The 30 Year Horizon

Manuel Bronstein
James Davenport
Albrecht Fortenbacher
Jocelyn Guidry
Michael Monagan
Jonathan Steinbach
Stephen Watt

William Burge
Michael Dewar
Patrizia Gianni
Richard Jenks
Scott Morrison
Robert Sutor
Jim Wen

Timothy Daly
Martin Dunstan
Johannes Grabmeier
Larry Lambe
William Sit
Barry Trager
Clifton Williamson

Volume 2: Axiom Users Guide
Portions Copyright (c) 2005 Timothy Daly

The Blue Bayou image Copyright (c) 2004 Jocelyn Guidry

Portions Copyright (c) 2004 Martin Dunstan
Portions Copyright (c) 2007 Alfredo Portes
Portions Copyright (c) 2007 Arthur Ralfs
Portions Copyright (c) 2005 Timothy Daly

Portions Copyright (c) 1991-2002,
The Numerical ALgorithms Group Ltd.
All rights reserved.

This book and the Axiom software is licensed as follows:

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.

- Neither the name of The Numerical ALgorithms Group Ltd.
nor the names of its contributors may be used to endorse
  or promote products derived from this software without
  specific prior written permission.

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 OWNER 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.
Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

Michael Albaugh    Cyril Alberga    Roy Adler
Christian Aistleitner    Richard Anderson    George Andrews
S.J. Atkins    Henry Baker    Martin Baker
Stephen Balzac    Yuri Baransky    David R. Barton
Gerald Baumgartner    Gilbert Baumslag    Michael Becker
Nelson H. F. Beebe    Jay Belanger    David Bindel
Fred Blair    Vladimir Bondarenko    Mark Botch
Raoul Bourquin    Alexandre Bouyer    Karen Braman
Peter A. Broadberr    Martin Brock    Manuel Bronstein
Stephen Buchwald    Florian Bundschuh    Luanne Burns
William Burge    Ralph Byers    Quentin Carpent
Robert Caviness    Bruce Char    Ondrej Certik
Tzu-Yi Chen    Cheekai Chin    David V. Chudnovsky
Gregory V. Chudnovsky    Mark Clements    James Cloos
Jia Zhao Cong    Josh Cohen    Christophe Couil
Don Coppersmith    George Corliss    Robert Corless
Gary Cornell    Meino Cramer    Jeremy Du Croz
David Cyganski    Nathaniel Daly    Timothy Daly Sr.
Timothy Daly Jr.    James H. Davenport    David Day
James Demmel    Didier Deshommes    Michael Dewar
Jack Dongarra    Jean Della Dora    Gabriel Dos Reis
Claire DiCrescendo    Sam Dooley    Lionel Ducos
Iain Duff    Lee Duhem    Martin Dunstan
Brian Dupee    Dominique Duval    Robert Edwards
Heow Eide-Goodman    Lars Erickson    Richard Fateman
Bertfried Fauser    Stuart Feldman    John Fletcher
Brian Ford    Albrecht Fortenbacher    George Frances
Constantine Frangos    Timothy Freeman    Korrinn Fu
Marc Gaetano    Rudiger Gebauer    Van de Geijn
Kathy Gerber    Patricia Gianni    Samantha Goldrich
Holger Gollan    Teresa Gomez-Diaz    Laureano Gonzalez-Vega
Stephen Gortler    Johannes Grabmeier    Matt Grayson
Klaus Ebbe Grue    James Griesmer    Vladimir Grinberg
Oswald Gschnitzer    Ming Gu    Jocelyn Guidry
Gaetan Hache    Steve Hague    Satoshi Hamaguchi
Sven Hammarling    Mike Hansen    Richard Hanson
Richard Harke    Bill Hart    Vilya Harvey
Martin Hassner    Arthur S. Hathaway    Dan Hatton
Waldek Hebisch    Karl Hegbloom    Ralf Hemmecke
## Contents

1 Axiom and Category Theory  
   1.1 Covariance and Contravariance .......................... 1  
   1.2 Axiom Type Lattice ..................................... 2  
   1.3 Terms to Understand .................................... 2  
   1.4 Category Definition .................................... 3  
   1.5 Monoids and Groups ................................... 4  

2 Axiom Implementation Details  
   2.1 Makefile .............................................. 5  

3 Writing Spad Code  
   3.1 The Description: label and the )describe command ....................... 7
New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation’s website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we’ve broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We’ve also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I’m looking forward to future milestones.

With that in mind I’ve introduced the theme of the “30 year horizon”. We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How will we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The “30 year horizon” is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 (iHy)
Chapter 1

Axiom and Category Theory

1.1 Covariance and Contravariance

Axiom has an order relation between types. The types can be in one of five possible relationships.

