Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Identifying and Explaining (Non-)Equivalence of First-Order Logic Formulas

Fabian Vehlken[1], **Thomas Zeume**[1], **Emilio Carrasco Bustamante**[2], **Maëlle Cornély**[3], **Lukas Pradel**[1]
[1]Ruhr University Bochum
[2]TU Dortmund University
[3]Université Paris-Saclay, ENS Paris-Saclay
{fabian.vehlken, thomas.zeume}@rub.de

Abstract

First-order logic is the basis for many knowledge representation formalisms and methods. Providing technological support for learning to write first-order formulas for natural language specifications requires methods to test formulas for (non-)equivalence and to provide explanations for non-equivalence. We propose such methods based on both theoretical insights and existing tools, implement them, and report on experiments testing their effectiveness on a large educational data set with > 100.000 pairs of first-order formulas.

1 Introduction

First-order logic is central for multiple areas of computer science. It is the basis of formal languages for representing knowledge and reasoning, of specification languages for formal verification, and of query languages for databases, among others. In knowledge representation and reasoning it is used in many subdisciplines, including argumentation (e.g. to formalize arguments and their relationships), belief change, logic programming, and reasoning about knowledge and belief. Also, many other knowledge representation formalisms are based on (fragments of) first-order logic, e.g. description logics, modal logics, and non-monotonic logics.

Basics of first-order logic – modelling, normal forms, and reasoning – are therefore often included in mandatory courses in Bachelor Computer Science programmes; for instance, guidelines of the German Association for Computer Science (GI) recommend to include such basics in computer science study programmes (Gesellschaft für Informatik e. V. 2016). Figure 1 sketches a typical exercise of such a course which students learn to solve by (1) designing a suitable first-order vocabulary (i.e., which relation, function, and constant symbols to use), (2) describing the scenario by first-order formulas, (3) transforming formulas into a suitable normal form (e.g. prenex or Skolem form), and (4) inferring knowledge using an inference mechanism (e.g. resolution).

In this paper we continue a line of research aimed at supporting educational tasks (1) – (4) for learning first-order logic in educational support systems. While basic support for all four tasks is provided by state-of-the-art systems such as the Iltis system (Schmellenkamp, Vehlken, and Zeume 2024; Kneisel, Vehlken, and Zeume 2025), not much work has been done on providing advanced feedback, even though individual, timely feedback is essential for learning success. In this work we focus on advanced feedback mechanisms for writing first-order formulas, i.e. for educational task (2).

Writing first-order formulas is a challenge for students and prone to many errors (Schmellenkamp, Schmalstieg, and Zeume 2025). Besides stating that a student attempt is incorrect and providing counter examples, feedback ideally pinpoints conceptual mistakes. For instance, when asked to write a first-order formula for the natural language statement

"*System libraries depend only on system libraries.*"

with solution

$$\forall x \forall y \big((S(x) \wedge D(x,y)) \rightarrow S(y) \big),$$

we envision that students receive individual conceptual feedback depending on their attempt, for instance:

(a) Hints at quantification mistakes:
- Student: $\forall x \exists y \big((S(x) \wedge D(x,y)) \rightarrow S(y) \big)$
- Feedback: It might help to double check the quantifier structure of your formula.

(b) Hints at guarding mistakes:
- Student: $\forall x \forall y (D(x,y) \rightarrow S(y))$
- Feedback: It seems that you need to ensure that the variable x ranges only over some domain elements.

(c) Hints at missing parts or symbols of a formula:
- Student: $\forall x \forall y (S(x) \rightarrow S(y))$
- Feedback: The statement includes a dependency between libraries, but your formula does not use the relation symbol D that models such dependencies.

Algorithmically providing feedback faces two obstacles. First, providing feedback for first-order formulas is an inherently hard algorithmic problem. Already determining the correctness of student attempts requires checking logical equivalence of first-order formulas, often modulo a background theory for knowledge implicit in the given scenario. While this algorithmic problem is undecidable in general, one approach is to use a standard reduction to the satisfiability problem for first-order logic and to apply a theorem prover such as Vampire (Kovács and Voronkov 2013; Bártek et al. 2025). A natural question is whether this suffices and also whether small, meaningful counter examples can be computed this way for non-equivalent formulas.

