Lurch: A Word Processor that Can Grade Students' Proofs

Nathan C. Carter[1] and Kenneth G. Monks[2]

[1] Bentley University, Waltham, MA, USA
ncarter@bentley.edu
[2] University of Scranton, Scranton, PA, USA
monks@scranton.edu

Abstract. *Lurch* [1] is a free word processor that can check the mathematical reasoning in a document. Most notably, it can check the steps of a mathematical proof, even one not written in a formal style. The current version works best for undergraduate introduction-to-proof courses, and this paper covers our goals, current interface, and some results of classroom testing.[3]

1 Introduction

1.1 Background

Lurch is a free word processor that can check the steps of a mathematical proof, and we have focused the current version on the needs of introduction-to-proof courses, covering topics such as logic, introductory set theory, and number theory. As we add features it becomes more able to handle a wider variety of mathematics courses.

Lurch is built on OpenMath [2, 3] (as well as several other technologies, including Qt [4]) and is a free and open-source desktop application for Windows, Mac, and Linux. Section 2 gives a brief overview of the user interface. It has been tested at the two institutions at which the authors teach, and results from that testing are discussed briefly in Section 3.

1.2 Goals

The *Lurch* project has a two-part mission statement; we state and explain each part here.

[3] *Lurch* was supported from 2008-2012 in 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.

Mission Statement, Part 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.

This is our user interface guideline. Users should be able to write math in whatever style or notation they prefer, with exposition inserted where they feel it's helpful. They can already do so on a chalkboard, on paper, or in LaTeX, and especially in a pedagogical context it is important for the software to match the textbook and lecture notes, so as not to be an obstacle for the student. So mathematics in *Lurch* must be as similar as possible to mathematics in the other media just mentioned. Except, of course, that pencil and paper won't tell the user whether his or her proof is correct, but *Lurch* will, in addition to the other benefits that any word processor provides, such as cut and paste.

Mission Statement, Part 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.

Of course, there are several impressive pieces of software in existence (e.g., *Mizar* [5] or *Coq* [6]) for validating rigorous mathematics, and *Lurch* is not claiming that it will supersede these offerings. Our goals are very different from those of proof checkers like *Mizar* and *Coq*, and thus *Lurch* is not trying to compete with those packages. We list here the important differences between our goals and theirs.

1. **More flexibility**
   Most existing proof checkers require the user to learn a specific language and rules, while *Lurch* lets the user (or his or her instructor) define the rules from scratch. This includes adding new definitions, or even replacing the rules of logic themselves with new ones. Future versions will also allow the user to define the mathematical language (i.e., notation, syntax) as well.
2. **No help with proofs**
   Proof checkers automate some proof steps, and an ongoing field of research furthers the frontiers of this capability. *Lurch* has no such capabilities, nor should it, because if it did proofs for the student, that would defeat the purpose of teaching the student how to do them. In *Lurch*, the student types his or her work while *Lurch* grades, encourages, and coaches.

3. **Familiar user interface**
   Proof checkers often have interfaces requiring advanced users, such as those familiar with the command line or shell scripts. *Lurch* is for the typical student, and its user interface is therefore a familiar one: It is a word processor that gives feedback visually in the document. We want typical mathematics students to encounter no learning curve except what the mathematics itself presents, not *Lurch*.

We're targeting students in their first proof-based courses, giving frequent, immediate, clear feedback on the steps in their work. That's the part of the mathematical community we're excited about reaching. Other projects are working on some of these same goals, such as simpler and more flexible notation for provers [7, 8] and integration of a prover into a graphical user interface [9]. But to our knowledge, no one is writing a learning tool like *Lurch* for our target audience.

Existing research on educational technology suggests that our goals make sense and are achievable. Investigations into the effects of automated assessment systems (AiM [10] and MyMathLab [11]) show the value of computers in giving high-frequency, individualized, and immediate feedback ([12–15]). The value of such feedback is well-documented in educational research (reviewed in [16]). This research matches our common sense that more feedback, delivered in immediate response to each of the student's actions, is better for learning.

