MAA, VSU, Bentley

Using *Lurch* in the Classroom

Presented at the MA-DC-VA Section of the MAA Spring Meeting
April 16, 2010

Handout

This handout and other materials can be found online at the Lurch Website:

Main Site URL: **http://lurch.sourceforge.net**

Workshop URL: **http://lurch.sourceforge.net/madcva10**

The Lurch Project is supported in large part by the National Science Foundation's Division of Undergraduate Education, in the Course, Curriculum, and Laboratory Improvement program (grant #0736644). Views expressed in this document are not necessarily those of the National Science Foundation.

Bentley University 175 Forest St.
Morison 317 Waltham, MA 02452 **T** (781) 891-3171 **F** (781) 891-2457 ncarter@bentley.edu http://web.bentley.edu/empl/c/ncarter

Launching Lurch Lite

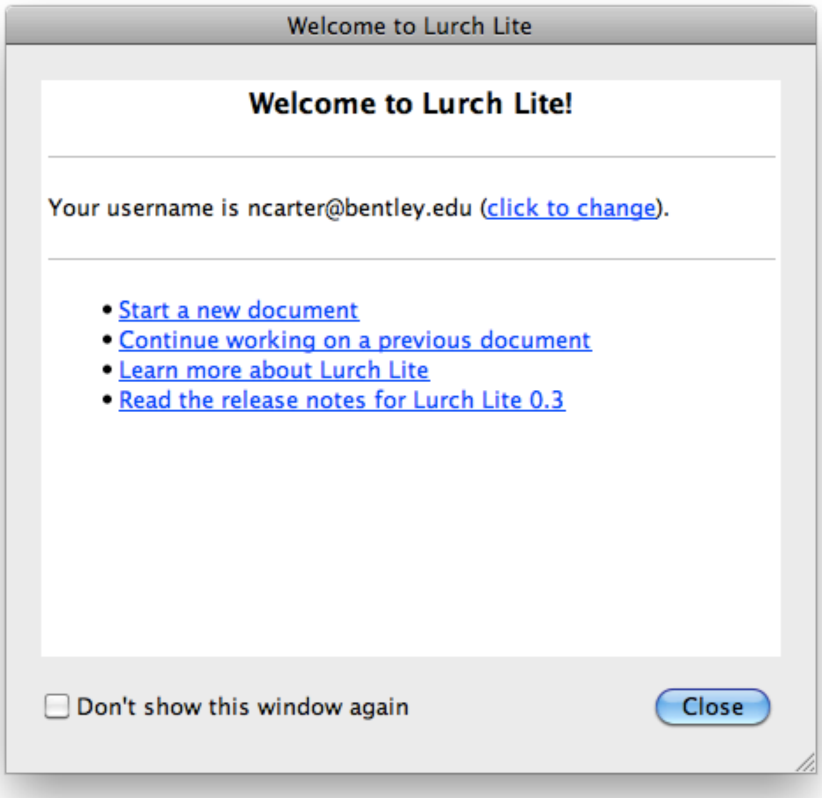
What is Lurch Lite?

Lurch Lite is not the final release of Lurch, and we call it Lurch Lite in order to emphasize that fact. During this workshop, we'll discuss some goals of the Lurch project that have not yet been achieved, and which motivate this modest naming convention. Lurch Lite is the application we will be using throughout the workshop today.

Running Lurch Lite

Lurch has been pre-installed on the computers in the workshop room. You can find it on the Start Menu, under Programs, in the Lurch folder.

The first time Lurch Lite launches, it shows you an welcoming message. You may close that window.



Welcome to Lurch Lite!

Your username is ncarter@bentley.edu (click to change).

- Start a new document
- Continue working on a previous document
- Learn more about Lurch Lite
- Read the release notes for Lurch Lite 0.3

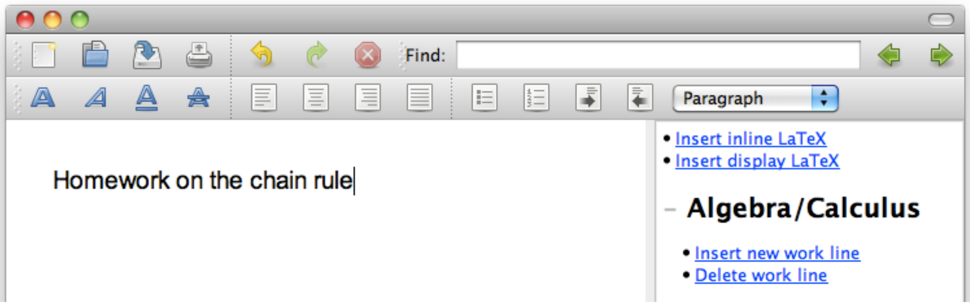
Example 1: Algebra/Calculus

Opening the Algebra and Calculus math topic

Once the Lurch Lite application is running, from the File menu, click “Choose topic.” Select the “Algebra and Calculus” topic and press Enter. The document will still be blank, waiting for you to type.

On the right hand side you’ll see hyperlinks that execute some commands. We’ll get to them soon.

Let’s pretend you’re a student doing a homework assignment for a differential calculus course. Go ahead and type “Homework on the chain rule” at the top, as shown here.



