quad \Gamma \vdash t_1 \in T \quad \Gamma \vdash t_2 \in T \quad \Gamma \vdash E(t_1, t_2)}{\Gamma \vdash t_1 = t_2 \in T//E}$$

Quotient types

$$\frac{\Gamma \vdash (T//E)\ \mathbf{type} \quad \Gamma \vdash t_1 \in T \quad \Gamma \vdash t_2 \in T \quad \Gamma \vdash E(t_1, t_2)}{\Gamma \vdash t_1 = t_2 \in T//E}$$

- $\mathbb{Z}_2 = \mathbb{Z}//(\lambda i, j.(i - j)\ \mathbf{mod}\ 2 = 0 \in \mathbb{Z})$

$$\frac{\Gamma \vdash 3 \in \mathbb{Z} \quad \Gamma \vdash 1 \in \mathbb{Z} \quad \Gamma \vdash (3 - 1)\ \mathbf{mod}\ 2 = 0 \in \mathbb{Z}}{\Gamma \vdash 3 = 1 \in \mathbb{Z}_2}$$

- The intersection type $\bigcap_{x \in A} B[x]$

$$\frac{\Gamma, x\colon A \vdash t_1 = t_2 \in B[x]}{\Gamma \vdash t_1 = t_2 \in \bigcap_{x \in A} B[x]}$$

- What is $\mathbb{Z}_2 \cap \mathbb{Z}_3$?

Quotient types

- What is $\mathbb{Z}_2 \cap \mathbb{Z}_3$?
- Derivation

$$\dfrac{\dfrac{\Gamma \vdash (i - j)\ \mathbf{mod}\ 2 = 0 \in \mathbb{Z}}{\Gamma \vdash i = j \in \mathbb{Z}_2} \quad \dfrac{\Gamma \vdash (i - j)\ \mathbf{mod}\ 3 = 0 \in \mathbb{Z}}{\Gamma \vdash i = j \in \mathbb{Z}_3}}{\Gamma \vdash i = j \in (\mathbb{Z}_2 \cap \mathbb{Z}_3)}$$

- So $\mathbb{Z}_2 \cap \mathbb{Z}_3 \simeq \mathbb{Z}_6$

Formalizing Abstract Algebra

- We'll mainly cover groups
- First objective: define a type for groups, including the usual stuff
  - *A carrier, representing the set*
  - *A binary operator that is associative*
  - *A unit*
  - *An inverse*

Records

- Records are tuples of typed fields.
  - Syntax: $\{x_1 = a_1; ...; x_n = a_n\}$, x_i is a label of the string type.
  - Type: $\{x_1 : A_1; \ldots; x_n : A_n\}$ $(a_i \in A_i)$.
- Dependent records (or dependently typed records)
  - The type of fields may depend on values of the previous fields.
  - $\{x_1 : A_1; x_2 : A_2\,[x_1]; \ldots; x_n : A_n\,[x_1, \ldots, x_{n-1}]\}$.

Record Constructors

- For records:
  - $\{\}$ for the empty record;
  - $\{r; x = a\}$ for record extension.
- For record types:
  - $\{\}$ for the empty record type (contains *all* records);
  - $\{self : R; x : A[self]\}$
    - left associative
    - *self* can be omitted
    - e.g. $\{x : A; y : B; z : C\} \leftrightarrow \{\{\{\}; x : A\}; y : B\}; z : C\}$
      $\{R; A[x; y]\} \leftrightarrow \{self : R; A[self.x; self.y]\}$

Basic Rules for the Record Calculus

Reduction rules

$\{r; x = a\}.x \longrightarrow a$ $\qquad$ $\{r; y = a\}.x \longrightarrow r.x$ when $x \neq y$

In particular: $\{x_1 = a_1; \ldots; x_n = a_n\}.x_i \longrightarrow a_i$ when all x_i's are distinct.

