A Misconception-Driven Adaptive Tutor for Linear Temporal Logic

Siddhartha Prasad[1][0000−0001−7936−8147], Ben Greenman[2][0000−0001−7078−9287], Tim Nelson[1][0000−0002−9377−9943], and Shriram Krishnamurthi[1][0000−0001−5184−1975]

[1] Brown University, USA
[2] University of Utah, USA

Abstract. Linear Temporal Logic (LTL) is used widely in verification, planning, and more. Unfortunately, users often struggle to learn it. To improve their learning, they need drill, instruction, and adaptation to their strengths and weaknesses. Furthermore, this should fit into whatever learning process they are already part of (such as a course).
In response, we have built a *misconception-based* automated tutoring system. It assumes learners have a basic understanding of logic, and focuses on their understanding of LTL operators. Crucially, it takes advantage of multiple years of research (by our team, with collaborators) into misconceptions about LTL amongst both novices and experts.
The tutor generates questions using these known learner misconceptions; this enables the tutor to determine which concepts learners are strong and weak on. When learners get a question wrong, they are offered immediate feedback in terms of the concrete error they made. If they consistently demonstrate similar errors, the tool offers them feedback in terms of more general misconceptions, and tailors subsequent question sets to exercise those misconceptions.
The tool is hosted for free on-line, is available open source for self-hosting, and offers instructor-friendly features.

Keywords: LTL, misconceptions, adaptive tutor

1 Introduction

Linear Temporal Logic is a cornerstone of verification [62], and is also used for synthesis [3,4,9,12,16,44,48,58,66], robotics [6,7,11,24,32,34,35,40,55,63,65], embedded systems [10,59], business processes [1,14,17,18,43], and more [46,60].

Since an incorrect specification can cause bugs to go undetected or derail a system's functionality, it is critical that users (from students to professionals) can write and understand LTL specifications correctly. This has spurred a growing body of research focused on improving the process of authoring and interpreting LTL specifications. These efforts include tools designed to represent specifications in terms of alternate formalisms (e.g., Büchi automata [8,20]), scaffold

common specification patterns [23,57], explain formulae via visualization [36,64], and generate specifications from natural language [15,25].

Another way to tackle the problem is to better understand *what* aspects of LTL are difficult. For several years [28,31], with collaborators, we have focused on this question. Our findings have resulted in a catalog of common misconceptions, and these have been distilled into multiple-choice question/answer sets.

However, simply deploying a question/answer sheet as, say, an electronic survey is not enough. To learn well, learners need lots of examples and periodic drill (even refreshers). When they make mistakes, they need feedback to understand what they got wrong. This feedback should occur at two levels. On individual questions, learners may have made a mis*take*. Though they need concrete feedback (e.g., showing traces), the error could also be due to a lack of attention or even a slip of the finger. If, however, they make the same kind of error consistently, they may have a mis*conception*; addressing that requires correcting conceptual knowledge. In turn, they need additional questions that target their errors to confirm that they have internalized the feedback.

We have operationalized all of the above into an on-line, adaptive *tutor*, the LTL Tutor. It is hosted for free online:

`https://ltl-tutor.xyz`

To be privacy-protecting, the tutor does not gather any identifying information. Educators can create a "course" and get aggregated data of how students in it are performing. For those who have additional data privacy concerns, the software is also available open source and designed for self-hosting. The tutor also supports both the classical LTL syntax (used in this paper) as well as variants used by recent tools [13,45] that use keywords (e.g., `always`, `after`).

Some tutoring systems are designed to teach a topic from scratch. As educators, we recognize this can be very disruptive because it assumes a specific context, preparation, amount of available time, and so on. Instead, the LTL Tutor is designed to be a *companion* that complements whatever pedagogy is already in use, rather than a substitute. We assume students have a basic grounding in formal logic, and may have heard a lecture or two about LTL in the instructor's preferred style. What the LTL Tutor does is save the instructor from having to provide *drill*, feedback, and corrections; and it leverages our extensive catalog of LTL difficulty without the instructor needing to learn it in depth themselves. Effectively, the LTL Tutor tries to learn, and then correct, the *latent* conceptual model of LTL that the student has in their mind—however it is obtained.

Finally, we note the increased interest in LTL in industrial settings. It can be especially difficult for industrial practitioners to get assistance the way a student in a course can from instructors and teaching assistants. Thus, the LTL Tutor should be of particular value to practitioners.

This paper describes the design and implementation of the LTL Tutor. After providing a high-level overview (section 3), it especially focuses on two aspects:

- A misconception driven process for generating novel question sets tailored to individual learners (section 4).

– Mechanisms designed to provide learners with insight both into the questions they get wrong and the underlying misconceptions that may be driving these errors (section 5).

2 Related Work

Our work is inspired by the seminal work on *concept inventories* [33] from physics education. A concept inventory is a collection of multiple-choice questions where each wrong answer (often called a *distractor*) is not merely wrong, but corresponds to a *specific misconception*. Thus, if students choose a certain distractor, the instructor can be confident about what the student's confusion is. Our prior work [28,29,30,31] takes steps toward such an instrument for LTL (and introduces a catalog of misconceptions), which this work leverages to make generative.

To make it generative, we need a way to not only create new problems but also create misconception-based distractors. While mutation testing [2] is appealing here, the mutants created may be trivial, redundant [53], or even functionally identical to the original [42]. We thus draw inspiration from Prasad et al. [50]'s work on "conceptual" mutation to address these problems. Their work (not for LTL!), however, is only partially automated and requires significant expert intervention. A key technical contribution of this work (section 4) is to perform conceptual mutation in a completely automated way.

When learners make conceptual errors, we have to provide high-level feedback (section 5.2). We draw on the literature of conceptual change [49], specifically using *refutation texts*, which has been found effective in many settings [54]. (The SMoL Tutor for programming language semantics [41] also uses these, and is also driven by misconceptions.)

