Epistemology
Software Engineering
and
Formal Methods

C. Michael Holloway

Assessment Technology Branch
Information & Electromagnetic Technology Division
Research & Technology Group
Langley Research Center
National Aeronautics and Space Administration
United States Government

Introduction

- One of the most basic questions anyone can ask is

  "How do I know that what I think I know is true?"

- The study of this question is called ***epistemology***

- Traditionally, epistemology has been considered to be of legitimate interest only to philosophers, theologians, and three-year-old children

- At least one other group should be very interested in epistemology -- software engineers -- because lack of understanding in this area plagues the field

The Basics of Epistemology

- There are three basic answers to the question of how do we know what is true
  - -- *Authority:* truth is given to us by a knowledgeable person
  - -- *Reason:* truth is what can be proven using the rules of deductive logic
  - -- *Experience:* truth is what can be encountered through one or more of the senses
    - * *Anecdotal experience:* truth is what an individual or a group of individuals experiences personally
    - * *Empirical evidence:* truth is what can be verified through carefully controlled experiments

Examples of Truth by Authority

- The Ten Commandments

  (omniscient authority)

- 1-year-old, pointing to the family cat: "Whatsthat?"

  father: "Kitty"

  (human authority)

Examples of Truth by Reason

- If that creature is a tove, then it is slithy
  That creature is a tove
  Therefore, that creature is slithy

- If the airplane was built by Boeing, then it is a jet
  The airplane is not a jet
  Therefore, the airplane was not built by Boeing

- $X + Y = 7$
  $3Y - 2X = 1$
  Therefore, $X = 4$ and $Y = 3$

Examples of Truth by Anecdotal Experience

- Smoking doesn’t shorten your life because my father smoked all his life and lived to be 95.
- Whenever I have the hiccups, I hold my breath and count to 10 and they go away. Therefore, holding your breath and counting to 10 cures the hiccups.
- We used method M and had 40% fewer bugs in testing. You should use method M, too.

Examples of Truth by Empirical Evidence

- The dive-recovery flap for the P38 in World War II developed through tests in Langley's 8-Foot High Speed Tunnel

- 5,000 patients were given drug X. 5,000 patients were given no drugs at all. 4,998 of the patients given drug X got better within 1 week. 3 of the patients given no drugs at all got better within 1 week. Drug X helps.

Relative Strengths

- Omniscient Authority: *absolute truth*
- Reason: *conditional absolute truth*
- Empirical Evidence: *probable truth*
- Anecdotal Experience: *possible truth*
- Human Authority: *opinion*

How Does This Apply to Software Engineering?

- The software engineering community is full of claims

  “The best way to develop reusable software is to use object-oriented design.”

  “Programmers should never be allowed to test their own code.”

  “Getting control of the software process is the key -- SEI’s CMM is the way to do this.”

  “We need more standards!”

  “Much progress has been made in the last few years in improving the way we develop software.”

  “GOTO’s are harmful.”

  “CASE tools are the best way to improve software productivity.”

- Many people accept these, or other similar, claims as being true

The Fundamental Question

What is the

epistemological foundation

for accepting

these claims?

The Answer

- Logically sound arguments are rarely given
- Virtually no empirical evidence is cited
- Instead, software engineering is based almost entirely on a combination of human authority and anecdotal experience
  -- We know that technique C is good because Jane Doe, who is a recognized authority in the field, says that it is good (*human authority*)
  -- Jane Doe knows that it is good because she used it on a project once and got good results (*anecdotal experience*)
- This is a weak epistemological foundation, one on which no legitimate claims of success can be based

Implications of This Epistemological Weakness

- Until we get adequate evidence, we should be very cautious in the claims we make and the standards we set
  - It is fine to say, "*Method M seems to have improved our productivity, so you might want to try it.*"

    But it is dishonest to say, "*If you want to improve your productivity, you must use Method M.*"
  - "*Company R used method F and found errors they don't think they would have found using their old methods*," is fine

    "*Method F finds errors that other methods do not find*," is dishonest
- The software engineering community should be investigating methods for obtaining strong (that is, logical or empirical) evidence

Why Has More Not Been Done?

