Type inference and finite group theory

Jeremy Avigad

Departments of Philosophy and Mathematical Sciences
Carnegie Mellon University

January, 2011

Contents

This is largely a field report from a year spent working on a formalization of the Feit-Thompson theorem in Georges Gonthier's "mathematical components" group.

Outline:

1. Type inference in mathematics
2. Type inference in Coq / Ssreflect
3. Type inference in finite group theory

Type inference

Consider the following mathematical statements:

"For every $x \in \mathbb{R}$, $e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$."

"If G and H are groups and f is a homomorphism from G to H, then for every $a, b \in G$, $f(ab) = f(a)f(b)$."

"If F is a field of characteristic p and $a, b \in F$, then $(a+b)^p = \sum_{i=0}^{p} \binom{p}{i} a^i b^{p-i} = a^p + b^p$."

How do we parse these?

Type inference

Observations:

1. The index of the summation is over the natural numbers.
2. $\mathbb{N}$ is embedded in $\mathbb{R}$.
3. In "$a \in G$," G really means the underlying set.
4. ab means multiplication in the relevant group.
5. p is a natural number (in fact, a prime).
6. The summation operator make sense for any monoid (written additively).
7. The summation enjoys extra properties if the monoid is commutative.
8. The additive part of any field is so.
9. $\mathbb{N}$ is also embedded in any field.
10. Alternatively, any abelian is a $\mathbb{Z}$-module, etc.

Type inference

Spelling out these details formally can be painful.

Typically, the relevant information can be *inferred* by keeping track of the *type* of objects we are dealing with:

- In "$a \in G$," the "$\in$" symbol expects a set on the right.
- In "ab," multiplication takes place in "the" group that a is assumed to be an element of.
- In "$x^i/i!$," one expects the arguments to be elements of the same structure.

Type inference: not only inferring types, but also inferring information from type considerations.

Type inference

Structure hierarchies:

- Subclasses: every abelian group is a group
- Reducts: the additive part of a ring is an abelian group
- Instances: the integers are an abelian group
- Embedding: the integers are embedded in the reals
- Uniform constructions: the automorphisms of a field form a group

Advantages:

- Reusing notation: 0, $+$, $a \cdot b$
- Reusing definitions: $\sum_{i \in I} a_i$
- Reusing facts: identities involving sums

Type inference

Observations:

- Type inference occurs when one parses an expression, but also when one applies a lemma).
- The goal is to omit information systematically.
- There are really two kinds of information that are omitted:
  - data: the relevant group multiplication, the relevant embedding
  - facts: the fact that an operation is associative, the fact that a set is closed under an operation
- Under the Curry-Howard isomorphism, facts and data look the same.
- A good deal of technology is imported from the theory of programming languages (but there are differences).
- There is no sharp line between "type" information and genuinely mathematical information.

Type inference

System	Framework	Type inference
Isabelle	Simple type theory	Axiomatic type classes
Mizar	Set theory	Soft typing
Coq	Dependent type theory	Canonical structures or Type classes, etc.

Features of Coq:

- It is based on a expressive dependent type theory.
- The underlying logic is constructive.
- Every term has a computational interpretation.
- Type checking is, in principle, decidable.
- For that reason, it is also rigid.

Type inference in Coq

Mechanisms for type inference in Coq:

- *Implicit arguments*: one can omit arguments that can be inferred from a dependent type
- *Coercions*: cast objects to different types
- *Canonical structures*: can view a particular structure as an instance of a class

In addition, Coq's type inference engine makes use of the computational interpretation, e.g. expanding definitions and simplifying terms as necessary.

Dependent types

```
Record group : Type := Group
{
  carrier : Type;
  mulg : carrier -> carrier -> carrier;
  oneg : carrier;
  invg : carrier -> carrier;
  mulgA : associative mulg;
  ...
}
```

The components of `G : group` are `carrier G`, `mulg G`, ...

