The 30 Year Horizon

Manuel Bronstein  
William Burge  
Timothy Daly  
James Davenport  
Michael Dewar  
Martin Dunstan  
Albrecht Fortenbacher  
Patrizia Gianni  
Johannes Grabmeier  
Jocelyn Guidry  
Richard Jenks  
Larry Lambe  
Michael Monagan  
Scott Morrison  
William Sit  
Jonathan Steinbach  
Robert Sutor  
Barry Trager  
Stephen Watt  
Jim Wen  
Clifton Williamson

Volume 13: Proving Axiom Correct
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.
or 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  Yurij 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. Broadbery  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 Coul
Don Coppersmith  George Corliss  Robert Corless
Gary Cornell  Meino Cramer  Jeremy Du Croz
David Cyganski  Nathaniel Daly  Timothy Daly Jr.
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  Korinn 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 Here is a problem ................................................. 3
   1.1 Approaches ................................................. 4

2 Theory ..................................................................... 7

3 Software Details ....................................................... 9
   3.1 Installed Software ............................................. 9
   3.2 Coq Spad proofs ................................................. 11
   3.3 ACL2 Lisp proofs ................................................. 11
   3.4 Lisp to Hardware ................................................. 11
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))
Our basic premise is that the ability to construct and modify programs will not improve without a new and comprehensive look at the entire programming process. Past theoretical research, say, in the logic of programs, has tended to focus on methods for reasoning about individual programs; little has been done, it seems to us, to develop a sound understanding of the process of programming — the process by which programs evolve in concept and in practice. At present, we lack the means to describe the techniques of program construction and improvement in ways that properly link verification, documentation and adaptability.

— Scherlis and Scott (1983) in [Mason 86]
Chapter 1

Here is a problem

The goal is to prove that Axiom’s implementation of the Euclidean GCD algorithm is correct. From category EuclideanDomain (EUCDOM) we find the implementation of the Euclidean GCD algorithm:

```spad
gcd(x,y) == --Euclidean Algorithm
   x:=unitCanonical x
   y:=unitCanonical y
   while not zero? y repeat
      (x,y):= (y,x rem y)
      y:=unitCanonical y -- this doesn’t affect the
      -- correctness of Euclid’s algorithm,
      -- but
      -- a) may improve performance
      -- b) ensures gcd(x,y)=gcd(y,x)
      -- if canonicalUnitNormal

   x
```

The unitCanonical function comes from the category IntegralDomain (INTDOM) where we find:

```spad
unitNormal: % -> Record(unit:%,canonical:%,associate:%)
   ++ unitNormal(x) tries to choose a canonical element
   ++ from the associate class of x.
   ++ The attribute canonicalUnitNormal, if asserted, means that
   ++ the "canonical" element is the same across all associates of x
   ++ if \spad{unitNormal(x) = [u,c,a]} then
      ++ \spad{u*c = x}, \spad{a*u = 1}.
unitCanonical: % -> %
   ++ \spad{unitCanonical(x)} returns \spad{unitNormal(x).canonical}.
```

implemented as
UCA ==> Record(unit:%,canonical:%,associate:%)
if not (% has Field) then
    unitNormal(x) == [1$%,x,1$%]$UCA -- the non-canonical definition
    unitCanonical(x) == unitNormal(x).canonical -- always true
    recip(x) == if zero? x then "failed" else _exquo(1$%,x)
    unit?(x) == (recip x case "failed" => false; true)
if % has canonicalUnitNormal then
    associates?(x,y) ==
        (unitNormal x).canonical = (unitNormal y).canonical
else
    associates?(x,y) ==
        zero? x => zero? y
        zero? y => false
        x exquo y case "failed" => false
        y exquo x case "failed" => false
        true

1.1 Approaches

There are several systems that could be applied to approach the proof.
The plan is to initially look at Coq and ACL2. Coq seems to be applicable at the Spad level.
ACL2 seems to be applicable at the Lisp level. Both levels are necessary for a proper proof.
Coq is very close to Spad in spirit so we can use it for the high-level proofs.
ACL2 is a Lisp-level proof technology which can be used to prove the Spad-to-Lisp level.
There is an LLVM to ACL2 translator which can be used to move from the GCL Lisp level
to the hardware since GCL compiles to C.
Quoting from Hardin [Hardin 14]

