Bentley University

Lurch Workshop

Lurch: Software for teaching mathematical proof
Presented at the NES MAA Conference at Bentley University

November 21, 2008
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/nesmaa08**

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

Bentley University

Installing Lurch Alpha

What is Lurch Alpha?

Lurch Alpha is the alpha release of Lurch, representing the work done from June to October of 2008. The final version of the Lurch project will be released in Fall 2010. The main differences are these.

- Lurch Alpha has a very rudimentary user interface; Lurch will have a word-processor like UI.
- Lurch will be able to handle many branches of mathematics, while Lurch Alpha just does a few simple things.

Downloading

Point your browser to the following URL and click the download link. Run the downloaded application, saying yes to all its suggestions, until installation completes. Lurch should open with a blank document in it.

http://lurch.sourceforge.net/nesmaa08

Using Lurch

Although Lurch launches when the download process completes, the Windows Start menu contains a Lurch Alpha folder with a list of shortcuts that makes it easier to launch Lurch in various modes, including the following three (among others). You will be encouraged to use each of these three in the scheduled time of "structured play." (See next page.)

- Lurch with the Circle-Dot System
- Lurch with Ken's Propositional ND
- Lurch with Fitch-style Number Theory

Credits

Some portions of this document were taken from the Lurch Jr. manuals written by Kenneth Monks of the University of Scranton for use in his Axiomatic Geometry class during Fall 2008.

Bentley University

Play: the Circle-Dot system

I don't know what I mean, but I know I'm right!

Welcome to Circle-Dot! The object of the game is to construct a circle-dot word made of entirely o's and .'s. Thus the words in this toy logical system do not mean anything, but the system teaches us the fundamentals of formal proofs, including axioms, rules, lines, reasons, etc.

You do not have any words to begin with. On each of your turns you can apply one of the five available rules to your current list of constructed circle-dot words to produce a new word. It may require some study and investigation on your part to come up with an efficient strategy for beating this game at the higher levels of difficulty.

To Begin

Launch "Lurch with the Circle-Dot System" from the Start Menu. Notice the two types of actions available to you on the right of the window, as links. Click "Add a new work section" to get started.

In the Circle-Dot System, we do work that is very much like a formal proof, in that we have axioms to start with, rules to make progress with, and a goal for which we aim. They are summarized here; note that o and . are valid symbols in the system, and w and v are variables that stand for any sequence of o's and .'s.

Axiom A	Axiom B	Rule 1	Rule 2	Rule 3
o.	.o	Given wv and vw Conclude w	Given w and v Conclude w.v	Given wv. Conclude wo

A First Proof

To prove the "Theorem" that is just the string "o" try the following sequence of actions:

1. Click Axiom A to insert o. as line 1.
2. Click Axiom B to insert .o as line 2.
3. Click Rule 1 to open up a window that allows you to choose the values of w and v. Edit the w to stand for o and the v to stand for . (dot, period). Click "Apply Rule." (Continued on next page.)

4. Click the “Turn current work into a Theorem” link to change your “work” section into a completed “Theorem” section.

Structured Play

Try proving the following “Theorems” in the Circle-Dot System on your own. For each one, start by adding a new work section to your existing document. For some of them, you need to use Rule 2 and/or Rule 3.

1. . (just a single dot)
2. ...
3. ..o
4. .oo
5. o..o
6. oooo
7. .o.
8. .ooo

More Circle-Dot

For a complete introduction to the Circle-Dot System, together with an interesting theoretical question about provability in the system, see the following URL on Ken Monks’s website.

http://math.scranton.edu/monks/courses/math448/ToyProofsV2.html

Bentley University

Play: Ken's Propositional ND

Natural Deduction and Lurch

Gerhard Gentzen in 1935 proposed a formal system for propositional (and predicate) logic, Natural Deduction. On the following page you will find a handout that my collaborator Ken Monks gives to his upper-division courses when introducing formal logic. It contains his version of Gentzen's rules. Note that the symbols have the following meanings.

