Z3Guide: A Scalable, Student-Centered, and Extensible Educational Environment for Logic Modeling

Ruanqianqian (Lisa) Huang*
r6huang@ucsd.edu
UC San Diego
La Jolla, CA, USA

Ayana Monroe*
aam285@cornell.edu
Cornell University
Ithaca, NY, USA

Peli de Halleux
jhalleux@microsoft.com
Microsoft Research
Redmond, WA, USA

Sorin Lerner
lerner@cs.ucsd.edu
UC San Diego
La Jolla, CA, USA

Nikolaj Bjørner
nbjorner@microsoft.com
Microsoft Research
Redmond, WA, USA

ABSTRACT

Constraint-satisfaction problems (CSPs) are ubiquitous, ranging from budgeting for grocery shopping to verifying software behavior. Logic modeling helps solve CSPs programmatically using SMT solvers. Despite its importance in many Computer Science disciplines, resources for teaching and learning logic modeling are scarce and scattered, and challenges remain in designing educational environments for logic modeling that are accessible and meet the needs of teachers and students. This paper explores how to design such an environment and probes the impact of the design on the learning experience. From a need-finding interview study and a design iteration with teachers of logic modeling, we curated 10 design guidelines spanning three main requirements: providing easy access, supporting various educational modalities, and allowing extensions for customized pedagogical needs. We implemented nine guidelines in Z3Guide, an open-source browser-based tool. Using Z3Guide in a logic modeling learning workshop with more than 100 students, we gathered positive feedback on its support for learning and identified opportunities for future improvements.

CCS CONCEPTS

• **Applied computing** → **Interactive learning environments**; **Computer-assisted instruction**; • **Human-centered computing** → *User centered design.*

KEYWORDS

logic modeling, Z3, human-centered design, web environment

1 INTRODUCTION

Constraint-**S**atisfaction **P**roblems (CSPs) involve finding an optimal solution under prescribed constraints. Most decision making involves solving a CSP, from budgeting groceries to scheduling interviews. Logic modeling is an approach to solving CSPs *programmatically*, by translating constraints to a set of logical formulas and using "**S**atisfiability **M**odulo **T**heories" or SMT solvers to determine their satisfiability [24, 29].

With advances in SMT solvers like Z3 [20] and their interfaces to widely-used programming languages such as C++ and Python, logic modeling is useful in many aspects of Computer Science, from Artificial Intelligence to Program Verification, and appears as a part of many standard computing curricula [36]. As such, there is a great need for accessible and student-centered educational environments for logic modeling, especially since the COVID-19 pandemic [45]. However, creating an educational environment for logic modeling that is accessible, meets the needs of various teachers and students, and provides different learning modalities [23], imposes a design challenge [19] that prior work has yet to fulfill [2, 9, 10, 13, 43, 51].

Motivated by these concerns, we conducted a design exploration to identify design guidelines for an educational environment for logic modeling. We interviewed six university faculty who have taught logic modeling with Z3 [20], which has been widely used for logic modeling, to understand user needs and technical requirements. We then developed a prototype learning environment, and iterated on the design with four faculty. The design exploration surfaced 10 guidelines that suggest three main requirements for an educational environment for logic modeling: (1) providing easy access (2) supporting various educational modalities (3) allowing extensions for customized pedagogical needs.

We implemented nine of the 10 guidelines in Z3Guide, a web-based tool for logic modeling. Z3Guide has an interactive textbook, a freeform editor, games, and access to external resources and the Z3 community, supporting both formal education and casual learning. Z3Guide is 100% client-side—with no server-side computation—and open-source[1], minimizing its maintenance costs, allowing easy extension, customization, and contribution, and addressing the scalability and maintainability issues in its predecessor `RiSE4Fun` [1, 13]. With Z3Guide, students can learn the basics of logic modeling and programming with the Z3 API in various programming languages, by engaging in the following activities, all directly within a web browser: (1) interacting with textbook-like tutorials that include real-world problems and code examples (2) writing larger programs in a playground (3) solving logic puzzles. Teachers can further use the tool for in-class demonstrations, supplementary exercises, or custom extensions for their own pedagogical needs, such as promoting active learning [11].

We used Z3Guide in a three-hour online workshop where more than 100 participants learned logic modeling with Z3Guide. In a post-workshop survey ($N = 21$), students rated Z3Guide to be easy to use and found it enjoyable to learn logic modeling and Z3 using Z3Guide. We reflect on the design guidelines, in combination with

*Work done while interning at Microsoft Research.

[1]Maintained at https://github.com/microsoft/z3guide

our own usage experience, to inform future educational systems for logic modeling and beyond.

2 RELATED WORK

2.1 Background: Logic Modeling and SMTs

A Constraint-Satisfaction Problem (CSP) decides whether a given set of constraints can be satisfied. Logic modeling solves a CSP programmatically by formally representing the CSP in a set of logical formulas and solving it using Satisfiability Modulo Theories.

Satisfiability Modulo Theories or SMT solvers support a formalism based on simply typed first-order logic with built-in theories for domains used in software. The theories include the theory of integer and real arithmetic, arithmetic over bit-vectors that correspond to the operations made by assembly code, as well as arrays and algebraic data types. These theories, along with the first-order logic, further enable modeling program behavior. As such, SMT solvers have been established since the early 2000s as a foundation for symbolic program analysis, verification, and testing.

We show an example of how logic modeling translates statements to first-order predicate formulas to be solved by the SMT solvers. Given the following statements:

All humans are mortal. Socrates is a human.
Therefore Socrates is mortal.

The statements can be encoded as first-order predicate formulas:

$$((\forall x \,.\, human(x) \implies mortal(x)) \land human(Socrates)) \implies mortal(Socrates)$$

One could further translate the formulas to a format recognized by SMT solvers (e.g., SMTLIB [14]) for their satisfaction to be checked.

Logic modeling with SMT solvers, grounded on first-order logic, is distinct from writing *algorithmic* code in an imperative, functional, or object-oriented programming language. Unlike imperative programs, logical formulas have no notion of side effects. Even when compared to functional programs, logical formulas cannot be evaluated. Logic modeling is also distinct from modeling using linear algebra, which is widely used in operations research and data science disciplines.

Z3 is a state-of-the-art SMT solver widely used for logic modeling [20]. Z3 supports many theories, reasoning with quantifiers, and customizing the solving process, enabling logic modeling in an extensive range of domains used in Formal Methods. Our work resulted in Z3Guide, an educational environment that assembles information about the full capability of Z3 starting from an elementary introduction to more advanced applications.

2.2 Web Tools for Formal Methods Education

Logic modeling is part of the Formal Methods discipline, which integrates mathematically rigorous techniques for the analysis and construction of systems [49]. There have been increased interests, especially since the COVID-19 pandemic [45], in online curricula and tools that facilitate the education of Formal Methods. Here we review web-based tools for the education of Formal Methods, including e-textbooks, programming environments, and tools that integrate them both.

E-textbooks allow for both guided and self-paced learning. The Software Foundations series [43] covers a broad range of Formal Methods topics from programming language theories to separation logic foundations. SAT/SMT by Example [51] contains real-world examples of problems solvable by SAT/SMT tools. Sage Tutorial [10] introduces using Sage, a tool for modeling and solving math problems, with code examples. These e-textbooks, however, are *static*, requiring students to use one or more external environments to run the embedded code examples. Iltis [25, 26] is *interactive*, letting students learn and assess their learning of Logic and Theoretical Computer Science concepts via multiple-choice and short-response questions, but it does not have any code examples for applying the concepts to programming. Finally, for all these e-textbooks, an important limitation is that there are no effective ways for students to ask questions about any text sections or examples.

There are also web-based programming environments for the education of Formal Methods that contain no conceptual tutorial content. KalkulierbaR [37] provides a series of games that help practice concepts of logic modeling. DiMO [34] supports using a SAT solver without installation. Alloy4fun [42] enables writing programs of Alloy [35], a language for modeling software behavior, within the browser. SageMathCell [9] runs Sage code from the web, supplementing the Sage Tutorial [10]. One limitation of all of the tools above is that they rely on server backends, which induce costs of tool maintenance and scalability concerns. A bigger limitation is their detachment from the tutorial materials of the corresponding languages/tools: while they ease writing programs in these languages/tools by not requiring software installation, the user could not learn about these languages/tools within the environments themselves.

