Lurch: A General-Purpose Math Checker

Nathan Carter

Bentley University

April 13, 2011
Background

1996  Initial ideas, in a Chaos & Fractals class with Ken Monks, University of Scranton
1999  Small prototype built
2006  Design discussions restarted in earnest
2007  Proposal accepted for exploratory funding from NSF, DUE #0763344
2008  Work began on current version
2011  Current state of project
Outline

Vision: What is the Lurch project aiming to do?
  Long-term goal
  Existing software
  How Lurch is different

Accomplishments: What have we built so far?
  Algebra and calculus
  Create your own equation-based math topics
  Create more advanced math topics

Future: Where is the project going from here?
  What we’ve learned
  Planned improvements
  Long-term goals
What would it be like to have a spell checker for mathematics?
\[ x^2 + 2x + 1 = (x + 1)^2 \text{ valid} \]
\[ = x^2 + 1 \text{ invalid} \]
\[ \frac{d}{dx} (\sin x^2) \]

\[ = (\cos x^2) \frac{d}{dx} x^2 \quad \text{valid} \]

\[ = (\cos x^2) \cdot 2x \quad \text{valid} \]
Theorem 3: In the Klein four-group, \((ab)a = b\).

Proof: We have

\[
(ab)a = (ba)a \quad \text{Abelian}
\]

\[
= b(aa) \quad \text{ Associativity}
\]

\[
= be \quad \text{Nilpotence of } a
\]

\[
= b \quad \text{Right Identity}
\]
Proof #6

Given: trapezoid ABCD with $\overline{AD} \cong \overline{BC}$

Prove: $\overline{AC} \cong \overline{BD}$

<table>
<thead>
<tr>
<th>Statements</th>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Trapezoid ABCD, $\overline{AD} \cong \overline{BC}$</td>
<td>1. Given</td>
</tr>
<tr>
<td>2. ABCD is isosceles Trapezoid</td>
<td>2. Definition of isosceles trapezoid</td>
</tr>
<tr>
<td>3. $\overline{DC} \cong \overline{DC}$</td>
<td>3. Reflexive prop of $\cong$</td>
</tr>
<tr>
<td>4. $\angle BCD \cong \angle ADC$</td>
<td>4. Base $\angle$'s in an isosceles trapezoid are $\cong$</td>
</tr>
<tr>
<td>5. $\triangle BCD \cong \triangle ADC$</td>
<td>5. SAS</td>
</tr>
<tr>
<td>6. $\overline{AC} \cong \overline{BD}$</td>
<td>6. CPCTC</td>
</tr>
</tbody>
</table>
Assuming $a_n \to L$ and $k \in \mathbb{R}$, I prove $ka_n \to kL$. We’ve assumed $a_n \to L$, which by definition of limit means

$$\forall \varepsilon_a > 0, \exists N_a \in \mathbb{R}, \forall n > N_a, |a_n - L| < \varepsilon_a.$$ 

Let $\varepsilon > 0$. I choose $\varepsilon_a = \frac{\varepsilon}{|k|}$, and so I have

$$\exists N_a \in \mathbb{R}, \forall n > N_a, |a_n - L| < \frac{\varepsilon}{|k|}.$$ 

Let $N = N_a$, and let $n > N$ be arbitrary. I now show that $|ka_n - kL| < \varepsilon$.

$$|ka_n - kL| = |k(a_n - L)| \quad \text{algebra}$$
$$= |k||a_n - L| \quad \text{abs. val. algebra}$$
$$\leq |k|\varepsilon_a \quad n > N = N_a$$
$$= |k| \cdot \frac{\varepsilon}{|k|} \quad \text{choice of } \varepsilon_a$$
$$= \varepsilon \quad \text{algebra}$$

Thus I have shown the definition of $ka_n \to kL$, as desired. \qed
Assuming \( a_n \to L \) and \( k \in \mathbb{R} \), I prove \( ka_n \to kL \).
We’ve assumed \( a_n \to L \), which by definition of limit means

\[
\forall \varepsilon_a > 0, \exists N_a \in \mathbb{R}, \forall n > N_a, |a_n - L| < \varepsilon_a.
\]

Let \( \varepsilon > 0 \). I choose \( \varepsilon_a = \frac{\varepsilon}{|k|} \), and so I have

Thus I have shown the definition of \( ka_n \to kL \), as desired. \( \square \)
Where should the green check marks go?