Our tutor is inspired by vanLehn's two-loop model for tutoring systems [61], in which an inner loop provides immediate feedback and an outer loop selects the next task. The LTL Tutor builds on this framework, extending the outer loop to also target misconceptions (fig. 5).

We are also inspired by works on cognitive tutoring [5,47,56], which capture how an expert would solve a problem and try to get learners to mimic that approach. This approach is too resource-intensive for our lightweight setting, as it requires extensive effort to model and encode expert problem-solving methods.

Finally, we describe existing tutors for formal logic. These systems rely on hand-crafted questions or generate questions without a guiding principle (akin to conventional mutation testing). In contrast, the LTL Tutor stands out by generating novel questions based on an inventory of conceptual errors.

Iltis is a web-based tutor designed to teach learners about the logical foundations of computer science [26,27]. The system has two primary focuses: allowing instructors to easily construct, compose, and pipeline questions and question sets, and the ability to provide students with instant meaningful feedback and explanations for errors. Unlike the LTL Tutor, this means that Iltis modules are designed to be closely tied to specific courses of study (e.g., a modal logic module [19]).

Lodder et al. have developed logic tutors [37,38,39]. Rather than requiring experts to specify the steps of a solution, the tutors automatically generate authoritative step-by-step proofs for instructor-specified problems. Students are given feedback when their proof steps diverge from the generated authoritative proof. However, this work is (a) proof-, not model-theoretic and (b) not for LTL.

3 The LTL Tutor

A user of the LTL Tutor sees a series of multiple-choice questions. There are two kinds of problems:

English-to-LTL questions ask learners to identify which LTL formula best captures a given English description. Figure 1 shows an example of a question and the feedback for a wrong answer.

Trace Satisfaction questions ask learners to decide whether a temporal trace satisfies a given LTL formula. These questions come in two forms: yes/no questions (fig. 3) in the style of the quizzes we used to build the misconception catalog [29,30], or a multiple-choice variant that asks learners to choose the one satisfying trace from among several possibilities (fig. 2). The LTL Tutor also provides an LTL Stepper (fig. 4), where learners can step through a formula and trace simultaneously to develop a better operational understanding of the language.

The LTL Tutor had about 254 unique users by April 2025, who answered a total of 2261 questions. Of these, 530 (23.44%) answers were incorrect. The wrong answers most commonly corresponded to the following misconceptions:

1. Implicit $\boldsymbol{G}$ (28.68%): Expecting a $\boldsymbol{G}$ operator to apply, even when it was not explicitly present. For example, expecting $\boldsymbol{G}(x \implies y)$ to behave like $\boldsymbol{G}(x \implies (\boldsymbol{G}y))$.
2. Other Implicit (23.58%): Expecting formulae to be stronger than their actual meaning. For example, expecting $\boldsymbol{F}x$ to capture the behavior of $(\neg x)\boldsymbol{U}x$.
3. Bad State Quantification (19.25%): Confusing how "fan out" operators ($\boldsymbol{F}$, $\boldsymbol{G}$, $\boldsymbol{U}$) apply to different states in a trace. For example, expecting $(\boldsymbol{G}x)\boldsymbol{U}y$ to behave like $x\,\boldsymbol{U}y$.

Detailed explanations of these misconceptions are available in [31, Figure 4].

Because of the nature of generated formulas (section 4), when learners make mistakes, we can associate these with known misconceptions. Therefore, over time, the LTL Tutor builds a model of the learner's overall understanding of LTL. Each learner mistake is given a score, with recent mistakes weighted more heavily using a decay function inversely proportional to the time elapsed since the mistake. Starting from a uniform prior, each misconception's relative likelihood is then calculated from the sum of the associated mistake scores. The effect of this is that the predicted likelihood of a learner having a misconception is higher if they have made recent mistakes associated with that misconception.

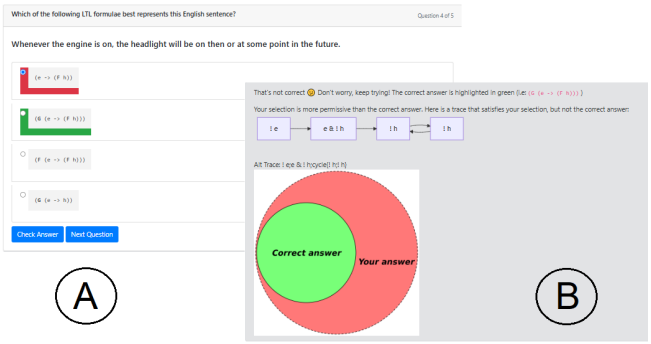

Fig. 1: An English-to-LTL question, with feedback about a learner's mistake. Here (A) the learner selects an incorrect answer, associated with the Implicit $\boldsymbol{G}$ misconception, and is shown (B) a concrete example of why their answer is incorrect, and the relationship between their answer and the correct solution.

Each time the tutor generates a question set, it uses these predictions to inform the kinds of questions it generates. If learners consistently demonstrate a misconception, then the tutor provides *concept-level* feedback, which we discuss in section 5.2. The overall flow is shown in fig. 5.

4 Generating Problems

From the above, we can see that the heart of the tutor lies in generating good sets of related formulas. Both kinds of questions are generated from a *seed formula*. This formula is randomly generated by the `SPOT randltl` tool [20], with the likelihood of each operator's occurence determined by the learner's predicted likelihood of having a related misconception. For example, if the learner has a high likelihood of the Implicit $\boldsymbol{G}$ misconception, the formula generation process will bias the seed formula towards the $\boldsymbol{G}$ operator. All seed formulae are generated to have at most 4 unique propositions, enough to allow for reasonably complex formulas, but low enough to limit extraneous cognitive load.

For English-to-LTL, the LTL Tutor generates a simple English description of the seed formula, which is used as the question prompt. We describe this translation process in section 7. The seed formula represents the correct answer to the question. It is mutated to create distractors, as we describe below.

