Lurch
Software for Teaching Mathematical Proofs

Nathan Carter
Bentley University
Northeast Section of the MAA
Fall 2008 Meeting
November 21, 2008
How do you teach proofs?
How did you learn them?
God’s Own Proofs!

This is a list of rules you are to obey when doing proofs. As these were given to me by God Himself you must heed them without fail!

1. To show $A \subseteq B$:
   
   (a) Let $x \in A$.
   
   (b) Show $x \in B$.

2. To show $A = B$ when $A$ and $B$ are sets:
   
   (a) Method 1
       
       i. Show $A \subseteq B$.
       ii. Show $B \subseteq A$.

   (b) Method 2
       
       i. Show $x \in A \iff x \in B$. 
3. To show $f = g$ when $f$ and $g$ are functions:

   (a) Show that $f$ and $g$ have the same domain and codomain.
   (b) Let $x \in Domain(f)$.
   (c) Show $f(x) = g(x)$.

4. To show a function $f$ is one to one:

   (a) Let $a, b \in Domain(f)$.
   (b) Assume $f(a) = f(b)$.
   (c) Show that $a = b$. 
God was Just Kidding

by Dr. Monks

God told me he was just kidding us with the last G.O.P. sheet. Here is what he told me he really wants us to do in our proofs.

1 Commandments

1.1 Let

Axiom 1. Thou shalt only use the word Let to declare variables.

Axiom 2. Thou mayest declare any variable one wishes.
[@#module@]
[@module level comments@]
[@include files@]
[@macro definitions@]

[@preprocessor directive@]...

[@data type or declaration@]...;

[@function definition@]...;
[End of file]
#define: Token replacement (replace the identifier with the given string)
#include: File inclusion (include the contents of the file)
#if: Conditional compilation (the constant expression is nonzero (TRUE)?)
#ifdef: Conditional compilation (the identifier is defined?)
#endif: Conditional compilation (the identifier is undefined?)
#line: Line number specification for purposes of error diagnostics
#undef: Token cancellation (cancel the preprocessor definition of the ident
Lurch, Lurch, Lurch, Lurch, ...

by Ken Monks
Dept. of Mathematics
University of Scranton

Thm: Let $f$ be a group homomorphism.
Then $f$ is injective $\iff$ Kernel($f$)={$e$}

(Hide lines 45,54-57)

Pf:
1. Given $f: (G,*,e)$->$\rightarrow(H,*,2,e2)$ is a group homomorphism
2. $(G,*,e)$ is a group
group homomorphism- 1
3. $(H,*,2,e2)$ is a group
group homomorphism- 1
4. $f: G\rightarrow H$
group homomorphism- 1
5. $(\forall a,(\forall b,f((a*b)) = (f(a) *2 f(b)))$
   $=>$
   group homomorphism- 1
6. Assume $f$ is injective
7. Let $x$ be arbitrary
   Assume $x$ in Kernel($f$)
8. $f(x) = e$
   Kernel- 1,8
9. $f(e) = e$
   Homomorphisms preserve identity 1
10. $f(x) = f(e)$
    Substitution 10,9
11. $x = e$
    injective- 6,11
12. $x$ in {$e$}
    in {$a$}+ 12
13. $<-$

Lurch 1999 Limitations

- Text only, no mathematical symbols
- One line at a time
- Very slow
- Many file types (parsing, logical system, documents)
- Extensible only by coding in Java
What should Lurch be?

- Lurch should be as indistinguishable from the ordinary activities of mathematics as possible, except for the additional services it provides.
- Lurch should provide the software infrastructure the mathematical community needs for validating rigorous mathematics.
What should Lurch be?

- Lurch should be as indistinguishable from the ordinary activities of mathematics as possible, except for the additional services it provides.
- Lurch should provide the software infrastructure the mathematical community needs for validating rigorous mathematics.

**Explicit validation:**
The user types in work, and then Lurch checks to be sure the work is correct.

**Implicit validation:**
Lurch provides ways to manipulate the document that guarantee correct results.
What should Lurch be?

- Lurch should be as indistinguishable from the ordinary activities of mathematics as possible, except for the additional services it provides.

- Lurch should provide the software infrastructure the mathematical community needs for validating rigorous mathematics.
Today’s Outline

• Introduction (almost done with this)
• Show you the long-term plan
• Show you the current state of affairs
• Let you tinker with the existing product
• Suggestions, comments, discussion, etc.
Current State of Affairs

Two bars
Today’s Purpose

• Get the word out to you about the project

• If it will help you already, take it!

• Or just keep it on your radar as it matures.

• Get feedback from you at this early stage.
Comparison: Mathematica, Maple, etc.

Mathematica

Use lists to represent collections of things

Lists are used to represent vectors, matrices, tensors, sets, ranges of integrals and plots, and groups of arguments in functions. Lists are given in curly brackets and their elements are separated by commas.