- The development of software engineering is following the same pattern as the development of other disciplines
  - –– Civil engineering, chemical engineering, aeronautical engineering, etc. all had periods in which they relied almost exclusively on anecdotal evidence
  - –– Often, it took major disasters to prompt changes
- It is hard
- It is expensive
- It is not glamorous
- Few people care: We haven't had a *major* disaster yet

Why Must More Be Done?

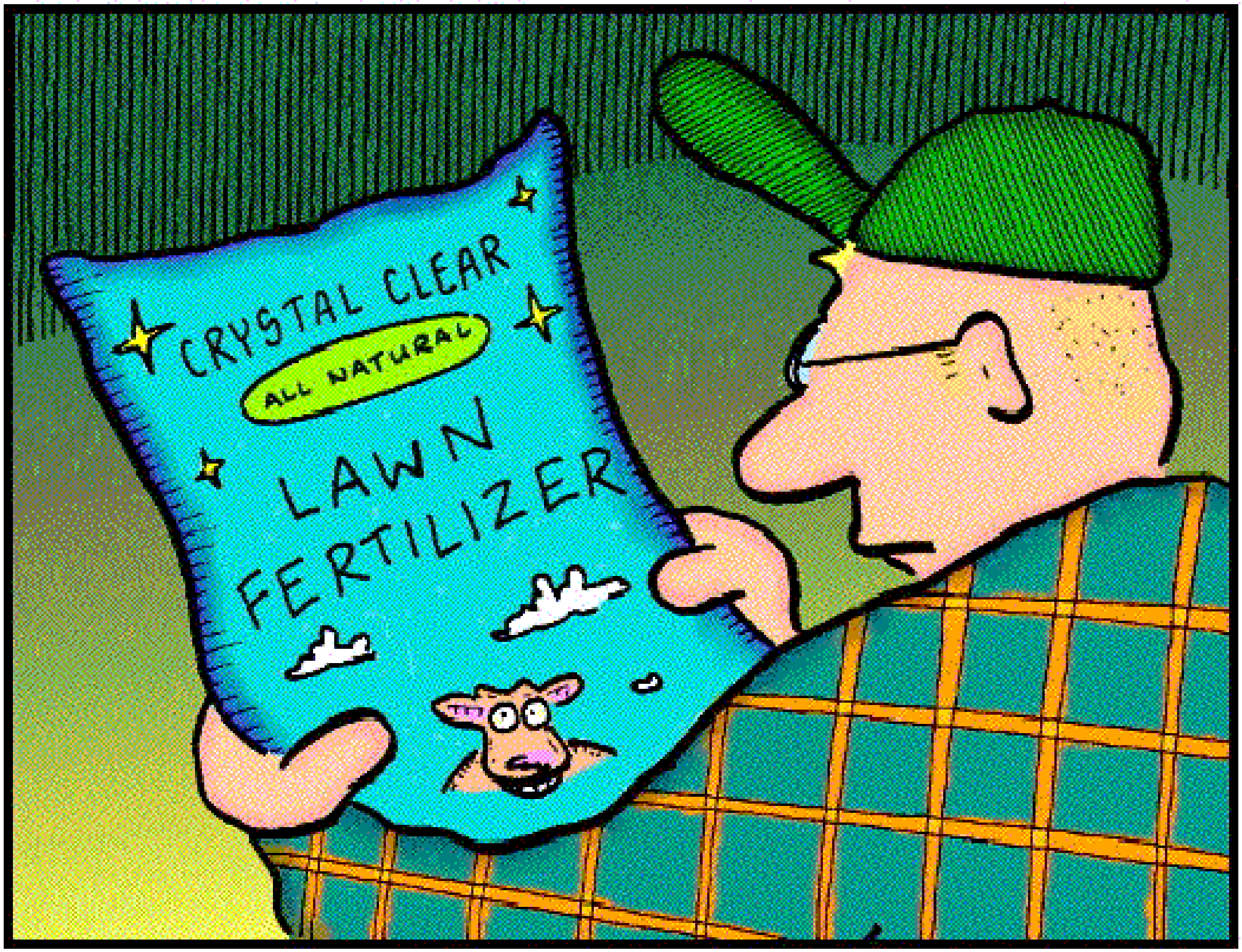
- Without adequate evidence, we are easily influenced by the latest bandwagon that goes rumbling by
- Without adequate evidence, we may well “cast-in-concrete” something that ought not even be “cast-in-mud”
- Without adequate evidence, the following two statements are equally as meaningless:

  -- You shall use method M in developing your software

  -- ‘Twas brillig by the slithy tove
- Without adequate evidence, disasters are inevitable

DOCTOR FUN

1-14-94



More marketing scams.

Towards Establishing a Valid Epistemological Foundation

- Recognize the fundamental need for such a foundation

- Understand the different approaches needed for *process* and *product*
  - Process questions (e.g., *Is method A better than method B?*) need to be answered empirically
  - Product questions (e.g., *Are my requirements consistent?*) need to be answered by an appropriate combination of logical and empirical methods

- Refuse to accept claims based on insufficient evidence

Current Research at LaRC

- Kelly Hayhurst (IETD/ATB) is leading an effort to develop an empirical evaluation of a particular approach to IV & V

  For more information, contact Kelly

  Email: k.j.hayhurst@LaRC.NASA.GOV

  Phone: 46215

- The formal methods team led by Ricky Butler (IETD/ATB) is investigating logical methods for answering product-type questions

  -- Other team members are Jim Caldwell, Victor Carreño, Michael Holloway, and Paul Miner

  -- Remainder of talk concerns this work

Further Reading on Epistemology

- If you are interested in more information on epistemology, I recommend you start with the following two books:

  – *Thales to Dewey*, by Gordon H. Clark, 2nd edition, 1989, ISBN 0-940931-26-5

  – *The Philosophy of Science*, by Gordon H. Clark, 2nd edition, 1987, ISBN 0-940931-18-4

- These two books contain pointers to most of the important philosophical works throughout the ages

Singular or Plural?

- Which of the following is correct?

Formal methods is the applied mathematics ...

OR

Formal methods are the applied mathematics ...

- Answer depends on the writer or speaker
- I will tend to use “formal methods” as singular

What is Formal Methods?

- Formal methods is the applied mathematics of computer systems engineering

- The mathematics of formal methods includes:

  -- predicate calculus (1st order logic)

  -- recursive function theory

  -- lambda calculus

  -- programming language semantics

  -- discrete mathematics: number theory, abstract algebra, etc.

What is Formal Methods?
(continued)

System Designed	*Engineering*	*Theory*
Bridge	**Civil**	**Classical Mechanics**
Airframe	**Aeronautical**	**Fluid Dynamics**
Nuclear Reactor	**Nuclear**	**Quantum Mechanics**
Digital Avionics System	**Software**	**Formal Methods**

Classical vs Computer Systems

Classical Systems	*Computer Systems*
continuous state space	discrete state space
smooth transitions	abrupt transitions
finite testing & interpolation acceptable	finite testing inadequate, interpolation unsound
mathematical modeling	prototyping & testing
build to withstand additional stress	build to specific assumptions
predictable	surprising

What Makes a Technique a Formal Method?