For trace satisfaction, the seed formula serves as the question prompt. Traces accepted by the seed formula and its mutants (also created as below) are then used as candidate answers and distractors, respectively. These traces are generated by by translating the LTL formulas into Büchi automata using the `SPOT` tool for ω-automata manipulation [20,21] and then extracting accepting runs. If an accepting run includes a state with multiple possible transitions, we randomly choose one of the branches to resolve the ambiguity. No specific preference is given to one trace over another beyond this random selection.

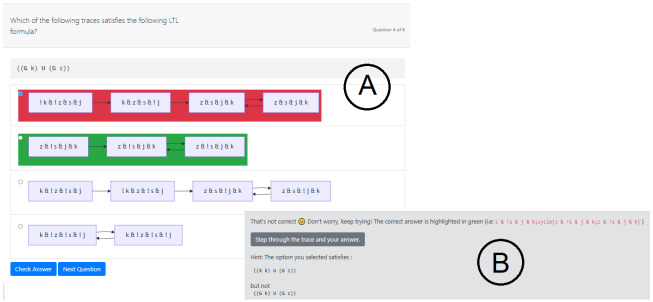

Fig. 2: A Trace Satisfaction (Multiple Choice) (A) question, with (B) feedback about a learner's mistake.

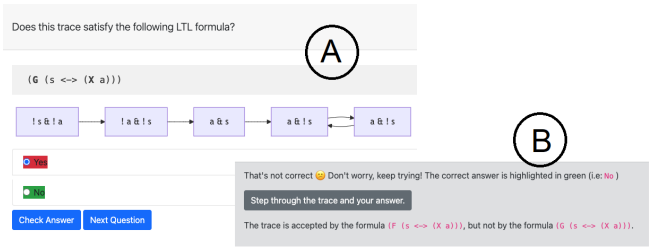

Fig. 3: A Trace Satisfaction (Y/N) (A) question, with (B) feedback about a learner's mistake. Figure 4 shows how the stepper can help shed further light on the mistake.

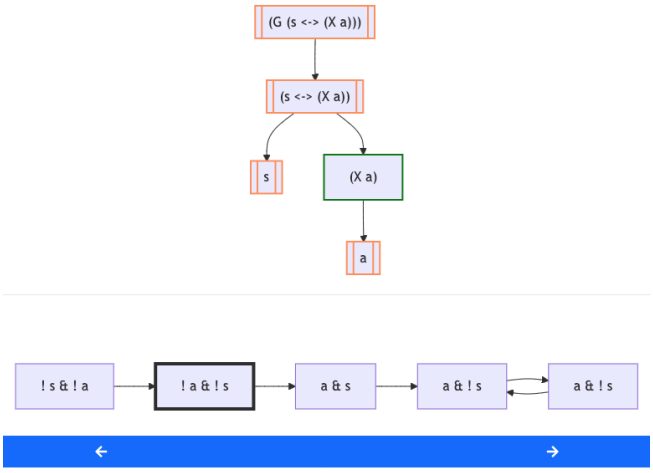

Fig. 4: LTL Stepper: The syntax tree (top) shows how each sub-formula of $\boldsymbol{G}(s \iff (\boldsymbol{X}a))$ is satisfied (green border) or not satisfied (orange double border) at a given trace step. The trace (bottom) highlights the trace step under study and shows the assignment of truth values to literals at each step.

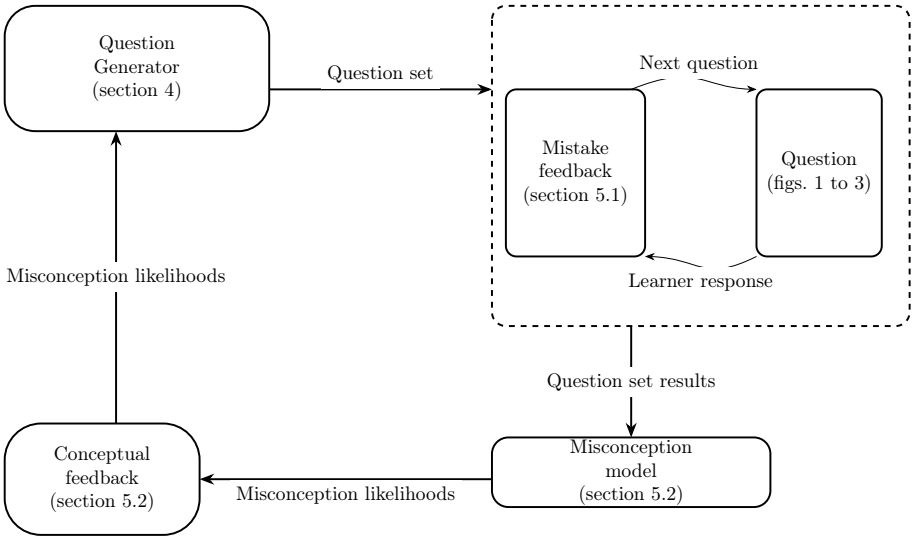

Fig. 5: The overall flow of the LTL Tutor.

Generating Good Mutants The key to generating useful problems comes down to creating good mutants. Doing so well is a central contribution of the LTL Tutor.