- RQ1: Can the correctness of student attempts be determined in practice for first-order logic modelling tasks?
- RQ2: In case of an incorrect attempt, can a counter example witnessing incorrectness be provided?

Second, no approaches for providing higher level explanations (such as explanations of types (a) – (c) from above) for explaining why an attempt is wrong or, in other words, why two first-order formulas are non-equivalent, have been explored in prior work. In particular, no algorithmic methods for finding such explanations exist so far.

- RQ3: In case of an incorrect attempt, how can highlevel explanations be computed in practice?

Contributions

We address research questions RQ1, RQ2, and RQ3 by (1) designing algorithmic methods for explaining non-equivalence of first-order formulas and (2) evaluating equivalence testing, finding counter models, and the methods for explaining non-equivalence from (1) on large educational data sets with > 100.000 pairs of first-order formulas.

Our explanations for non-equivalence of two formulas ψ and φ come in two flavours. A *blocker* is a syntactic property of one of them that prevents equivalence; examples are different numbers of free variables or missing vocabulary symbols. A *bugfixing modification* is a small modification of one of the formulas that makes it equivalent to the other; examples are bugfixes that modify a quantifier, a guarding attempt or small careless mistakes. For computing explanations, we exploit several techniques:

- We use consequences of characterizations of Beth definability (Hodges 1993, Theorem 6.6.4) to computationally discover vocabulary symbols that must be used for expressing some property.
- We use specifically designed profiling information for quantifier patterns of atoms and (attempted) guards to compute candidates for bugfixing modifications, whose correctness is then tested by applying them and testing equivalence to the resulting formula.
- We use a naïve Erdös-Rényi-style random model generator to generate counter examples to supplement built-in finite model building algorithms of Vampire.

We evaluate all methods – equivalence testing, finding counter models, methods for explaining non-equivalence – on an educational data set collected anonymously with an educational support system in mandatory Bachelor-level courses at several large universities. At the time of data collection, only feedback indicating whether the input was correct, possibly augmented by a counter example, was provided (with the possibility of receiving feedback that the system was not able to determine the correctness of the input).

In total, the data set contains 125,970 pairs of formulas (distinct: 37,159), of which 27,655 pairs are equivalent (distinct: 3,019) and 98,315 pairs are non-equivalent (distinct: 34,140). Each pair consists of a first-order formula modelling a natural language statement and a formula by a student attempting to model the same statement. In total, there are pairs for 62 statements spread over 14 scenarios, 11 of them with a background theory of implicit knowledge.

Exercise: From modelling to inference

Julia is investigating a collection of software packages. Each software package has exactly one maintainer, and software packages may depend on other software packages. She has identified the following dependencies:

(a) No software package is both a program library and a system library.

(b) System libraries depend only on system libraries.

(c) Program libraries may only depend on system libraries which are not maintained by the same maintainer.

(d) Every software package that must be explicitly installed by the user depends on at least one program library directly.

She concludes that there is no system library that must be explicitly installed by the user. Can you confirm her conclusion using methods you learned for first-order logic?

Figure 1: A first-order logic modelling exercise in which students have to choose a first-order vocabulary, write first-order formulas for natural language specifications in this vocabulary, and infer knowledge with an inference mechanism such as resolution.

For an evaluation summary we refer to Figure 2. We answer research questions RQ1 and RQ2 affirmatively. For all 125,970 pairs of formulas, equivalence is determined by the Vampire theorem prover. For 98.48% of non-equivalent pairs of formulas a counter example is found either by Vampire's finite model builder (93.34%, exclusively 17.68%) or our naïve random model generator (80.80%, exclusively 5.14%). Towards answering research question RQ3, our computational strategies for computing explanations yield at least one explanation for 52.14% of all non-equivalent pairs. Strategies based on missing symbols, quantifiers, and guarding each explain $> 20\%$ of non-equivalent pairs.