A type can be more general than another type. For example, Integer is more general than PositiveInteger.

A type can be more specific than another type. Conversely PositiveInteger is more specific than Integer.

A type can be equal to another type.

A type can be converted or coerced to another type. For example, Fraction(Polynomial(Integer)) can be coerced to Polynomial(Fraction(Integer)).

A type can be unrelated to another type. String and Expression are not related.

Covariance is converting from a wider type to a narrower type. For instance, converting from Matrix(Float) to Matrix(Integer).

Contravariance is converting from a narrower type to a wider type. For instance, converting from Matrix(Integer) to Matrix(Float).

Invariance means that one type cannot convert to another. For instance, a Matrix(Float) which contains numbers which cannot be represented as Integers cannot be converted to a Matrix(Integer).

These facts form an order relation, which by definition is reflexive, transitive and antisymmetric.

Reflexive means that Integer = Integer.

Transitive means that PositiveInteger < Integer < Float implies that PositiveInteger < Float.

Antisymmetric means that PositiveInteger < Float implies not(Float < PositiveInteger).
1.2 Axiom Type Lattice

The types in Axiom form a lattice based on the order relationship. It is a lattice because Axiom supports multiple inheritance.

References of interest include:
Saunders Mac Lane “Categories for the Working Mathematician”
Steve Awodey “Category Theory”
ftp://sumin.in.ua/Books/DVD-021/Awodey_S._Category_Theory(en)(305s).pdf
“Introduction to Category Theory”
www.youtube.com/watch?v=eu0rj5C20tg
lucacardelli.name/Papers/OnUnderstanding.A4.pdf
staffwww.dcs.shef.ac.uk/people/A.Simons/research/reports/addaxiom.pdf
Dana Scott “Data Types as Lattices”
www.cs.ox.ac.uk/files/3287/PRG05.pdf
Roland Backhouse and Marcel Bijsterveld “Category Theory as Coherently Constructive Lattice Theory” November 1994

1.3 Terms to Understand

Suppose we wish to join Complex with Polynomial(Integer). What would elements of this combination look like?
The union of the two is a co-product of topological spaces.
The simple combination is not simply adding elements since
\[ i + x^2 \]

is not a valid combination.
We need the algebraic co-product, known as the tensor product. We end up with a domain of Complex(Polynomial(Integer)).

\[ \rightarrow a:\text{Complex(POLY(INT))}:=\%i+3\times \]
\[ 3\times + \%i \]

Type: Complex(Polynomial(Integer))

\[ \rightarrow a::\text{POLY(COMPLEX(INT))} \]
1.4 Category Definition

A category has four parts. We need a set of objects, usually represented as dots. We need a set of arrows (maps, morphisms), from dot to dot. We need a way to compose arrows in an associative manner. We need an identity arrow from a dot to itself.

The set of all arrows from dot A to dot B is written as $\text{Hom}_c(A, B)$ or, sometimes $C(A, B)$. Notice that the set $C(A, B)$ is disjoint from $C(A, D)$ since each arrow has a unique domain and co-domain.

For the example of the category Set, the objects are sets and the arrows are functions between sets. For the category Ring, the objects are rings and the arrows are ring homomorphisms. Similarly for the category Group, the dots are groups and the arrows are group homomorphisms. For a fixed Ring $R$, the category $R$-Mod has dots which are left $R$-modules and the arrows are $R$-module homomorphisms. We can also look at the category $\text{Mod-R}$ which has dots of right $R$-modules and arrows which are $R$-module homomorphisms. For the category $K$, if $K$ is a field, the dots are $K$-vector spaces and the arrows are $K$-linear transformations.

In Axiom the dots are Types (such as Integer or Character) and the arrows are functions between them with signature:

$$f : \text{Integer} \rightarrow \text{Character}$$

Relations between categories is called a [functor](https://en.wikipedia.org/wiki Functor). A functor $F$ takes things in category $C$ into things in category $D$. We need a function on objects which maps objects of $C$ to objects of $D$. We need a function on arrows which take arrows of $C$ to arrows of $D$.

The categories $C$ and $D$ well defined structure. They have a domain and co-domain of arrows. They have identity arrows. There is a rule of composition of arrows. These form commutative diagrams.