An English-to-LTL question, for example, might be founded in the following seed formula and English sentence pair:

$\boldsymbol{G}(e \implies (\boldsymbol{F}h))$	*Whenever the engine is on, the headlight will be on then or at some point in the future.*

As we have noted, it is natural to mutate this seed formula to create mutants. Because LTL equality is decidable [62], we can easily rule out syntactic variants that are not semantically different. Thus, we could adapt typical syntactic mutation techniques from programming to LTL formulae: for instance, we might randomly change logical operators (eq. (1)), change operands (eq. (2)), or swap operand order (eq. (3)):

$$\boldsymbol{G}(e \implies (\boldsymbol{F}h)) \xrightarrow{\text{mutate}} \boldsymbol{G}(e \wedge (\boldsymbol{F}h)) \tag{1}$$

$$\boldsymbol{G}(e \implies (\boldsymbol{F}h)) \xrightarrow{\text{mutate}} ((\boldsymbol{G}e) \implies (\boldsymbol{F}h)) \tag{2}$$

$$\boldsymbol{G}(e \implies (\boldsymbol{F}h)) \xrightarrow{\text{mutate}} \boldsymbol{G}((\boldsymbol{F}h) \implies e) \tag{3}$$

While these syntactic mutants of the original formula could be used as distractors in a multiple choice question, they are very unlikely to capture the actual difficulties that learners have. For example, since the *and* operator is nowhere in the English sentence to be translated, it is highly unlikely that a learner would pick $\boldsymbol{G}(e \wedge (\boldsymbol{F}h))$ (the mutant in eq. (1)) as an answer. Furthermore, if a learner were to select this option, it is unclear *why* they did so. This limits the kinds of feedback that can be provided to the learner.

Table 1: Conceptual mutation rules used by the LTL Tutor, alongside their associated misconceptions. Arbitrary LTL formulae are represented by α, β, and δ. Arbitrary binary operators are represented by $\bowtie$.

Misconception	Mutation Rules
Implicit $\boldsymbol{G}$	$\boldsymbol{G}\alpha \xrightarrow{\text{mutate}} \alpha$
Implicit $\boldsymbol{F}$	$\boldsymbol{F}\alpha \xrightarrow{\text{mutate}} \alpha$
Bad State Quantification	$\boldsymbol{G}\alpha \xrightarrow{\text{mutate}} \boldsymbol{F}\alpha$
	$\boldsymbol{F}\alpha \xrightarrow{\text{mutate}} \boldsymbol{G}\alpha$
	$\alpha\boldsymbol{U}\beta \xrightarrow{\text{mutate}} (\boldsymbol{F}\alpha)\boldsymbol{U}\beta$
	$\alpha\boldsymbol{U}\beta \xrightarrow{\text{mutate}} \alpha\boldsymbol{U}(\boldsymbol{F}\beta)$
	$\alpha\boldsymbol{U}\beta \xrightarrow{\text{mutate}} (\boldsymbol{G}\alpha)\boldsymbol{U}\beta$
	$\alpha\boldsymbol{U}\beta \xrightarrow{\text{mutate}} \alpha\boldsymbol{U}(\boldsymbol{G}\beta)$
	$\alpha\boldsymbol{U}\beta \xrightarrow{\text{mutate}} (\beta\boldsymbol{U}\alpha)$
Precedence	$\alpha \bowtie (\beta \bowtie \delta) \xrightarrow{\text{mutate}} (\alpha \bowtie \beta) \bowtie \delta$
Exclusive $\boldsymbol{U}$	$\alpha\boldsymbol{U}(\neg\alpha \wedge \beta) \xrightarrow{\text{mutate}} \alpha\boldsymbol{U}\beta$
Weak $\boldsymbol{U}$	$\alpha\boldsymbol{U}\beta \xrightarrow{\text{mutate}} (\alpha\boldsymbol{U}\beta) \wedge \boldsymbol{F}\beta$
Bad State Index	$\delta\boldsymbol{U}(\alpha \wedge (\boldsymbol{F}\beta)) \xrightarrow{\text{mutate}} (\delta\boldsymbol{U}\alpha) \wedge (\boldsymbol{F}\beta)$
	$\delta\boldsymbol{U}(\alpha \wedge (\boldsymbol{G}\beta)) \xrightarrow{\text{mutate}} (\delta\boldsymbol{U}\alpha) \wedge (\boldsymbol{G}\beta)$
	$\delta\boldsymbol{U}(\alpha \wedge (\boldsymbol{F}\beta)) \xrightarrow{\text{mutate}} (\delta\boldsymbol{U}\alpha) \wedge (\boldsymbol{F}\beta)$
	$\delta\boldsymbol{U}(\alpha \vee (\boldsymbol{G}\beta)) \xrightarrow{\text{mutate}} (\delta\boldsymbol{U}\alpha) \vee (\boldsymbol{G}\beta)$
	$\delta\boldsymbol{U}(\alpha \implies (\boldsymbol{F}\beta)) \xrightarrow{\text{mutate}} (\delta\boldsymbol{U}\alpha) \implies (\boldsymbol{F}\beta)$
	$\delta\boldsymbol{U}(\alpha \implies (\boldsymbol{G}\beta)) \xrightarrow{\text{mutate}} (\delta\boldsymbol{U}\alpha) \implies (\boldsymbol{G}\beta)$
	$\boldsymbol{X}(\alpha \wedge \beta) \xrightarrow{\text{mutate}} (\boldsymbol{X}\alpha) \wedge \beta$
	$\boldsymbol{X}(\boldsymbol{X}(\boldsymbol{X} \ldots \boldsymbol{X}\alpha)) \xrightarrow{\text{mutate}} \boldsymbol{X}\alpha$
Other Implicit	$(\neg\alpha\boldsymbol{U}\alpha) \xrightarrow{\text{mutate}} \boldsymbol{F}\alpha$
	$(\alpha\boldsymbol{U}(\boldsymbol{G}\alpha)) \xrightarrow{\text{mutate}} \alpha \wedge (\boldsymbol{F}(\boldsymbol{G}\alpha))$
	$(\boldsymbol{F}\alpha) \wedge (\boldsymbol{G}(\beta \implies (\boldsymbol{X}(\boldsymbol{G}\beta)))) \xrightarrow{\text{mutate}} \alpha \wedge (\boldsymbol{X}(\boldsymbol{G}\beta))$
	$\boldsymbol{X}\alpha \xrightarrow{\text{mutate}} \boldsymbol{F}\alpha$
	$\alpha \bowtie \beta \xrightarrow{\text{mutate}} \alpha$
	$\alpha \bowtie \beta \xrightarrow{\text{mutate}} \beta$
	$\neg\alpha \xrightarrow{\text{mutate}} \alpha$