These results lay the foundation for conceptual feedback for first-order modelling in educational support systems. They also quantitatively confirm conceptual challenges encountered by students in first-order modelling tasks, including the use of quantifiers and guarding, and thus can be the basis for both fine-grained analysis and intervention design from a CS education research perspective.

Related work

How to explain student mistakes and how to provide adequate feedback has also been explored for educational modelling tasks in other subfields of formal foundations of computer science including propositional logic (Schmellenkamp, Latys, and Zeume 2023), temporal logics (Greenman et al. 2023), regular languages (D'Antoni et al. 2015), and context-free languages (Schmellenkamp et al. 2025).

Outline

After formalizing the setting and recalling methods for determining (non-)equivalence of formulas and finding counter examples in Section 2, we discuss methods for explaining

non-equivalence in Section 3. All methods are evaluated in Section 4. We discuss impact and future work in Section 5.

An extended version of this paper with an appendix containing details on the evaluations and sample exercises is available on arXiv (Vehlken et al. 2026).

2 Setting, Equivalence, Counter Examples

In educational tasks that ask to write first-order formulas for natural language statements, typically students are given a natural language description of a scenario together with the statement to be modelled. For writing the formula, a first-order vocabulary σ with a description of the intended meaning of its symbols is provided by instructors or designed by students in preliminary educational tasks. An instructor specifies a first-order formula ψ over σ that correctly models the statement as well as additional first-order constraints Γ over σ that model constraints implicit in the scenario description. The first-order constraints Γ induce a first-order *background theory* $\mathrm{Th}(\Gamma) \stackrel{\text{def}}{=} \{\mathcal{S} \mid \mathcal{S} \models \Gamma\}$. For checking whether a first-order formula φ over σ written by a student correctly describes the statement to be modelled, one checks whether ψ and φ agree for all σ-structures $\mathcal{S} \in \mathrm{Th}(\Gamma)$.

For determining correctness and certifying incorrectness, the following algorithmic problem needs to be solved:

Problem: FO-EQUIVALENCE-MODULO-THEORY
Input: First-order formulas ψ, φ and a set Γ of first-order formulas over σ.
Question: Are ψ and φ equivalent modulo $\mathrm{Th}(\Gamma)$ (denoted by $\psi \equiv_{\mathrm{Th}(\Gamma)} \varphi$), that is, does $\mathcal{S} \models \psi$ iff $\mathcal{S} \models \varphi$ hold for all $\mathcal{S} \in \mathrm{Th}(\Gamma)$? If not, which $\mathcal{S} \in \mathrm{Th}(\Gamma)$ is a counter example?

We explain existing algorithmic approaches for this problem, as they are the basis for more advanced explanations (see Section 3) and because their performance on educational data sets is evaluated later on (see Section 4).

The decision version of the problem above can be easily reduced to first-order (un-)satisfiability as follows:

$$\begin{aligned} \psi \equiv_{\mathrm{Th}(\Gamma)} \varphi &\iff \bigwedge \Gamma \to (\psi \leftrightarrow \varphi) \text{ is a tautology} \\ &\iff \bigwedge \Gamma \wedge \neg(\psi \leftrightarrow \varphi) \text{ is unsatisfiable.} \end{aligned}$$

Thus, it can be solved by invoking a theorem prover such as Vampire (Kovács and Voronkov 2013; Bártek et al. 2025).

For providing counter examples for non-equivalent formulas, one can use Vampire's finite model building implementation (Reger, Suda, and Voronkov 2016). Since initial experiments indicated that this does not produce counter examples for all pairs judged to be non-equivalent by Vampire, we also use a naïve random model generator approach. We compare both approaches in Section 4.