For example, do we need one after “Let $\varepsilon > 0$”? That is, what is to be validated?

Questions of mathematics, logic, and pedagogy

\[
|ka_n - kL| = |k(a_n - L)| \quad \text{algebra}
\]
\[
= |k||a_n - L| \quad \text{abs. val. algebra}
\]
\[
\leq |k|\varepsilon_a \quad n > N = N_a \quad \text{choice of } \varepsilon_a
\]
\[
= |k| \cdot \frac{\varepsilon}{|k|} \quad \text{algebra}
\]
\[
= \varepsilon
\]

Thus I have shown the definition of $ka_n \to kL$, as desired. \qed
Assuming $a_n \to L$ and $k \in \mathbb{R}$, I prove $ka_n \to kL$. We’ve assumed $a_n \to L$, which by definition of limit means

$$\forall \varepsilon > 0, \exists N_a \in \mathbb{R}, \forall n > N_a, |a_n - L| < \varepsilon.$$ 

Let $\varepsilon > 0$. I choose $\varepsilon_a = \frac{\varepsilon}{|k|}$, and so I have

Where should the green check marks go?

For example, do we need one after “Let $\varepsilon > 0$”? That is, what is to be validated?

Questions of mathematics, logic, and pedagogy

| $ka_n - kL$ | $= |k(a_n - L)|$ | algebra |
|-------------|-----------------|---------|
|             | $= |k||a_n - L|$ | abs. val. algebra |

Would too much validation clutter the document?

When do the green check marks decrease usability? That is, what is the best help we can provide?

Thus I have shown the definition of $ka_n \to kL$, as desired. □
Assuming $a_n \to L$ and $k \in \mathbb{R}$, I prove $ka_n \to kL$. We’ve assumed $a_n \to L$, which by definition of limit means

$$\forall \varepsilon_a > 0, \exists N_a \in \mathbb{R}, \forall n > N_a, |a_n - L| < \varepsilon_a.$$

Let $\varepsilon > 0$. I choose $\varepsilon_a = \frac{\varepsilon}{|k|}$, and so I have

\[
|ka_n - kL| = |k(a_n - L)| \quad \text{algebra}
\]
\[
= |k||a_n - L| \quad \text{abs. val. algebra}
\]

Where should the green check marks go?

For example, do we need one after “Let $\varepsilon > 0$”? That is, what is to be validated?

Questions of mathematics, logic, and pedagogy

Would too much validation clutter the document?

When do the green check marks decrease usability? That is, what is the best help we can provide?

Questions of user interface design, and what’s possible

Thus I have shown the definition of $ka_n \to kL$, as desired. □
Please solve this equation for $u$.

$$-2 + \frac{u}{7} = -6u + \frac{2}{7}.$$

$$\frac{u-2}{7} = 2 - 6u.$$

$$u - 2 = 7 \cdot 2 - 6u.$$

This equation has a different solution.
Existing software: Maple

\[
\int \sin^2 x \, dx = \int \left( \frac{1}{2} + \frac{1}{2} \cos(2x) \right) \, dx \\
= \frac{1}{2} \int dx + \frac{1}{2} \int \cos(2x) \, dx \\
= \frac{1}{2} x + \frac{1}{2} \int \cos(2x) \, dx \\
= \frac{1}{2} x + \frac{1}{2} \int \cos(u) \, du \\
= \frac{1}{2} x + \frac{1}{4} \sin(u) 
\]
Existing software: *Fitch*
## Types of validation

### Implicit validation

The user directs the software, which performs each step of work.

- Such steps are automatically valid because a procedure that is known to be correct did them.
- The user does not get their hands dirty with syntax or rules.

The *Maple* tools shown a few slides ago use implicit validation.

### Explicit validation

The user performs each step of work, and the software checks it.

- Not restricted to a set of pre-set tools that do the work for them, they can enter arbitrary math.
- Validation must then be done after the user’s input.

*xyAlgebra* automatically validates each line of user input. *Fitch* validates a step after the user supplies a reason for it.
So why build something else?