This feedback is also noteworthy in another respect. When we think of students' experiencing mathematical objects on a computer, we tend to think of interactive graphs with sliders, rotating three-dimensional surfaces, or other flashy graphics. In *Lurch*, students experience a far more abstract mathematical object. The positive and negative feedback they get while working within a logical system is their experience *of that system,* including its rules, axioms, and theorems. In Section 3, quotes from student surveys support this assertion.

2 Current Interface

This section summarizes briefly how far we have come (as of this writing) towards achieving the goals in Section 1.2.

2.1 A word processor with semantics

First, we have built a word processor that supports mathematical expressions with actual mathematical semantics. Consider the screenshot of the

Mac version of our application in Figure 1. The red, yellow, and green icons interspersed throughout the user's document are *Lurch*'s feedback on the correctness of each step of the user's work. (This feedback can be hidden to obtain a clean view for printing.)

Green thumbs-up icons follow a correct step of work that is correctly justified, red thumbs-down icons follow a step that contains an error or is incorrectly justified, and yellow lights follow mathematical expressions that are used as premises to justify later steps of work, but for which the user provided no justification (e.g., the hypotheses in a proof).

Students can hover their mouse over any of these colored icons for a brief explanation of its meaning, or double-click it for a detailed report. In order to provide this kind of feedback, *Lurch* needs to know the *meaning* of the mathematical expressions in the document, and from the fact that the feedback in Figure 1 is correct, you can see that it does.

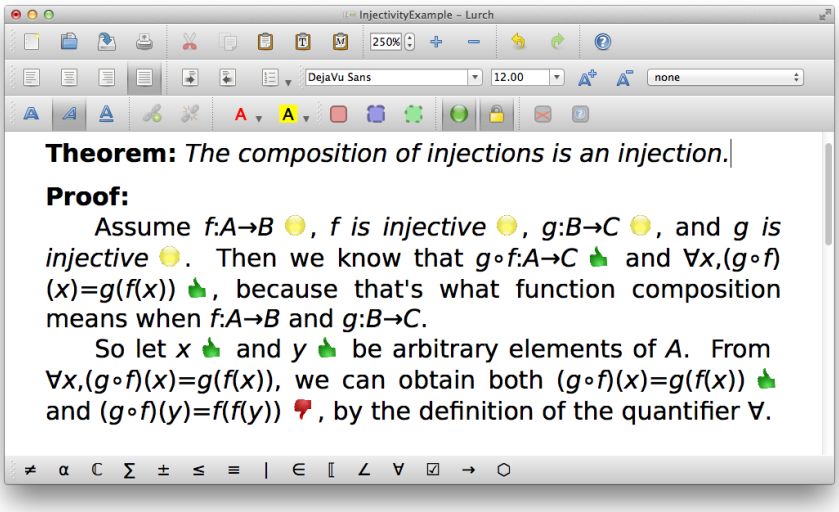

Fig. 1. A *Lurch* screenshot explained in Section 2.1. Although this screenshot was captured on a Mac, *Lurch* is free and open-source for all three major platforms, Mac, Windows, and Linux. The default view uses three colors of "traffic lights," but the thumb icons shown here are a variant useful to colorblind users and in black-and-white printing.

Lurch knows the meanings of mathematical expressions in the sense that it is aware of the rules defined elsewhere in the same document (or documents on which it depends), and those rules define the valid uses of each type of expression. It is only this sense in which *Lurch* knows mathematical meaning; it knows the valid ways to work with each type of expression. A fuller description of meaning and validation in *Lurch* appears in [17]. So how does *Lurch* read the user's meaning?

2.2 The bubble interface: Marking expressions as meaningful

The second major component of what we have built so far is the user interface through which users tell *Lurch* the meaning of the mathematics in their documents. Of course, the ideal interface would be for the computer to read a document and, without any help from a human, infer correctly which parts are meaningful and what their meanings are.[4]

Not only do current limitations on technology for natural language processing make this a great challenge, but there are actually pedagogical benefits to be derived from expecting the user to be involved in the process. But it is essential that *Lurch* not require much of the user beyond simply typing in their document, so that he or she is not distracted from the mathematics, and his or her work is slowed only in the slightest. To solve this problem, we employ a user interface paradigm we call *bubbles.*

In order for a section of text to be treated as a mathematical expression, the user must mark it as such with a single click of a toolbar button (or the corresponding keyboard shortcut), as illustrated in Figure 2. The user selects a portion of text that he or she wishes *Lurch* to treat as meaningful, and then clicks the "Meaningful expression" button on the toolbar. *Lurch* then wraps the text in a bubble and reports the understood meaning in a tag atop the bubble.[5] If *Lurch* cannot understand the content, it calls it a string, treating it as if it were surrounded in quotes.