The naïve random model approach uses an Erdös-Rényi-style construction (see e.g. (Libkin 2004)). Given a universe size n and a probability p, construct a structure $\mathcal{S}$ with universe $A = \{1, \ldots, n\}$. For each k-ary relation symbol $R \in \sigma$, each tuple in A^k will be added to $R^{\mathcal{S}}$ independently with probability p. For k-ary function symbols $f \in \sigma$, the image under $f^{\mathcal{S}}$ of each tuple in A^k is chosen randomly, each element in A having probability $\frac{1}{n}$ to be picked. The universe sizes, probability, and a number of models to be generated for each universe size are parameters.

To find counter examples, one generates random structures in ascending size and checks if they are a model of $\bigwedge \Gamma \wedge \neg(\psi \leftrightarrow \varphi)$. If so, it is a counter example certifying $\psi \not\equiv_{\mathrm{Th}(\Gamma)} \varphi$. By checking $\bigwedge \Gamma \wedge \psi$ and $\bigwedge \Gamma \wedge \varphi$ separately, additionally feedback as to whether a student attempt φ is too restrictive or too permissive can be provided.

Our implementation allows to randomly generate structures satisfying Γ in a preprocessing step, and only to subsequently test against ψ and φ. This is helpful in cases where Γ severely restricts permissible models. In such cases, this preprocessing speeds-up the counter example search.

3 Explanations for Non-Equivalence

Our conceptual explanations for non-equivalence are motivated from our educational application scenario. A recent interview study on errors and misconceptions in first-order logic modelling (Schmellenkamp, Schmalstieg, and Zeume 2025) confirmed the common perception among instructors that when writing first-order formulas, students struggle with the use of (i) quantifiers, (ii) guarding, and (iii) free variables. Additionally, students often (iv) forget to model parts of statements, or (v) do careless syntactical mistakes including wrong use of Boolean operators. Our goal is to explain non-equivalences of first-order formulas that occur because of mistakes due to (i) – (v).

In this section we always assume non-equivalence of the two first-order formulas ψ and φ. Our explanations for non-equivalence are *directional* in the sense that they try to explain why φ is not equivalent to ψ from φ's perspective. This allows to derive appropriate feedback for student attempts from computed explanations.

Roughly speaking, our explanations come in two flavours: *blockers* and *bugfixing modifications*. A *blocker* is a syntactic property of φ that no formula equivalent to ψ (modulo $\mathrm{Th}(\Gamma)$) can have. An example is that ψ uses a relation symbol R and no equivalent formula without R exists; thus if φ does not use R, this explains non-equivalence of φ and ψ. A *bugfixing modification* is a small modification of φ such that applying it to φ leads to a formula equivalent to ψ.

We next describe algorithmic strategies to discover explanations for non-equivalence. A challenge is efficiency: for finding bugfixing explanations, our algorithms enumerate promissing candidate modifications for φ and test whether they transform φ into φ' such that $\varphi' \equiv_{\mathrm{Th}(\Gamma)} \psi$. Thus, each candidate requires a computationally expensive equivalence test (e.g. via Vampire). Our strategies are designed to minimize equivalence tests and to find good candidates.

Table 1 provides an overview of our computational strategies as well as examples for their application.

3.1 Preprocessing

In a preprocessing step, we compute information that profiles (i) use of quantifier in φ and ψ, (ii) use of guards in φ and ψ, and (iii) necessary vocabulary symbols in ψ.