Shortcomings of current offerings:

▶ No interoperability (cannot transfer data among them)
▶ No customizability (cannot change notation, rules, etc.)
▶ Some are commercial
▶ None are a word processor (cannot explain your work)
▶ Not all topics covered, and barriers to building more
Goals

How *Lurch* should avoid those shortcomings:

- Interoperability extends beyond *Lurch*, due to OpenMath.
- Each topic should be customizable (notation, rules, etc.).
- *Lurch* is free.
- *Lurch* should be a word processor, with math a central part.
- *Lurch* should be easy to extend to new topics.
What Lurch is NOT

*Lurch* is not for doing computations.

Maple does math for you.

*Lurch* checks your work.

*Lurch* is not a homework-grading system.

WeBWorK checks answers when you submit.

*Lurch* checks steps as you work.

*Lurch* is not a “proof assistant.”

Coq helps automate the proving process.

*Lurch* is a learning tool.
Lurch validates your work...

...and if all goes well, you get the stamp of approval.
Features of Lurch Lite 0.4 (current version)

- Simple word processor
- Math typesetting for output
- Math palette for input

Several math topics come with Lurch

- All algebra, trig, precalculus
- All differential calculus
- Very simple integral calculus
- Boolean algebras
- Group axioms
- Propositional logic
- Logic games

Create your own topics

- If they are based on axioms
- Equational or line-by-line format
Demonstration
Basic math topics, using a built-in CAS

1. Algebra and differential calculus
2. Simplifications only (not solving equations)
3. Cannot use earlier equations for substitution (thus no $u$-sub for $\int$, f.ex.)

Necessary content, easy to implement
Demonstration part 2 of 3

More advanced math topics, using rules the software learns from your documents

1. Show Ken’s group theory document and walk through it
2. Change an axiom and see the consequences
3. Show how you would create boolean algebra lib from scratch (just a bit)
4. Show final version and do a short proof using its rules
5. If time, show SKI calculus on Wikipedia, and its incarnation in *Lurch*

Good examples of what we want: easy creation of topics
Demonstration part 3 of 3

Non-equational math topics

1. Show example use of propositional logic document
2. Explain difference between this explicit version and the old implicit version
3. Open the document that defines it, and show how its rules were created
4. Examples of how this could be useful:
   4.1 Other logical systems
   4.2 Other formulations of the same system
   4.3 Other notations

We begin here to see use interface barriers. *Lurch* is starting to get clunky.
(However, the implicit validation version was too easy, based on student feedback.)
Demonstration
What we’ve learned from students

Constant feedback is valuable

Quotes

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

Likert-scale: Strongly disagree = 1, strongly agree = 5

▶ The constant feedback Lurch provides about my work is valuable. — 4.2
▶ I used Lurch for experimentation; I tinkered to learn the results of various actions. — 4.6
What we’ve learned from students

Taking away some of the tedium can be helpful

Quote

▶ “It was also easier to move lines around than erasing everything on paper.”

Allowing customizable notation is important

Likert-scale: Strongly disagree = 1, strongly agree = 5

▶ It was helpful that proofs in Lurch looked just like proofs in our textbook. — 4.2
This academic software project is unlike most in industry, in that the final product is not well-known in advance.

Sometimes that’s good because
- we and our students are the clients,
- deadlines are more flexible, and
- sometimes you build something better than you anticipated.

Sometimes it’s bad because
- it’s almost impossible to design the right thing the first time, and
- as we learn from what we’ve built, we often back up to change things.
What we’ve learned as developers: 1. Mission statement

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

That is, it should only add to your math experience, not change your math experience.

2. *Lurch* should provide the software infrastructure the mathematical community needs for validating rigorous mathematics.

That is, it should validate mathematical content created by you—a “spell-checker” for mathematical rigor.
What we’ve learned as developers: 2. Taxonomy

**Step** any mathematical expression that has a reason attached (whether that reason was attached automatically or manually)

\[
\text{Step} = (\text{expression, reason})
\]

**Reason** any Javascript function attached to an expression
Such functions, when run with steps as input, yield judgments, hints, and/or explanations as outputs.

\[
R(\text{step}) = \text{“Invalid (an incorrect substitution).”}
\]

**Editable item** a small portion of the document which, when the user attempts to edit it, performs an action specific to its meaning (e.g., opens up a list of choices, toggles true/false, shows a palette of relevant math symbols, etc.)

