Formalizing Abstract Mathematics: Issues and Progress

Lawrence C. Paulson

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Milestones in Formalized Mathematics

- Gödel's incompleteness theorem (Shankar, 1985)
- Quadratic reciprocity (Russinoff, 1992)
- Real analysis, measure theory and probability (Harrison, 1996; Hurd, 2002)
- Continuous lattices (Bancerek & Rudnicki, 2003)
- Relative consistency of the axiom of choice (2003)

Abstract Mathematics

- concerns classes of objects specified by axioms, not concrete objects like the integers or reals
- objects are typically structures: $(G, \cdot, 1, ^{-1})$
  - groups, rings, lattices, topological spaces, ...
- concepts are frequently combined and extended
- instances may be concrete (the integers are a ring) or abstract (the product of two groups is a group)

Essentials for Formalization

- Structures are not "theories" (of proof tools)
  - Carriers must be sets, not types.
  - Structures must be first-class values.
- Syntax should reflect the context: if G is a group, then $(xy)^{-1} = y^{-1}x^{-1}$ refers implicitly to G.
- Inheritance of syntax and theorems should be automatic.

Isabelle Overview

- A generic proof tool supporting (among others)
  - higher-order logic with polymorphism
  - ZF set theory
- rewriting, classical reasoning, arithmetic
- flexible mixfix syntax with LaTeX output
- User interface: Proof General

Support for Abstraction

- `\<index>` arguments in syntax declarations
- extensible records (HOL only)
- locales: portable contexts
- locale instantiation
- choice of typed or untyped formalism

`\<index>` Arguments in Syntax Declarations

- One function argument may be `\<index>`
- Even works for infix operators: $\mathtt{a} \otimes_{\mathtt{G}} \mathtt{b}$
- Good for denoting record fields
- Can declare a default by `(structure G)`
- Yields a concise syntax for *G* while allowing references to other groups.

Records

- Can have polymorphic types
- Can be extended with additional fields
- Fields are functions and can have special syntax

base type for carrier

subscripted operator

```
record 'a monoid =
  carrier :: "'a set"
  mult    :: "['a, 'a] ⇒ 'a" (infixl "⊗ı" 70)
  one     :: 'a ("1ı")
```

subscripted constant

Locales: A Lightweight Module System

- Named contexts including variables (with syntax), assumptions and theorems
  - “G is a group”
  - “h is a homomorphism between G and H”
- Multiple inheritance
- One can reason within a locale but also reason about a locale: it is simply a predicate.

A Locale for Monoids

Eliminates the need for subscripts on the operators

```
locale monoid = struct G +
  assumes m_closed [intro, simp]:
        "⟦x ∈ carrier G; y ∈ carrier G⟧ ⟹ x ⊗ y ∈ carrier G"
      and m_assoc:
        "⟦x ∈ carrier G; y ∈ carrier G; z ∈ carrier G⟧
         ⟹ (x ⊗ y) ⊗ z = x ⊗ (y ⊗ z)"
      and one_closed [intro, simp]: "𝟏 ∈ carrier G"
      and l_one [simp]: "x ∈ carrier G ⟹ 𝟏 ⊗ x = x"
      and r_one [simp]: "x ∈ carrier G ⟹ x ⊗ 𝟏 = x"
```

Axioms for monoids

A Locale for Groups

A group is a monoid whose elements have inverses.

```
locale group = monoid +
  assumes inv_ex:
    "⋀x. x ∈ carrier G ⟹ ∃y ∈ carrier G. y ⊗ x = 𝟏 & x ⊗ y = 𝟏"
```

Reasoning in locale *group* makes implicit the assumption that G is a group.

We can use the syntax defined in the locale.

