

# 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 1, 2010

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Copyright and License</b>	<b>3</b>
<b>3</b>	<b>Installation</b>	<b>4</b>
3.1	Downloading alg . . . . .	4
3.2	Installation for Linux and MacOS . . . . .	4
3.2.1	Prerequisites . . . . .	4
3.2.2	Compiling to native code . . . . .	5
3.2.3	Compiling to bytecode . . . . .	5
3.2.4	Installation without Make . . . . .	5
3.3	Installation for Microsoft Windows . . . . .	6
<b>4</b>	<b>Input Files</b>	<b>7</b>
<b>5</b>	<b>Output Files</b>	<b>9</b>
<b>6</b>	<b>Command-line Options</b>	<b>10</b>
<b>7</b>	<b>Optimization</b>	<b>11</b>

# 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.<sup>1</sup> 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<sup>2</sup> 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.

---

<sup>1</sup>Strictly speaking, the axioms of an algebraic theory must be equations, but alg can handle all of first-order logic.

<sup>2</sup>A model is indecomposable if it cannot be written as a non-trivial product of two smaller models.

## Chapter 2

# Copyright and License

Copyright © 2010, Aleš Bizjak and Andrej Bauer

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

This software is provided by the copyright holders and contributors “as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the copyright holder or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

# 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 Files

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: inv(1) = 1.
Axiom: inv(inv(x)) = x.
Axiom inverse_left: x * inv(x) = 1.
Axiom inverse_right: inv(x) * x = 1.
Axiom associativity: (x * y) * z = x * (y * z).
```

The precise syntax rules are as follows.

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

**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.

**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 preceded 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.



**Declaration of operations.** The declarations

```
Constant c1 c2 ... ck.  
Unary u1 u2 ... um.  
Binary b1 b2 ... bn.
```

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))`. 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 string beginning with `$`, `!` or `~` is a *prefix* operator. Note that `~` is reserved for negation.

## Chapter 5

# Output Files

## Chapter 6

# Command-line Options

## Chapter 7

# Optimization