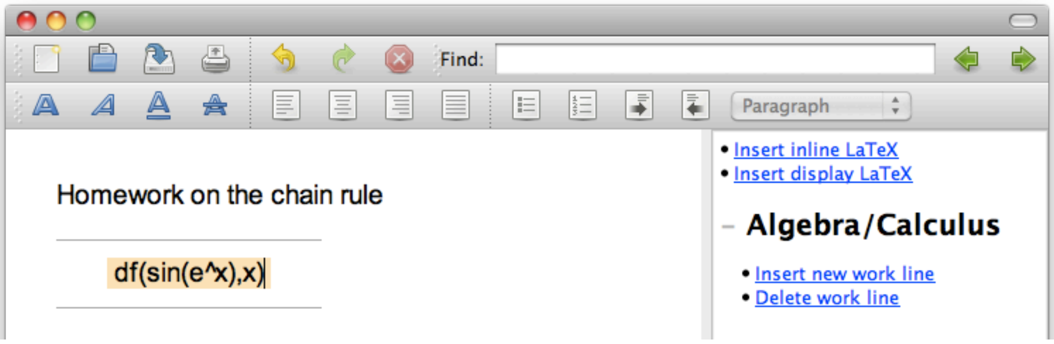
Doing some math

To tell Lurch that you are about to type some meaningful mathematics (not just exposition, but mathematics you want validated), click the “Insert new work line” link on right hand side. (You can also access this action through the tools menu at the top, or with its shortcut key sequence, Ctrl+Shift+N.) Lurch sets off your work with horizontal lines and places your cursor in a highlighted zone into which you can type algebra or calculus expressions using typical calculator notation, such as 2*x^5 and sin(3+y). The only notation that may be unexpected is that for calculus:

df(A,x) stands for the derivative of A with respect to x

int(A,x) stands for the integral of A with respect to x

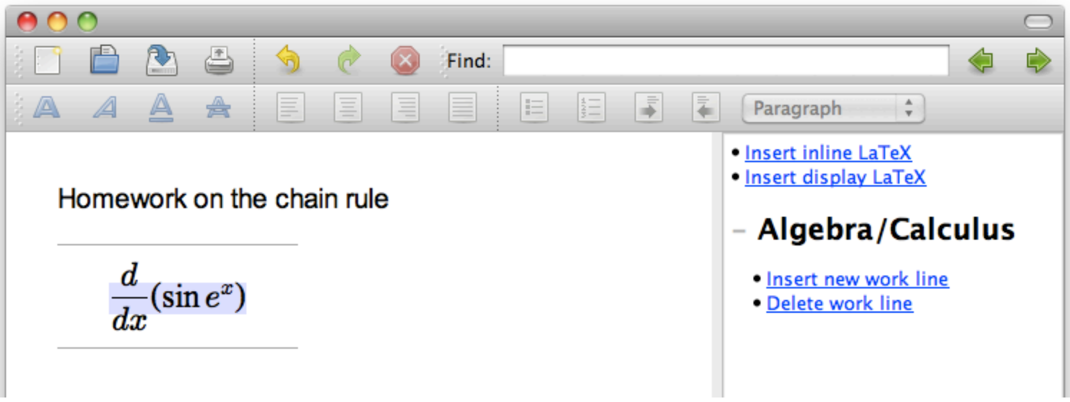
So let’s try to differentiate sin(e^x) and have Lurch validate our work. Enter the expression df(sin(e^x),x) as shown below.



Homework on the chain rule

df(sin(e^x),x)

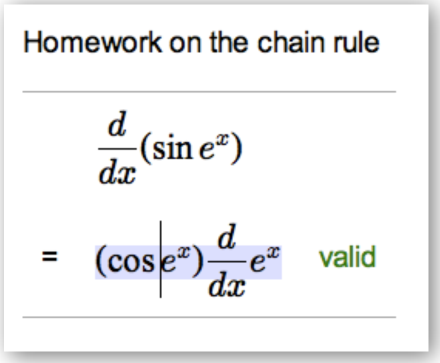
When you press enter, Lurch typesets your mathematics and changes the highlighting to show that you completed your edit. (To change it, you can just hit enter again and it will go back to editing mode; then press enter again when you're done editing.)



Homework on the chain rule

$$\frac{d}{dx}(\sin e^x)$$

To continue our math problem, click to insert a new work line (or press Ctrl+Shift+N). In this one, let's just do part of the problem. On the new line, enter cos(e^x) * df(e^x,x) and see that Lurch validates your work.



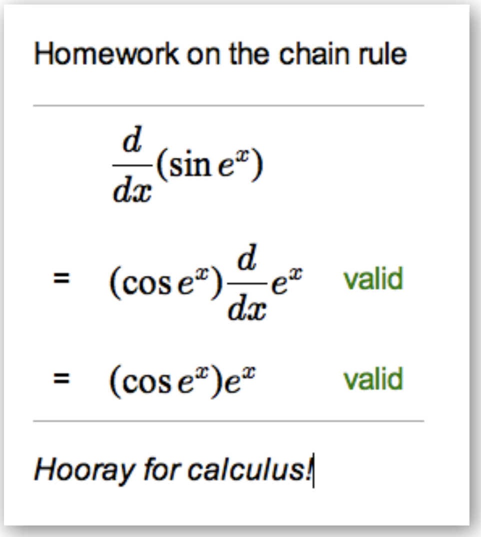
Homework on the chain rule

$$\frac{d}{dx}(\sin e^x)$$

$$= (\cos e^x)\frac{d}{dx}e^x \quad \text{valid}$$

If you were to change that new line (for example, with -cos instead of cos, or with sin instead of cos, or without the chain rule term) Lurch would mark it with "invalid" in red, rather than "valid" in green.

Use the insert-new-work-line tool again to insert a final step in this problem. My final screen looks like the one shown on the bottom right here.



Homework on the chain rule

$$\frac{d}{dx}(\sin e^x)$$

$$= (\cos e^x)\frac{d}{dx}e^x \quad \text{valid}$$

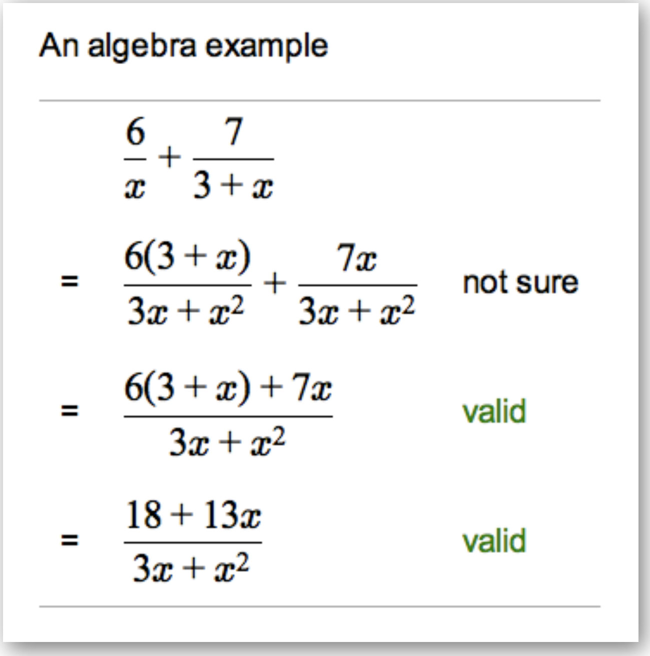
$$= (\cos e^x)e^x \quad \text{valid}$$

Hooray for calculus!