Type formation

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

Introduction (membership rules)

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

Elimination (inverse typing rules)

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

Formalizing groups

- Groupoid
  - $Groupoid_i \leftrightarrow \{\mathrm{Car}: \mathbb{U}_i; * : \mathrm{Car} \to \mathrm{Car} \to \mathrm{Car}\}$.
- Semigroup
  - $isSemigroup(g) \leftrightarrow \forall x, y, z : \mathrm{Car}_g . (x *_g y) *_g z = x *_g (y *_g z) \in \mathrm{Car}_g$
  - $Semigroup_i \leftrightarrow \{g : Groupoid_i \mid isSemigroup(g)\}$
- Monoid
  - $Premonoid_i \leftrightarrow \{Groupoid_i; 1 : \mathrm{Car}\}$
  - $isMonoid(g) \leftrightarrow isSemigroup(g) \wedge \forall x : \mathrm{Car}_g .\ (1_g *_g x = x \in \mathrm{Car}_g \wedge x *_g 1_g = x \in \mathrm{Car}_g)$
  - $Monoid_i \leftrightarrow \{g : Premonoid_i \mid isMonoid(g)\}$

Formalizing groups

- Group
  - $Pregroup_i \leftrightarrow \{Premonoid_i; \mathrm{inv}\colon \mathrm{Car} \to \mathrm{Car}\}$
  - $isGroup(g) \leftrightarrow \quad isSemigroup(g)$
    $\wedge \quad \forall x\colon \mathrm{Car}_g.(1_g *_g x = x \in \mathrm{Car}_g)$
    $\wedge \quad \forall x\colon \mathrm{Car}_g.(x_g^{-1} *_g x = 1_g \in \mathrm{Car}_g)$
  - $Group_i \leftrightarrow \{g\colon Pregroup_i \mid isGroup(g)\}$
- Groupoid $\supseteq$ Semigroup $\supseteq$ Monoid $\supseteq$ Group

Substructures

- Subsemigroup, submonoid, subgroup, etc.
  - Substructure: $h \subseteq_{Str} g \leftrightarrow \mathrm{Car}_h \subseteq \mathrm{Car}_g \wedge *_g = *_h \in \mathrm{Car}_h \to \mathrm{Car}_h \to \mathrm{Car}_h$
  - submonoid: $h \subseteq_{Monoid_i} g \leftrightarrow h \in Monoid_i \wedge g \in Monoid_i \wedge h \subseteq_{Str} g$
  - subgroup: $h \subseteq_{Group_i} g \leftrightarrow h \in Group_i \wedge g \in Group_i \wedge h \subseteq_{Str} g$
- Commutative semigroup, commutative monoid, abelian group:
  - $isCommut(g) \leftrightarrow \forall x, y : \mathrm{Car}_g.(x *_g y = y *_g x \in \mathrm{Car}_g)$
  - $Csemigroup_i \leftrightarrow \{g : Semigroup_i \mid isCommut(g)\}$
  - $Cmonoid_i \leftrightarrow \{g : Monoid_i \mid isCommut(g)\}$
  - $Abelg_i \leftrightarrow \{g : Group_i \mid isCommut(g)\}$

Maps

- Group homomorphisms:

$$isGroupHom(f;h;g) \leftrightarrow \forall x : \mathrm{Car}_h.\forall y : \mathrm{Car}_h.(f\ (x *_h y) = f\ x *_g f\ y \in \mathrm{Car}_g);$$

$$GroupHom(h;g) \leftrightarrow \left\{ f : \mathrm{Car}_h \to \mathrm{Car}_g \mid isGroupHom(f;h;g) \right\}.$$

- Other group mappings

$$GroupMono(h;g) \leftrightarrow \left\{ f : GroupHom(h;g) \mid isInjective(f;\mathrm{Car}_h;\mathrm{Car}_g) \right\}$$

$$GroupEpi(h;g) \leftrightarrow \left\{ f : GroupHom(h;g) \mid isSurjective(f;\mathrm{Car}_h;\mathrm{Car}_g) \right\}$$

$$GroupIso(h;g) \leftrightarrow \left\{ f : GroupHom(h;g) \mid isBijective(f;\mathrm{Car}_h;\mathrm{Car}_g) \right\}$$