A, B, C, D, ... Propositions (statements)

~ Negation (not)

and, or Their usual meanings

=> Implication (conditional)

<=> If and only if (biconditional)

In order to support Ken's use of this system to introduce the idea of a proof, we wanted Lurch to be able to operate with the same system, written exactly the same way, following exactly the same rules. We did so by adding to our library of Lurch documents one containing the appropriate Javascript code to describe the system. In later versions of Lurch, we aim for such extensions to new logical systems (and rules, theorems, etc.) to not require writing any code; instead, the user should be able to describe directly in a Lurch document exactly what their logical system is like.

A first proof

The following sequence of actions will help you prove a simple theorem in propositional logic: "and" is commutative. After you have followed these steps to get your feet wet, try proving some of the theorems below as exercises. (Or just experiment on your own! This is play time, after all.)

1. From the Start Menu, from the Lurch folder, launch Lurch with Ken's Propositional ND.
2. Click the link to Add a new work section.
3. Click the link to Add a proof premise.
4. In place of P, type P and Q. Click Apply Rule.
5. Press the down arrow to move your cursor to be after the new premise.

Table 1: **Rules of inference for Propositional Logic**

and + To show W and V 1. Show W 2. Show V	and – To show W 1. Show W and V and – To show V 1. Show W and V
$\Rightarrow$ + To show $W \Rightarrow V$ 1. Assume W 2. Show V 3. $\leftarrow$	$\Rightarrow$ – (modus ponens) To show V 1. Show W 2. Show $W \Rightarrow V$
$\Leftrightarrow$ + To show $W \Leftrightarrow V$ 1. Show $W \Rightarrow V$ 2. Show $V \Rightarrow W$	$\Leftrightarrow$ – To show $W \Rightarrow V$ 1. Show $W \Leftrightarrow V$ $\Leftrightarrow$ – To show $V \Rightarrow W$ 1. Show $W \Leftrightarrow V$
or + To show W or V 1. Show W *or* + To show W or V 1. Show V	*or* – (proof by cases) To show U 1. Show W or V 2. Show $W \Rightarrow U$ 3. Show $V \Rightarrow U$
$\sim$ + (proof by contradiction) To show $\sim W$ 1. Assume W 2. Show $\rightarrow\leftarrow$ 3. $\leftarrow$	$\sim$ – (proof by contradiction) To show W 1. Assume $\sim W$ 2. Show $\rightarrow\leftarrow$ 3. $\leftarrow$
$\rightarrow\leftarrow$ + To show $\rightarrow\leftarrow$ 1. Show W 2. Show $\sim W$	

Remark *Note that the inputs "Assume -" and "$\leftarrow$" are not themselves statements but rather inputs to rules of inference that may be inserted into a proof at any time. There is no reason however, to insert such statements unless you intend to use one of the rules of inference that*

6. Click the link for And elimination rule #1, and in place of W type P and press enter, then in place of V type Q and press enter. Click Apply Rule.

7. Repeat the previous step for And elimination rule #2, and you should then have two new lines in your proof, one with P and one with Q.

8. Click the link for And introduction rule, and enter the values of W and V now in reverse order (Q first, P second). Click Apply Rule.

9. This is now a proof that from any statement P and Q, you can derive Q and P. So let's save it for re-use. Click the link for Turn current work into a theorem. Give it a name like "And is commutative" and press Apply Rule.

10. You can try using your new theorem by starting a new work section and clicking the link for your new theorem (way down the bottom of the right hand pane, after scrolling).

Exercises

Try proving each of these theorems in a new work section.

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

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

3. "Law of the excluded middle," from no premises, prove P or ~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 or B) <=> ~A and ~B ~(A and B) <=> ~A or ~B

Bentley University

Play: Fitch-style Number Theory

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

I taught MA305H this fall, "Introduction to Mathematical Logic for honors students." 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 first-order predicate logic, and do not include any "mathematics" (other than logic). In order to be able to reason about what we would call mathematics (numbers, sets, functions, spaces, etc.), the axioms and notation for at least one branch of mathematics need to be added to the system.