Lurch draws a bubble around a mathematical expression if and only if the user's cursor is inside it; hence a bubble appears in Figure 2 but none in Figure 1. So when the cursor is not in a mathematical expression, no extra visual clutter asserts itself. But when the cursor is in one, the interface makes this fact clear, and right-clicking a bubble exposes a menu of actions relevant to the bubble, such as changing its formatting or asking

[4] In this context, a portion of the document is "meaningful" if it is an essential part of the mathematical structure of the user's argument. In other words, meaningful here means meaningful *to the grading algorithm.*

[5] Currently only Unicode math symbols from a palette are supported; full WYSIWYG math editing is still in progress.

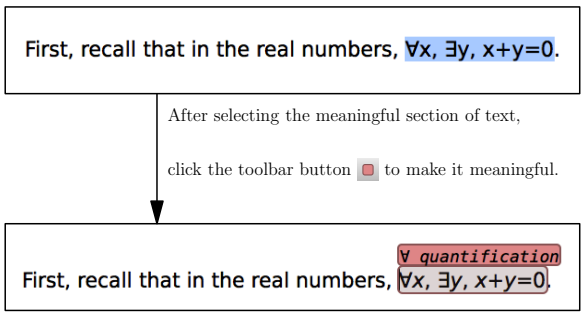

Fig. 2. How a sentence within a *Lurch* document changes when a user marks a section of text as meaningful. On the top, the user has selected a section of text. After telling *Lurch* to mark it as meaningful, the red bubble shown on the bottom appears, indicating that *Lurch* has read the expression, and knows that it is a universal quantification.

for help on it. This paradigm is not completely unique to *Lurch*; both *LyX* and *Word* have somewhat similar interfaces for editing mathematics.

Thus it is a trivial task to mark expressions as meaningful. It is something the user will do often, and thus must take near-zero time, and be intuitive as well. In Figure 3, you can see the toolbar in which the "Meaningful expression" button appears. It is just as easy to mark text as meaningful as it is to make it bold or italic; a single click or keystroke accomplishes it. This also makes the action of marking text as meaningful seem natural to the user, by emphasizing its similarity to familiar word processing actions like bold, italic, and underline.

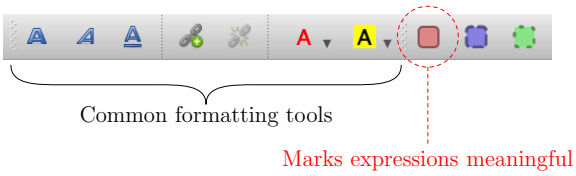

Fig. 3. One of the *Lurch* toolbars, shown here as it appears on Mac OS X. Some of the buttons are the typical formatting buttons one expects to see in any word processor, such as bold, italic, and underline. The most important new one to note here is the red button for marking text as meaningful.

2.3 What bubbles accomplish: A thin user interface layer

The bubble interface enables several important features. First, *Lurch* can *know* which portions of the document the author considers meaningful and which he or she does not. It does no natural language processing or guesswork, which may be prone to misinterpreting the author's wishes, but the correct classification of meaningful vs. non-meaningful content is assured, because it is fully under the user's control.

Second, in line with the first item in our mission statement (Section 1.2), the user is stylistically free to structure their document their own way. Because meaning is determined by what sections of text have been bubbled, the user can arrange expository portions and mathematics however they like, from informal sentences to formal proof structures and varying levels in between.

Third, the user is free to use any natural language, from Polish to Portuguese, because *Lurch* is ignoring all of the prose, concentrating only on mathematical expressions the user has singled out as meaningful.