**Flarf** superfluous, uneditable filler
What we’ve learned as developers: 3. Don’t take shortcuts

We used a WebKit “widget” as our word processor.

That was not its intended purpose.
What we’ve learned as developers: 3. Don’t take shortcuts

The word processor in *Lurch* needs to satisfy the following requirements.

1. It must support all common document structures (lists, fonts, tables, etc.).
2. It must display mathematics in an attractive way.
3. It must be kept in one-to-one correspondence with an underlying document in OpenMath format.
4. It must allow editing of that underlying OpenMath content.
5. It must permit some regions of the document to be read-only.
OpenMath: The standard format for math semantics

Syntax

\[ \sum_{i=1}^{n} (i^2 + i) \]

Semantics

- Bind variable \( i \)
- Begin at integer 1
- End at free variable \( n \)
OpenMath: The standard format for math semantics

Syntax
\[ \sum_{i=1}^{n} (i^2 + i) \]

Semantics
- Bind variable \( i \)
- Begin at integer 1
- End at free variable \( n \)
- \( i \)
- 2
OpenMath: The standard format for math semantics

\[
\begin{align*}
\text{<OMBIND>}
\text{<OMS name="sum" cd="..."/>}
\text{<OMV name="i"/>}
\text{<OMI>1</OMI>}
\text{<OMV name="n"/>}
\text{<OMA>}
\text{<OMA>}
\text{<OMS name="plus" cd="openmath.org/cd/arith1"/>}
\text{<OMA>}
\text{<OMS name="power" cd="openmath.org/cd/arith1"/>}
\text{<OMV name="i"/>}
\text{<OMI>2</OMI>}
\text{</OMA>}
\text{<OMV name="i"/>}
\text{</OMA>}
\text{</OMA>}
\text{</OMBIND>}
\end{align*}
\]
What we’ve learned as developers: 3. Don’t take shortcuts

The word processor in *Lurch* needs to satisfy the following requirements.

1. It must support all common document structures (lists, fonts, tables, etc.).
2. It must display mathematics in an attractive way.
3. It must be kept in one-to-one correspondence with an underlying document in OpenMath format.
4. It must allow editing of that underlying OpenMath content.
5. It must permit some regions of the document to be read-only.
What we’ve learned as developers: 3. Don’t take shortcuts

The word processor in *Lurch* needs to satisfy the following requirements.

1. It must support all common document structures (lists, fonts, tables, etc.).
2. It must display mathematics in an attractive way.
3. It must be kept in one-to-one correspondence with an underlying document in OpenMath format.
4. It must allow editing of that underlying OpenMath content.
5. It must permit some regions of the document to be read-only.

WebKit provided items 1. and 2. instantly.
This was its attraction.
What we’ve learned as developers: 3. Don’t take shortcuts

The word processor in *Lurch* needs to satisfy the following requirements.

1. It must support all common document structures (lists, fonts, tables, etc.).
2. It must display mathematics in an attractive way.
3. It must be kept in one-to-one correspondence with an underlying document in OpenMath format.
4. It must allow editing of that underlying OpenMath content.
5. It must permit some regions of the document to be read-only.

We made significant changes to achieve items 4. and 5. This took a lot of time and effort.
What we’ve learned as developers: 3. Don’t take shortcuts

The word processor in *Lurch* needs to satisfy the following requirements.

1. It must support all common document structures (lists, fonts, tables, etc.).
2. It must display mathematics in an attractive way.
3. It must be kept in one-to-one correspondence with an underlying document in OpenMath format.
4. It must allow editing of that underlying OpenMath content.
5. It must permit some regions of the document to be read-only.

We asymptotically approached completion of item 3. This was how we learned not to take shortcuts.
What we’ve learned as developers: 4. Allow freedom

- Since 2008 we’ve described *Lurch* as “a word processor that can check your math.”
- All that time we built math topics that were very rigidly structured.
- The more freedom the user has, the more they can mess things up.
- That’s okay. Sometimes it’s even better—as long as the software can handle a messed-up document.