In MA305H, after the students learned predicate logic, I introduced them to axiomatic number theory as one of the applications of logic. After loading Lurch with the number theory content included, you will see the appropriate axioms and rules in the right-hand pane, as you have with the previous two examples.

An example Number Theory proof

The following steps walk you through the creation of a proof that every natural number besides zero is the successor of some other natural number. This proof works backwards from the desired goal, showing an alternative (and intuitive) way to use Lurch for constructing proofs.

1. Use the Start Menu shortcut to open Lurch with Fitch-style Number Theory.

2. Use the link to Add a new work section, and then click the link to Insert a goal (under "Proof utilities"). We need to express the statement "For all x, if x is not zero then there exists y such that x=Sy," where S is the successor function S(n)=n+1. The limitations of the Lurch Alpha user interface require that we type the "for all" symbol (usually written as ∀) as @ and the "there exists" symbol (usually written as ∃) as #. In fact, here is a complete table of all the symbols you may need to type in during work in Number Theory, and their Lurch Alpha shortcuts. Keep in mind that future versions of Lurch will have a much more sophisticated user interface that does not require this notation.

Symbol	Meaning	Lurch Alpha notation	Symbol	Meaning	Lurch Alpha notation
∀	Universal quantifier ("for all")	@	∃	Existential quantifier ("there exists")	#
→	Implication ("if...then")	-> (hyphen, greater than)	¬	Negation ("not")	- (hyphen)

So we type in our goal statement as follows: @x (-(x=0) -> #y x=Sy)
(How many spaces you include is irrelevant, and including extra parentheses is optional.)
After typing this expression in and pressing enter, then Apply Rule, it is inserted with reason "Unproven," so that you know you have a line in your proof that still requires justification.

3. Press the up arrow to move your cursor above the unproven line, so that we can insert work that justifies it. We want to prove a universal statement about all natural numbers, so we will do so by induction. Click the link under Axioms of Number Theory called "NT 3: Mathematical Induction."
   The variable x is correct and does not need to be changed; we want to introduce a "for all x" statement.
   The statement P is just an example; replace it with the statement about x that we want to create: -(x=0) -> #y x=Sy.
   Press Apply Rule. Now your original goal is proven, but we have two new unproven statements, two new goals.

4. Place your cursor above the second new goal by pressing the up arrow once. We wish to prove this universally quantified statement, so click the link for Universal introduction.
   My class's textbook encourages students to prove a statement about an arbitrary constant (like the c shown on your screen) and then generalize to a variable like x. So we will follow this pattern.
   For the statement A, type in the desired conclusion, with c in place of x:
   (-(c=0) -> #y c=Sy) -> (-(Sc=0) -> #y Sc=Sy)
   Click Apply Rule and see a new goal created.

5. Place your cursor above the conditional statement in line 2, which we now need to prove. Click the link for Conditional introduction, so we can prove a conditional statement.
   For the premise A, enter the premise of our conditional, -(c=0) -> #y c=Sy.
   For the conclusion B, enter the conclusion of our conditional, -(Sc=0) -> #y Sc=Sy.
   Click Apply Rule and see a new subproof created, with the premise at the top of the subproof, and the conclusion as the unproven goal. It is indented to indicate that the assumption is a temporary one that does not apply to the whole proof. (This is a part of any Fitch-style logical system.)

6. Place your cursor above line 3, our new unproven goal. Again, we wish to prove a conditional, so use Conditional introduction with premise -Sc=0 and conclusion #y Sc=Sy.

7. Place your cursor above the unproven line #y Sc=Sy. It is clear that this is true because c=y, so let us tell Lurch that fact. Click the Existential introduction link. We find that we must provide a statement A that we can prove, specify

which variable will follow the existential, and what statement C we want inside the existential.
For A, type in the statement Sc=Sc, which should be easy to prove.
For x, type in y, because our existential quantifier is about y.
For C, type in Sc=Sy, the statement we want to have inside our existential conclusion.
Click Apply Rule, and you'll see the only unproven statement left in our subproof is an easy one, Sc=Sc.