- Group kernel

$$GroupKer(f;h;g) \leftrightarrow \left\{ \mathrm{Car} = \left\{ x : \mathrm{Car}_h \mid f\ x = 1_g \in \mathrm{Car}_g \right\}; * = *_h; 1 = 1_h; \mathrm{inv} = \right.$$

Cyclic groups

- Power operation

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

- Cyclic groups

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

- Cyclic subgroup generated by a

$$cycSubg(g;a) \leftrightarrow \left\{\mathrm{Car} = \left\{x\colon \mathrm{Car}_g \mid \exists n\colon \mathbb{Z}.x = a_g^n \in \mathrm{Car}_g\right\}; * = *_g;\right.$$

Cosets and normal subgroups

- Left/right coset

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

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

- Normal subgroup

$$normalSubg_i(s;g) \leftrightarrow s \subseteq_{Group_i} g \wedge \forall x : \mathrm{Car}_g . Lcoset(s;g;x) =_e Rcoset(s;g;x)$$

($=_e$ is extensional equality.)

Quotient groups

- Quotient Group

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

How to form a group

- For example, $\langle \mathbb{Z}, + \rangle$ is a group:

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

- Introduction rules:

$$\frac{\Gamma \vdash g \in \mathit{Pregroup}_i \qquad \Gamma \vdash \mathit{isGroup}(g)}{\Gamma \vdash g \in \mathit{Group}_i}$$

$$\frac{\Gamma \vdash g \in \{\mathrm{Car}: \mathbb{U}_i; *: \mathrm{Car} \to \mathrm{Car} \to \mathrm{Car}; 1: \mathrm{Car}; \mathrm{inv}: \mathrm{Car} \to \mathrm{Car}\}}{\Gamma \vdash g \in \mathit{Pregroup}_i}$$

$$\frac{\Gamma \vdash \mathrm{Car}_g\ \mathrm{Type} \quad \Gamma \vdash \mathit{isSemigroup}(g) \qquad \Gamma; x : \mathrm{Car}_g \vdash 1_g *_g x = x \in \mathrm{Car}_g \quad \Gamma; x : \mathrm{Car}_g \vdash x_g^{-1} *_g x = 1_g \in \mathrm{Car}_g}{\Gamma \vdash \mathit{isGroup}(g)}$$

MetaPRL

Left and right identities

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.

$$\frac{\Gamma \vdash g \in \mathit{Group}_i \qquad \Gamma \vdash a \in \mathrm{Car}_g}{\Gamma \vdash a *_g 1_g = a \in \mathrm{Car}_g}.$$

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

1. $\dfrac{\Gamma \vdash g \in \mathit{Group}_i \quad \Gamma \vdash a \in \mathrm{Car}_g}{\Gamma \vdash 1_g = a_g^{-1} *_g a \in \mathrm{Car}_g}$ (trivial since $\dfrac{\Gamma \vdash g \in \mathit{Group}_i \quad \Gamma \vdash a \in \mathrm{Car}_g}{\Gamma \vdash a_g^{-1} *_g a = 1_g \in \mathrm{Car}_g}$)
   - BY substT $(a_g^{-1} *_g a = 1_g \in \mathrm{Car}_g)$ 0 ttca

Left and right identities

2. $$\frac{\Gamma \vdash g \in Group_i \quad \Gamma \vdash a \in \mathrm{Car}_g}{\Gamma \vdash a *_g (a_g^{-1} *_g a) = a \in \mathrm{Car}_g}$$

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

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

– BY equalSymT

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

– BY substT $(a *_g a_g^{-1} = 1_g \in \mathrm{Car}_g)$ 0 ttca (right inverse property)

MetaPRL

Formalization

- What do we gain?
  - *Automation*
  - *Assurance that proofs are correct*
  - *New insight*
- Algebra in MetaPRL is young
  - *Proofs are currently ad-hoc*
    - *And too long*