A web-based *integrated environment* supports both learning and programming Formal Methods concepts by combining an e-textbook and a programming environment. The most noticeable and closest to our work is `RiSE4Fun` [13], a web environment for multiple logic modeling tools. `RiSE4Fun` combined tutorials for logic modeling with a playground for each tool. However, `RiSE4Fun` shared the same limitation as the e-textbooks [10, 25, 43, 51] that there was no effective support for students to ask questions about the tutorial. Also, like the environments above [9, 34, 37, 42], `RiSE4Fun` was limited in its implementation: it relied on a server backend that was vulnerable to attacks and large traffic [13], and the maintenance costs were high due to its lack of extensibility [1]. For `RiSE4Fun`, these isues became severe to the point that it was discontinued [1]. Our system Z3Guide addresses these limitations by adding shortcuts for asking tutorial-related questions in its interactive examples, implementing a completely client-side architecture, and enforcing extensibility.

More importantly, despite the variations of web-based educational tools for Formal Methods, comparatively few works explore how to design them to effectively support teaching and learning. Runge et al. [45] found through two qualitative studies with students of Formal Methods that, for these tools to facilitate online teaching, they must support various educational modalities and be interactive, accessible, and engaging. Our work reveals more pedagogical needs for an educational tool for logic modeling through a design exploration and a workshop evaluation.

Table 1: Need-finding study participants. "Teaching Focus" is the main topic of their classes where logic modeling is taught.

ID	Gender	Title	Main Area of Expertise	Teaching Focus
P1	M	Assoc. Prof.	Compilers	Compilers
P2	F	Assoc. Prof.	Formal Methods	Verification
P3	M	Prof.	Verification	Verification
P4	M	Prof.	Automated Reasoning	Verification
P5	F	Assoc. Prof.	Program Synthesis	SAT/SMT
P6	M	Asst. Prof.	Formal Methods	Verification

2.3 Web Tools for Programming Education

Beyond logic modeling and Formal Methods, there is a rich line of web-based tools that support programming education. Online interactive textbooks like Stepik [7] and Runestone [22] let students learn programming concepts and complete programming practices in the browser. There are also various programming environments covering a broad range of topics. Python Tutor [31] has allowed millions of users to program, share, and visualize Python code in the browser, and led to design guidelines [30] for scalable research software in academia. Inspired by Python Tutor, Pandas Tutor and Tidy Data Tutor [38] support data science education at scale. Godbolt [27] allows students learning compilers to compare compilation outputs from different compilers without any installation, which is usually platform- and architecture-dependent. CS50 uses GitHub Codespaces [6] to support introductory programming without local configuration [4]. Finally, integrated environments like Ed [5] combine digital textbooks, programming experience, and online discussions. Our work is complementary to these efforts by focusing on the education of logic modeling and adopting a human-centered design.

3 DESIGN EXPLORATION WITH TEACHERS

To derive design guidelines for an educational logic modeling tool, we conducted a two-stage exploration: a need-finding interview study with six university faculty (Sec. 3.1), and a design iteration with four previously interviewed faculty on a prototype we built based on the interview findings (Sec. 3.2). Both studies were run by the first two authors via web conferencing, involved no compensation, and received IRB approval. This section reports the methodology of each study. Detailed findings are reported in Sec. 4.

3.1 Need-Finding Interview Study

We recruited participants by emailing faculty with publications and teaching experience in logic modeling according to their websites. We required that participants have experience with `RiSE4Fun` [13], a deprecated web environment for logic modeling that had been widely used by teachers. Eventually, six faculty from six institutions across five countries gave their consent to participate. Table 1 shows their demographics. We deem the sample size of six participants reasonable due to the small size of the logic modeling community.

Each participant went through a 60-minute semi-structured interview. We first asked how they taught logic modeling and related tools. Then, we asked questions from four categories: (1) the integration of `RiSE4Fun` into the curriculum (2) failures and successes of the integration in supporting their and/or the students' needs (3) challenges faced teaching logic modeling and Z3 (4) expectations for a better tool. Each category had a list of questions that interviewers could ask out of order and follow up on when necessary.

We recorded the audio and video of each interview. The web-conferencing software transcribed the audio, and the first two authors fixed transcript errors by revisiting the video recordings. We analyzed the data using thematic analysis [17]. The second author coded all data, and the first author coded 25% of the data to compute percentage agreement [48] and establish reliability. Eventually, the coders agreed upon 88% of the sampled data. The first author then grouped the codes into themes, which are reported as paragraph headings in Sec. 4.

3.2 Design Iteration

Based on findings from the interviews, we developed a prototype similar to our final design (Sec. 5). We re-recruited the interview participants to evaluate the initial prototype per the iterative nature of the study. Four participants (P1, P2, P4, P5 in Table 1) agreed to be in the prototype evaluation.

Each 45-minute evaluation consisted of a 15-minute open-ended exploration and a 30-minute cognitive walkthrough [41], a commonly used technique for evaluating early prototypes. Anyone experienced in UX research or the corresponding domains could perform cognitive walkthroughs while revealing usability problems via realistic use cases. In our study, the participants are experts in the education of logic modeling: they are familiar with the topic and the kinds of problems the students could encounter.

During the initial exploration, each participant commented on things they noticed, asked clarifying questions, and summarized how they understood the prototype. In the cognitive walkthrough, they picked one topic in the prototype they would cover (e.g., "Logic") when teaching a related course (*e.g.*, software verification) and used the tool following an imaginary scenario:

> "You were a student in [a specified course] learning logic modeling and Z3, and you were asked to use the tool as supplementary reading and exercises. How would you use it to learn concepts in [the selected section]?"

For each action the participants performed, we asked how they expected the interface to react, whether the existing design matched their expectations, and their ideal design and why.

We recorded the audio, video, and the participant's screen of each session, and noted down interactions with the prototype. We used the same audio transcription approach in Sec. 3.1. Relating to the findings from Sec. 3.1, the first author used top-down coding to code the transcripts and notes of interactions. Like in Sec. 3.1, the first two authors used 25% of the coded data to compute agreement, which was 87.5%. The authors agreed upon themes emerging from the codes reported as paragraph headings in Sec. 4.

4 INITIAL DESIGN GUIDELINES

We derived 10 design guidelines (denoted as **D**s below) under three categories for educational tools for logic modeling from the interviews and design iterations (Sec. 3).

Providing Easy Access. All participants considered easy access to the tool—regardless of educational setting and role—a priority, particularly with regard to programming experience.

D1: *Fast Execution*. In the interview, five participants recalled unpredictable slowdowns in `RiSE4Fun` that interrupted lectures. They remarked that *interactive speed* is necessary for the tool to facilitate effective classroom interactions.

During the design iteration, all participants liked the *"snappy"* (P4) speed of our prototype most of the time. However, the tool froze for P2 and P5 at the end of their sessions after several code executions due to a memory leak bug that we later fixed.

D2: *Code Sharing*. Four interview participants (P1, P2, P3, P6) deemed it important to easily share code snippets with others. With `RiSE4Fun`, one could share permalinks with others to recreate the state of the tool and their code. While this feature might not be essential (due to workarounds like copying/pasting), the four participants considered it to be useful, as *"people send links to each other all the time [to] teach each other"* (P1).

Our prototype did not enable direct code sharing or a shortcut for copying code snippets. However, P2 attempted to copy code snippets across editors multiple times when using the prototype with the select-all and copy keyboard shortcuts, and P1 suggested that students should be able to share/replicate both a code snippet and its execution state with their classmates or teachers.

D3: *Editing Support*. In the need-finding interview, P4 and P6 found it necessary to have visual indicators in the code editor and output display, such as syntax highlighting and visually formatting long output, which `RiSE4Fun` lacked. They believed that these features would help students catch errors earlier on and understand the output. Compared to debugging algorithmic code, debugging logic modeling code has been a hard problem [28, 40], but part of it could be mitigated with appropriate editor support for getting the syntax correct and avoid *"the [output] given to the students [not being] straightforward"* (P6).

The design iteration further reinforced the necessity of appropriate editing support. Our prototype lacked syntax highlighting and autocompletion, while Z3 could be written in SMTLIB, a format that adopts a Lisp-like syntax with a heavy use of parentheses. As such, every participant made at least one mistake in balancing parentheses that took a long time to debug during the cognitive walkthrough due to the lack of IDE support. In addition, P4 further found the localization of error messages difficult because the editor did not have line numbers. Finally, all four participants demanded that the tool support resetting an edited snippet to its original content and reverting any accidental reset.

Supporting Multi-Modal Education. Participants proposed engaging and supporting students via the tool from five aspects.

D4: *Small Examples*. Most interview participants requested that the tool explain logic modeling concepts through small code examples. Participants believed that by building content with small code examples, the tool would encourage students to run the examples and lead to a better understanding of the concepts. In addition, the small examples might also encourage students to make edits, *"play around, and [re-execute]"* (P3), which could deepen their understanding of the connection between the code and the output.

The design iteration confirmed the benefits of executable and editable code examples, as all participants tweaked and executed the examples that came with the prototype while going through the material. P2 further pointed out that allowing the user to run each code snippet on the fly could avoid *"the hassle of copying and pasting code examples"* into an external editor. She imagined letting her students *"run [each code example] themselves"* in class to reinforce their understanding of the related topics. Finally, participants suggested having code examples with intentional errors so students could learn about repairing them.