First we have to make sure the functor $F$ maintains the domain and co-domain structure of $C$. When we apply functor $F$ to $C$ we need to preserve all of the structure so $F$ has to be defined on all of these properties. If we look at two dots in category $C$ and a function $f$ which is an arrow in $C$

$$\begin{align*}
    f & : A \rightarrow B \\
    Ff & : FA \rightarrow FB
\end{align*}$$

then the functor $F$ has to operate on everything so we get:

$$\begin{align*}
    F \circ \text{dom} & = \text{dom} \circ F
\end{align*}$$

This means that if $\text{dom}$ is the domain function in $C$ then the functor $F$ commutes with $\text{dom}$. That is, applying $F(\text{dom}(f)) = \text{dom}(F(f))$. 

$$3x + \%i \quad \text{Type: Polynomial(Complex(Integer))}$$
Next we have to make sure the functor $F$ maintains the identity arrow of $C$. From the above we know that $F(\text{id}(x)) = \text{id}(F(x))$.

Finally we have to make sure that the rule for composition of arrows in $C$ is preserved. So the functor $F$ has to make sure that what composes in $C$ also composes with the same diagram in $D$.

Some standard functors are the identity functor $1_c$ which just maps $C$ to $C$. We can form a functor which forgets properties so that the category Group could map to its underlying set. We can lift a category by forgetting properties, for example, lifting the category of Abelian Group $C$ to Group $D$ by “forgetting” the commutative property of $C$. Similarly the category Ring or the category Module can be mapped to the underlying Abelian Group. There is also the Constant functor which maps all of the dots in $C$ to a single dot in $D$ and all of the arrows in $C$ to the identity arrow in $D$.

The category CommutativeRing $R$ can be mapped to a Group with the functor $GL_n$ which is the group of invertible $NxN$ matrices with entries in the CommutativeRing $R$.

### 1.5 Monoids and Groups

Given a single element set and a set of arrows from that element to itself we know from the associative property that $(fg)h = f(gh)$ and from the identity property that $ef = f = fe$.

A 1-object category is a monoid. A 1-object category where all of the arrows are invertible is a group.

If we restrict the category so there is at most one arrow between any two objects in the set then we have an ordered set.

A functor $F$ from category $C$ to category $D$ consists of

- object function takes objects of $C$ to objects of $D$
- arrow function takes arrows of $C$ to arrows of $D$

Structurally we have 3 things to preserve.

- domains and co-domains of arrows. In order to preserve structure the functor $F$ has to commute with the domain and co-domain functions. That is, $F(\text{dom}(f)) = \text{dom}(F(f))$ and $F(\text{co} - \text{dom}(f)) = \text{co} - \text{dom}(F(f))$.
- identity arrows. The functor $F$ must preserve identity so $F(\text{id}(x)) = \text{id}(F(x))$.
- composition properties of arrows. The functor $F$ must take commuting diagrams to commuting diagrams.
Chapter 2

Axiom Implementation Details

2.1 Makefile

This book is actually a literate program\cite{2} and can contain executable source code. In particular, the Makefile for this book is part of the source of the book and is included below. Axiom uses the “noweb” literate programming system by Norman Ramsey\cite{6}.
Chapter 3

Writing Spad Code

3.1 The Description: label and the )describe command

The `describe` command will print out the comments associated with Axiom source code elements. For the category, domain, and package sections the text is taken from the `Description:` keyword.

This information is stored in a database and can be queried with

```lisp
)lisp (getdatabase '|Integer| 'documentation)
```

for the Integer domain. However, this information has other uses in the system so it contains tags and control information. Most tags are removed by the `describe` function since the output is intended to be displayed in ASCII on the terminal.

The `Description:` keyword is in the comment block just after the `abbreviation` command. It is freeform and the paragraph will be reflowed automatically to allow for about 60 characters per line, adjusted for spaces. The `Description:` section should be written after the keyword in the “+++” comments as in:

```lisp
)abbrev package D03AGNT d03AgentsPackage
++ Description:
++ This package does some interesting stuff. We can write multiple
++ lines but they should all line up with the first character of
++ the Description keyword. Special \spad{terms} will be removed.
++
++ The above line will force a newline. So will ending a line with \br
++ \tab{5}This will allow primitive formatting\br
++ \tab{5}So you can align text\br
++ \tab{10}Start in column 11\tab{5}and skip 5 spaces\br
++ \tab{10}End in column 11\tab{7}and count out the needed spaces\br
++ \tab{5}note that the last line will not need the br command
```
As the comment says, the Description should all be aligned under the “D” in Description. You can indent using \(\texttt{\textbackslash tab}\{n\}\) which will insert \(n\) spaces. You can force a newline in two ways. Either include a blank line (with the “++” comments) or use the \(\texttt{\textbackslash br}\) keyword.