A more complex example

Try an algebraic example rather than a calculus one. I did one in which two fractions needed to be combined by finding a common denominator. Note that in my example, the internal computer algebra system (CAS) wasn't perfectly confident about one of the steps. "Not sure" usually happens when there are potential dividing-by-zero issues that make the simple CAS we imported into our project uncertain whether it's gotten the right answer.

This is perhaps a good time to bring up the bullet points on the overhead that you can be thinking about as you play with the CAS: What are some of its other limitations? What are some of the educational benefits to this software as is?



An algebra example

$$\frac{6}{x}+\frac{7}{3+x}$$

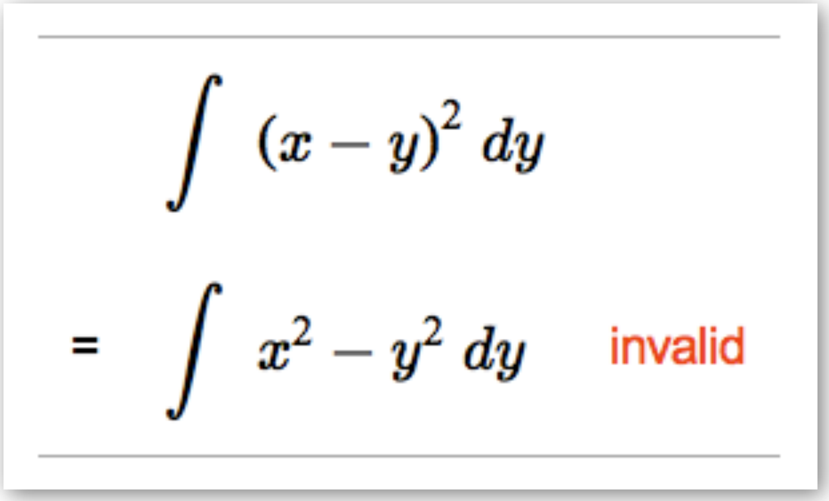
$$= \frac{6(3+x)}{3x+x^2}+\frac{7x}{3x+x^2} \quad \text{not sure}$$

$$= \frac{6(3+x)+7x}{3x+x^2} \quad \text{valid}$$

$$= \frac{18+13x}{3x+x^2} \quad \text{valid}$$

Experiment!

Try tinkering with this math topic as much as you like. (We'll spend about 15 minutes on it in the workshop.) Note that integration is also available, using the notation introduced earlier.



$$\int (x-y)^2\,dy$$

$$= \int x^2 - y^2\,dy \quad \text{invalid}$$

Example 2: Classical Propositional Logic

P.D. Magnus's "forall*x*"

I teach MA305H, "Introduction to Mathematical Logic for honors students." The most recent time I taught it (Fall 2008) I chose a textbook entitled "forall*x*" that is available for free, as a PDF, from P.D. Magnus's website, released under a Creative Commons license. It is available from the link below.

http://www.fecundity.com/logic

Two pages from the back of that textbook are included in this handout immediately following this section; they are reference sheets for the rules of inference in the Fitch-style deductive system the book teaches. Those rules cover just classical propositional and first-order logic; the math topic we will explore in Lurch is just for classical propositional logic.

When creating this math topic in Lurch, I chose to make it look as identical to the textbook as possible, to help students have a seamless experience from textbook to software. We'll talk more about this later.

Opening this topic

Now, from the File menu, click "Choose topic," select "Classical Propositional Logic," and press OK. This topic comes with some introductory text, and yet does not let you place your cursor in that text to alter it. In fact, you'll find this math topic a lot more restrictive than the previous one, which has advantages and disadvantages.

A first proof

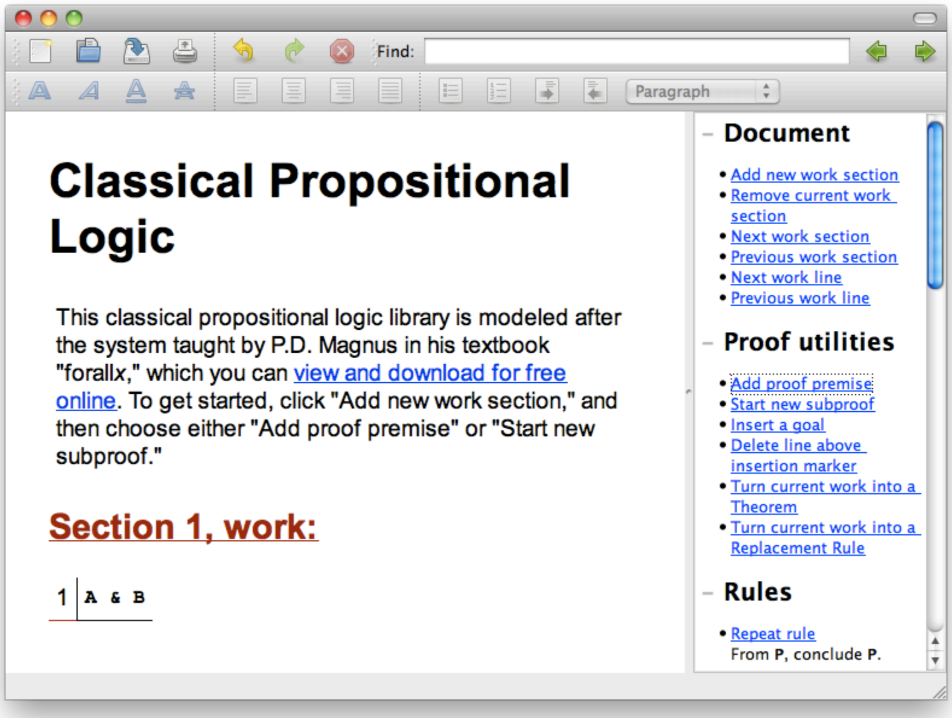
Because most people are not familiar with fitch-style formal proofs in classical propositional logic, I provide a careful walkthrough of how to do such a proof in Lurch. Try the following steps to prove the simple theorem that the "and" operator is commutative.

