Alg User Manual

Aleš Bizjak
Ales.Bizjak@gmail.com
Faculty of Mathematics and Physics
University of Ljubljana

Andrej Bauer
Andrej.Bauer@andrej.com
Faculty of Mathematics and Physics
University of Ljubljana

December 3, 2010

Contents

Chapter 1

Introduction

Alg is a program for enumeration of finite models of algebraic theories. An algebraic theory is given by a signature (a list of constants and operations) and axioms expressed in first-order logic.[1] Examples of algebraic theories include groups, lattices, rings, fields, and many others. Alg can do the following:

- list or count all non-isomorphic models of a given theory,
- list or count all non-isomorphic indecomposable[2] models of a given theory.

Currently alg has the following limitations:

- only unary and binary operations are accepted,
- it is assumed that constants denote pairwise distinct elements.

This manual describes how to install and use alg. For a quick start you need Ocaml 3.11 or newer and the menhir parser generator. Compile alg with

```
make
```

and run

```
./alg.native --size 8 theories/unital_commutative_ring.th
```

For usage information type `./alg.native -help` and for examples of theories see the `theories` subdirectory.

Alg is released under the open source simplified BSD License, as detailed in the next chapter.

[1]Strictly speaking, the axioms of an algebraic theory must be equations, but alg can handle all of first-order logic.

[2]A model is indecomposable if it cannot be written as a non-trivial product of two smaller models.

Chapter 2

Copyright and License

Chapter 3

Installation

3.1 Downloading alg

Alg is available at `http://hg.andrej.com/alg/`. You have three options:

1. download the ZIP file with source code from

   `http://hg.andrej.com/alg/archive/tip.zip`

2. clone the repository with the Mercurial revision control system:

   ```
   hg clone http://hg.andrej.com/alg/
   ```

3. download a precompiled executable for your architecture from

   `http://hg.andrej.com/alg/file/tip/precompiled`

   if one is available. If you choose this option, make sure that you still obtain the ZIP file because the `theories` subdirectory contains a number of useful examples.

3.2 Installation for Linux and MacOS

3.2.1 Prerequisites

To compile alg you need the Make utility, Ocaml 3.11 or newer, and the menhir parser generator higher. We will assume you have Make. You can get Ocaml and menhir in several ways:

1. On Ubuntu, install the packages `ocaml` and `menhir`:

   ```
   sudo apt-get install ocaml menhir
   ```

   Similar solutions are available on other Linux distributions.

2. On MacOS the easiest way to install Ocaml and menhir is with the macports utility:

   ```
   sudo port install ocaml
   sudo port install caml-menhir
   ```

3. If you have GODI installed then you already have Ocaml. Install menhir with the `godi_console` command, if you do not have it yet.

4. Ocaml is also available from

   `http://caml.inria.fr/`

   and menhir from

   `http://pauillac.inria.fr/~fpottier/menhir/`

3.2.2 Compiling to native code

To compile alg, type `make` at the command line. If all goes well ocamlbuild will generate a subdirectory `_build` and in it the `alg.native` executable. It will also create a link to `_build/alg.native` from the top directory. To test alg type

```
./alg.native --count --size 8 theories/group.th
```

It should tell you within seconds that there are 5 groups of size 8.

We provided only a very rudimentary installation procedure for alg. First edit the `INSTALL_DIR` setting in `Makefile` to set the directory in which alg should be installed, then run

```
sudo make install
```

This will simply copy `_build/alg.native` to `$(INSTALL_DIR)/alg`. You may also wish to stash the `theories` subdirectory somewhere for future reference.

3.2.3 Compiling to bytecode

If your version of Ocaml does not compile to native code you can try compiling to bytecode with

```
make byte
```

This will generate a (significantly slower) `alg.byte` executable.

3.2.4 Installation without Make

If you do not have the Make utility (how can that be?) you can compile alg directly with ocamlbuild:

```
ocamlbuild -use-menhir alg.native
```

To install alg just copy `_build/alg.native` to `/usr/local/bin/alg` or some other reasonable place.

3.3 Installation for Microsoft Windows

Sorry, this has not been written yet. But if you have Make and Ocaml 3.11 and menhir, you should be able to just follow the instructions for Linux.

Note that a Windows precompiled executable may be available at

```
http://hg.andrej.com/alg/tip/precompiled/
```

Chapter 4

Input

An alg input file has extension `.th` and it describes an algebraic theory. The syntax vaguely follows the syntax of the Coq proof assistant. A typical input file might look like this:

```
# The axioms of a group.
Theory group.
Constant 1.
Unary inv.
Binary *.
Axiom unit_left: 1 * x = x.
Axiom unit_right: x * 1 = x.
Axiom inverse_left: x * inv(x) = 1.
Axiom inverse_right: inv(x) * x = 1.
Axiom associativity: (x * y) * z = x * (y * z).
```

There is an optional `Theory` declaration which names the theory, then we have declarations of constants, unary and binary operations, and after that there are the axioms. The precise syntax rules are as follows.