Finally, there is also a pedagogical benefit. Student users, just learning what a proof is, are forced to indicate which parts of their own proofs are meaningful and which are not. Experience from testing shows us that this is perhaps the clearest lesson students learn from *Lurch*, compared to our experience in courses without *Lurch*. Students come away from a *Lurch*-integrated course knowing very solidly how a proof is structured out of statements, reasons, and premises. (In addition to marking statements as meaningful, there are similar actions for marking which text represents reasons and premises.) In fact, it was not until *after* this learning had fully sunk in that students perceived the bubbling process as a burden.

Obviously, these four benefits come at the cost of giving the user an added burden when typing their document into *Lurch*. But that burden is nearly as minimal as one could want (assuming that *Lurch* is not doing natural language processing). Furthermore, it has educational benefits as described above, as it requires the student to be explicit about their proof structure. And we have plans to simplify the bubble interface further.

3 Classroom Testing

Lurch was tested throughout an introduction to formal logic class the first author taught at Bentley University in the fall of 2008, and an introduction to proofs course for mathematics and mathematics education majors the second author taught at the University of Scranton in the spring of 2013. In both cases, student response was very positive. Quotes

from student surveys mentioned all the main benefits that we as developers expected the software to have. We provide sample student quotes in each of two main themes here.

On the value of frequent and immediate feedback

"I liked that we could check our work as we are doing it—it makes the learning more immediate by providing constant feedback even outside of class."

"It allowed me to guarantee when I got something right, which was helpful early on when I was unsure about what a proof was."

"I was intimidated by proofs before taking this class, but *Lurch* helped boost my confidence."

On experiencing a mathematical system through *Lurch*

"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."

"*Lurch* became easier as I learned the rules and definitions, so using it in turn helped me learn those."

"It made it easier to identify statements, reasons, and premises."

"Really helped me learn what a proof is."

Some of the education research cited in Section 1.2 finds that students with access to frequent, immediate feedback spend more time working on homework, because they do not want to turn in work that they know is incorrect. Students mentioned this on our surveys as well, one explicitly saying, "I worked until I was correct."

One natural worry, given these positive student responses, is that repeated guesswork alone might create a proof in *Lurch*. With enough feedback, do students lose the need to be creative nor actually learn anything, and complete their homework with persistent experimentation alone? Our surveys asked for agreement or disagreement with the sentence "It is possible to do a proof in *Lurch* by experimental clicking and typing, without thinking." Students from 2008 (whose version of *Lurch* was much more automated than the current version) responded neutrally, 3.2 on average. This latest version of *Lurch*, however, has significantly improved this score to a 1.6, part way from disagree to strongly disagree.

Students also reported being surprised at how long it took them to learn what was required to form a correct proof; that difficulty would

have consumed more of the beginning of the course if *Lurch* hadn't made it abundantly clear when their homework still contained errors. One said, "If I did not have *Lurch*, I would have had a lot of incorrect proofs."

And regarding the software learning curve, the average of student responses *disagreed* with the sentence, "Learning to use Lurch took a lot of time that I could have spent learning logic instead."

Lurch use in the test courses was optional for each student. Instructors observed that students who did not choose to use *Lurch* for their homework fell into two categories. Either they were doing very well in the class and did not feel a need for extra help, or they didn't like being told that they were wrong, and thus avoided the software. This latter group was small, and most of them immediately switched to using *Lurch* after their first poor homework grades. One student said that getting back her first graded homework showed her why *Lurch* wouldn't accept her proofs—because they were wrong![6]

4 Conclusion

Lurch has many opportunities for improvement, including testing in courses not taught by the developers and new features (typeset mathematics, more efficient validation algorithms, a type system, more attractive visual representations of premise relationships, and an even less obtrusive bubbling interface). But the current version, despite its opportunities to improve in many ways, shows all the signs of having been very beneficial in the courses in which it has been used.

The *Lurch* team invites collaborators for writing mathematical topics in *Lurch*, doing further classroom testing, and software development.

References

1. Carter, N.C., Monks, K.G.: *Lurch:* a word processor that can check your math. [online] (2013) A free and open-source software project hosted on SourceForge.net, available at `http://lurch.sf.net`.
2. Arsmac, O., Dalmas, S.: OpenMath INRIA C/C++ libraries. [online] (Downloaded May 18, 2008) Available at `http://www.openmath.org/software/index.html`.
3. Buswell, S., Caprotti, O., Carlisle, D., Dewar, M., Gaëtano, M., Kohlhase, editors, M.: The OpenMath standard, version 2.0. The OpenMath Society (June 2004) Available at `http://www.openmath.org/standard/`.