1. Click the link in the sidebar to Add a new work section (under "Document").

2. Click the link in the sidebar to Add a proof premise (under "Proof utilities").

3. It prompts you to enter a premise, and suggests P. Let's change it to A & B instead, the expression meaning "A and B." Once you have changed the P on the right hand side of the table to A & B, click Apply Rule.

4. Press the down arrow to move the little red cursor below the newly inserted premise. Your document should now look like this one.



Classical Propositional Logic

This classical propositional logic library is modeled after the system taught by P.D. Magnus in his textbook "forallx," which you can view and download for free online. To get started, click "Add new work section," and then choose either "Add proof premise" or "Start new subproof."

Section 1, work:

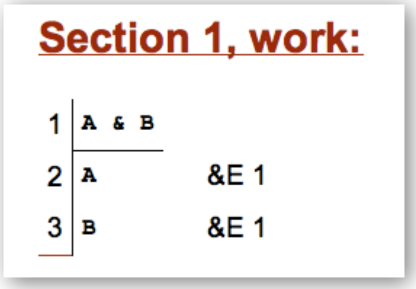
1	A & B

Document
- Add new work section
- Remove current work section
- Next work section
- Previous work section
- Next work line
- Previous work line

Proof utilities
- Add proof premise
- Start new subproof
- Insert a goal
- Delete line above insertion marker
- Turn current work into a Theorem
- Turn current work into a Replacement Rule

Rules
- Repeat rule
From P, conclude P.

5. We have one premise, and wish to use it in our proof. The only way to do so is to disassemble the "and." So click the link for "And elimination rule." Conveniently, the example variables A and B are exactly what we need in our proof. Click Apply Rule.
Next to the two newly inserted lines, you can see that reasons have been automatically inserted. This is part of the implicit validation style of this library. The user directs Lurch what to do (supplying the creativity) but Lurch does the low-level work (handling the details).
You might find it interesting at this point to move your cursor up and down and use Backspace to delete lines in the proof (which you can then put back using Undo). Watch how the latter two lines go from justified to unproven if you delete the line on which they depend.
When you're ready to continue with the proof, put your cursor at the bottom, below line 3.



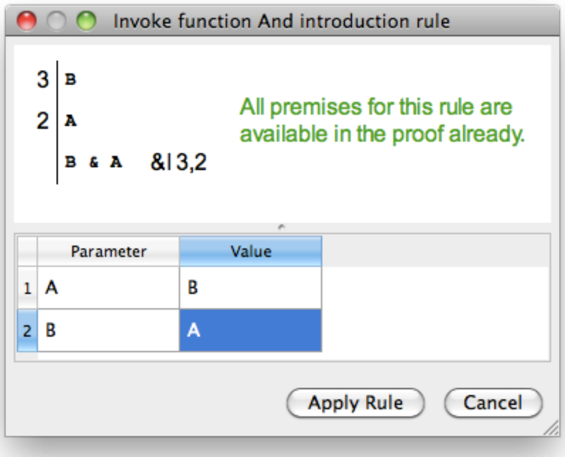
Section 1, work:

1	A & B	
2	A	&E 1
3	B	&E 1

6. We wish now to prove B & A as a conclusion, so click the link for "And introduction rule," to draw a conclusion which has an "and" in it. This time

reverse the roles of A & B, so that the rule will conclude B & A instead, as shown to the right. Click Apply Rule.

7. This is now a proof that from any statement of the form A & B, you can derive B & A. So let's save it as a (simple) theorem for later re-use in other proofs. Click the link for Turn current work into a theorem (under "Proof utilities"). Give it a name like "And is commutative" and press Apply Rule.
Your new theorem appears at the very bottom of the sidebar, under a new section called "Theorems."



```
3 | B
2 | A
  | B & A   &I 3,2
```

All premises for this rule are available in the proof already.

	Parameter	Value
1	A	B
2	B	A

Simple customization

We've just seen the simplest way an instructor can customize Lurch to the needs of their class: By proving theorems, and then sharing those theorems with their students in a Lurch document on which the students can base their own work. I used this extensively in my Fall 2008 logic class, and will demonstrate this in the workshop.

Further exploration

If you have time left after proving your first theorem, try using your theorem to prove that from the premise (A v B) & C you can conclude C & (A v B). (Type lower case vee for the disjunction symbol v). Or try proving one of these theorems.

1. "Modus Tollens," from the two givens A -> B and -B, prove -A.

2. "Or is commutative," from the given A v B, prove B v A.

3. "Law of the excluded middle," from no premises, prove P v -P.

4. If you want an extra challenge, try both versions of De Morgan's laws, that is, both of the following statements (from no premises).