Instead, the LTL Tutor uses our well-substantiated catalogs of LTL misconceptions [28,31] to guide the mutation process. This process of *conceptual* mutation is achieved by associating each misconception with mutation rules (table 1). Applying any of these mutation rules to a given LTL formula (or sub-formula) generates a mutant that embodies the corresponding misconception. The seed formula above, for example, could be mutated to explicitly embody multiple misconceptions, with the misconception given as a label:

$$\boldsymbol{G}(e \implies (\boldsymbol{F}h)) \xrightarrow{\text{mutate}} e \implies (\boldsymbol{F}h) \qquad \text{(Implicit } \boldsymbol{G}\text{)}$$

$$\boldsymbol{G}(e \implies (\boldsymbol{F}h)) \xrightarrow{\text{mutate}} \boldsymbol{G}(e \implies h) \qquad \text{(Implicit } \boldsymbol{F}\text{)}$$

$$\boldsymbol{G}(e \implies (\boldsymbol{F}h)) \xrightarrow{\text{mutate}} \boldsymbol{F}(e \implies h) \qquad \text{(Bad State Quantification)}$$

Crucially, each distractor is now associated with a known misconception.[3] Not only do these distractors reflect the actual difficulties that learners are known to have, when chosen, they also provide insight into the underlying misconceptions. Thus, for instance, if a learner selects the first conceptual mutant above (as shown in fig. 1), it is likely that they have the Implicit $\boldsymbol{G}$ misconception. We next discuss how we operationalize this insight.

5 Helping Learners Learn

As mentioned in section 1, we draw a meaningful distinction between learner mis*takes* and mis*conceptions*. While mistakes can be addressed via feedback in terms of the problem at hand (section 5.1), misconception feedback must be provided in terms of the misunderstood concept (section 5.2).

5.1 Addressing Learner Mistakes

The first thing the LTL tutor does is give feedback at the question level. When a learner selects a distractor to an English-to-LTL question (e.g., fig. 1), they are shown a concrete example of why their answer is incorrect and the relationship between their answer and the correct solution. Feedback for trace satisfaction questions involves the formula used to generate the (incorrect) trace alongside the correct formula (fig. 2, fig. 3). If the learner wants further insight, they can walk through the evaluation of the trace they selected over the correct formula trace using an interactive trace stepper. Figure 4 shows how the stepper can help shed light on the learner's mistake in fig. 3.

5.2 Addressing Learner Misconceptions

A single mistake, however, is not enough to identify a pervasive misconception. A learner could have misread the question, mis-clicked, or just had a minor

[3] When multiple mutants are *syntactically* equal, we present only one to the learner, but associate all relevant misconceptions with that distractor.

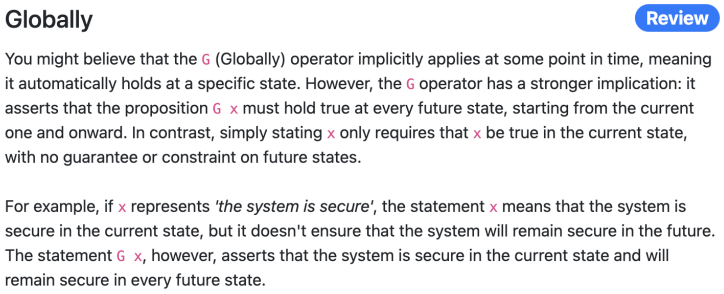

Fig. 6: Feedback for the Implicit ***G*** misconception.

misunderstanding. However, if a learner consistently makes the same mistake, it is likely that they have a misconception.

As described in section 3, the LTL Tutor models the likelihood of a learner having a misconception based on their previous mistakes. This model informs seed formula generation, and thus the likelihood of a learner encountering a question that exercises a particular misconception.

Once the learner has got at least 5 questions incorrect, the tutor provides textual feedback about their most probable misconception.[4] Crucially, this feedback does not refer to a specific question encountered by the learner, but rather the general misconception itself. Using the refutation text format (section 2), this feedback confronts the learner with their misconception and provides a rebuttal to it. For example, the feedback for the Implicit ***G*** misconception (fig. 6) presents a hypothesis of the learner's idea of how the ***G*** operator behaves, explains the correct semantics of the operator, and provides an example to illustrate the difference.

6 Instructor Support

Many of our existing users are instructors who employ the LTL Tutor in the context of a course. They would benefit from having feedback on how their students are doing, not only to track progress but also to detect class-wide persistent misconceptions (which may suggest weaknesses in their materials).

Therefore, the LTL Tutor provides instructors the option of creating a notion of a "course". This generates a code that students use when submitting work. Instructors can then use the course instance to track student progress by identifier or class progress via aggregate statistics.

Because we host the LTL Tutor, this can create discomfort or problems for some instructors regarding student privacy. For that reason, the tutor is available

[4] This threshold is arbitrary but chosen with some thought. We wanted it to be high enough that users gain familiarity with the tutor, to avoid misidentifying early mistakes as misconceptions. At the same time, it is low enough to ensure that users don't go too long without receiving conceptual feedback.

Table 2: Example patterns used to translate LTL to English.

LTL Pattern	English Translation
$\boldsymbol{G}(\alpha \implies (\boldsymbol{F}\beta))$	Whenever α (holds), eventually β will (hold)
$\boldsymbol{G}(\boldsymbol{F}\alpha)$	There will always be a point in the future where α (holds)
$\boldsymbol{F}(\boldsymbol{G}\neg\alpha)$	Eventually, it will never be the case that α (holds)
$\boldsymbol{X}\boldsymbol{X}...\boldsymbol{X}\alpha$	In n states, α (will hold)
$\boldsymbol{G}(\alpha \implies (\boldsymbol{X}(\beta \boldsymbol{U} \delta)))$	Whenever α (happens), β (will hold) until δ (holds)

as an open-source system [52] with instructions for running local copies [51]. While we appreciate instructors providing us with summary statistics (which help us keep track of both global student understanding of LTL[5] and the tutor's performance), we do not require this.