- Formal method = logic + programming language concepts
- Important attributes:

  -- logic based

  -- programming language concepts (e.g., data types, module structure, generics)

  -- fully and formally specified semantics

  -- should be able to express what is done without saying how it is done (i.e., non-procedural)

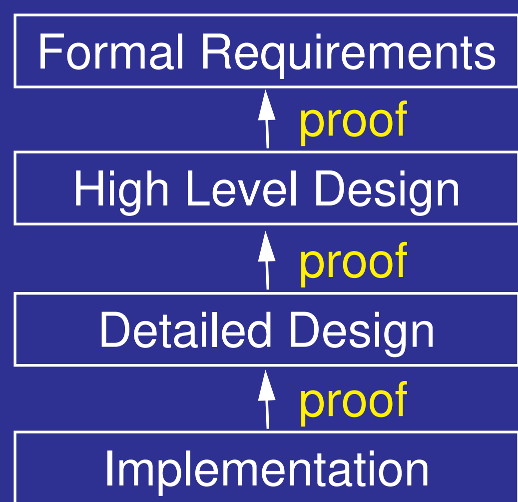
  -- supports the building of useful tools for analysis

Levels of Rigor of Formal Methods

- *Level 0*: No use of formal methods
- *Level 1*: Formal specification (using mathematical logic or a specification language with formal semantics) of all or part of a system
- *Level 2*: Formal specification at two or more levels of abstraction and paper-and-pencil proofs that the detailed specification satisfies the abstract one
- *Level 3*: Like level 2, except paper-and-pencil proofs are replaced by formal proofs checked by a semi-automatic theorem prover

Extent of Application

- Formal Methods is not an all-or-nothing approach
- Complete formal verification of a large complex system is impractical at this time



- Application of formal methods to critical portions of a system is practical and useful

Extent of Application (example)

- In the Reliable Computing Platform, we use formal methods to establish:

  ENOUGH_WORKING_HARDWARE

  $\supset$

  PROPER_OPERATION

- We use reliability analysis to calculate:

  Probability[ENOUGH_WORKING_HARDWARE]

- Reliability analysis relies on physical testing of devices to establish some important parameters

Level 1 Example: Phone Book
English Requirements

- The phone book shall store the phone numbers of a city
- Given a name, there shall be a way to retrieve an associated phone number
- It shall be possible to add and delete entries from the phone book

Level 1 Example: Phone Book
Choosing a Specification Approach

- How do we represent the phone book mathematically?

  1. A set of ordered pairs $\langle$name, number$\rangle$. Adding and deleting entries is by set addition and deletion.
  2. A function whose domain is all possible names and range is all phone numbers. Adding and deleting entries is by modification of function values.
  3. A function whose domain is only names currently in the phone book and range is phone numbers. Adding and deleting entries is by modification of the function domain and values. (Z style)

- We choose to use approach 2

Level 1 Example: Phone Book
Specifying the Book

- Using traditional mathematical notation, we would write:

Let $\boldsymbol{N}$ = set of names

$\boldsymbol{P}$ = set of phone numbers

$\textit{book}: \boldsymbol{N} \rightarrow \boldsymbol{P}$

- To indicate that we do not have a phone number for all possible names, but only for names of real people, we decide to use a special number: $\rho \in \boldsymbol{P}$

- An empty phone book is specified as follows:

$$\textit{emptybook}: \boldsymbol{N} \rightarrow \boldsymbol{P}$$

$$\textit{emptybook}(\textit{nm}) \equiv \rho$$

Level 1 Example: Phone Book
Accessing an Entry

Let $\mathbf{N}$ = set of names

$\mathbf{P}$ = set of phone numbers

$\mathbf{book:}\ \mathbf{N} \rightarrow \mathbf{P}$

$\mathbf{B}$ = set of functions: $\mathbf{N} \rightarrow \mathbf{P}$

$\mathbf{FindPhone:}\ \mathbf{B} \times \mathbf{N} \rightarrow \mathbf{P}$

`FindPhone(bk,name) = bk(name)`

Note that ***FindPhone*** is a higher-order function, because its first argument is a function