-(A v B) <-> -A & -B   -(A & B) <-> -A v -B

Basic Rules of Proof

Reiteration

$$\begin{array}{r|ll}
m & \mathcal{A} & \\
 & \mathcal{A} & \text{R } m
\end{array}$$

Conjunction Introduction

$$\begin{array}{r|ll}
m & \mathcal{A} & \\
n & \mathcal{B} & \\
 & \mathcal{A} \,\&\, \mathcal{B} & \&\text{I } m, n
\end{array}$$

Conjunction Elimination

$$\begin{array}{r|ll}
m & \mathcal{A} \,\&\, \mathcal{B} & \\
 & \mathcal{A} & \&\text{E } m \\
 & \mathcal{B} & \&\text{E } m
\end{array}$$

Disjunction Introduction

$$\begin{array}{r|ll}
m & \mathcal{A} & \\
 & \mathcal{A} \vee \mathcal{B} & \vee\text{I } m \\
 & \mathcal{B} \vee \mathcal{A} & \vee\text{I } m
\end{array}$$

Disjunction Elimination

$$\begin{array}{r|ll}
m & \mathcal{A} \vee \mathcal{B} & \\
n & \neg\mathcal{B} & \\
 & \mathcal{A} & \vee\text{E } m, n
\end{array}$$

$$\begin{array}{r|ll}
m & \mathcal{A} \vee \mathcal{B} & \\
n & \neg\mathcal{A} & \\
 & \mathcal{B} & \vee\text{E } m, n
\end{array}$$

Conditional Introduction

$$\begin{array}{r|ll}
m & \quad \vert\ \underline{\mathcal{A}} & \text{want } \mathcal{B} \\
n & \quad \vert\ \mathcal{B} & \\
 & \mathcal{A} \to \mathcal{B} & \to\text{I } m\text{–}n
\end{array}$$

Conditional Elimination

$$\begin{array}{r|ll}
m & \mathcal{A} \to \mathcal{B} & \\
n & \mathcal{A} & \\
 & \mathcal{B} & \to\text{E } m, n
\end{array}$$

Biconditional Introduction

$$\begin{array}{r|ll}
m & \quad \vert\ \underline{\mathcal{A}} & \text{want } \mathcal{B} \\
n & \quad \vert\ \mathcal{B} & \\
p & \quad \vert\ \underline{\mathcal{B}} & \text{want } \mathcal{A} \\
q & \quad \vert\ \mathcal{A} & \\
 & \mathcal{A} \leftrightarrow \mathcal{B} & \leftrightarrow\text{I } m\text{–}n,\ p\text{–}q
\end{array}$$

Biconditional Elimination

$$\begin{array}{r|ll}
m & \mathcal{A} \leftrightarrow \mathcal{B} & \\
n & \mathcal{B} & \\
 & \mathcal{A} & \leftrightarrow\text{E } m, n
\end{array}$$

$$\begin{array}{r|ll}
m & \mathcal{A} \leftrightarrow \mathcal{B} & \\
n & \mathcal{A} & \\
 & \mathcal{B} & \leftrightarrow\text{E } m, n
\end{array}$$

Negation Introduction

$$\begin{array}{r|ll}
m & \quad \vert\ \underline{\mathcal{A}} & \text{for reductio} \\
n-1 & \quad \vert\ \mathcal{B} & \\
n & \quad \vert\ \neg\mathcal{B} & \\
 & \neg\mathcal{A} & \neg\text{I } m\text{–}n
\end{array}$$

Negation Elimination

$$\begin{array}{r|ll}
m & \quad \vert\ \underline{\neg\mathcal{A}} & \text{for reductio} \\
n-1 & \quad \vert\ \mathcal{B} & \\
n & \quad \vert\ \neg\mathcal{B} & \\
 & \mathcal{A} & \neg\text{E } m\text{–}n
\end{array}$$

Quantifier Rules

Existential Introduction

m	$\mathcal{A}$	
	$\exists \chi \mathcal{A}[\chi \| \| c]$	$\exists$I m

χ may replace some or all occurrences of c in $\mathcal{A}$.

Existential Elimination

m	$\exists \chi \mathcal{A}$	
n	$\quad \mid \underline{\mathcal{A}[c \| \chi]}$	
p	$\quad \mid \mathcal{B}$	
	$\mathcal{B}$	$\exists$E m, n–p

The constant c must not appear in $\exists \chi \mathcal{A}$, in $\mathcal{B}$, or in any undischarged assumption.

Universal Introduction

m	$\mathcal{A}$	
	$\forall \chi \mathcal{A}[\chi \| c]$	$\forall$I m

c must not occur in any undischarged assumptions.

Universal Elimination

m	$\forall \chi \mathcal{A}$	
	$\mathcal{A}[c \| \chi]$	$\forall$E m

Identity Rules

	$c = c$	$=$I

m	$c = d$	
n	$\mathcal{A}$	
	$\mathcal{A}[c \| \| d]$	$=$E m, n
	$\mathcal{A}[d \| \| c]$	$=$E m, n

One constant may replace some or all occurrences of the other.

Derived Rules

Dilemma

m	$\mathcal{A} \vee \mathcal{B}$	
n	$\mathcal{A} \to \mathcal{C}$	
p	$\mathcal{B} \to \mathcal{C}$	
	$\mathcal{C}$	$\vee *$ m, n, p

Modus Tollens

m	$\mathcal{A} \to \mathcal{B}$	
n	$\neg \mathcal{B}$	
	$\neg \mathcal{A}$	MT m, n

Hypothetical Syllogism

m	$\mathcal{A} \to \mathcal{B}$	
n	$\mathcal{B} \to \mathcal{C}$	
	$\mathcal{A} \to \mathcal{C}$	HS m, n

Replacement Rules

Commutivity (Comm)

$$(\mathcal{A} \,\&\, \mathcal{B}) \iff (\mathcal{B} \,\&\, \mathcal{A})$$
$$(\mathcal{A} \vee \mathcal{B}) \iff (\mathcal{B} \vee \mathcal{A})$$
$$(\mathcal{A} \leftrightarrow \mathcal{B}) \iff (\mathcal{B} \leftrightarrow \mathcal{A})$$

DeMorgan (DeM)

$$\neg(\mathcal{A} \vee \mathcal{B}) \iff (\neg \mathcal{A} \,\&\, \neg \mathcal{B})$$
$$\neg(\mathcal{A} \,\&\, \mathcal{B}) \iff (\neg \mathcal{A} \vee \neg \mathcal{B})$$

Double Negation (DN)

$$\neg\neg \mathcal{A} \iff \mathcal{A}$$

Material Conditional (MC)

$$(\mathcal{A} \to \mathcal{B}) \iff (\neg \mathcal{A} \vee \mathcal{B})$$
$$(\mathcal{A} \vee \mathcal{B}) \iff (\neg \mathcal{A} \to \mathcal{B})$$