Strategy	Example: Solution formula ψ	Example: Student attempt φ	Explanation
Strategies for Symbols			
S-1 Missing symbol names	$\forall x\,(Q(x) \to P(x))$	$\forall x\,P(x)$	Relation symbol Q does not occur in φ, but is required
S-2 Permuted arguments of a symbol	$\forall x \exists y\,R(x,y)$	$\forall x \exists y\,R(y,x)$	Wrong quantification pattern due to permutation
S-3 Different relation symbol names	$\exists x\,P(x)$	$\exists x\,Q(x)$	Wrong relation symbol used
S-4 Different terms	$\exists x\,P(f(x))$	$\exists x\,P(x)$	Terms in $P(\cdot)$ differ
Strategies for Quantifiers			
Q-1 Different quantifier prefixes	$\forall x \exists y(P(x) \to G(x,y))$	$\forall x \forall y(P(x) \to G(x,y))$	Wrong quantifier prefix
Q-2 Different order of quantifiers	$\forall x(S(x) \to \exists y \forall z\,R(x,y,z))$	$\forall x \exists y \exists z(S(x) \to R(x,y,z))$	Wrong quantification pattern for $R(x,y,z)$
Q-3 Different free variables	$\forall x\,P(x)$	$P(x)$	x is free only in φ
Strategies for Guarding			
G-1 Different guards	$\forall x\,(P(x) \to Q(x))$	$\forall x\,Q(x)$	Missing universal guard for x
	$\exists x\,Q(x)$	$\exists x\,(P(x) \wedge Q(x))$	Superfluous existential guard for x
G-2 Wrong guarding operators	$\forall x\,(P(x) \to Q(x))$	$\forall x\,(P(x) \wedge Q(x))$	$\wedge$ is the wrong guard operator for universally quantified x
Strategies for Boolean Operators*			
B-1 Different negation patterns	$\forall x\,P(x)$	$\forall x\,\neg P(x)$	Wrong negation prefix for $P(x)$
B-2 Swapping Implication	$\forall x\,(P(x) \to Q(x))$	$\forall x\,(Q(x) \to P(x))$	Implication in wrong direction

Table 1: Overview of strategies for explaining non-equivalence of first-order formulas. Only most successful strategies for Boolean operators are listed (*).

Profiling the Use of Quantifiers. We precompute quantifier profiles of atoms occuring in ψ and φ.

A first-order formula is in *prenex normal form*, if it is of the form $Q_1x_1 \dots Q_kx_k\,\mu$ for quantifiers $Q_i \in \{\exists, \forall\}$ and quantifier-free formula μ. We call $Q_1x_1 \dots Q_kx_k$ *quantifier prefix* and $Q_1Q_2 \dots Q_k$ *quantifier prefix type*. Denote by $\varphi^\neg$ the *negation normal form* obtained from φ by "moving all negations inwards" using De Morgans law.

The *quantifier prefix of an atom* $\mathcal{A} \stackrel{\text{def}}{=} R(x_1, \dots, x_k)$ in formula φ captures how variables in the atom are quantified and in which order. Formally, it is the sequence $Q_1y_1 \dots Q_\ell y_\ell$ where Q_i are quantifiers and y_i variables with

- $\{y_1, \dots, y_\ell\} \cup \{z_1, \dots, z_m\} = \{x_1, \dots, x_k\}$, where z_i are free variables of φ, and
- $y_1, \dots, y_\ell$ are bound by $Q_1y_1, \dots, Q_\ell y_\ell$ in this order in the syntax tree of $\varphi^\neg$ (if y_i is quantified multiple times, the sequence contains the quantification closest to $\mathcal{A}$).

The *quantifier prefix type* of $\mathcal{A}$ captures how and in which order positions of $\mathcal{A}$ are quantified. Formally, it is the sequence $Q_1P_1, \dots, Q_\ell P_\ell$ where $P_i \stackrel{\text{def}}{=} \{j \mid x_j = y_i\}$. The address of an atom $\mathcal{A}$ in a formula φ is a path $\pi_\mathcal{A}$ from the root of the syntax tree of φ to $\mathcal{A}$. The *profile* $\text{profile}(\varphi, \mathcal{A}, \pi_\mathcal{A})$ of an atom $\mathcal{A} \stackrel{\text{def}}{=} R(x_1, \dots, x_k)$ with address $\pi_\mathcal{A}$ in φ is the tuple (R, v, q) where the valence v is $v = +$ if no negation is in front of $\mathcal{A}$ in $\varphi^\neg$ and $v = -$ otherwise, and q is the quantifier prefix type of $\mathcal{A}$ in φ. The *profile* $\text{profile}(\varphi)$ of a formula φ is the set $\{\text{profile}(\varphi, \mathcal{A}, \pi_\mathcal{A}) \mid \mathcal{A} \text{ is an atom with address } \pi_\mathcal{A} \text{ in } \varphi\}$.