8. Place your cursor above Sc=Sc and Click Equality introduction. Enter Sc as the term t, and click Apply Rule.

9. Proving the remaining unproven statement in the proof (the base case of our induction) is left as an exercise to the reader. Here are some hints:
   Do indeed begin with a conditional introduction, as you might expect.
   Assume the opposite of what you're trying to prove, and prove it by contradiction. (See the reference sheets for how to accomplish this using the "negation elimination" rule.)

If that wasn't enough...

If you can't get enough axiomatic number theory, try proving these theorems. See Principia Mathematica for more.

1. ∀x (0+x=x)

2. ∀x ∀y (x+Sy = Sx+y)

Basic Rules of Proof

Reiteration

m	$\mathcal{A}$	
	$\mathcal{A}$	R m

Conjunction Introduction

m	$\mathcal{A}$	
n	$\mathcal{B}$	
	$\mathcal{A} \,\&\, \mathcal{B}$	$\&$I m, n

Conjunction Elimination

m	$\mathcal{A} \,\&\, \mathcal{B}$	
	$\mathcal{A}$	$\&$E m
	$\mathcal{B}$	$\&$E m

Disjunction Introduction

m	$\mathcal{A}$	
	$\mathcal{A} \vee \mathcal{B}$	$\vee$I m
	$\mathcal{B} \vee \mathcal{A}$	$\vee$I m

Disjunction Elimination

m	$\mathcal{A} \vee \mathcal{B}$	
n	$\neg\mathcal{B}$	
	$\mathcal{A}$	$\vee$E m, n

m	$\mathcal{A} \vee \mathcal{B}$	
n	$\neg\mathcal{A}$	
	$\mathcal{B}$	$\vee$E m, n

Conditional Introduction

m	$\mathcal{A}$	want $\mathcal{B}$
n	$\mathcal{B}$	
	$\mathcal{A} \to \mathcal{B}$	$\to$I m–n

Conditional Elimination

m	$\mathcal{A} \to \mathcal{B}$	
n	$\mathcal{A}$	
	$\mathcal{B}$	$\to$E m, n

Biconditional Introduction

m	$\mathcal{A}$	want $\mathcal{B}$
n	$\mathcal{B}$	
p	$\mathcal{B}$	want $\mathcal{A}$
q	$\mathcal{A}$	
	$\mathcal{A} \leftrightarrow \mathcal{B}$	$\leftrightarrow$I m–n, p–q

Biconditional Elimination

m	$\mathcal{A} \leftrightarrow \mathcal{B}$	
n	$\mathcal{B}$	
	$\mathcal{A}$	$\leftrightarrow$E m, n

m	$\mathcal{A} \leftrightarrow \mathcal{B}$	
n	$\mathcal{A}$	
	$\mathcal{B}$	$\leftrightarrow$E m, n

Negation Introduction

m	$\mathcal{A}$	for reductio
$n-1$	$\mathcal{B}$	
n	$\neg\mathcal{B}$	
	$\neg\mathcal{A}$	$\neg$I m–n

Negation Elimination

m	$\neg\mathcal{A}$	for reductio
$n-1$	$\mathcal{B}$	
n	$\neg\mathcal{B}$	
	$\mathcal{A}$	$\neg$E m–n

Quantifier Rules

Existential Introduction

$$\begin{array}{l|ll} m & \mathcal{A} & \\ & \exists\mathcal{X}\mathcal{A}[\mathcal{X}||\mathcal{c}] & \exists\text{I } m \end{array}$$

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

Existential Elimination

$$\begin{array}{l|ll} m & \exists\mathcal{X}\mathcal{A} & \\ n & \quad \mathcal{A}[\mathcal{c}|\mathcal{X}] & \\ p & \quad \mathcal{B} & \\ & \mathcal{B} & \exists\text{E } m,\ n\text{–}p \end{array}$$

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

Universal Introduction

$$\begin{array}{l|ll} m & \mathcal{A} & \\ & \forall\mathcal{X}\mathcal{A}[\mathcal{X}|\mathcal{c}] & \forall\text{I } m \end{array}$$

$\mathcal{c}$ must not occur in any undischarged assumptions.