Biconditional Exchange ($\leftrightarrow$ex)

$$[(\mathcal{A} \to \mathcal{B}) \,\&\, (\mathcal{B} \to \mathcal{A})] \iff (\mathcal{A} \leftrightarrow \mathcal{B})$$

Quantifier Negation (QN)

$$\neg \forall \chi \mathcal{A} \iff \exists \chi \neg \mathcal{A}$$
$$\neg \exists \chi \mathcal{A} \iff \forall \chi \neg \mathcal{A}$$

Example 3: Line-numbered Proofs

High-school geometry

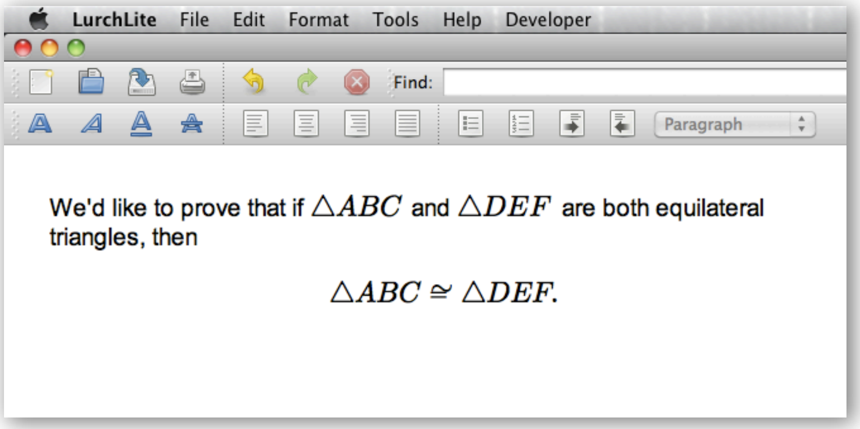
In high school geometry courses, it is common to teach and write two-column proofs, statements on the left and reasons on the right. Sometimes we number the lines of the proof, so that we can refer back to statements in line 2 from the reasons column in line 7, for example. It's a simple setup.

However, when you realize you need an additional step at line 3, then all reasons that referred to lines 3 through the end of the proof need to be hunted down and renumbered to be one higher. The same goes if you need to switch lines 10 and 11, or delete line 5, etc. This is a perfect task for a computer, and a very simple kind of proof-writing help Lurch can give. It's so simple, in fact, that it doesn't even fit my original definition of "validation," unless you really stretch it.

Math word processing

So far in Lurch, we've only typed calculator notation, logic notation, and a little text. We haven't really exploited Lurch as a math word processor. Let's begin there before we get into two-column proofs. From the File menu, click Choose topic, and choose Line-numbered proofs. Then type "We'd like to prove that if" and stop there.

The next thing we'd like to type is a mathematical expression, so insert an inline latex block using the link in the sidebar. In the highlighted zone, type \triangle ABC and press enter. You should see nicely typeset mathematics. Move your cursor out of it to the right, and continue the sentence shown below. Use the TeX code \cong to get the "is congruent to" symbol. I made the final math block display math just to have a chance to use that link in the sidebar.



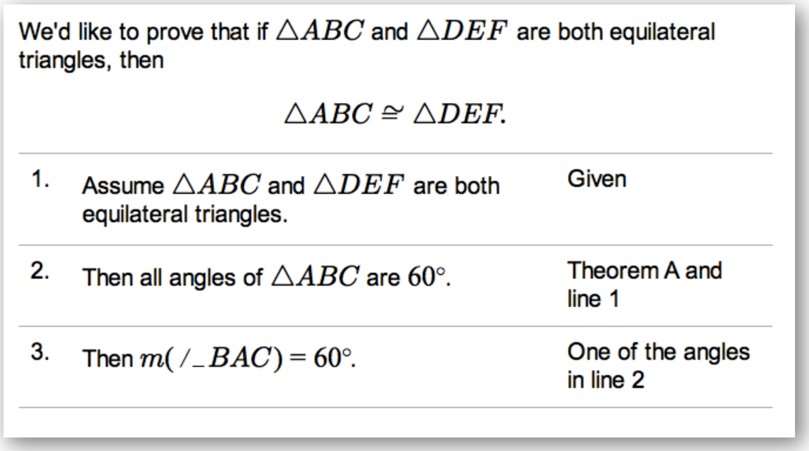
We'd like to prove that if $\triangle ABC$ and $\triangle DEF$ are both equilateral triangles, then

$$\triangle ABC \cong \triangle DEF.$$

Just as you could go back to algebra/calculus expressions and edit them, you can also put your cursor back in a LaTeX expression you've created, press enter, and change the TeX source in order to change it. This word processor does have several limitations, but its typesetting of mathematics has reached a reliable level, and the results are high-quality. (The underlying TeX engine is jsMath.)

Starting a proof

Click the sidebar link under "Line numbered proofs" to add a new line, and it gives you a filler statement and filler reason that you can change. Let's make them our assumption, and call it a given. I'm not going to complete this proof, but just do enough to show you how the line-numbered proofs library works. Try inserting two additional lines, each of which references other lines in its reason column, like this.