7 LTL to English

The creation of English-to-LTL questions requires the translation of a seed formula into English. To do this, the LTL Tutor first attempts to match the LTL formula to a set of common patterns (inspired by Dwyer et al.'s work on patterns in property specifications [22,23]), some of which are described in table 2.

Formulae that do not match any of these patterns are recursively translated into English via mechanical description of their logical operators. For instance, a formula like $\boldsymbol{G}(p \wedge \boldsymbol{X}q)$ might be broken into "Globally, p and $\boldsymbol{X}q$," where $\boldsymbol{X}q$ is further translated as "in the next state, q." This ensures even complex or unconventional formulae receive a systematic, if literal, English description. However, these translations may be stilted or ambiguous, as the translation method ignores systematic dependencies between subformulae. For instance, the formula $\boldsymbol{F}(n \rightarrow \boldsymbol{G}z)$, translates to "Eventually, globally, z holds is necessary for n holds". We readily acknowledge that this recursively-generated phrasing is both confusing and unnatural; future work should look to improve this output.

While translations could be improved with language models, we are wary of potential "hallucinations" that could lead to incorrect translations. The demands of the educational context require that English translations never be *incorrect*.

8 Limitations

Given the tutor's support for English-to-LTL, it is natural to wonder why it does not also support LTL-to-English. This is particularly relevant since the work we build upon [29,30] identified misconceptions in both directions. However, LTL-to-English requires the ability to check English output. Naturally, it may be possible to employ language models for this. However, we have not done this because of our desire for reliability in evaluating output, which seems hard to

[5] Because LTL operators are tied to natural language, it is conceivable that different linguistic backgrounds would have different performance and misconceptions.

achieve. Furthermore, in prior work, people demonstrated strong performance in this direction [31], reducing its priority. In addition, language models can significantly drive up computational costs (complicating our hosting) or require use of external paid services (which is difficult at scale).

A natural weakness of the current tutor is that it centers around our existing catalog of misconceptions. Though this has been built up over many years, there may yet be other misconceptions in the wild. One of our goals is to adapt the tutor to be more open to these: e.g., using some of the purely syntactic mutants that we rejected earlier (section 4) to see whether they yield unexpected answers. The reason we have not done this already is that turning these mistakes into misconceptions ideally requires learners to provide textual explanations of their choices, and making the interface for this useful to us while not irritating to them is a challenge.

Acknowledgments

This work was partially supported by US NSF grant DGE-2208731. We thank Sriram Sankaranarayanan for his advice. We are also grateful to Skyler Austen, Elijah Rivera, Gavin Gray, and Kuang-Chen Lu for alpha testing the LTL Tutor and providing valuable feedback. Finally, we thank the instructors and learners who have used the LTL Tutor, both in classrooms and in the wild.

Disclosure of Interests

All authors declare that they have no competing interests.

References

1. Van der Aalst, W.M., de Beer, H.T., van Dongen, B.F.: Process mining and verification of properties: An approach based on temporal logic. In: On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31-November 4, 2005, Proceedings, Part I. pp. 130–147. Springer (2005)
2. Acree, A., Budd, T., Demillo, R., Lipton, R., Sayward, F.: Mutation analysis. Tech. Rep. ADA076575, Georgia Inst. of Tech. Atlanta School of Information and Computer Science (09 1979), https://apps.dtic.mil/sti/citations/ADA076575
3. Alur, R., Bansal, S., Bastani, O., Jothimurugan, K.: A framework for transforming specifications in reinforcement learning. In: Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday. pp. 604–624. Springer (2022). https://doi.org/10.1007/978-3-031-22337-2_29
4. Amram, G., Bansal, S., Fried, D., Tabajara, L.M., Vardi, M.Y., Weiss, G.: Adapting behaviors via reactive synthesis. In: CAV. pp. 870–893. Springer (2021). https://doi.org/10.1007/978-3-030-81685-8_41
5. Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R.: Cognitive tutors: Lessons learned. The journal of the learning sciences **4**(2), 167–207 (1995)

6. Antoniotti, M., Mishra, B.: Discrete events models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers. In: ICRA. pp. 1441–1446. IEEE (1995). https://doi.org/10.1109/ROBOT.1995.525480
7. Araki, B., Li, X., Vodrahalli, K., DeCastro, J.A., Fry, M.J., Rus, D.: The logical options framework. In: ICML. vol. 139, pp. 307–317. PMLR (2021), http://proceedings.mlr.press/v139/araki21a.html
8. Babiak, T., Křetínskỳ, M., Řehák, V., Strejček, J.: LTL to Büchi automata translation: Fast and more deterministic. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 95–109. Springer (2012)
9. Bansal, S., Li, Y., Tabajara, L.M., Vardi, M.Y., Wells, A.: Model checking strategies from synthesis over finite traces. In: ATVA. pp. 227–247. Springer (2023). https://doi.org/10.1007/978-3-031-45329-8_11
10. Benny, A., Chandran, S., Kalayappan, R., Phawade, R., Kurur, P.P.: faRM-LTL: A domain-specific architecture for flexible and accelerated runtime monitoring of LTL properties. In: International Conference on Runtime Verification. pp. 109–127. Springer (2024)
11. Bhatia, A., Kavraki, L.E., Vardi, M.Y.: Sampling-based motion planning with temporal goals. In: ICRA. pp. 2689–2696. IEEE (2010). https://doi.org/10.1109/ROBOT.2010.5509503
12. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. Journal of Computer and System Sciences **78**(3), 911–938 (2012). https://doi.org/10.1016/j.jcss.2011.08.007
13. Brunel, J., Chemouil, D., Cunha, A., Macedo, N.: The Electrum analyzer: Model checking relational first-order temporal specifications. In: ASE. pp. 884–887. ACM (2018)
14. Ciccio, C.D., Montali, M.: Declarative process specifications: Reasoning, discovery, monitoring. In: Process Mining Handbook, Lecture Notes in Business Information Processing, vol. 448, pp. 108–152. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_4
15. Cosler, M., Hahn, C., Mendoza, D., Schmitt, F., Trippel, C.: nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In: CAV. pp. 383–396. Springer (2023). https://doi.org/10.1007/978-3-031-37703-7_18
16. Cui, L., Rothkopf, R., Santolucito, M.: Towards Reactive Synthesis as a Programming Paradigm (5 2024). https://doi.org/10.1184/R1/25587741.v1, https://kilthub.cmu.edu/articles/conference_contribution/Towards_Reactive_Synthesis_as_a_Programming_Paradigm/25587741
17. De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: Monitoring business metaconstraints based on LTL and LDL for finite traces. In: BPM. pp. 1–17. Springer (2014). https://doi.org/10.1007/978-3-319-10172-9_1
18. De Giacomo, G., Maggi, F.M., Marrella, A., Patrizi, F.: On the disruptive effectiveness of automated planning for LTLf-based trace alignment. In: Artificial Intelligence. pp. 1–7. AAAI (2017). https://doi.org/10.1609/aaai.v31i1.11020
19. Dortmund, T.U.: Logic WiSe 2022 (2022), https://iltis.cs.tu-dortmund.de/Logic-WiSe2022-external/en/#chapterB1
20. Duret-Lutz, A.: Manipulating LTL formulas using Spot 1.0. In: Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13). pp. 442–445. Springer (2013). https://doi.org/10.1007/978-3-319-02444-8_31

21. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From Spot 2.0 to Spot 2.10: What's new? In: Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22). Lecture Notes in Computer Science, vol. 13372, pp. 174–187. Springer (Aug 2022). https://doi.org/10.1007/978-3-031-13188-2_9
22. Dwyer, M.B.: Patterns for LTL translation (2025), https://matthewbdwyer.github.io/psp/patterns/ltl.html, accessed: 2025-01-08
23. Dwyer, M., Avrunin, G., Corbett, J.: Patterns in property specifications for finite-state verification. In: Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002). pp. 411–420 (1999). https://doi.org/10.1145/302405.302672
24. Fainekos, G.E., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for mobile robots. In: ICRA. pp. 2020–2025. IEEE (2005). https://doi.org/10.1109/ROBOT.2005.1570410
25. Fuggitti, F., Chakraborti, T.: NL2LTL — a Python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. AAAI Conference on Artificial Intelligence **37**(13), 16428–16430 (2023). https://doi.org/10.1609/aaai.v37i13.27068
26. Geck, G., Ljulin, A., Peter, S., Schmidt, J., Vehlken, F., Zeume, T.: Introduction to Iltis: an interactive, web-based system for teaching logic. In: ITiCSE. pp. 141–146. ACM (2018). https://doi.org/10.1145/3197091.3197095
27. Geck, G., Quenkert, C., Schmellenkamp, M., Schmidt, J., Tschirbs, F., Vehlken, F., Zeume, T.: Iltis: Teaching logic in the Web. CoRR **abs/2105.05763** (2021)
28. Greenman, B., Prasad, S., Di Stasio, A., Zhu, S., De Giacomo, G., Krishnamurthi, S., Montali, M., Nelson, T., Zizyte, M.: Misconceptions in finite-trace and infinite-trace linear temporal logic. In: International Symposium on Formal Methods. pp. 579–599. Springer (2024)
29. Greenman, B., Prasad, S., Stasio, A.D., Zhu, S., De Giacomo, G., Krishnamurthi, S., Montali, M., Nelson, T., Zizyte, M.: Artifact for misconceptions in finite-trace and infinite-trace linear temporal logic (Jul 2024). https://doi.org/10.5281/zenodo.12770102
30. Greenman, B., Saarinen, S., Nelson, T., Krishnamurthi, S.: Accepted Artifact for Little Tricky Logic: Misconceptions in the Understanding of LTL (Aug 2022). https://doi.org/10.5281/zenodo.6988909
31. Greenman, B., Saarinen, S., Nelson, T., Krishnamurthi, S.: Little tricky logic: Misconceptions in the understanding of LTL. Programming **7**(2), 7:1–7:37 (2023). https://doi.org/10.22152/programming-journal.org/2023/7/7
32. Gundana, D., Kress-Gazit, H.: Event-based signal temporal logic synthesis for single and multi-robot tasks. IEEE Robotics and Automation Letters **6**(2), 3687–3694 (2021). https://doi.org/10.1109/LRA.2021.3064220
33. Hestenes, D., Wells, M., Swackhamer, G.: Force concept inventory. The Physics Teacher **30**(3), 141–158 (1992). https://doi.org/10.1119/1.2343497
34. Kantaros, Y., Zavlanos, M.M.: STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. International Journal of Robotics Research **39**(7), 812–836 (2020). https://doi.org/10.1177/0278364920913922
35. Lahijanian, M., Almagor, S., Fried, D., Kavraki, L., Vardi, M.: This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In: AAAI. pp. 3664–3671. AAAI Press (2015), https://shaull.github.io/pub/LAFKV15.pdf