D5: *Freeform and Exploratory Programming*. Four interview participants (P2, P4, P5, P6) wanted *"a simple playground where [students] can try certain things out"* (P4) in the tool. P6 deemed a freeform editor also important for the teacher to quickly demo some logic modeling code (i.e., *live coding* [46]) for the students to follow along and further explore.

Our prototype came with a freeform editor that, compared to other concept-related interactive examples, was on its own page and not attached to any concept. In her walkthrough, P5 demonstrated how a student could use the editor to create logic formulas and solve an example problem in her course slides. However, the freeform editor was not editable until after the user ran the sample code inside, which the participants found confusing and contrary to its intention for freeform explorations. This design inherited that of the concept-related examples. While all participants liked this design for those examples, they suggested that the freeform editor be always editable to encourage exploratory programming. The free editor in the prototype was also small, which P1 thought should be larger to encourage various sizes of exploration.

D6: *Gamification*. Two interview participants (P1, P3) believed that games would benefit logic modeling students. P1 particularly referred to competitions (a form of games) for compiler optimization: he would use competitions in his class to encourage students to write compilation code as optimized as possible, then release the competitions to the public with rewards for the winner, and people in the compilers community *"love these kinds of competitions."* He suggested that such competitions could be part of the tool to help practice concepts and engage students.

Our prototype did not include any games, and P1 reiterated the importance of having alternative forms of educational materials beyond readings and exercises in the tool.

D7: *References to Basic Concepts in Logic*. Logic modeling has a basic logic prerequisite, but according to the interview participants, students still occasionally need references to these concepts.

Our prototype lacked such references, to which P4 remarked that most students at the start of the academic term *"[didn't] remember anything [...] about basic logic"* although *"everybody had taken [the prerequisite course]"* (P4). This quote exemplifies the need new logic modeling students often have for a basic logic concepts refresher. It is thus beneficial to provide basic logic references in the tool.

D8: *Question Asking*. Four interview participants (P1, P3, P4, P5) mentioned that students learn through asking questions and obtaining answers [11], but many of them feel uncomfortable asking or answering questions during class. Asynchronous question-asking thus became a feature of interest. For example, P1 used Slack [47] in his class so that students could ask questions and receive help from the teaching staff and other students asynchronously. The participants would particularly like the tool to support asking questions about individual code snippets because most questions involve specific code examples. Moreover, they saw the potential of native

question-asking inside the tool for casual learners to receive help from the entire logic modeling community.

In our prototype, each code example came with a "Discuss" button that took the user to the GitHub Discussion of the Z3 repository, which is active with people asking/answering questions about Z3 and logic modeling. The button, which was next to the "Run" button for each code snippet, received compliments for its functionality (P2, P5) and complaints about its appearance (P1). P1 suggested a redesign of the button for it to be noticeable but not distracting.

Allowing for Extensions. Participants hoped for open source in both the tool and related educational resources.

D9: *Resources Sharing*. Four interview participants (P1, P2, P3, P5) said that teachers of logic modeling typically share teaching resources (e.g., course slides) with one another and reuse existing ones. However, there was no dedicated space for the sharing of such resources. Participants saw an opportunity in an educational tool for logic modeling for *"sharing and publishing [these teaching resources] systematically"* (P2) that makes them accessible, credits their authors, and enables community-driven improvements. The tool could also serve as a platform for informing users of related technologies. For example, user propagators are techniques that allow users to write custom theory extensions for the Z3 solver [16]. P3 and P4 had created user propagators, and they would like the work built by themselves and others to be available to teachers and students. In particular, P3 suggested that there could be a centralized space for Z3 users to access these propagators, whether they are work-in-progress or ready-to-use.

Our prototype provided links to other Z3- and logic modeling-related resources in the bottom, which caught the attention of P1 and P2, both of whom found them to be useful.

D10: *Tool Extensibility*. Three interview participants (P2, P3, P4) desired the ability for teachers to easily extend the tool for their pedagogical needs. An extensible tool could allow teachers to use its technical infrastructure while adding their own content, including text and code examples. Participants also hoped for integrations with existing learning platforms for activities such as grading and assignment generation. Ideally, the tool should enable connections to these services, e.g., at the source code level.

Although our prototype did not implement extensions facing the teachers, in the design iteration we mentioned that the tool would be open source on GitHub for contributions and extensions, about which all participants were excited.

5 DESIGN OF Z3GUIDE

We designed Z3Guide, a web-based tool for the education of logic modeling with the Z3 SMT solver [20] that is scalable, student-centered, and extensible, addressing all design guidelines (Sec. 4) except **D2**. Sec. 5.1 previews the design via usage scenarios. Sec. 5.2–Sec. 5.4 each details how we accomplished each of the three categories of design guidelines reported in Sec. 4.

5.1 Usage Scenarios

We describe two usage scenarios below to demonstrate the functionality of Z3Guide. The first scenario is fictional based on contents in Z3Guide, whereas the second scenario is based on how P5 used it in the design iteration (Sec. 3.2).

Informal Learning. Sofia is a working professional. Although she is not in the tech industry, she got her college degree in Applied Math and knows logic and basic programming. As such, she is always curious about tools that help solve mathematical problems programmatically. Recently, she heard about Z3 and logic modeling from a friend who used it to arrange seats for an event, which involves dealing with various constraints. She decides to learn about logic modeling and Z3 in her spare time using Z3Guide.

(1) Sofia presses "SMTLIB Tutorial" (Fig. 1-A) to learn the basics of logic modeling using the Z3 SMT solver in SMTLIB format [15], a syntax many SMT solvers use, following a textbook-like tutorial (Fig. 2). She runs the Z3 code snippets embedded in the tutorial, sometimes with changes, to understand the concepts. When she has questions about a snippet, she presses a button inside the editor to go to the Z3 GitHub discussion page (Fig. 2-C) to look for related discussions on the topic or start a new discussion thread.
(2) After Sofia acquires the basics of logic modeling and Z3, thinking about using Z3 in JavaScript, one of the languages she is most familiar with, she presses "Programming Z3" (Fig. 1-B) to follow tutorials on using Z3 in JavaScript. The tutorials follow the same structure as the SMTLIB tutorial with editable/executable code snippets. She finds the "Dogs, cats and mice" example interesting (Fig. 3), so she first runs the example to inspect its output, and then edits the code to see how the modeling results will update if she changes the quantity thresholds for cats from $cats >= 1$ to $10 <= cats <= 20$ (line 10, Fig. 3); it turns out that such constraints are unsatisfiable (not shown in figure).
(3) In the same editor, she applies the concepts behind the "Dogs, cats and mice" problem to budgeting for grocery shopping (not shown in figure): *Given $30 for purchasing toilet papers, fruits, and snacks, and the constraints that (1) she must buy a pack of toilet papers, which costs $8.99 (2) she wants to buy some fruits and some snacks (3) she cannot spend more on snacks than on fruits (4) she wants to spend all the money, how should she allocate the money?* While writing Z3 JavaScript code for the problem, she explores using `Real` as opposed to `Int` (lines 4-6, Fig. 3) to encode constraints for budgets, since money does not need to be integers. Within seconds, the model came back to her with a solution: spend $10.25 on snacks and $10.75 on fruits.
(4) Sometimes, she wants more programming challenges beyond constructing logic models. She presses "Playground" (Fig. 1-C) to play formula guessing games (Fig. 4), which let her iterate on the logic model based on feedback from the modeling tool to guess the secret model.
(5) Sofia finds the linked materials towards the bottom of the page (Fig. 1-D) useful for her future reference, including slides for learning advanced topics of logic modeling.
(6) Throughout her usage, she finds materials for a specific concept with the built-in search feature (Fig. 1-E).