Universal Elimination

$$\begin{array}{l|ll} m & \forall\mathcal{X}\mathcal{A} & \\ & \mathcal{A}[\mathcal{c}|\mathcal{X}] & \forall\text{E } m \end{array}$$

Identity Rules

$$\begin{array}{l|ll} & \mathcal{c} = \mathcal{c} & =\text{I} \end{array}$$

$$\begin{array}{l|ll} m & \mathcal{c} = \mathcal{d} & \\ n & \mathcal{A} & \\ & \mathcal{A}[\mathcal{c}||\mathcal{d}] & =\text{E } m,\ n \\ & \mathcal{A}[\mathcal{d}||\mathcal{c}] & =\text{E } m,\ n \end{array}$$

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

Derived Rules

Dilemma

$$\begin{array}{l|ll} m & \mathcal{A} \vee \mathcal{B} & \\ n & \mathcal{A} \to \mathcal{C} & \\ p & \mathcal{B} \to \mathcal{C} & \\ & \mathcal{C} & \vee* \ m,\ n,\ p \end{array}$$

Modus Tollens

$$\begin{array}{l|ll} m & \mathcal{A} \to \mathcal{B} & \\ n & \neg\mathcal{B} & \\ & \neg\mathcal{A} & \text{MT } m,\ n \end{array}$$

Hypothetical Syllogism

$$\begin{array}{l|ll} m & \mathcal{A} \to \mathcal{B} & \\ n & \mathcal{B} \to \mathcal{C} & \\ & \mathcal{A} \to \mathcal{C} & \text{HS } m,\ n \end{array}$$

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)
$\neg\forall\mathcal{X}\mathcal{A} \iff \exists\mathcal{X}\neg\mathcal{A}$
$\neg\exists\mathcal{X}\mathcal{A} \iff \forall\mathcal{X}\neg\mathcal{A}$

Bentley University

For the nonconformist

What to do if you don't like the given Structured Play options

Feel free to diverge from the suggested path; here are some other paths you may like to try.

1. Check out the Lurch website, at http://lurch.sourceforge.net. It contains the following content you may find interesting.
   Videos introducing Lurch on the "About Lurch" page:
   The first video covers material already introduced in this workshop, so that would be redundant.
   The second video allows you to watch me do several of the proofs in this handout, in a screencast.
   Source code documentation, for the technically inclined who are interested in the nitty-gritty details of implementation.

2. We will probably not get to the Axiomatic Number Theory proofs during the workshop itself, so you can try the section entitled **"Play: Fitch-style Number Theory"** on your own.

3. Launch Lurch in **Developer Mode**, using the Start Menu shortcut entitled "Lurch in Developer Mode." After Launch, be sure to open up a document (using Open from the File menu or toolbar) so that you have something to inspect with the developer tools. Then try these things:
   Shift+F1: Brings up the list of scripts in the existing document, some of which are auto-run scripts that are executed when the document loads, and others of which are user-action scripts that respond to the links clicked in the right pane. Check out their code using the handy buttons and menus.
   Shift+F2: Brings up the developer console, into which you can type Javascript code and it will be executed, including code that inspects and modifies the current document. Try these for example:
   document() - returns the current document, in all its underlying XML format glory
   numDependencies() - tells you how many dependencies the document has
   dependency(0) - returns the first dependency (counting from zero)
   document().child(0) - returns the first child of the current document (which will be the "document" symbol)
   More indications of what commands will work can be inferred from the scripts list (Shift+F1, as above) and from the source code documentation for the Lob class on the Lurch website.