Example 1. *Consider the first-order formula*

$$\varphi \stackrel{\text{def}}{=} \exists y\big(S(y) \wedge \forall x \neg \forall y(T(x,y,x) \vee S(y))\big).$$

The quantifier prefix of the atom $T(x,y,x)$ is $\forall x \exists y$, and its quantifier prefix type is $\forall\{1,3\}\exists\{2\}$. The profile of φ is

$$\{(S, +, \exists\{1\}), (S, -, \exists\{1\}), (T, -, \forall\{1,3\}\exists\{2\})\}.$$

For the sake of readability, we introduced a simplified notion of profiles that only talks about variables. In our implementation, profiles contain information about terms occuring in atoms and about the quantifier structure of variables occuring in those terms.

In a preprocessing step for φ and ψ, our procedure computes $\text{profile}(\psi)$, $\text{profile}(\varphi)$, and $\text{prefix}(\psi)$ and $\text{prefix}(\varphi)$ (if φ, ψ are in prenex form).

Profiling the Use of Guards. The use of a variable z in an atom $\mathcal{A}(\overline{u})$ within a formula ψ is *guarded* by an atom $\mathcal{G}(\overline{v})$, if $z = y_\ell$ in the quantifier prefix $Q_1y_1 \dots Q_ky_\ell$ of $\mathcal{G}(\overline{u})$ and there is a subformula of the formula ψ of the form

- $\mathcal{G} \to \rho(\overline{w})$ and $\mathcal{A}(\overline{u})$ occurs in $\rho(\overline{w})$ if $Q_\ell = \forall$, and
- $\mathcal{G} \wedge \rho(\overline{w})$ and $\mathcal{A}(\overline{u})$ occurs in $\rho(\overline{w})$ if $Q_\ell = \exists$.

It is *wrongly guarded* if the conditions are replaced by

- $\mathcal{G} \wedge \rho(\overline{w})$ and $\mathcal{A}(\overline{u})$ occurs in $\rho(\overline{w})$ if $Q_\ell = \forall$, and

- $\mathcal{G} \to \rho(\overline{w})$ and $\mathcal{A}(\overline{u})$ occurs in $\rho(\overline{w})$ if $Q_\ell = \exists$.

As an example, x in the atom $R(x, y, z)$ within $\forall y \exists x(S(x) \wedge \exists z R(x, y, z))$ is guarded by $S(x)$, while it is wrongly guarded within $\forall y \exists x(S(x) \to \exists z R(x, y, z))$.

Let guards(ψ) be the set that contains all pairs $(\text{PROFILE}(\mathcal{G}), \text{PROFILE}(\mathcal{A}))$ of profiles of atoms $\mathcal{G}$ and $\mathcal{A}$ such that there is a variable z in $\mathcal{A}$ guarded by $\mathcal{G}$ within ψ. Similarly, let wrong-guards(ψ) contain all such pairs where z is wrongly guarded.

In a preprocessing step for φ and ψ, our procedure computes guards(ψ), guards(φ), wrong-guards(ψ), and wrong-guards(φ).

Profiling Necessary Symbols. If ψ and φ are non-equivalent and φ does not use a symbol R used by ψ, one possible explanation for the non-equivalence is that R is necessary for expressing the property expressed by ψ.

Formally, let ψ be a first-order formula over σ that expresses a property $\mathcal{P}$ modulo a theory $\mathrm{Th}(\Gamma)$. A symbol $R \in \sigma$ is *necessary* to express $\mathcal{P}$ modulo $\mathrm{Th}(\Gamma)$, if there is no φ over $\sigma \setminus \{R\}$ that is equivalent to ψ.