Using Z3Guide, Sofia studies logic modeling at her own pace, with the ability to (1) read, write, edit, and run Z3 code in a variety of formats, (2) join discussions about materials in Z3Guide, and (3) access other resources of more advanced concepts related to logic modeling, all without setting up a local development environment. Without Z3Guide, she would have needed to rely on web search to

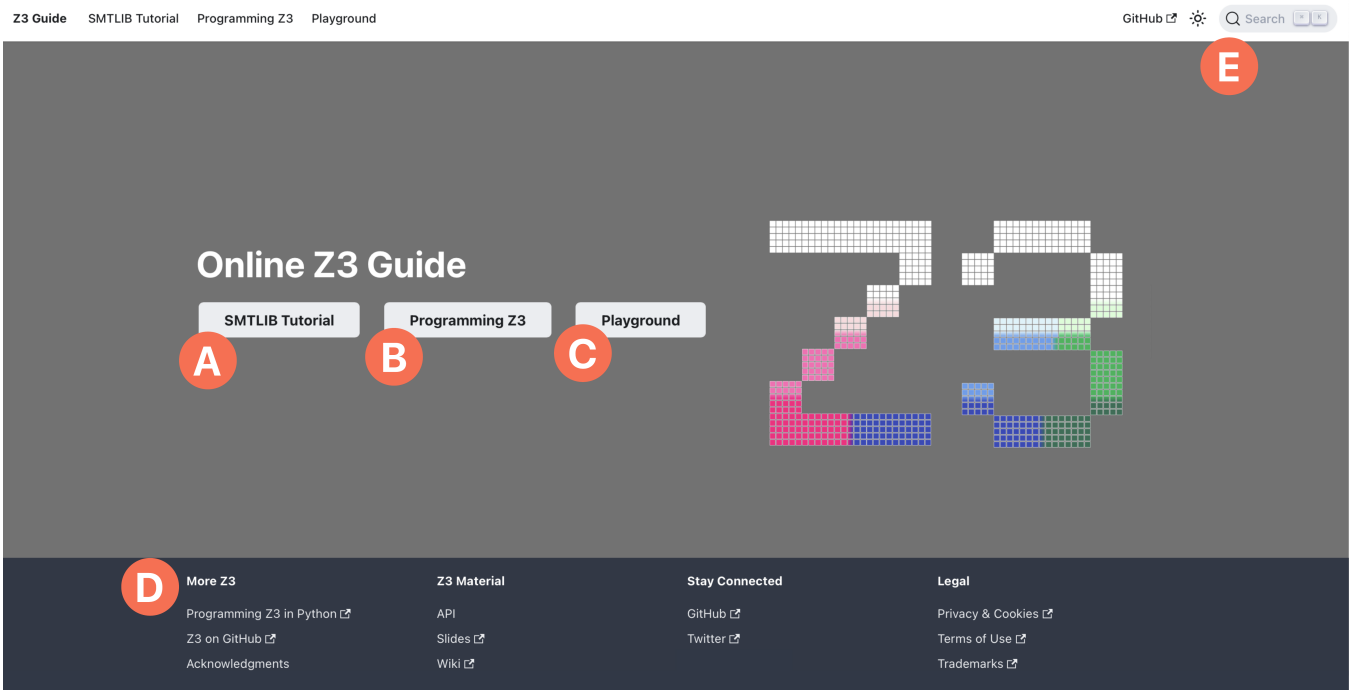

Figure 1: The Z3Guide interface: *SMTLIB Tutorial* (A), upon pressing, shows Fig. 2 that includes tutorial content and editable code examples for logic modeling in SMTLIB format. *Programming Z3* (B), upon pressing, shows logic modeling with Z3 bindings in JavaScript (Fig. 3) and Python. *Playground* (C), upon pressing, shows logic formula guessing games (Fig. 4) and a freeform editor (right half of Fig. 5). It also comes with links to external resources (D) and a built-in search (E).

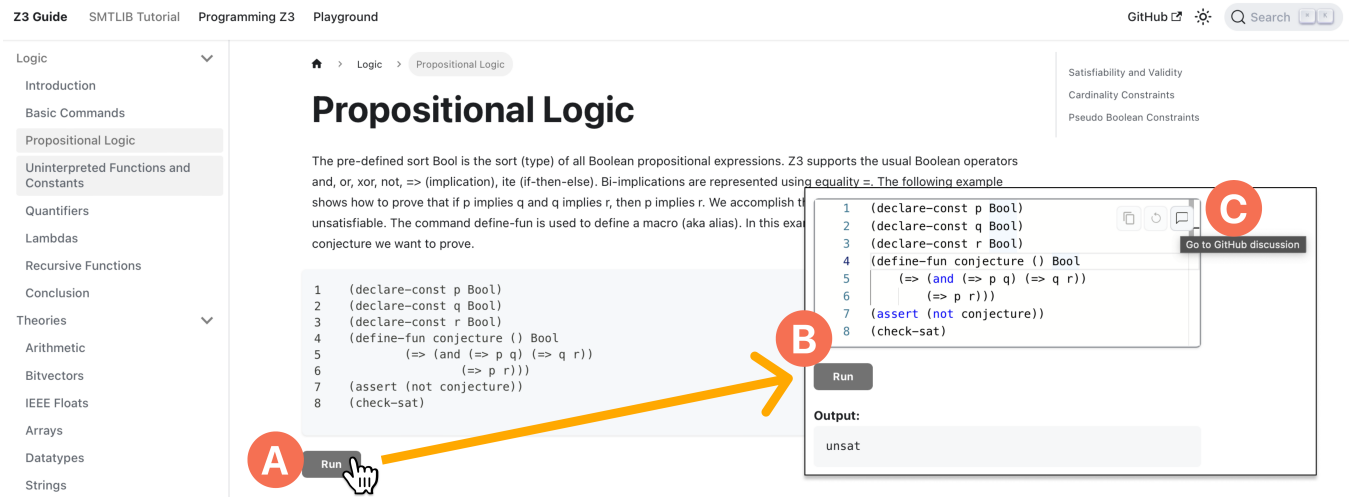

Figure 2: *SMTLIB Tutorial* in Z3Guide discusses basics of logic modeling with textual explanations, practice problems, and interactive code examples in SMTLIB [14] (a syntax many SMT solvers including Z3 uses). (A) Each code example is static at first. (B) The user clicks "Run" to see code output, potentially editing and rerunning the code. (C) Each editor in Z3Guide comes with three buttons at the upper right corner, from left to right: a copy button for its content, a reset button (↺) that reverts the content to original, and a discussion button that by default takes the user to the Z3 discussion on GitHub.

look for tutorials, spend hours setting up multiple local development environments for different language bindings, and potentially sign up for workshops or courses to receive more guidance.

Formal Education. Miles is a Professor specializing in Formal Methods. He is teaching logic modeling for 150 students in the coming school term. This is the second time he has taught the course, and he has been thinking about possible pedagogical improvement.

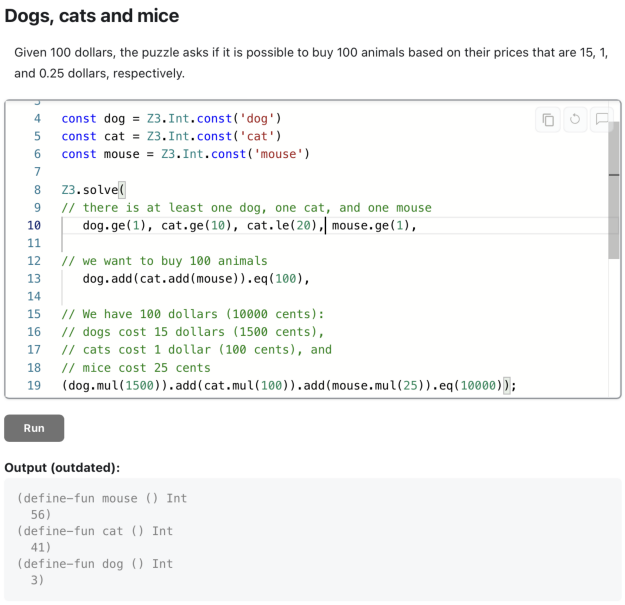

Figure 3: An interactive logic modeling problem solved with Z3 in JavaScript. The *Programming Z3* section of Z3Guide includes many of such Z3 examples as well as tutorials, introducing using Z3 in JavaScript or Python. For every interactive example, if the user presses "Run" to see the output, and then edits the code, the output area fades to prompt the user to rerun the code for an up-to-date output.

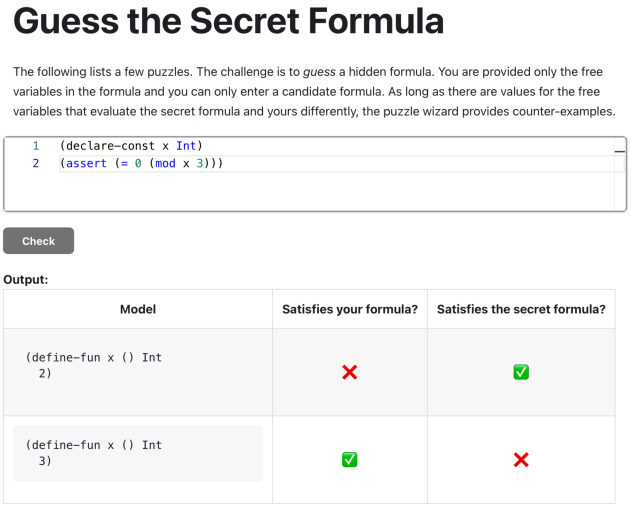

Figure 4: *Playground* in Z3Guide contains formula guessing games. Each game encourages students to create a logic model that matches the secret model behind the game based on the output, which shows instances that satisfy and/or fail the user-specified model.