Level 1 Example: Phone Book
Adding/Deleting an Entry

Let $\boldsymbol{N}$ = set of names

$\boldsymbol{P}$ = set of phone numbers

$\boldsymbol{book}: \boldsymbol{N} \rightarrow \boldsymbol{P}$

$\rho \in \boldsymbol{P}$

$\boldsymbol{B}$ = set of functions: $\boldsymbol{N} \rightarrow \boldsymbol{P}$

$$\boldsymbol{AddPhone}: \boldsymbol{B} \times \boldsymbol{N} \times \boldsymbol{P} \rightarrow \boldsymbol{B}$$

$$\boldsymbol{AddPhone(bk, name, num)\ (x)} = \begin{cases} \boldsymbol{bk\ (x)} & \text{if } \boldsymbol{x} \neq \boldsymbol{name} \\ \boldsymbol{num} & \text{if } \boldsymbol{x} = \boldsymbol{name} \end{cases}$$

$$\boldsymbol{DelPhone}: \boldsymbol{B} \times \boldsymbol{N} \rightarrow \boldsymbol{B}$$

$$\boldsymbol{DelPhone(bk, name)\ (x)} = \begin{cases} \boldsymbol{bk\ (x)} & \text{if } \boldsymbol{x} \neq \boldsymbol{name} \\ \rho & \text{if } \boldsymbol{x} = \boldsymbol{name} \end{cases}$$

Level 1 Example: Phone Book
Complete Specification

Let $\boldsymbol{N}$ = set of names
$\boldsymbol{P}$ = set of phone numbers
$\boldsymbol{book}: \boldsymbol{N} \rightarrow \boldsymbol{P}$
$\rho \in \boldsymbol{P}$
$\boldsymbol{B}$ = set of functions: $\boldsymbol{N} \rightarrow \boldsymbol{P}$

$$\boldsymbol{emptybook}: \boldsymbol{N} \rightarrow \boldsymbol{P}$$
$$\boldsymbol{emptybook(nm)} \equiv \rho$$

$$\boldsymbol{FindPhone}: \boldsymbol{B} \times \boldsymbol{N} \rightarrow \boldsymbol{P}$$
$$\boldsymbol{FindPhone(bk, name)} = \boldsymbol{bk(name)}$$

$$\boldsymbol{AddPhone}: \boldsymbol{B} \times \boldsymbol{N} \times \boldsymbol{P} \rightarrow \boldsymbol{B}$$
$$\boldsymbol{AddPhone(bk, name, num)\,(x)} = \begin{cases} \boldsymbol{bk(x)} & \text{if } \boldsymbol{x} \neq \boldsymbol{name} \\ \boldsymbol{num} & \text{if } \boldsymbol{x} = \boldsymbol{name} \end{cases}$$

$$\boldsymbol{DelPhone}: \boldsymbol{B} \times \boldsymbol{N} \rightarrow \boldsymbol{B}$$
$$\boldsymbol{DelPhone(bk, name)\,(x)} = \begin{cases} \boldsymbol{bk(x)} & \text{if } \boldsymbol{x} \neq \boldsymbol{name} \\ \rho & \text{if } \boldsymbol{x} = \boldsymbol{name} \end{cases}$$

Level 2 Example: Phone Book
Putative Theorems

A putative theorem is a theorem that we know must be true if we have formulated the specification correctly.

Lemma putative 1:

```
FindPhone(AddPhone(bk,name,num),name) = num
```

Proof:

```
FindPhone(AddPhone(bk,name,num),name) = num

AddPhone(bk,name,num)(name) = num

num = num
```

Q.E.D.

Level 2 Example: Phone Book

Putative Theorems (continued)

Lemma putative 2: $\textit{bk(name)} = \rho \supset$

$\textit{DelPhone(AddPhone(bk,name,num),name)} = \textit{bk}$

Lemma putative 3: $(\forall i : name_i \neq name) \wedge$

$$
\begin{aligned}
&book = AddPhone(bk,\ name,\ num) \wedge \\
&book_1 = AddPhone(book,\ name_1\ ,\ num_1\ ) \wedge \\
&book_2 = AddPhone(book_1\ ,\ name_2\ ,\ num_2\ ) \wedge \\
&\quad \vdots \qquad\qquad\qquad \vdots \\
&book_n = AddPhone(book_{n-1}\ ,\ name_n\ ,\ num_n\ ) \\
\supset\ & \\
&FindPhone(book_n\ ,\ name) = num
\end{aligned}
$$

Formal methods can establish that a property holds even in the presence of an arbitrary number of operations; testing can never establish this.

