The Soft Type System of GAP

Dennis Müller, Markus Pfeiffer, Florian Rabe

June 6, 2016

Abstract

1 Introduction

2 Concepts

2.1 Object Level

GAP allows arbitrary run-time representations of mathematical objects (which is natural for efficiency). These types of the run-time system are called families. Each GAP object is typed by exactly one family.

An **object** is

- a literal,
- a list of objects,
- a function on object,
- any other object introduced by a user-declared family.

Objects are typed. A **type** is

- a base type (called a **family**)
- a predicate subtype of some type by a unary predicate on objects (called a **filter**)

Users can declare new families. But some families are built-in:

- one base type for each type of built-in literals:
  - cyclotomic numbers (elements of the algebraic closure of the rationals),
  - booleans,
  - strings,
- one base type each for several built-in operators that allow forming complex objects
  - homogeneous lists (called **collections**): lists of objects that have the same family,
  - arbitrary lists of objects,
  - functions on objects.

A **filter** is a unary predicate on objects.

- the universal filter `IsObject` (the filter of all objects)
- a category C,
- a property P,
- a conjunction $F \wedge G$ of filters.

By convention, the names of atomic filters are of the form `IsXXX`.

The **typing relation** is a binary relation between objects and filters. We write it as $O\#F$. It is defined by

- $O\#$`IsObject` always
- $O\#C$ if O was returned by a constructor of category C,
- $O\#P$ if evaluating P on O returns `true`,
- $O\#F \wedge G$ if $O\#F$ and $O\#G$.

For atomic filters F, the known state of the relation $O\#F$ is cached with O. Thus, the type of every object can be inferred as the conjunction of atomic filters that are known to hold. This type changes dynamically as more properties are evaluated.

2.2 Declaration Level

Categories A **category** declaration introduces a definition-less filter. A category declaration consists of

- a name,
- a filter (called the superfilter).

The concrete syntax is `DeclareCategory(name: String, superfilter: Filter)`.

Categories can be used to represent the type of models of an abstract specification. The details of the specification are formulated by declaring operations and properties on the category.

Because categories are definition-less, all categories are created empty. The objects typed by the category are introduced by declaring constructors. These are operations whose implementation explicitly marks the returned objects as having the category as a filter.

Operations An **operation** declaration introduces an n-ary[1] function on objects. Operations are softly typed: Each n-ary operations provides a list of length n providing the input filter of the respective argument. Operations may also carry an optional return type, which defaults to `IsObject` if omitted.[2]

The concrete syntax is `DeclareOperation(name: String, inputfilters: Filter *, outputfilter: Filter?)`.

An **property** declaration can be seen as a special case of a unary operation that returns a boolean. The concrete syntax is `DeclareProperty(name: String, inputfilter: Filter)`.

An **constructor** declaration can be seen as a special case of an operation that returns an object of a given category. The concrete syntax is `DeclareConstructor (name: String, inputfilters: Filter*, outputfilter: Filter?)`. [3] [4]

Conceptually, all operations are defined. However, the definiens is declared separately through methods.

[1] GAP has an implementation restriction of $n \leq 6$.

[2] This is not implemented yet.

[3] The return argument is not implemented yet.

[4] The current implementation of constructors is somewhat awkward and may be subject to change. Currently, a constructors first argument is special: It must be the expected return filter (rather than an object). This is used to allow method dispatch to choose a different method for different special cases. A more elegant solution would be to allow every operation to declare that some of its arguments must be filters. This would yield a very nice untyped version of bounded polymorphism, which is routine in typed programming languages.

Methods Every operation can have multiple definitions, which are declared by methods. A method declaration consists of

- the named of the operation,
- the input and output filters,
- the actual definition, as a function in the underlying programming language.

The concrete syntax of a method declaration is `InstallMethod(operationname : String, inputfilters: Filter*, outputfilter: Filter?, definition: function )`.

The input and return filters of a method may be more restrictive than the filters used in the operation declaration. More restrictive input filters can be used to represent overloading of operations or run-time polymorphism. A more restrictive output filter can be used to indicate a sharper type than required by the operations.

When evaluating the application of an operation to arguments, one method is selected and its definition executed. If more than one method is found, whose input filters are type the operation arguments, an internal ranking is used to disambiguate.

2.3 Theory Level

There is no explicit theory level. Instead, theories are represented as categories, and theory morphisms as operations, and their relation is a special case of typing.

Therefore, we can treat each source file as a theory.

2.4 Document Level

Source files are grouped into folders and **packages**. The package bundled with GAP is called the **library**.

3 Relating Explicit Theories to GAP's Implicit Theories

A Logic for GAP Theories We can identify a logic and a group of theories that can be naturally embedded into GAP's type system.

Any GAP filter can be used as a type.

Every theory implicitly declares a fixed base type u for the universe.

Then it may have two kinds of declarations:

- includes of another theory,
- function symbols $f : a_1 \times \ldots \times a_n \to a_0$ where each a_i is a type (either u or some GAP type),
- potential axioms: code in GAP's underlying programming language that evaluates to a boolean

Representing Theories in GAP A theory with name T, includes $T_1, \ldots, T_k$, function symbols $f_1, \ldots, f_l$, and potential axioms $p_1, \ldots, p_m$ is translated to GAP as follows:

- We declare a category with name T and superfilter is $T_1 \wedge \ldots \wedge T_n$.

- We declare an operation for each f_i.
- We declare a property for each p_i.

Now the filter $T \wedge p_1$ represents the type of models of T that satisfy the axiom p_1, and accordingly for every subset of the potential axioms.

Extracting Explicit Theories from GAP's Because GAP does not enforce an abstraction boundary between theories and types, it is not generally feasible to extract explicit theories from GAP.

A heuristic extraction might be possible by trying to identity groups of GAP declarations for which the above operation can be inverted to yield a theory.

4 Conclusion

References