A Proof In Locale *group*

The default group is *G*

```
lemma (in group) l_cancel [simp]:
  assumes [simp]: "x ∈ carrier G" "y ∈ carrier G" "z ∈ carrier G"
  shows "(x ⊗ y = x ⊗ z) = (y = z)"
proof
  assume eq: "x ⊗ y = x ⊗ z"
  hence  "(inv x ⊗ x) ⊗ y = (inv x ⊗ x) ⊗ z"
    by (simp only: m_assoc inv_closed prems)
  thus "y = z" by simp
next
  assume eq: "y = z"
  then show "x ⊗ y = x ⊗ z" by simp
qed
```

Theorem of the group locale

Axiom of the monoid locale

Defining the Direct Product

- The carrier is the Cartesian product of G and H.
- The operator and unit are the pairwise combination of those of G and H.

$$\text{"}G \times\times H \equiv (\!|\, carrier = carrier\ G \times carrier\ H,\\ mult = (\lambda(g, h)\ (g', h').\ (g \otimes_G g', h \otimes_H h')),\\ one = (\mathbf{1}_G, \mathbf{1}_H)\,|\!)\text{"}$$

Subscripting identifies the group

The Product of Two Groups is a Group

```
lemma DirProd_monoid:
  includes monoid G + monoid H
  shows "monoid (G ×× H)"
proof -
  from prems
  show ?thesis by (unfold monoid_def DirProd_def, auto)
qed
```

Two instances of a locale: one each for *G* and *H*

```
lemma DirProd_group:
  includes group G + group H
  shows "group (G ×× H)"
  by ...
```

Locales express the premises and conclusion

The Set of Homomorphisms

```
"hom G H ≡
  {h. h ∈ carrier G → carrier H &
    (∀x ∈ carrier G. ∀y ∈ carrier G. h (x ⊗_G y) = h x ⊗_H h y)}"
```

Two trivial consequences

lemma *hom_mult:*
```
  "⟦h ∈ hom G H; x ∈ carrier G; y ∈ carrier G⟧ ⟹ h (x ⊗_G y) = h x ⊗_H h y"
```
by *(simp add: hom_def)*

lemma *hom_closed:*
```
  "⟦h ∈ hom G H; x ∈ carrier G⟧ ⟹ h x ∈ carrier H"
```
by *(auto simp add: hom_def funcset_mem)*

A Locale for Homomorphism Proofs

G and *H* are groups

h is a homo-
morphism

```
locale group_hom = group G + group H + var h +
  assumes homh: "h ∈ hom G H"
  notes hom_mult [simp] = hom_mult [OF homh]
    and hom_closed [simp] = hom_closed [OF homh]
```

installing two simplification rules

A Proof: Homomorphisms Preserve Inverses

Facts about *h* and about groups *G* and *H* are implicitly present.

```
lemma (in group_hom) hom_inv [simp]:
  assumes [simp]: "x ∈ carrier G"  shows "h (inv x) = inv_H (h x)"
proof -
  have "h x ⊗_H h (inv x) = 1_H"
    by (simp add: hom_mult [symmetric] del: hom_mult)
  also have "... = h x ⊗_H inv_H (h x)"
    by simp
  finally have "h x ⊗_H h (inv x) = h x ⊗_H inv_H (h x)" .
  thus ?thesis by (simp del: inv add: is_group)
qed
```

More Formalized Algebra

- Groups of permutations and automorphisms
- Cosets, normal subgroups, Lagrange's theorem
- The first Sylow theorem (Kammüller, 1999)
- Quotient groups, the first isomorphism theorem
- Beginnings of ring theory

Related Work: Mizar

- structures with multiple inheritance
- adjectives for constraining structures
- coercions by widening and by proved closure properties
- Substantial formal developments
  - commutative algebra
  - *Compendium of Continuous Lattices*

Related Work: Others

- Pioneering attempts: E. Gunter (1989); Yu (1990)
- Recent experiments by Arthan (2004)
- Massive development by Kobayashi et al. (2004)
    - Jordan-Hölder theorem
    - Chinese remainder theorem for rings
    - Modules: exact sequences, tensor products

Conclusions

- Without notational support, users can still do much by pure stamina (Kobayashi).
- Excellent support for abstraction can be hard-wired into an assertion language (Mizar).
- The elements of formal abstract mathematics are records, subscripting (with infixes and defaults) and locales (contexts formalized as predicates).