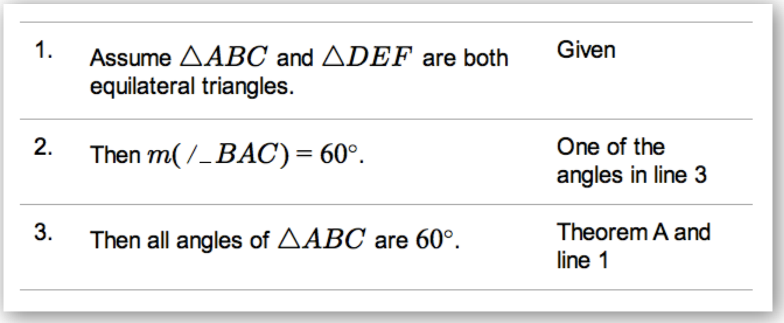


We'd like to prove that if $\triangle ABC$ and $\triangle DEF$ are both equilateral triangles, then

$$\triangle ABC \cong \triangle DEF.$$

1.	Assume $\triangle ABC$ and $\triangle DEF$ are both equilateral triangles.	Given
2.	Then all angles of $\triangle ABC$ are $60°$.	Theorem A and line 1
3.	Then $m(\angle BAC) = 60°$.	One of the angles in line 2

If we then place our cursor in any line in the proof and use the command to move that line up or down (Ctrl+Up, Ctrl +Down), the lines renumber automatically. Here is the result of swapping lines 2 and 3:



1.	Assume $\triangle ABC$ and $\triangle DEF$ are both equilateral triangles.	Given
2.	Then $m(\angle BAC) = 60°$.	One of the angles in line 3
3.	Then all angles of $\triangle ABC$ are $60°$.	Theorem A and line 1

Now, swapping those two lines doesn't make a lot of sense, but Lurch is providing only a very tiny amount of validation here; it cannot (yet?) check a geometry proof written in English and TeX! Try inserting new lines in the middle of the proof, moving lines around, and deleting old lines. In every case, Lurch should keep the line numbers in sync.

Although this is a very limited, small amount of validation, it is also very flexible because of how little it gets in your way.

Student Projects Available

We list all these tasks as potential student projects because we enjoy working with students, know it has value to both the students and the project, and because we've had success working with students on this project and others in the past. However, collaboration with other instructors/faculty is quite welcome, also!

For mathematics/computer science students with good writing ability

Much of the work of a software project is making it easy for people to use it, which means good tutorials, documentation, and web pages. Students who understand the software and/or the mathematics in it and are able to explain that understanding clearly to others could work on projects like these.

- Write a user manual for the Lurch Lite application itself
- Write web tutorials for each of the main topics in Lurch, like much expanded versions of the sections in this handout
- Improve and expand the existing tutorials for how to become a Lurch script author

For students with knowledge of C++

The Lurch User Interface is built in C++ using the Qt GUI toolkit. There are many opportunities to add new features and fix old bugs, including these examples.

- Fixing bugs in the underlying OpenMath library on which Lurch is built
- Adding to the File menu commands for Import, Recent documents, and Recent topics
- Making program startup time more efficient
- Implement an auto-save feature

For students with knowledge of JavaScript

Lurch math topics and the foundational tools on which they're built are written in JavaScript. There are opportunities to create new mathematics topics and/or general tools that will be used in creating new math topics, including these examples.

- Extending the algebra/calculus math topic to handle using and solving equations, including integration by substitution
- Taking old math topics and updating them to work with the new Lurch user interface (for example, there is a predicate logic topic that needs updating)
- Create a topic that makes it easy for non-programmers to define a new language/parser
- Create a topic in which users can play Ehrenfeucht-Fräissé (logic) games against one another

For students with knowledge of both C++ and JavaScript

There is hybrid territory in Lurch, where the JavaScript interpreter communicates with the underlying software. We occasionally add new features there, and students able to program in both worlds could work on implementing projects like these examples.

- Create a package that makes it easy for Lurch to communicate with external processes (e.g., Mathematica, Sage) using stdin/stdout
- Implement Copy+Paste functionality in the word processor
- Fixing bugs in the Lurch Lite word processing engine

For students with knowledge of Unix

Tasks for the Unix-savvy include these examples.

- Create a cron job on the Lurch server to regularly run our unit test suite and email nicely-formatted results to the developers' email list.
- Create a script that creates .deb files for installation of Lurch on Debian-based Unix systems.
- Create a script that creates .rpm files for installation of Lurch on Redhat-based Unix systems.
- Improve usability of our existing build-and-install shell script.

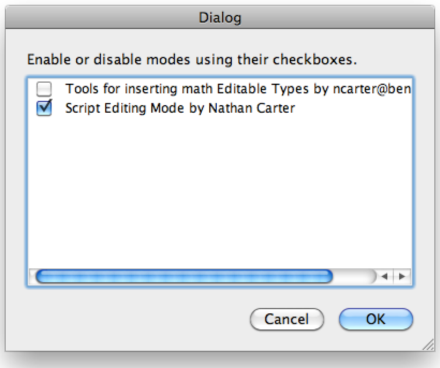
Intro to *Lurch* Programming

Optional

If there is time left after discussion at the end of the workshop, those interested in seeing how Lurch programming is done can stick around for a demonstration/exploration of some simple coding within Lurch. This section of the handout covers that material. More can be found on our website, under "Developer Resources."

The Scripting Mode

Start any new document (preferably in a word processing topic, for simplicity) and save it to disk under any filename. Then from the File menu, click "Choose modes." Turn on the Scripting mode and click OK. New actions have appeared in your sidebar (and Tools menu), related to writing scripts.



Enable or disable modes using their checkboxes.

- [ ] Tools for inserting math Editable Types by ncarter@ben
- [x] Script Editing Mode by Nathan Carter

Every time you start Lurch after this, it will default to using this mode, which you can turn off again by clicking File, Choose modes, and unchecking it.

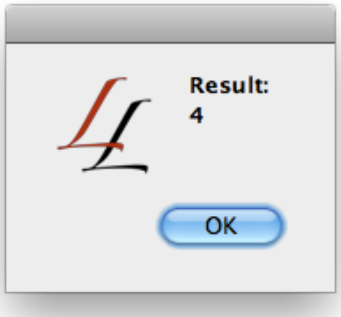
Writing a script

Let's create a piece of JavaScript code in our document. From the sidebar, under Scripting, click the link to Insert new script after cursor. A script appears, containing just a one-line comment inviting you to edit it. Place your cursor inside the script box and press enter. A script editing window opens, and your cursor goes inside.

Modify the comment line to say something different, and click Save (or press Ctrl+S, or Command+S on a Mac). Notice that your updates are only saved to the document when you click the Save button on the script window.
Important: Saving script code into the document does **not** save the document to disk! You need to click Save from the document window for that to happen.