Miles decides to use Z3Guide to make the lectures more interactive and consolidate course materials. To that end, he adds customized configuration and content to Z3Guide.[2]

(1) In the customized Z3Guide, Miles changes the "GitHub Discussion" button inside its editor to direct students to the GitHub Discussion used by the class, as opposed to the Z3 solver discussion. From there, students could ask the class questions about a particular code snippet or concept in the tutorial.
(2) When preparing for teaching, he refers to slides made by other people in the community for inspiration (Fig. 1-D).
(3) Miles assigns some sections of the SMTLIB tutorial (Fig. 2) as pre-class readings and exercises, including some additional examples he adds on top of the original ones. In each lecture, he discusses key concepts with the class, and, to encourage participation, changes parts of the built-in code snippets and asks students to predict the outcome.
(4) During a lecture, Miles shows on a slide a problem where Santa Claus needs to make presents given some constraints around quantity, cost, and time. Miles asks his students to pair up and model the problem in the freeform editor in Z3Guide (Fig. 5). All 150 students access Z3Guide all at once. Miles later demonstrates solving the problem using the same editor himself.
(5) At the end of each lecture, he spends five minutes solving one formula guessing question (Fig. 4) with the class. His students love brainstorming solutions with one another.

Miles bases his teaching on a customized Z3Guide, aggregating all components needed for the class in one platform: readings, programming environment, and online discussions (GitHub for the class), while promoting student engagement via group work and games. Had Miles not used Z3Guide, he would have needed to use multiple tools for all of his pedagogical needs. In addition, he might have needed to troubleshoot the programming environment configuration for his students or even himself. Most importantly, his customization needs might not have been satisfied without reimplementing some existing solutions from scratch (and knowing how to do so in the first place).

5.2 Providing Easy Access

Z3Guide is web-based and completely client-side with no capacity limit, allowing hundreds of users to use it simultaneously. This is because the Z3 formulas (in SMTLIB and JavaScript formats[3]) are compiled to Web Assembly (WASM) [8] and then executed client side, without going through any external hosts. As such, users can access Z3Guide via a URL anytime, from any web browser that supports WASM, to model Z3 logical formulas and have them run directly on their machines. The client-side implementation thus enables **D1: *Fast Execution***: when the user changes an interactive code block, the code is immediately recomputed in the browser.

Z3Guide also provides **D3: *Editing Support*** for programming—such as syntax highlighting and parentheses autocomplete—and additional shortcuts for copying code, undo-/redo-ing edits, and asking for help, in all editors. When the user hovers over an editor, three buttons appear in the upper right corner (Fig. 2-C): (1) a copy button saves the current content in the editor to the clipboard for

[2]The customization process is not shown in the figures.
[3]While Z3Guide includes Z3 examples in Python, they are read-only due to the lack of client-side compilation support at the time of our implementation.

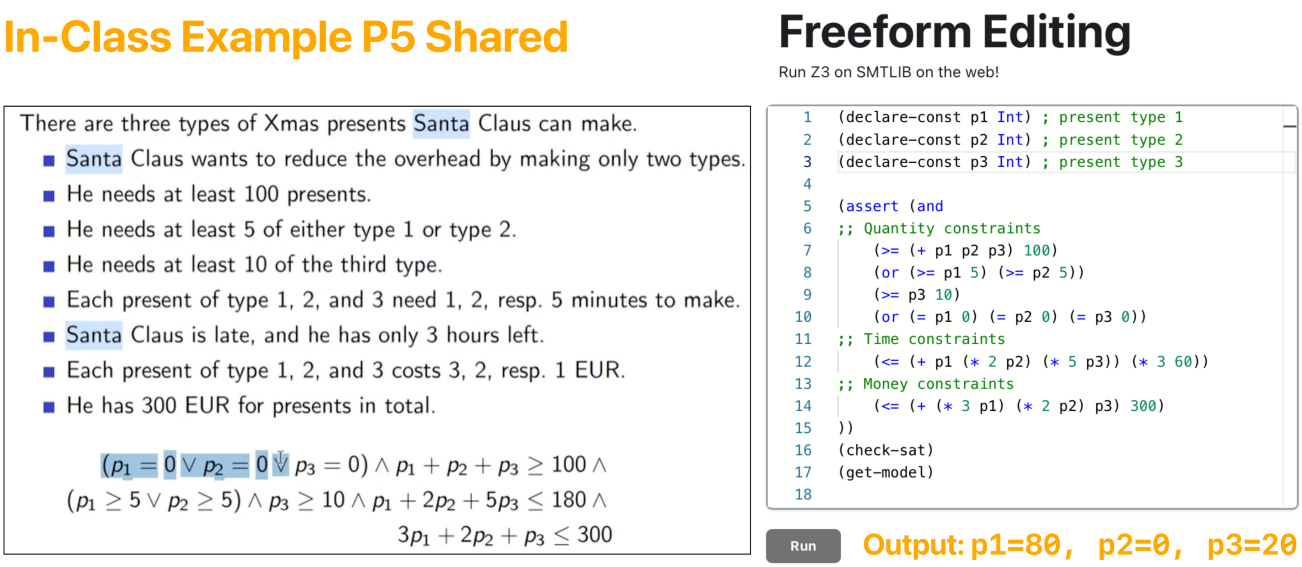

Figure 5: *Playground* in Z3Guide also includes a freeform editor for Z3 in SMTLIB. Left: An example P5 shared in the design iteration study. Right: Solving the example problem in the freeform editor. The model output is summarized in orange text.

potential reuse and sharing (partially **D2** as Z3Guide does not fully address this design goal) (2) a `reset` button resets the editor content to its original state; in case the user accidentally resets the state, the `reset` button (↺) changes to an `undo` button (↺) for three seconds to allow the user to undo the reset action and retain their work (3) a `discussion` button takes the user to some online discussion forum (the GitHub Discussion for Z3 by default) in a new tab for help with content within the guide.

5.3 Supporting Multi-Modal Education

Z3Guide provides a variety of contents. It comes with three main sections, respectively reached via pressing Fig. 1-A to C: *SMTLIB Tutorial* reviews the basics of logic and contains basic- to advanced-level topics of logic modeling with interactive examples in SMTLIB; *Programming Z3* introduces more complex Z3 program examples and API references in JavaScript (executable) and Python (read-only); *Playground* includes a freeform editor for the SMTLIB binding and a series of formula guessing games. In particular, the formula guessing games differ from all other examples throughout Z3Guide because they "reverse" the logic modeling process: while logic modeling builds a model given pre-defined constraints, the formula guessing games require changing the constraints given concrete instances of modeling successes and failures (Fig. 4) to build a satisfying model. Such differences aim to promote a deeper understanding of logic modeling concepts.

With the variety of contents, Z3Guide thus enables several use cases for the classroom. First, via interactive **D4: *Small Examples*** throughout the tutorials, Z3Guide supports *learning-by-doing* [12], enabling students to internalize concepts by editing and running the examples as they go through the tutorial. Second, with editable code blocks throughout the tool and a freeform editor in Playground, Z3Guide not only facilitates students to perform **D5: *Freeform and Exploratory Programming*** but also helps educators easily live-code [46] in front of the class with the students following along. This can all be done interactively, directly within the browser, because of its programming support and client-side execution (Sec. 5.2). Third, the formula guessing games encourage learning via **D6: *Gamification*** and allow educators to run engaging group activities. Fourth, the tutorials can be supplemental materials for a course to support concept previewing and reviewing (**D7**). Lastly, with a button that redirects to some online discussion forum (customizable) in each code example, Z3Guide supports asynchronous **D8: *Question Asking*** during and after class.

Finally, and more importantly, all Z3Guide contents are self-contained, allowing students to learn the topics at their own pace.

5.4 Allowing for Extensions

Z3Guide achieves **D9: *Resources Sharing*** by both providing links to related Z3 resources (Fig. 1-D) and allowing contributions to its content. Z3Guide is open-source on GitHub (**D10**).[4] All of the tutorial contents are written in Markdown. As such, anyone can fork the repository, edit existing Markdown files for the content or add new ones, and submit a pull request for content contribution. This allows educators to add their teaching materials to Z3Guide for sharing with all users of the tool.