Level 3 Example: Phone Book
PVS Specification

```
phonebook: THEORY
BEGIN

  names: TYPE
  name0: names
  ph_number: TYPE
  p_0: ph_number
  book: TYPE = [names -> ph_number]

  name: VAR names
  emptybook(name): ph_number = p_0
  bk: VAR book

  FindPhone(bk, name): ph_number = bk(name)

  num: VAR ph_number
  AddPhone(bk, name, num): book = bk WITH [name := num]

  DelPhone(bk, name): book = bk WITH [name := p_0]

  putative_1: LEMMA FindPhone(AddPhone(bk,name,num),name) = num

  putative_2: LEMMA bk(name) = p_0 IMPLIES
                    DelPhone(AddPhone(bk,name,num),name) = bk

END phonebook
```

Level 3 Example: Phone Book
Proof Using PVS

```
putative_1 :

  |-------
{1}   (FORALL (bk: book), (nm: names),(num: ph_number):
             FindPhone(AddPhone(bk, nm, num), nm) = num)


Rule? (skosimp*)
Repeatedly Skolemizing and flattening,
this simplifies to:
putative_1 :

  |-------
{1}   FindPhone(AddPhone(bk!1, nm!1, num!1), nm!1) = num!1



Rule? (expand "FindPhone" )
```

Level 3 Example: Phone Book
Proof Using PVS (continued)

```
Rule? (expand “FindPhone” )
Expanding the definition of FindPhone,this simplifies to:
putative_1 :

  |-------
{1}   AddPhone(bk!1, nm!1, num!1)(nm!1) = num!1

Rule? (expand “AddPhone” )
Expanding the definition of AddPhone,this simplifies to:
putative_1 :

  |-------
{1}   TRUE

which is trivially true.
Q.E.D.

Run time  = 1.02 secs.
Real time = 20.00 secs.
```

Level 1 Example: Phone Book
Deficiencies in the Specification

- Our specification does not rule out the possibility of someone having a “ρ” phone number
- We have not allowed multiple phone numbers per single name
- Our specification does not say anything about whether there should be a warning if a deletion is requested on name that is not in the phone book

How do we remedy these deficiencies?

Level 1 Example: Phone Book
Overcoming Deficiencies 1 & 2

Let $\boldsymbol{N}$ = set of names
$\boldsymbol{P}$ = set of phone numbers
$\boldsymbol{book}: \boldsymbol{N} \to \boldsymbol{2}^{\boldsymbol{P}}$
B = set of functions: $\boldsymbol{N} \to \boldsymbol{2}^{\boldsymbol{P}}$

$$\boldsymbol{emptybook(name)} \equiv \varnothing$$

$$\boldsymbol{FindPhone}: B \times \boldsymbol{N} \to \boldsymbol{2}^{\boldsymbol{P}}$$
$$\boldsymbol{FindPhone(bk,name)} = \boldsymbol{bk(name)}$$

$$\boldsymbol{AddPhone}: B \times \boldsymbol{N} \times \boldsymbol{P} \to B$$
$$\boldsymbol{AddPhone(bk,name,num)(x)} = \begin{cases} \boldsymbol{bk(x)} & \text{if } \boldsymbol{x} \neq \boldsymbol{name} \\ \boldsymbol{bk(name)} \cup \boldsymbol{\{num\}} & \text{if } \boldsymbol{x} = \boldsymbol{name} \end{cases}$$

$$\boldsymbol{DelPhone}: B \times \boldsymbol{N} \to B$$
$$\boldsymbol{DelPhone(bk,name)(x)} = \begin{cases} \boldsymbol{bk(x)} & \text{if } \boldsymbol{x} \neq \boldsymbol{name} \\ \varnothing & \text{if } \boldsymbol{x} = \boldsymbol{name} \end{cases}$$