LLVM is a register-based intermediate in Static Single Assignment (SSA) form. As such, LLVM supports any number of registers, each of which is only assigned once, statically (dynamically, of course, a given register can be assigned any number of times). Appel has observed that “SSA form is a kind of functional programming”; this observation, in turn, inspired us to build a translator from LLVM to the applicative subset of Common Lisp accepted by the ACL2 theorem prover. Our translator produces an executable ACL2 specification that is able to efficiently support validation via testing, as the generated ACL2 code features tail recursion, as well as in-place updates via ACL2’s single-threaded object (stobj) mechanism. In order to ease the process of proving properties about these translated functions, we have also developed a technique for reasoning about tail-recursive ACL2 functions that execute in-place, utilizing a formally proven “bridge” to primitive-recursive versions of those functions operating on lists.
Our translation toolchain architecture is shown in Figure 1. The left side of the figure depicts a typical compiler frontend producing LLVM intermediate code. LLVM output can be produced either as a binary “bitcode” (.bc) file, or as text (.ll file). We chose to parse the text form, producing an abstract syntax tree (AST) representation of the LLVM program. Our translator then converts the AST to ACL2 source. The ACL2 source file can then be admitted into an ACL2 session, along with conjectures that one wishes to prove about the code, which ACL2 processes mostly automatically. In addition to proving theorems about the translated LLVM code, ACL2 can also be used to execute test vectors at reasonable speed.

Note that you can see the intermediate form from clang with

```
clang -04 -S -emit-llvm foo.c
```

Both Coq and the Hardin translator use OCAML [OCAML 14] so we will have to learn that language.
CHAPTER 1. HERE IS A PROBLEM
Chapter 2

Theory

The proof of the Euclidean algorithm has been known since Euclid. We need to study an existing proof and use it to guide our use of Coq along the same lines, if possible. Some of the “obvious” natural language statements may require Coq lemmas.

From WikiProof [Wiki 14a] we quote:

Let

\[ a, b \in \mathbb{Z} \]

and \( a \neq 0 \) or \( b \neq 0 \).

The steps of the algorithm are:

1. Start with \((a, b)\) such that \(|a| \geq |b|\). If \( b = 0 \) then the task is complete and the GCD is \( a \).
2. if \( b \neq 0 \) then you take the remainder \( r \) of \( a/b \).
3. set \( a \leftarrow b, b \leftarrow r \) (and thus \(|a| \geq |b|\) again).
4. repeat these steps until \( b = 0 \)

Thus the GCD of \( a \) and \( b \) is the value of the variable \( a \) at the end of the algorithm.

The proof is:

Suppose

\[ a, b \in \mathbb{Z} \]

and \( a \neq 0 \) or \( b \neq 0 \).

From the division theorem, \( a = qb + r \) where \( 0 \leq r \leq |b| \)

From GCD with Remainder, the GCD of \( a \) and \( b \) is also the GCD of \( b \) and \( r \).

Therefore we may search instead for the \( \gcd(b, r) \).
Since $|r| \geq |b|$ and $b \in \mathbb{Z}$, we will reach $r = 0$ after finitely many steps.

At this point, $gcd(r, 0) = r$ from GCD with Zero.

We quote the Division Theorem proof [Wiki 14b]:

For every pair of integers $a, b$ where $b \neq 0$, there exist unique integers $q, r$ such that $a = qb + r$ and $0 \leq r \leq |b|$.

\section*{CHAPTER 2. THEORY}
Chapter 3

Software Details

3.1 Installed Software

Install CLANG, LLVM

http://llvm.org/releases/download.html

Install OCAML

sudo apt-get install ocaml

An OCAML version of gcd would be written

let rec gcd a b = if b = 0 then a else gcd b (a mod b)
val gcd : int -> int -> int = <fun>
Bibliography

3.2  Coq Spad proofs

[Bertot 04] Bertot, Yves; Castéran, Pierre
 "Interactive Theorem Proving and Program Development"
 Springer ISBN 3-540-20854-2

[OCAML 14] .
 The OCAML website
 ocaml.org

3.3 ACL2 Lisp proofs

[Kaufmann 14] Kaufmann, Matt; Moore, J Stromher
 "ACL2 Version 6.4"
 www.cs.utexas.edu/users/moore/ac2

3.4 Lisp to Hardware

[Daly 10] Daly, Timothy
 "Intel Instruction Semantics Generator"
 daly.axiom-developer.org/TimothyDaly_files/publications/sei/intel/intel.pdf

[Hardin 13] Hardin, David S.; McClurg, Jedidiah R.; Davis, Jennifer A.
 "Creating Formally Verified Components for Layered Assurance with an LLVM to ACL2 Translator"

[Hardin 14] Hardin, David S.; Davis, Jennifer A.; Greve, David A.; McClurg, Jedidiah R.
 "Development of a Translator from LLVM to ACL2"
 arxiv.org/pdf/1406.1566
[Mason 86] Mason, Ian A.
“The Semantics of Destructive Lisp”
Center for the Study of Language and Information ISBN 0-937073-06-7

[Wiki 14a] ProofWiki
“Euclidean Algorithm”
proofwiki.org/wiki/Euclidean_Algorithm

[Wiki 14b] ProofWiki
“Division Theorem”
proofwiki.org/wiki/Division_Theorem