A common way to generate a list is with the Table command. Here \((x, 2, 10)\) is also a list, which specifies that \(x\) goes from 2 to 10. If \(x\) starts at 1, you can use \((x, 10)\) instead.

\[
\begin{align*}
\text{In[1]} &= \text{Table}[2^x, \{x, 2, 10\}] \\
\text{Out[1]} &= \{4, 8, 16, 32, 64, 128, 256, 512, 1024\}
\end{align*}
\]

A matrix is a list of lists. The equality sign is used to define \(m_x\), a \(3 \times 5\) matrix.

\[
\begin{align*}
\text{In[2]} &= m_x = \text{Table}[a \times b, \{a, 3\}, \{b, 5\}] \\
\text{Out[2]} &= \{(x, x^2, x^3, x^4, x^5), (2x, 2x^2, 2x^3, 2x^4, 2x^5), (3x, 3x^2, 3x^3, 3x^4, 3x^5)\}
\end{align*}
\]

Use \(\text{Part}[m, i, j]\) to get the element in the \(i\)th row and \(j\)th column of a matrix \(m\). The short notation \(m[[i, j]]\) works as well. You can use lists to group multiple results together.

\[
\begin{align*}
\text{In[3]} &= \text{Part}[m_x, 2, 3], m_x[[2, 3]] \\
\text{Out[3]} &= \{2x^5, 2x^3\}
\end{align*}
\]
Mathematica: Similarities

- Nice **user interface** for typing in mathematics
- Able to mix **comment text** with meaningful mathematical expressions
- The computer helps you in some way with some kind of **decision engine**
Mathematica: Differences

• Lurch is pedagogical; it helps the user learn. It may or may not also help find the answer.

• Lurch makes you do the work, and it just verifies your reasons.

• Lurch is for step-by-step work; Mathematica shows only the answer.

• Lurch will be able to do proofs.
Comparison: Fitch
Fitch: Similarities

- Same type of decision engine in the program, one that verifies the user’s work
- Each step is verified (not just an answer)
- The reasons provided are verified also
Fitch: Differences

- Limited to one logical system (can’t extend it in any way)
- Limited to logic only (can’t do actual mathematics, just generic predicates like $P$)
- Limited to one notation only (built-in)
- Limited to one type of verification only (type line, specify rule, specify supports)
Big Picture Goals

• **User interface** like that of Mathematica or Maple (or Scientific Workplace, etc.)

• **Decision engine** like that of Fitch (and similar applications), but much more robust

• **Extensibility and flexibility** to new topics, notations, and rules
Big Picture Goals

- **User interface** like that of Mathematica or Maple (or Scientific Workplace, etc.)
- **Decision engine** like that of Fitch (and similar applications), but much more robust
- **Extensibility and flexibility** to new topics, notations, and rules
- **Doing math in Lurch** should be just like doing it on paper, only better.
OpenMath

International standard for mathematical semantics
OpenMath
International standard for mathematical semantics

Javascript
Among the most popular scripting languages in the world
Lurch Core

Extends OpenMath functionality, incl. scriptability.
Introduces documents, with load, save, dependence.

OpenMath
International standard for mathematical semantics

Javascript
Among the most popular scripting languages in the world
Lurch Core

Extends OpenMath functionality, incl. scriptability. Introduces documents, with load, save, dependence.

OpenMath
International standard for mathematical semantics

Javascript
Among the most popular scripting languages in the world
**Lurch Core**

Extends OpenMath functionality, incl. scriptability. Introduces documents, with load, save, dependence.

**OpenMath**

International standard for mathematical semantics

**Javascript**

Among the most popular scripting languages in the world

**Lurch User Interface**
Work Done So Far

- **OpenMath** incorporated into Lurch Project, and extensions to it implemented
- **Javascript** incorporated into Lurch, and connected to OpenMath data type
- **Lurch Core** complete (filesystem interface, relationships among documents, handling script actions and document changes)
- **Lurch Alpha** created (simple user interface, used in classroom testing during Fall 2008)
Work Remaining

• Evaluating results of Fall 2008 classroom testing, as well as continued testing (*Winter 2008-2009* and thereafter)

• Build packages that make it easy for Lurch to be used in many areas of math (*Spring 2009* and thereafter)

• User Interface (*Summer 2009*)

• Continual ongoing enhancements
Structured Play

http://lurch.sourceforge.net/nesmaa08
Excessive Formality?

- There is a lot of formality in my examples because they are from a formal logic course.
- Lurch is quite flexible, and need not always require such formality; consider these rules:
  - Insert any line that is a propositional consequence of previous lines.
  - Insert any line that a computer algebra system verifies follows from earlier lines.
  - You can leave obvious statements “Unproven.”
Structured Play

http://lurch.sourceforge.net/nesmaa08
Testing Results: Quotes

- **Constant feedback**
  “It helped me learn how to do proofs through trial and error.”
  “...I liked using Lurch because I was able to see what was needed for rules to work.”
  “It told me if I was right or wrong”

- **Interface**
  “It was also easier to move lines around than erasing everything on paper.”
I used Lurch for experimentation;
I tinkered to learn the results of various actions.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Disagree nor Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
</table>

4.6
Testing Results: Likert

It is possible to do a proof in Lurch by experimental clicking and typing, without thinking.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neither Agree nor Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
</table>

3.2
Testing Results: Likert

This early version of Lurch is not yet beneficial to students.
The constant feedback Lurch provides about my work is valuable.

4.2
I found the unusual notation (such as @ and #) a big drawback to the current version.
It was helpful that proofs in Lurch looked just like proofs in our textbook.

Testing Results: Likert

Strongly Disagree | Disagree | Neither Agree nor Disagree | Agree | Strongly Agree

4.2
Testing Results: Likert

Learning to use Lurch took a lot of time that I could have spent learning logic instead.
Discussion
White Sheets