Level 1 Example: Phone Book
An Additional Deficiency

- Notice that the function ***DelPhone*** deletes all of the phone numbers associated with a name
- There is no way to remove just one of the phone numbers that is associated with a given name
- The original requirements did not address this situation; to address it, we must define an additional function:

$$\textbf{\textit{DelPhoneNum:}}\; B \times \boldsymbol{N} \times \boldsymbol{P} \rightarrow B$$

$$\textbf{\textit{DelPhoneNum(bk,name,num)(x)}} = \begin{cases} \textbf{\textit{bk(x)}} & \text{if } \boldsymbol{x} \neq \textbf{\textit{name}} \\ \textbf{\textit{bk(name)}} \setminus \{\textbf{\textit{num}}\} & \text{if } \boldsymbol{x} = \textbf{\textit{name}} \end{cases}$$

Example: Phone Book
Revised Requirements

Original Requirements

- The phone book shall store the phone numbers of a city
- Given a name, there shall be a way to retrieve an associated phone number
- It shall be possible to add and delete entries from the phone book

Revised Requirements

- For each name in the city, a set of phone numbers shall be stored
- Given a name, there shall be a way to retrieve the associated phone numbers
- It shall be possible to add a new name and phone number
- It shall be possible to add new phone numbers to an existing name
- It shall be possible to delete a name from the phone book
- It shall be possible to delete one of the phone numbers associated with a name
- A warning need not be given for a requested deletion of a name not in the city
- A warning need not be given for a requested deletion of a non-existent phone number

Example: Phone Book
Observations

- Our specification is abstract. The functions are defined over infinite domains.
- In translating the requirements from English into a more formal notation, many things that were left out of the English were explicitly enumerated.
- The formal process exposed ambiguities and deficiencies in the requirements. E.g., we had to choose between

$$\textbf{book}: N \rightarrow P$$

$$\textbf{book}: N \rightarrow 2^{P}$$

- Putative theorem proving and scrutiny revealed deficiencies in the formal specification

Example: Phone Book
More Observations

- There are many different ways to formally specify
- No matter what representation you chose you are making some decisions that bias the implementation
- The goal is to minimize this bias and yet be complete
- The process of formalizing the requirements can reveal problems and deficiencies and lead to a better English requirements document also
- The formal specification process is similar to the mathematical modeling process of engineering disciplines

Formal Methods Research at LaRC

- Detailed design with complete level 3 verification of a Reliable Computing Platform
- Design with level 2/3 verification of a transient fault-tolerant clock synchronization circuit and fabrication of the circuit
- With SRI International & Rockwell-Collins, level 3 specification and verification of the microcode of the AAMP5 microprocessor
- With Odyssey Research Associates & Union Switch and Signal, level 3 specification and verification of next-generation railroad control system
- ORA & Honeywell, level 3 specification and verification of aircraft navigation functions

Formal Methods Research at LaRC
(continued)

- Vigyan & SRI working with Loral, JSC, JPL on level 3 specification and verification of some Space Shuttle functions
- SRI working with Allied-Signal on level 3 specification and verification of important algorithms for fault-tolerance
- Conduct periodic workshops on formal methods; previous ones in 1990, 1992, with next one planned for 1995
- Maintain extensive collection of information on the research, accessible through the World Wide Web at URL

  http://shemesh.larc.nasa.gov/fm-top.html

Document Title: NASA Langley Formal Methods Program (07-April-1994)

Document URL: http://shemesh.larc.nasa.gov/fm-top.html

NASA Langley Formal Methods Program

The major goals of NASA Langley's research program are to make formal methods practical for use on life-critical systems developed in the United States, and to orchestrate the transfer of this technology to industry through use of carefully designed demonstration projects. Towards these goals, work is underway to design and formally verify a fault-tolerant computing platform suitable for advanced flight-control applications. Also several direct technology transfer efforts have been initiated that apply formal methods to critical subsystems of real aerospace computer systems.