36. Li, R., Gurushankar, K., Heule, M.J., Rozier, K.Y.: What's in a name? linear temporal logic literally represents time lines. In: 2023 IEEE Working Conference on Software Visualization (VISSOFT). pp. 73–83. IEEE (2023)
37. Lodder, J., Heeren, B., Jeuring, J.: A comparison of elaborated and restricted feedback in LogEx, a tool for teaching rewriting logical formulae. Journal of Computer Assisted Learning **35**(5), 620–632 (2019)
38. Lodder, J., Heeren, B., Jeuring, J.: Providing hints, next steps and feedback in a tutoring system for structural induction. arXiv preprint arXiv:2002.12552 (2020)
39. Lodder, J., Heeren, B., Jeuring, J., Neijenhuis, W.: Generation and use of hints and feedback in a Hilbert-style axiomatic proof tutor. International Journal of Artificial Intelligence in Education **31**, 99–133 (2021)
40. Loizou, S.G., Kyriakopoulos, K.J.: Automatic synthesis of multi-agent motion tasks based on LTL specifications. In: CDC. pp. 153–158. IEEE (2004). https://doi.org/10.1109/CDC.2004.1428622
41. Lu, K.C., Krishnamurthi, S.: Identifying and correcting programming language behavior misconceptions. Proceedings of the ACM on Programming Languages **8**(OOPSLA1), 334–361 (2024)
42. Madeyski, L., Orzeszyna, W., Torkar, R., Jozala, M.: Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. IEEE Transactions on Software Engineering **40**(1), 23–42 (2013)
43. Maggi, F.M., Montali, M., Westergaard, M., Van Der Aalst, W.M.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: Business Process Management: 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30-September 2, 2011. Proceedings 9. pp. 132–147. Springer (2011)
44. Manna, Z., Wolper, P.: Synthesis of communicating processes from temporal logic specifications. TOPLAS **6**(1), 68–93 (1984). https://doi.org/10.1145/357233.357237
45. Nelson, T., Greenman, B., Prasad, S., Dyer, T., Bove, E., Chen, Q., Cutting, C., Vecchio, T.D., LeVine, S., Rudner, J., Ryjikov, B., Varga, A., Wagner, A., West, L., Krishnamurthi, S.: Forge: A tool and language for teaching formal methods. Proceedings of the ACM on Programming Languages **8**(OOPSLA1), 613–641 (2024)
46. O'Connor, L., Wickström, O.: Quickstrom: Property-based acceptance testing with LTL specifications. In: PLDI. pp. 1025–1038. ACM (2022). https://doi.org/10.1145/3519939.3523728
47. Pane, J.F., Griffin, B.A., McCaffrey, D.F., Karam, R.: Effectiveness of Cognitive Tutor Algebra I at scale. Educational Evaluation and Policy Analysis **36**(2), 127–144 (2014)
48. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL. pp. 179–190. ACM (1989). https://doi.org/10.1145/75277.75293
49. Posner, G.J., Strike, K.A., Hewson, P.W., Gertzog, W.A.: Accommodation of a scientific conception: Toward a theory of conceptual change. Science education **66**(2), 211–227 (1982)
50. Prasad, S., Greenman, B., Nelson, T., Krishnamurthi, S.: Conceptual mutation testing for student programming misconceptions. The Art, Science, and Engineering of Programming **8**(2) (2023). https://doi.org/10.22152/programming-journal.org/2024/8/7
51. Prasad, S., Greenman, B., Nelson, T., Krishnamurthi, S.: Hosting the LTL Tutor (2024), https://github.com/brownplt/LTLTutor/wiki/Hosting-the-LTL-Tutor

52. Prasad, S., Greenman, B., Nelson, T., Krishnamurthi, S.: Ltl tutor (2024), https://github.com/brownplt/ltltutor
53. Rojas, J.M., White, T.D., Clegg, B.S., Fraser, G.: Code Defenders: Crowdsourcing effective tests and subtle mutants with a mutation testing game. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). pp. 677–688. IEEE (2017)
54. Schroeder, N.L., Kucera, A.C.: Refutation text facilitates learning: A meta-analysis of between-subjects experiments. Educational Psychology Review **34**(2), 957–987 (2022)
55. Shah, A., Kamath, P., Shah, J.A., Li, S.: Bayesian inference of temporal task specifications from demonstrations. In: NeurIPS. pp. 3808–3817 (2018)
56. Sieg, W.: The AProS project: Strategic thinking & computational logic. Logic Journal of the IGPL **15**(4), 359–368 (2007)
57. Smith, R., Avrunin, G., Clarke, L., Osterweil, L.: PROPEL: an approach supporting property elucidation. In: Proceedings of the 24th International Conference on Software Engineering. ICSE 2002. pp. 11–21 (2002). https://doi.org/10.1109/ICSE.2002.1007952
58. Tabajara, L.M., Vardi, M.Y.: LTLf synthesis under partial observability: From theory to practice. In: GandALF. p. 1–17. Open Publishing Association (2020). https://doi.org/10.4204/eptcs.326.1
59. Tracy II, T., Tabajara, L.M., Vardi, M., Skadron, K.: Runtime verification on FPGAs with LTLf specifications. In: FMCAD. pp. 36–46. IEEE Computer Society (2020). https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_10
60. Umili, E., Capobianco, R., De Giacomo, G.: Grounding LTLf specifications in images. In: KR. pp. 45–63. ACM (2023). https://doi.org/10.24963/kr.2023/65
61. VanLehn, K.: The behavior of tutoring systems. International journal of artificial intelligence in education **16**(3), 227–265 (2006)
62. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: 1st Symposium in Logic in Computer Science (LICS). IEEE Computer Society (1986)
63. Wang, Y., Figueroa, N., Li, S., Shah, A., Shah, J.: Temporal logic imitation: Learning plan-satisficing motion policies from demonstrations. In: Conference on Robot Learning, CoRL. pp. 94–105. PMLR (2022), https://proceedings.mlr.press/v205/wang23a.html
64. Wickström, O.: LTL visualizer (2023), https://github.com/quickstrom/ltl-visualizer
65. Wongpiromsarn, T., Ulusoy, A., Belta, C., Frazzoli, E., Rus, D.: Incremental temporal logic synthesis of control policies for robots interacting with dynamic agents. In: IROS. pp. 229–236. IEEE (2012). https://doi.org/10.1109/IROS.2012.6385575
66. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: Symbolic LTLf synthesis. In: IJCAI. pp. 1362–1369 (2017). https://doi.org/10.24963/ijcai.2017/189