Furthermore, anyone can easily extend Z3Guide (**D10**) as tool contributions or for their own use. Part of its implementation enables easy extensions (which we detail in Sec. 6): (1) All contents of Z3Guide are written in Markdown files because Z3Guide uses docusaurus [3], a static site generator, to compile the Markdown files to HTML and create its webpage (2) A single JSON file for *configuration of languages* specifies editing and execution support for all interactive code examples. As such, users can add interactive

[4] https://github.com/microsoft/z3guide

code examples for other Z3 bindings or other logic modeling tools by extending both the content in Markdown and the JSON configuration of languages. For example, if one wants to add interactive examples for Dafny [39] (another logic modeling tool), they can (1) create Dafny code blocks in Markdown starting with ```` ```dafny ```` (2) in the language configuration JSON file, declare a new language named `dafny` and configure support for it, such as syntax highlighting, path to the runtime for computing outputs, and destination for the Discussion button in the editor. To enable these extensions/-customizations for Z3Guide, users can either contribute directly to the Z3Guide repository via pull requests or deploy their fork to their own domains via mechanisms such as GitHub Pages.

6 IMPLEMENTATION CHALLENGES

When implementing Z3Guide, we realized that existing technologies could not fully address two categories of our design guidelines: **Providing Easy Access** and **Allowing for Extensions**. This section describes the technical challenges we resolved to meet the design guidelines. Solutions to these challenges are not specific to Z3Guide and can apply to other interactive programming environments written entirely in Markdown.

6.1 Interactive Code Examples in Markdown

Our design guidelines state that anyone, particularly those who teach logic modeling, should be able to easily extend Z3Guide (**D9-D10**), including (1) writing new Z3 examples that will be rendered as interactive code snippets and (2) adding examples in other language bindings for Z3 or other logic modeling tools, with proper syntax highlighting and runtime for execution. However, all Z3Guide contents are written in Markdown, which by default renders static content only, not editable content, and the syntax highlighting for code blocks in Markdown only supports a limited set of languages.

We contribute three techniques to support the intended use case: (1) a customized editor (as shown in Fig. 3) that provides proper syntax highlighting and autocomplete for a Z3 code example, the buttons for the copy, reset, and discussion actions, an output area that shows the result of execution, and a "Run" button that triggers computing the output; (2) a Markdown rendering plugin[5] that replaces the default read-only code blocks with interactive code blocks with our customized editor and computes the outputs of the code blocks when applicable; (3) a JSON file—configuration of the languages (denoted as `lang-config` below)—that specifies code blocks of which languages should be replaced with the customized editor, and additional language-specific information for the rendering and output computation.

We implemented these techniques for the Z3-SMTLIB and Z3-JavaScript examples in Z3Guide. Listing 1 shows part of the configuration for the Z3-SMTLIB code blocks in `lang-config`. Using `lang-config`, for each code block in Markdown, the plugin properly specifies the syntax highlighting (line 4), the process for computing the code (lines 10-11) during build (Sec. 6.2) or at runtime (associated with the "Run" button), the destination for the discussion button (line 14), etc. for each instance of the customized editor. The plugin further computes the output of the code (more in Sec. 6.2) if the `buildConfig` property is specified (lines 6-13, Listing 1).

[5]https://github.com/remarkjs/remark

Listing 1: Configuration of Z3-SMTLIB blocks

```
1  {
2    name: 'Z3', // your language name
3    label: 'z3', // label for markdown code blocks
4    highlight: 'clojure', // prism-supported syntax highlighting
5    showLineNumbers: true, // whether to show line numbers
6    buildConfig: {
7      timeout: 30000, // execution timeout of each snippet in ms
8      npmPackage: 'z3-solver',
9        // npm package name for the language runtime, if any
10     processToExecute: './src/remark/run-z3-smtlib.js',
11       // process for computing code outputs
12     ...
13   },
14   githubRepo: 'Z3Prover/z3', // discussion button destination
15   githubDiscussion: true // whether to show the discussion btn
16 }
```

Note that while Z3Guide is 100% client-side for Z3, users could still implement output computation that relies on external servers when extending Z3Guide with examples in other languages. They could write a script that handles the client-server communication and set `processToExecute` (line 10, Listing 1) to the script.

Because our approach combines Markdown rendering and configuration for specific language support, it is intended to generalize to other Markdown-based interactive environments. For example, tutorials for MSAGL[6] are built with the framework behind Z3Guide.

6.2 Computing Tutorial Examples

Our design guidelines also state that anyone should be able to easily access the tool and its interactive code examples (**D1-D3**). Since the static site generator used in Z3Guide, docusaurus [3], does not handle Z3 execution directly, we initially attached a Z3-to-WASM compiler to Z3Guide to compute Z3 examples client-side when the user runs the examples. However, larger Z3 examples might still take long to be computed at runtime, and the user's computing environment can affect the client-side computation efficiency. We deem these large examples essential as they could be important for learning, but Z3Guide should still deliver an interactive experience that responds to the user in seconds.

We implemented three approaches to ensure an interactive and responsive experience with Z3Guide in as many cases as possible. First, we precompute all code examples in Z3Guide when building from the source (prior to deployment), including the large examples that might be slow to compute. Second, we cache the precomputed outputs by writing each code output to a file on the disk and naming it with a hash of the code, the language runtime name, and the runtime version. This way, we avoid unnecessary re-computations during the build, only recomputing the output of an example if it is new or the language runtime is updated. Third, we configure the timeout of the runtime execution (Listing 1) to be 30000 milliseconds (30 seconds): if the user edits a large example and recomputes it at runtime, the execution will timeout after 30 seconds and prevent them from waiting too long. Most examples in the tutorial will not reach this timeout threshold, and if the user encounters an unexpected execution timeout or wants to get some long example to be computed, they can ask for help in the discussion.

With this approach, failures in precomputing code examples at build time will cause the build to fail and thus help contributors

[6]https://microsoft.github.io/msagljs/

catch examples with unintentional errors. However, contributors might want to create examples with *intentional* errors for educational purposes. They could skip computing these examples at the build by adding a `no-build` flag when creating the Markdown code blocks (e.g., ```` ```z3 no-build ````). Our Markdown rendering plugin (Sec. 6.1) will thus skip the computation of examples with this flag.

7 STUDENTS USING Z3GUIDE

The design of Z3Guide was driven by needs and feedback from the *teachers*. To understand how *students* perceive Z3Guide and gauge possible improvements for the tool, we conducted a free online workshop where participants used Z3Guide to learn logic modeling and Z3. We advertised the workshop through mailing lists of our institutions and on social media, eventually getting 112 attendees. We informed them of the workshop agenda and its research purpose and asked for consent to data collection during registration. Only 21 attendees gave consent, and per our IRB protocol, we only used their survey responses to report findings.

Participants. We recruited 21 adult participants (4 women, 16 men, and 1 unknown) from 11 countries and regions. All participants are *novices* to logic modeling and Z3: 19 participants had no experience in logic modeling or Z3, and two participants self-reported minimal prior experience. Four participants were pursuing a Bachelor's degree in Computer Science, while the rest were full-time working professionals in computing-related fields. All participants joined the workshop because of interest in the topic and/or using the technique in their work. Six belonged to the 18-24 age group, and two were above 60.

Procedure. We conducted the workshop via web conferencing. The workshop lasted three hours: two 75-minute tutorials with a 15-minute break in between, and 15 minutes at the end for a post-workshop survey. The first tutorial covered the basics of logic modeling with Z3 in the SMTLIB format following parts of the SMTLIB Tutorial in Z3Guide (Fig. 2) and the formula guessing games (Fig. 4), while the second introduced using the Z3 solver API in JavaScript and Python for solving realistic constraint-satisfaction problems (e.g., Fig. 3). Only Z3Guide was used in the workshop, with no other supplementary materials, although participants were free to search for related content on the internet. All workshop attendees, including those excluded from the analysis, used Z3Guide simultaneously while following the tutorials and programming within the environment. The last author led the workshop, lecturing and facilitating discussions and exercises. Workshop participants interacted with one another by unmuting themselves in the web conference or messaging within the chat. Three authors helped moderate the workshop, including admitting attendees, monitoring the chat, and troubleshooting Z3Guide when necessary.

Post-Workshop Survey. At the end of the workshop, participants filled out a survey reflecting on their experience using Z3Guide. The survey consisted of five Likert-scale rating questions about the programming experience within Z3Guide, its overall user experience, and whether participants would recommend the tool to other similar users; Table 2 summarizes the statements used in the questions. The survey also included three short response questions asking the participants about features they liked, disliked, and wished to be available in Z3Guide. In addition, participants provided demographic information in the survey.

Data Collection and Analysis. We monitored comments in the chat related to issues of Z3Guide, though we did not record any. In addition, we collected post-workshop survey responses, which included demographics, Likert-scale ratings, and open-ended comments on Z3Guide. The first author coded the comments using open coding [17]. Using the same approach in Sec. 3.1, the first two authors computed the inter-coder agreement using 25% of the data and achieved a 92% agreement.