Rationale for Formal Methods Research

- Why is Formal Methods Necessary?
- What is Formal Methods?

Goals of Our Program, Strategy, and Research Team

Research Focus Areas

- Fault-Tolerant Systems
- Verification of Software
- Verification of Hardware Devices
- Civil Air Transport Requirements

Things to Take Home with You

- Publications
- FTP Directory

We would appreciate your comments about this collection of information. Please click here, to submit your comments. There is a reward awaiting any and all who take the time to submit comments.

C. Michael Holloway
c.m.holloway@LaRC.NASA.GOV
+1 804 864 1701

LaRC Home Page. WWW Virtual Library: Formal Methods

Last updated: Tue May 10 1994 (08:25.40)

Back Forward Home Reload Open... Save As... Clone New Window Close Window

What is Formal Methods?

One day, we'll have available a tutorial on formal methods, which will be accessed by clicking here.

Formal methods is the applied mathematics of computer systems engineering. There are many different types of formal methods with various degrees of rigor. The following is a useful (first-order) taxonomy of the degrees of rigor in formal methods:

Level-1:
: Formal specification of all or part of the system.

Level-2:
: Formal specification at two or more levels of abstraction and paper and pencil proofs that the detailed specification implies the more abstract specification.

Level-3:
: Formal proofs checked by a mechanical theorem prover.

Level 1 represents the use of mathematical logic or a specification language that has a formal semantics to specify the system. This can be done at several levels of abstraction. For example, one level might enumerate the required abstract properties of the system, while another level describes an implementation that is algorithmic in style.

Level 2 formal methods goes beyond Level 1 by developing pencil-and-paper proofs that the more concrete levels logically imply the more abstract-property oriented levels. This is usually done in the manner illustrated below.

Level 3 is the most rigorous application of formal methods. Here one uses a semi-automatic theorem prover to make sure that all of the proofs are valid. The *Level 3* process of *convincing* a mechanical prover is really a process of developing an argument for an ultimate skeptic who must be shown every detail.

It is also important to realize that formal methods is not an all-or-nothing approach. The application of formal methods to the most critical portions of a system is a pragmatic and useful strategy. Although a complete formal verification of a large complex system is impractical at this time, a great increase in confidence in the system can be obtained by the use of formal methods at key locations in the system.

LaRC Formal Methods Publications Request Form

To submit a request for paper copies of some of our publications, complete this form. (Note: This requires "forms" support in your WWW browser [for example, Mosaic 2.0 or better is required]). Once you have completed the information, click on the "Submit" button at the end to send your request to us.

The following lists many of the publications that are we are likely to have available. Select all of them that you would like to receive. To select multiple publications, hold down *Control* when selecting. If a publication in which you are interested does not appear in the list, use the comments section below to request it.

The Infeasibility of Quantifying the Reliability of Life–Critical Real–Time Software
An Elementary Tutorial on Formal Specification and Verification Using PVS
Formal Methods for Life–Critical Software
Formal Methods and Digital Systems Validation for Airborne Systems
NASA Langley's Research Program in Formal Methods (revised)
Second NASA Formal Methods Workshop 1992
Formal Design and Verification of a Reliable Computing Platform For Real–Time Control (Phase 1 Results)
Formal Design and Verification of a Reliable Computing Platform For Real–Time Control (Phase 2 Results)
High Level Design Proof of a Reliable Computing Platform
Design Strategy for a Formally Verified Reliable Computing Platform
Design for Validation
Verification of Fault–Tolerant Clock Synchronization Systems

Your Name:

Company:

Address Line 1:

Address Line 2:

Address Line 3:

City: **State:** **Zip Code:**

Country: USA

EMail:

Phone: **Fax:**

Epistemology

It’s Important, Learn About It

Software Engineering

It’s Immature, Work On It

Formal Methods

It’s Promising, Look For It