Given `g, h : carrier G`, we have `mulg G g h : carrier G`.

So `mulg` has type `forall (G : group), carrier G -> carrier G -> carrier G`.

Implicit arguments and coercions

One can also write `mulg _ g h`, leaving the first argument *implicit*.

Type inference has to solve `carrier ? = carrier G`, which is easy.

```
Notation "g * h" := (mulg _ g h).
```

Now one can write `g * h` for group multiplication.

One can also define

```
Coercion carrier : group >-> Type.
```

Then `g : G` is interpreted as `g : carrier G`.

Canonical structures

Suppose we define

```
IntGroup := Group int addi zeroi negi addiA ...
```

Given `i,j : int`, this will let us (perversely) write `mulg IntGroup i j` for `i + j`, and (less perverseley) apply facts about groups.

What happens if write `i * j`?

Type inference has to solve `carrier ? = int`, and gets stuck.

Declaring

```
Canonical Structure IntGroup.
```

registers the hint `carrier IntGroup = int` for use in type inference.

Summary / recap

Type checking is triggered when:

- parsing an expression
- applying a lemma

Often implicit arguments or facts need to be inferred.

Mechanisms:

- Unification: pattern matching to infer implicit arguments.
- Coercions: cast objects to different types
- Canonical structures: register unification hints that associate structures with instances
- Unfolding definitions, simplifying terms

Finite group library

In the finite group library, type inference is used in a number of ways:

- To recognize when structures have decidable equality and choice functions, satisfy extensionality, and so on.
- To define "big operations" such as $\sum$, $\prod$, $\bigcap$, $\bigcup$, $\bigwedge$, $\bigvee$
- To mediate between sets and structures (e.g. $G \cap H$ and $C_G(A)$ act as both sets and groups).
- To manage class inclusions (rings, commutative rings, fields)
- To manage algebraic constructions (matrices over a ring, polynomials over a ring, quotient groups)
- To infer views (e.g. abelian group as a $\mathbb{Z}$-module)
- The mediate between functions and morphisms
- To view both predicates and lists as sets (e.g. Px vs. $x \in P$).

Examples

```
Lemma commg_subl : forall G H,
    ([~: G, H] \subset G) = (H \subset 'N(G)).

Lemma nilpotent_proper_norm : forall G H,
  nilpotent G -> H \proper G -> H \proper 'N_G(H).

Lemma morphim_center : forall rT A D
  (f : {morphism D >-> rT}),
    f @* 'Z(A) \subset 'Z(f @* A).

Lemma quotient_cents2 : forall A B K,
    A \subset 'N(K) -> B \subset 'N(K) ->
  (A / K \subset 'C(B / K)) = ([~: A, B] \subset K).
```

Examples

```
Theorem Sylow's_theorem :
  [/\ forall P,
      [max P | p.-subgroup(G) P] = p.-Sylow(G) P,
      [transitive G, on 'Syl_p(G) | 'JG],
      forall P, p.-Sylow(G) P ->
        #|'Syl_p(G)| = #|G : 'N_G(P)|
   &  prime p -> #|'Syl_p(G)| %% p = 1%N].

Lemma card_GL : forall n, n > 0 ->
  #|'GL_n[F]| = (#|F| ^ 'C(n, 2) *
    \prod_(1 <= i < n.+1) (#|F| ^ i - 1))%N.

Theorem Cayley_Hamilton : forall A,
  (Zpoly (char_poly A)).[A] = 0.
```

References

- http://www.msr-inria.inria.fr/Projects/math-components
- Gonthier and Mahboubi, "An introduction to small scale reflection in Coq," *Journal of Formalized Reasoning* 2010
- Garillot, Gonthier, Mahboubi, and Rideau, "Packaging mathematical structures," *TPHOLs* 2009
- Bertot, Gonthier, Biha, and Pasca, "Canonical big operators," *TPHOLs* 2008