[6] Very rarely, a student would find a case in which *Lurch* incorrectly graded their work. This happened only twice in the most recent semester of testing, and both times we fixed the bug and did an immediate point release for the students to upgrade.

4. The Qt Project: Qt 4.8. GNU General Public License (and the lesser) version 3.0 (2012) A cross-platform application and UI framework for developers using C++, QML, CSS, and JavaScript, available at http://qt-project.org/.
5. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Mizar in a nutshell. Journal of Formalized Reasoning (2010) Available at http://jfr.unibo.it/article/view/1980/1356.
6. Bertot, Y., Castéran, P.: Coq'Art: the calculus of inductive constructions. Springer, XXV, 472 p. ISBN 978-3-540-20854-9 (2004) Available at http://www.labri.fr/perso/casteran/CoqArt/index.html.
7. Dietrich, D., Schulz, E., Wagner, M.: Authoring verified documents by interactive proof construction and verification in text-editors. Springer Lecture Notes in Computer Science Volume 5144, 2008, pp.398–414 (2008)
8. Autexier, S., Fiedler, A., Neumann, T., Wagner, M.: Supporting user-defined notations when integrating scientific text-editors with proof assistance systems. In *Towards Mechanized Mathematical Assistants:* 14^{th} Symposium, Calculemus 2007, 6^{th} International Conference, MKM 2007, Hagenberg, Austria, Proceedings (2007)
9. Wenzel, M.: Isabelle/jEdit — a prover IDE within the PIDE framework. In J. Jeuring et al., editors, Conference on Intelligent Computer Mathematics (CICM 2012). Springer LNAI 7362. (2012)
10. Delius, G., Strickland, N.: AiM—assessment in mathematics. [online] (Accessed on April 22, 2013) A free and open-source software project hosted on SourceForge.net, available at http://sourceforge.net/projects/aimmath/.
11. Pearson Education: MyMathLab. [online] A series of mathematics courses on the World Wide Web, available at http://www.mymathlab.com/.
12. Klai, S., Kolokolnikov, T., Van den Bergh, N.: Using Maple and the web to grade mathematics tests. In IWALT 2000 Conference Proceedings (2000) Available at http://rc.fmf.uni-lj.si/matija/ACDCA2000/Kolokolnikov-Klai-Bergh-pdf.pdf.
13. Sangwin, C.J.: Assessing higher mathematical skills using computer algebra marking through AiM. In Proceedings of the Engineering Mathematics and Applications Conference (EMAC03, Sydney, Australia), pages 229–234 (2003) Available at http://web.mat.bham.ac.uk/C.J.Sangwin/Publications/Emac03.pdf.
14. Sangwin, C.J.: Assessing mathematics automatically using computer algebra and the internet. Teaching Mathematics and its Applications, 23(1):1–14 (2004) Available at http://web.mat.bham.ac.uk/C.J.Sangwin/Publications/tma03.pdf.
15. Speckler, M.D.: Making the grade: A report on the success of MyMathLab in higher education math instruction. Pearson Education, Boston, MA (2005) Available at http://www.mymathlab.com.
16. Chickering, A., Gamson, Z.: Seven principles for good practice in undergraduate education. American Association for Higher Education Bulletin, pages 3–7 (1987) Available at http://aahebulletin.com/public/archive/sevenprinciples1987.asp.
17. Carter, N., Monks, K.G.: *Lurch*: A word processor built on OpenMath that can check mathematical reasoning. In Aspinall, D., Carette, J., Davenport, J., Kohlhase, A., Kohlhase, M., Lange, C., Libbrecht, P., Quaresma, P., Rabe, F., Sojka, P., Whiteside, I., Windsteiger, W., eds.: Proceedings of the Workshops and Work in Progress at the Conference on Intelligent Computer Mathematics. Number 1010 in CEUR Workshop Proceedings, Aachen (2013)