Limitations. The deployment only evaluated Z3Guide in one aspect—how it could affect students' learning experience—while there are many other aspects of Z3Guide and the design guidelines behind to be more thoroughly evaluated, such as its impact on learning outcomes, the teachers' experience, and the generalizability of its implementation to other educational tools for logic modeling. Our evaluation is also limited to its sample size, with only a fifth of the 112 attendees giving consent to using their post-workshop survey responses. As such, the deployment should be viewed as an early step towards understanding the use of Z3Guide and its design guidelines, future summative assessments, and future tool designs in related domains.

7.1 Post-Workshop Survey Results

The workshop, with more than 100 people using Z3Guide simultaneously, went smoothly with no reported technical issues. Below we report findings from 21 post-workshop survey responses.

Participants Deemed Z3Guide Easy to Use. On average, participants rated 4.43 (sd = .60) [7] out of 5 (Strongly Agree) to the statement "Z3Guide was easy to use". Participants appreciated the ease of accessing everything used in the workshop within one tool, from tutorial materials to the programming environment, and that everything *"loaded up instantaneously."* They considered the content to be *"very structured,"* the examples to be *"clear,"* and *"there is a connection between most topics in the tutorial."* They found the tool to have promising usability *"considering students in different levels."* They also strongly agreed that "programming in Z3Guide was easy" (4.76 out of 5, sd = .54). They believed that it was easy to create and modify code examples to see *"different [Z3] features in context."* In addition, it is *"a great [tool] for playing with Z3,"* giving users *"the ability to try out [Z3] without installing [it] locally."*

Participants Found Z3Guide Helpful for Learning. Participants gave an average rating of 4.57 out of 5 (sd = .60) that "Z3Guide was helpful for learning," and an average of 4.71 out of 5 (sd = .46) that they "would recommend Z3Guide to other students." Participants commented that the inclusion and depth of *"a lot of important concepts [about logic]"* could *"apply to everyone,"* whether or not they intend to use Z3 for logic modeling, which provide *"theoretical reminders all along the way"* while learning new concepts. One participant noted that the tutorials went a bit too fast for them, but: *"now I can read the pages in my own pace, which is really great."* Programming in Z3Guide was considered as particularly helpful for learning ($avg.$ = 4.76, sd = .54): seeing the output of each snippet helped the participants refine *"[their]*

[7] sd - standard deviation.

Table 2: Likert-scale ratings on Z3Guide in the post-workshop survey. 1 - "Strongly Disagree", 5 - "Strongly Agree". Avg. - Average, Mdn. - Median, Dist. - Distribution.

	Avg.	Mdn.	Dist.
Z3Guide was easy to use.	4.43	4.0	
Programming in Z3Guide was easy.	4.76	5.0	
Z3Guide was helpful for learning.	4.57	5.0	
Programming in Z3Guide was helpful for learning.	4.76	5.0	
Would recommend Z3Guide to other students.	4.71	5.0	

understanding of the material and the examples." Furthermore, participants liked the editable examples throughout the guide as they were *"easy to run"*, with fast and stable execution, and encouraged learning-by-doing.

Participants Wanted More Programming Support and Engagement. Several participants pointed out issues with the programming experience in Z3Guide. One participant, new to Z3, was frustrated about not being able to *"get the syntax right"* as the editor had limited support in autocompleting keywords. Some participants hoped for more readable, *"pretty-printed"* modeling output messages that are usually long in logic modeling. Additionally, some participants hoped for more learning engagement. One of the highlights of Z3Guide is logic puzzles, with one dedicated section of formula guessing games in its Playground and realistic puzzles like Sudoku throughout the tutorial. Participants deemed the puzzles important for an engaging self-paced learning experience of logic modeling but expected more such *"guided modeling problem[s]."* They further yearned for the ability to share their work-in-progress logic modeling code with others for help, such as *"via permalinks,"* rather than having to manually copy and paste the examples.

8 DISCUSSION AND FUTURE WORK

We reflect on our design guidelines (Sec. 8.1) and the last author's experience using Z3Guide (Sec. 8.2) to discuss the implications of our work and future design opportunities.

8.1 Reflections on Design Guidelines

Student feedback from the formative workshop suggested three design requirements missing in Z3Guide: one was **D2: Code Sharing** that we did not implement, and the other two were possible new guidelines. Further assessments of **D6: Gamification** also remain: while students and teachers showed a positive attitude, prior work reveals unresolved debates around its impact on student learning.

Sharing of Learning Progress. The post-workshop survey shows that students hoped to share learning progress, particularly via permalinks. Indeed, code sharing (**D2**) is the only design guideline we did not fully address despite the request of teachers. We did not implement code sharing in Z3Guide due to security concerns: a permalink includes complex parameters, such as code to be parsed and evaluated, which must be done with care to prevent the possibility of malicious attacks especially when there is JavaScript execution of user-specified code using `eval()`, in our case the Z3 code in JavaScript format. In future work, we aim to improve how JavaScript code is parsed and executed to better support user needs.

Full Programming Support Even in a Lightweight Environment. In the post-workshop survey, students expected Z3Guide to incorporate similar programming support to existing IDEs, such as keyword autocomplete and API discovery. Such expectations might relate to the fact that most participants were casual learners experienced in computing, possibly with prior exposure to professional programming tools. Design tradeoffs remain between enabling a programming experience with full IDE support and delivering a lightweight educational environment that covers other experiences than programming. Our work prioritizes the latter but continues to seek better editing support in future iterations.

Comprehension Aids for Logic Modeling Outputs. Both P4 and P6 in the design exploration and students in the workshop wanted better aids for understanding the output of logic modeling. There is a long line of research in the output representations of algorithmic programs that facilitate code comprehension. Even if logic modeling differs from algorithmic programming, users share a similar desire to understand code behavior via some aids. Designing better representations for logic modeling outputs, however, has been an open problem [28] and warrants a separate investigation. Future work could leverage program visualization techniques such as in-situ visualizations [32, 33] to better connect the non-algorithmic logic-modeling programs with their outputs.

Understanding of the Impact of Gamification on Student Learning. Both our design exploration (Sec. 4) and post-workshop survey reveal a positive attitude towards the educational games in Z3Guide, while there have been debates on the use of gamification in education [18, 21]. Future work should evaluate Z3Guide in formal educational settings, particularly its gamification, to understand its impact on student learning.

8.2 Additional Roles for Interactive Textbooks

From the last author's year-long experience in using Z3Guide to answer questions about SMT solvers in an online forum, we identify two previously under-explored roles beyond an educational environment for Z3Guide and similar tools with interactive textbooks.

Z3Guide As A Reference Guide. Z3Guide is very suitable for a quick reference for logic modeling and Z3 because of its interactive examples. Indeed, the author has been referring to Z3Guide when answering user questions and demonstrating Z3 features. To this end, they added new examples to Z3Guide when there were no suitable ones or when new Z3 features came out. Such benefits are not special to Z3Guide or logic modeling; in fact, any extensible interactive textbooks for programming tools would be suitable for demonstrating the use and features of the tools.

Z3Guide As A Personalized Tutor. While one can obtain answers to many questions of logic modeling or Z3 by having a human expert direct them to respective sections in Z3Guide or searching within Z3Guide themselves, Z3Guide itself cannot answer "how-to" questions sufficiently. For example, to answer *"Is it possible to dump an SMT2 file after simplifications? If so, how?"*, one needs to (1) break the solution down to steps and (2) connect each step to sections in Z3Guide. Currently, Z3Guide directly assists with step (2) via built-in search, but a human expert is necessary for step (1). Large language models (LLMs) are, on the other hand, very suitable for automating constructing the how-to solution with decomposed

steps [50] and hence reducing the barriers to obtaining answers. We imagine Z3Guide and similar interactive textbooks, which already engage students more than static e-textbooks [44], to become even more active in providing quick answers and feedback to students with the help of LLMs. Future interactive textbooks could support learning new concepts and receiving timely feedback on learning via integration with LLMs.

9 CONCLUSION

To understand the pedagogical needs for educational tools for logic modeling, we conducted a needfinding interview and a design iteration with teachers of the subject. Driven by 10 design guidelines from the design exploration, we developed Z3Guide, a scalable, student-centered, and extensible educational environment for logic modeling with the Z3 SMT solver. We ran a workshop with more than 100 students using Z3Guide to learn about logic modeling with Z3, and a post-workshop survey ($N = 21$) shows that students deemed Z3Guide easy to use and helpful for learning. Our evaluation is limited in its formative nature; however, the lessons we learned from our design process, the workshop, and our year-long experience using the tool imply several directions for improving Z3Guide and further evaluations. We hope that our findings can inform future educational tools and their design guidelines for logic modeling and programming education.

ACKNOWLEDGMENTS

We thank all our study participants for their invaluable input.