- Mantra: “*Lurch* is a word processor.”
Plans for 2011

1. Remove faulty technology

Using a web browsing widget as our word processor gave us a quick burst of speed, but eventually proved problematic.

I am currently in the process of rebuilding the word processor using tools designed for that purpose.

This is not the fun part.
Plans for 2011

2. Allow greater freedom

Replace all rigid document structures with freely editable ones.

This will rely on the concept of “groupers,” the design of which is still being finished.

We aim to use groupers to indicate many different kinds of structure in the document as unobtrusively as possible.

Example

**Theorem:** \( \forall x \in \mathbb{R}, \ x^2 \geq 0 \)

**Proof:** Let \( x \in \mathbb{R} \).

By Theorem 12, either \( x < 0 \), \( x = 0 \), or \( x > 0 \).

Case 1, \( x < 0 \): ...
Case 2, \( x = 0 \): ...
Case 3, \( x > 0 \): ...

Since the theorem holds in all possible cases, it is true.

This is almost the fun part.
2. Allow greater freedom

Replace all rigid document structures with freely editable ones.

This will rely on the concept of “groupers,” the design of which is still being finished.

We aim to use groupers to indicate many different kinds of structure in the document as unobtrusively as possible.

Example

**Theorem:** \( (\forall x \in \mathbb{R}, \ x^2 \geq 0) \)

**Proof:** Let \( x \in \mathbb{R} \).

By Theorem 12, either \( x < 0 \), \( x = 0 \), or \( x > 0 \).

Case 1, \( x < 0 \): ...

Case 2, \( x = 0 \): ...

Case 3, \( x > 0 \): ...

Since the theorem holds in all possible cases, it is true. \( \)

This is almost the fun part.
Plans for 2011

2. Allow greater freedom

Replace all rigid document structures with freely editable ones.

This will rely on the concept of “groupers,” the design of which is still being finished.

We aim to use groupers to indicate many different kinds of structure in the document as unobtrusively as possible.

Example

**Theorem:** \( \forall x \in \mathbb{R}, \ x^2 \geq 0 \)

**Proof:** Let \( x \in \mathbb{R} \).

\[(S(\text{By Theorem 12,})_R (E \text{either } x < 0, \ x = 0, \text{ or } x > 0.)_E)S\]

Case 1, \( x < 0 \): ...
Case 2, \( x = 0 \): ...
Case 3, \( x > 0 \): ...

Since the theorem holds in all possible cases, it is true.

This is almost the fun part.
3. Classroom testing

Recent versions of *Lurch* have not had a sufficiently stable user interface for testing in classrooms (other than mine). Yet testing is important because

1. without it, we cannot be sure our product is beneficial;
2. feedback from users brings new ideas that improve the product;
3. it is a means of sharing our work and improving math education; and
4. it is a requirement for continued external support.

We aim for a stable enough user experience that several classrooms can try *Lurch* in Fall 2011 and measure impact on student learning.
Long-term goals: 1. Authoring arbitrary new topics

Groups

A group is a set \( G \) with a binary operation \( \cdot \), a unary operation \( ^{-1} \), and a particular element \( e \) in \( G \) (called the identity element) satisfying the following axioms.

For any elements \( x, y, z \) in \( G \)

\[
(xy)z = x(yz) \quad \text{Associativity}
\]

\[
ex = x \quad \text{Left Identity}
\]

\[
x e = x \quad \text{Right Identity}
\]

\[
xx^{-1} = e \quad \text{Right Inverse}
\]

\[
x^{-1}x = e \quad \text{Left Inverse} 
\]
Long-term goals: 2. Getting mathematicians to create topics
Long-term goals: 3. Reaching critical mass

Following the model of the highly successful open-source mathematics software SAGE:

When *Lurch* reaches sufficient maturity so that enough mathematicians, students, and developers have seen its benefits for them and their students, we aspire to see

1. *Lurch* benefiting many classrooms,
2. instructors creating, maintaining, and sharing new topics,
3. programmers joining us and improving the software itself,
4. *Lurch* workshops at many local conferences, and
5. *Lurch* mini-conferences in many locales.
Discussion