4.1 Comments

Comments are written as in Python, i.e., a comment begins with the `#` symbol and includes everything up to the end of line.

4.2 General syntactic rules

An alg input file consists of a sequence of declarations (`Theory`, `Constant`, `Unary`, `Binary`) and axioms (`Axiom`, `Equation`). Each declaration and axiom is terminated with a period.

4.3 The `Theory` keyword

You may give a name to your theory with the declaration

```
Theory theory_name.
```

at the beginning of the input file, possibly preceeded by comments and whitespace. The theory name consists of letters, numbers, and the underscore. If you do not provide a theory name, alg will deduce one from the file name.

4.4 Declaration of operations

The declarations

```
Constant c_1 c_2 ... c_k.
Unary u_1 u_2 ... u_m.
Binary b_1 b_2 ... b_n.
```

are used to declare constants, unary, and binary operations respectively. You may declare several constants or operations with a single declaration, or one at a time. You may mix declarations and axioms, although it is probably a good idea to declare the constants and operations first.

A constant may be any string of letters, digits and the underscore character. In particular, a constant may consist just of digits. Popular choices for neutral elements are `0` or `1`.

Unary and binary operations may be strings of letters, digits and the underscore character. For example, if we declare

```
Unary inv.
Binary mult.
```

then we can write expressions like `mult(x, inv(y))`. It is even possible to declare operations whose names are strings of digits, for example:

```
Unary 3 ten.
Binary +.
Axiom: 3(3(x)) + x = ten(x).
```

Alternatively, we can use *infix* and *prefix* operators. These follow the Ocaml rules for infix and prefix notation. An operator is a string of symbols

```
 ! $ % & * + - / \ : < = > ? @ \^ | ~
```

where:

- a *prefix operator* is one that starts with `?`, `!` or `~`. It can be used as a unary operation.
- *infix operators* can be used as binary operations and have four levels of precedence, listed from lowest to highest:
  - left-associative operators starting with `|`, `&`, `$`

– right-associative operators starting with `@` and `^`
– left-associative operators starting with `+`, `-`, and `\`
– left-associative operators starting with `*`, `/`, and `%`
– right-associative operators starting with `**`.

An operator $\circ$ is *left-associative* if $x \circ y \circ z$ is understood as $(x \circ y) \circ z$, and *right-associative* if $x \circ y \circ z$ is understood as $x \circ (y \circ z)$. If you look at the above list again, you will notice that operators have the expected precedence and associativity. However, if you are unsure about precedence, it is best to use a couple of extra parentheses.

4.5 Axioms

An axiom has the form

```
Axiom [name]: <formula>.
```

or

```
Theorem [name]: <formula>.
```

There is no difference between an axiom and a theorem as far as alg is concerned. We use `Axiom` for the actual axioms and `Theorem` for statements that are consequences of axioms and are worth including in the theory because they make alg run faster, see Chapter 7.

The optional *[name]* is a string of of letters, digits and the underscore chatacters. The *<formula>* is a first-order formula built from the following logical operations, listed in order of increasing precedence:

$\forall x \,.\, \phi$	is written as	`forall` x`,` ϕ,
$\exists x \,.\, \phi$	is written as	`exists` x`,` ϕ,
$\phi \Leftrightarrow \psi$	is written as	ϕ `<->` ψ or ϕ `<=>` ψ,
$\phi \Rightarrow \psi$	is written as	ϕ `->` ψ or ϕ `=>` ψ,
$\phi \vee \psi$	is written as	ϕ `\/` ψ or ϕ `or` ψ,
$\phi \wedge \psi$	is written as	ϕ `/\` ψ or ϕ `and` ψ,
$\neg\phi$	is written as	`not` ϕ,
$s = t$	is written as	s `=` t,
$s \neq t$	is written as	s `<>` t or s `!=` t,
$\top$ and $\bot$	are written as	`True` and `False`, respectively.

An iterated quantification $\forall x_1 \,.\, \forall x_2 \,.\, \cdots \forall x_n \,.\, \phi$ may be written as

$$\texttt{forall}\ x_1\ x_2\ \ldots\ x_n\ ,\ \phi.$$

and similarly for $\exists$.

Axioms may contain free variables. Thus we can write just

```
Axiom: x + y = y + x.
```

instead of

```
Axiom: forall x y, x + y = y + x.
```

Chapter 5

Output

At the moment alg prints results to standard output in text format. We plan to add various other output formats, such as LaTeX, JSON and HMTL.

Chapter 6

Command-line Options

Chapter 7

Optimization

Chapter 8

Examples