Due to lousy parsing algorithms for comments there are various ways this can all go wrong. There should not be any macros between the Description: section and the beginning of the definition. This is wrong. It will cause the

\[
\texttt{)}\texttt{describe package d03AgentsPackage}
\]

to give the wrong output because it does not find the end of the description section properly.

\[
\texttt{)}\texttt{abbrev package D03AGNT d03AgentsPackage}
\]

\[\texttt{++ Description:}\]
\[\texttt{++ This description does not work}\]

\[
\texttt{LED} = \Rightarrow \text{List Expression DoubleFloat}
\]

\[
d03AgentsPackage(): \texttt{E} = \texttt{I where}
\]

In the Description: section the \(\texttt{\textbackslash tab}\{nn\}\) function will be transformed into \(nn\) spaces. If you end each line with a \(\texttt{\textbackslash br}\) you can control alignment.

\[
\texttt{++ Description:}\]
\[\texttt{++ This is an example of a table alignment}\]
\[
\texttt{\textbackslash tab\{5\}First Item\textbackslash tab\{5\} This will line up with the following line}\]
\[
\texttt{\textbackslash tab\{5\}Second Item\textbackslash tab\{4\} This will line up with the following line}\]
\[
\texttt{\textbackslash tab\{5\}Third Item\textbackslash tab\{5\} This will line up with the following line}\]

If the main body of the category, domain, or package begins with properties rather than functions the Description will be incorrectly recorded. This is a known bug finding the end of the Description section. For instance, this

\[
\texttt{++ Description:}\]
\[\texttt{++ The category of Lie Algebras.}\]
\[\texttt{++ It is used by the domains of non-commutative algebra,}\]
\[\texttt{++ LiePolynomial and XPBWPolynomial.}\]

\[
\texttt{LieAlgebra(R: CommutativeRing): Category == Module(R) with}\]
\[\texttt{NullSquare}\]
\[
\texttt{\texttt{\textbackslash axiom\{NullSquare\}} means that \texttt{\textbackslash axiom\{[x,x] = 0\}} holds.}\]
\[\texttt{JacobiIdentity}\]
\[
\texttt{\texttt{\textbackslash axiom\{JacobiIdentity\}} means that}\]
\[\texttt{\texttt{\textbackslash axiom\{[x,[y,z]]+[y,[z,x]]+[z,[x,y]] = 0\}} holds.}\]
\[\texttt{construct: (\$\$) -> \$}\]
\[\texttt{\texttt{\textbackslash axiom\{construct(x,y)\}} returns the Lie bracket of \texttt{\textbackslash axiom\{x\}}}\]
\[\texttt{\texttt{\textbackslash axiom\{y\}}.}\]

will give the output
3.1. **THE DESCRIPTION: LABEL AND THE )DESCRIBE COMMAND**

JacobiIdentity means that \([x,[y,z]]+[y,[z,x]]+[z,[x,y]] = 0\) holds.

but reordering it to read:

```plaintext
++ Description:
++ The category of Lie Algebras.
++ It is used by the domains of non-commutative algebra,
++ LiePolynomial and XPBWPolynomial.
```

LieAlgebra(R: CommutativeRing): Category == Module(R) with
   construct: ($,$) -> $
      ++ \texttt{construct(x,y)} returns the Lie bracket of \texttt{x}
      ++ and \texttt{y}.
   NullSquare
      ++ \texttt{NullSquare} means that \texttt{[x,x] = 0} holds.
   JacobiIdentity
      ++ \texttt{JacobiIdentity} means that
      ++ \texttt{[x,[y,z]]+[y,[z,x]]+[z,[x,y]] = 0} holds.
```

will give the output

The category of Lie Algebras. It is used by the domains of non-commutative algebra, LiePolynomial and XPBWPolynomial.

which is correct.

```
PROJECT=bookvol2
TANGLE=/usr/local/bin/NOTANGLE
WEAVE=/usr/local/bin/NOWEAVE
LATEX=/usr/bin/latex
MAKEINDEX=/usr/bin/makeindex
```

all:
```
${WEAVE} -t8 -delay ${PROJECT}.pamphlet >${PROJECT}.tex
${LATEX} ${PROJECT}.tex 2>/dev/null 1>/dev/null
${MAKEINDEX} ${PROJECT}.idx
${LATEX} ${PROJECT}.tex 2>/dev/null 1>/dev/null
```
Bibliography


   http://axiom.axiom-developer.org

   http://www.aldor.org


   http://www.eecs.harvard.edu/~nr/noweb

[7] Daly, Timothy, "The Axiom Literate Documentation"
   http://axiom.axiom-developer.org/axiom-website/documentation.html