Running a script

Let's put in the script some content worth evaluating. Replace the comment with the text 2+2 all by itself. Click the Run button at the top of the window, in the toolbar (it looks like a green "play" triangle).



Result:
4

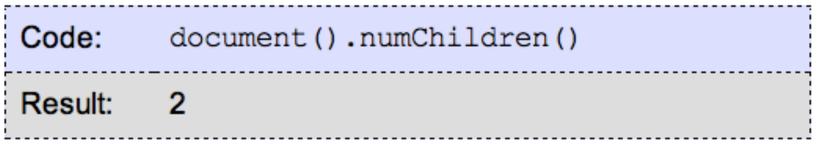
Lurch evaluated your script and told you the answer. You can do arbitrarily complicated scripts, and only the final value is reported when you run them. Try this one.

```
var a = 'hello';
var b = 'world';
a + ' to the ' + b
```

Note that scripts in the document do not run on their own by default; they only run if you open them and click Run. We'll see in a moment how you can change this behavior.

Script types

In the document (not the script editor), right-click your script and choose "Edit script attributes." Note the list of different types you can give your script. Type **example** in the "type" box and click Apply. This shows you that "example" scripts evaluate in the document, useful for making in-document examples of what a script can do.



```
Code:    document().numChildren()
Result:  2
```

Now try changing your script as follows: Make its code the following:

```
code.document().numChildren()
```

If you left the script with type example, you should see that this code works, even though you might not yet know how the document or numChildren functions work in general.

Here's a quick overview of the other script types in Lurch.

auto-run	automatically run every time the document is loaded (and at select other times that make sense; nevermind that now)
user-action	should show up on the Tools menu and sidebar, like the "Insert new script after cursor" action you used earlier
example	as you've seen, it just shows in the document the result of its evaluation
test	each expression in the script is treated as a test that will return a true value for "pass" or a false value for "fail"; more on this below

Test scripts

Let's do a slightly more realistic example of how you might use auto-run and test scripts in a Lurch document. Let's say you were creating a set of tools for use in algebra, and one of the first ones you had to create was a function for computing slope between two points, (y2-y1)/(x2-x1). Change your script as follows.

Make its type auto-run.

Make its code the following.

```
function slopeBetweenPoints ( x1, y1, x2, y2 )
{
        return ( y2 - y1 ) / ( x2 - x1 );
}
```

Place your cursor after that script block and again use the command Inset new script after cursor.

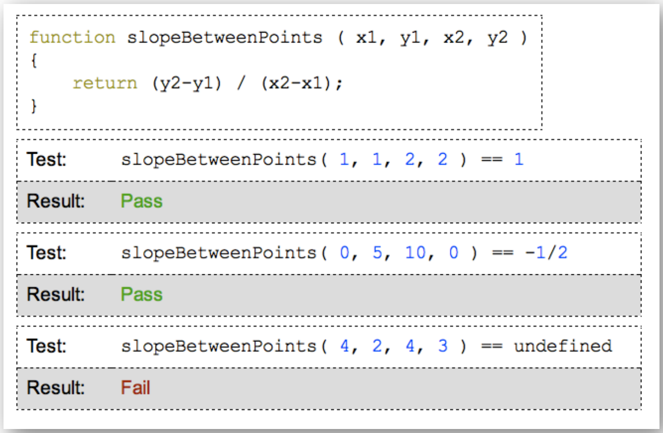
Make the new script have type test.

Make its code the following.

```
slopeBetweenPoints( 1, 1, 2, 2 ) == 1
slopeBetweenPoints( 0, 5, 10, 0 ) == -1/2
slopeBetweenPoints( 4, 2, 4, 3 ) == undefined
```

When you save, you should find that the first two tests pass and the third fails. Ah, the value of testing! Changing the original function as follows (and saving it) should make the test pass.

```
function slopeBetweenPoints ( x1, y1, x2, y2 )
{
        if ( x1 == x2 ) return undefined;
        return ( y2 - y1 ) / ( x2 - x1 );
}
```

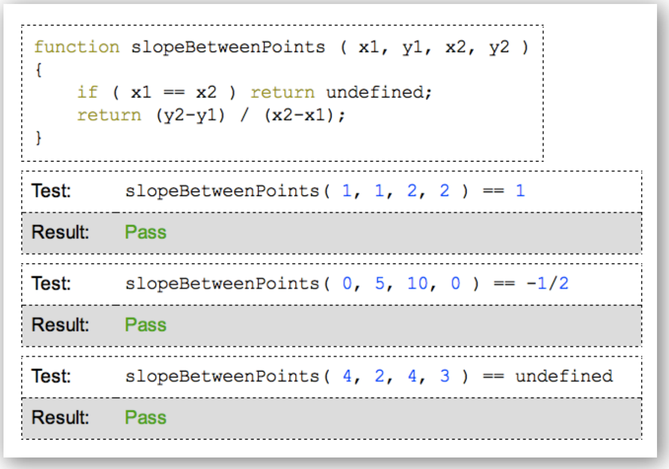


```
function slopeBetweenPoints ( x1, y1, x2, y2 )
{
    return (y2-y1) / (x2-x1);
}
```

Test: `slopeBetweenPoints( 1, 1, 2, 2 ) == 1`
Result: Pass

Test: `slopeBetweenPoints( 0, 5, 10, 0 ) == -1/2`
Result: Pass

Test: `slopeBetweenPoints( 4, 2, 4, 3 ) == undefined`
Result: Fail



```
function slopeBetweenPoints ( x1, y1, x2, y2 )
{
    if ( x1 == x2 ) return undefined;
    return (y2-y1) / (x2-x1);
}
```

Test: `slopeBetweenPoints( 1, 1, 2, 2 ) == 1`
Result: Pass

Test: `slopeBetweenPoints( 0, 5, 10, 0 ) == -1/2`
Result: Pass

Test: `slopeBetweenPoints( 4, 2, 4, 3 ) == undefined`
Result: Pass