REFERENCES

[1] 2021. rise4fun.com web page is down · Z3Prover/z3 · Discussion #5473. https://github.com/Z3Prover/z3/discussions/5473

[2] 2023. Alloy4fun. http://alloy4fun.inesctec.pt

[3] 2023. Build optimized websites quickly, focus on your content: Docusaurus. https://docusaurus.io/

[4] 2023. CS50. https://cs50.dev/

[5] 2023. Digital Learning Platform. https://edstem.org/

[6] 2023. GitHub Codespaces. https://github.com/features/codespaces

[7] 2023. Stepik is an educational marketplace and online course editor. https://stepik.org/

[8] 2023. Web Assembly. https://webassembly.org/

[9] 2024. Online Guide for SAGE. https://doc.sagemath.org/html/en/tutorial

[10] 2024. Sage Tutorial. https://doc.sagemath.org/html/en/tutorial/index.html

[11] Deborah Allen and Kimberly Tanner. 2005. Infusing active learning into the large-enrollment biology class: seven strategies, from the simple to complex. *Cell biology education* 4, 4 (2005), 262–268.

[12] Yuichiro Anzai and Herbert A. Simon. 1979. The theory of learning by doing. *Psychological Review* 86, 2 (1979), 124. https://doi.org/10.1037/0033-295X.86.2.124 Publisher: US: American Psychological Association.

[13] Thomas Ball, Peli de Halleux, Nikhil Swamy, and Daan Leijen. 2013. Increasing Human-Tool Interaction via the Web. In *Proceedings of the 11th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering* (Seattle, Washington) *(PASTE '13)*. Association for Computing Machinery, New York, NY, USA, 49–52. https://doi.org/10.1145/2462029.2462031

[14] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. [n. d.]. The Satisfiability Modulo Theories Library (SMT-LIB). https://www.SMT-LIB.org

[15] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. 2010. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, Vol. 13. 14.

[16] Nikolaj Bjørner, Clemens Eisenhofer, and Laura Kovács. 2023. Satisfiability Modulo Custom Theories in Z3. In *Verification, Model Checking, and Abstract Interpretation: 24th International Conference, VMCAI 2023, Boston, MA, USA, January 16–17, 2023, Proceedings*. Springer, 91–105.

[17] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa Publisher: Routledge _eprint: https://www.tandfonline.com/doi/pdf/10.1191/1478088706qp063oa.

[18] Ilaria Caponetto, Jeffrey Earp, and Michela Ott. 2014. Gamification and education: A literature review. In *European Conference on Games Based Learning*, Vol. 1. Academic Conferences International Limited, 50.

[19] Jennifer Davis, Matthew Clark, Darren Cofer, Aaron Fifarek, Jacob Hinchman, Jonathan Hoffman, Brian Hulbert, Steven Miller, Lucas Wagner, and Rockwell Collins. 2013. Study on the Barriers to the Industrial Adoption of Formal Methods. *Lecture Notes in Computer Science* 8187, 63–77. https://doi.org/10.1007/978-3-642-41010-9_5

[20] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[21] Edward L Deci, Richard Koestner, and Richard M Ryan. 1999. A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. *Psychological bulletin* 125, 6 (1999), 627.

[22] Barbara J Ericson and Bradley N Miller. 2020. Runestone: A platform for free, online, and interactive ebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1012–1018.

[23] Göran Folkestad. 2006. Formal and informal learning situations or practices vs formal and informal ways of learning. *British journal of music education* 23, 2 (2006), 135–145.

[24] Eugene C Freuder. 1997. In pursuit of the holy grail. *Constraints* 2 (1997), 57–61.

[25] Gaetano Geck, Artur Ljulin, Sebastian Peter, Jonas Schmidt, Fabian Vehlken, and Thomas Zeume. 2018. Introduction to Iltis: an interactive, web-based system for teaching logic. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. Association for Computing Machinery, New York, NY, USA, 141–146. https://doi.org/10.1145/3197091.3197095

[26] Gaetano Geck, Christine Quenkert, Marko Schmellenkamp, Jonas Schmidt, Felix Tschirbs, Fabian Vehlken, and Thomas Zeume. 2022. Iltis: Learning Logic in the Web. arXiv:2105.05763 [cs.LO]

[27] Matt Godbolt. 2023. Compiler Explorer. https://godbolt.org/

[28] F. Goualard and F. Benhamou. 1999. A visualization tool for constraint program debugging. In *14th IEEE International Conference on Automated Software Engineering*. 110–117. https://doi.org/10.1109/ASE.1999.802142

[29] Jun Gu, Paul W Purdom, John Franco, and Benjamin W Wah. 1996. Algorithms for the satisfiability (SAT) problem: A survey. *Satisfiability problem: Theory and applications* 35 (1996), 19–152.

[30] Philip Guo. 2021. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. ACM, Virtual Event USA, 1235–1251. https://doi.org/10.1145/3472749.3474819

[31] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 579–584.

[32] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2018. Augmenting Code with In Situ Visualizations to Aid Program Understanding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3174106

[33] Ruanqianqian Lisa Huang, Philip J. Guo, and Sorin Lerner. 2024. UNFOLD: Enabling Live Programming for Debugging GUI Applications. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 306–316. https://doi.org/10.1109/VL/HCC60511.2024.00041

[34] Norbert Hundeshagen, Martin Lange, and Georg Siebert. 2021. DiMo – Discrete Modelling Using Propositional Logic. In *Theory and Applications of Satisfiability Testing – SAT 2021*, Chu-Min Li and Felip Manyà (Eds.). Springer International Publishing, Cham, 242–250.

[35] Daniel Jackson. 2019. Alloy: a language and tool for exploring software designs. *Commun. ACM* 62, 9 (2019), 66–76.

[36] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, New York, NY, USA.

[37] Eduard Kamburjan and Lukas Grätz. 2021. Increasing Engagement with Interactive Visualization: Formal Methods as Serious Games. In *Formal Methods Teaching*, João F. Ferreira, Alexandra Mendes, and Claudio Menghi (Eds.). Springer International Publishing, Cham, 43–59.

[38] Sam Lau, Sean Kross, Eugene Wu, and Philip J. Guo. 2023. Teaching Data Science by Visualizing Data Table Transformations: Pandas Tutor for Python, Tidy Data Tutor for R, and SQL Tutor. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research* (Seattle, WA, USA) *(DataEd '23)*. Association for Computing Machinery, New York, NY, USA, 50–55. https://doi.org/10.1145/3596673.3596972

[39] K Rustan M Leino. 2010. Dafny: An automatic program verifier for functional correctness. In *International conference on logic for programming artificial intelligence and reasoning*. Springer, 348–370.

[40] Kevin Leo and Guido Tack. 2017. Debugging unsatisfiable constraint models. In *Integration of AI and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings 14*. Springer, 77–93.

[41] Clayton Lewis and Cathleen Wharton. 1997. Cognitive walkthroughs. In *Handbook of human-computer interaction*. Elsevier, 717–732.

[42] Nuno Macedo, Alcino Cunha, José Pereira, Renato Carvalho, Ricardo Silva, Ana C.R. Paiva, Miguel Sozinho Ramalho, and Daniel Silva. 2021. Experiences on teaching alloy with an automated assessment platform. *Science of Computer Programming* 211 (2021), 102690. https://doi.org/10.1016/j.scico.2021.102690

[43] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. 2017. *Software Foundations*. Electronic textbook. http://www.cis.upenn.edu/~bcpierce/sf

[44] Kerttu Pollari-Malmi, Julio Guerra, Peter Brusilovsky, Lauri Malmi, and Teemu Sirkiä. 2017. On the value of using an interactive electronic textbook in an introductory programming course. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*. 168–172.

[45] Tobias Runge, Tabea Bordis, Thomas Thüm, and Ina Schaefer. 2021. Teaching correctness-by-construction and post-hoc verification–the online experience. In *Formal Methods Teaching: 4th International Workshop and Tutorial, FMTea 2021, Virtual Event, November 21, 2021, Proceedings 4*. Springer, 101–116.

[46] Ana Selvaraj, Eda Zhang, Leo Porter, and Adalbert Gerald Soosai Raj. 2021. Live Coding: A Review of the Literature. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. ACM, Virtual Event Germany, 164–170. https://doi.org/10.1145/3430665.3456382

[47] Slack. 2023. Slack is your productivity platform. https://slack.com

[48] Moin Syed and Sarah C. Nelson. 2015. Guidelines for Establishing Reliability When Coding Narrative Data. *Emerging Adulthood* 3, 6 (Dec. 2015), 375–387. https://doi.org/10.1177/2167696815587648 Publisher: SAGE Publications Inc.

[49] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods: Practice and experience. *ACM computing surveys (CSUR)* 41, 4 (2009), 1–36.

[50] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601 [cs.CL]

[51] Dennis Yurichev. 2023. SAT/SMT by example. https://sat-smt.codes/