Whether a symbol is necessary can be inferred using the projective Beth definability theorem, see for instance (ten Cate and Comer 2025). The following theorem rephrases the formulation from (Hodges 1993, Theorem 6.6.4):

Theorem 1. *Let σ, σ' be vocabularies with $\sigma \subseteq \sigma'$, let $\psi(\bar{x})$ be a first-order formula over σ', and let $\mathcal{M}$ be a non-empty set of σ'-structures. Then the following are equivalent:*

(i) There exist two σ'-structures $\mathcal{A}$ and $\mathcal{B}$ in $\mathcal{M}$ such that
- *$\mathcal{A}|\sigma = \mathcal{B}|\sigma$ and*
- *there is a tuple $\bar{a}$ of $\mathcal{A}$: $\mathcal{A} \models \psi(\bar{a}) \iff \mathcal{B} \not\models \psi(\bar{a})$.*

(ii) There exists no first-order formula $\varphi(\bar{x})$ over σ that is equivalent to $\psi(\bar{x})$ on $\mathcal{M}$.

Theorem 1 immediately helps in our application. For determining whether symbols $\sigma' \setminus \sigma$ are necessary in a solution formula ψ and background theory $\mathcal{M} \stackrel{\text{def}}{=} \mathrm{Th}(\Gamma)$, we employ Padoa's method (see e.g. (ten Cate and Comer 2025)) to test whether structures $\mathcal{A}$ and $\mathcal{B}$ and a tuple $\bar{a}$ as in part (i) of the theorem exist. If so, the symbols in $\sigma' \setminus \sigma$ are necessary in any student attempt formula φ.

We reduce the testing of part (i) of Theorem 1, i.e. whether structures $\mathcal{A}$ and $\mathcal{B}$ exist, to the satisfiability problem for first-order logic and apply a theorem prover like Vampire. Such structures exist if and only if the following existential second-order (ESO) formula χ is satisfiable:

$$\exists \overline{R}\, \exists \overline{f} \exists \overline{T}' \exists \overline{g}' \exists \overline{T}'' \exists \overline{g}'' \exists \bar{x} \Big(\big(\varphi_{[\overline{T}/\overline{T}',\, \overline{g}/\overline{g}']}(\bar{x}) \leftrightarrow \varphi_{[\overline{T}/\overline{T}'',\, \overline{g}/\overline{g}'']}(\bar{x}) \big) \wedge \Gamma_{[\overline{T}/\overline{T}',\, \overline{g}/\overline{g}']} \wedge \Gamma_{[\overline{T}/\overline{T}'',\, \overline{g}/\overline{g}'']} \Big)$$

where

- $\exists \overline{R}\, \exists \overline{f}$ quantifies a structure over σ,
- $\exists \overline{T}' \exists \overline{g}'$ and $\exists \overline{T}'' \exists \overline{g}''$ quantify structures over $\sigma' \setminus \sigma$, and
- $\psi_{[S/S']}$ denotes the formula obtained from ψ by replacing every occurrence of S by S'.

The ESO formula χ is satisfiable if and only if the following first-order logic formula is satisfiable:

$$\Big(\big(\varphi_{[\overline{T}/\overline{T}',\, \overline{g}/\overline{g}']}(\bar{x}) \leftrightarrow \varphi_{[\overline{T}/\overline{T}'',\, \overline{g}/\overline{g}'']}(\bar{x}) \big) \wedge \Gamma_{[\overline{T}/\overline{T}',\, \overline{g}/\overline{g}']} \wedge \Gamma_{[\overline{T}/\overline{T}'',\, \overline{g}/\overline{g}'']} \Big)$$

We note that in (Hodges 1993), all first-order vocabularies contain equality. Therefore, in Theorem 1, if σ' contains equality, then σ also contains equality. However, in our application, we also want to determine whether the equality symbol is necessary to express the property expressed by ψ on $\mathrm{Th}(\Gamma)$. To accommodate this, we rephrase the question of whether equality is necessary as a question of whether a congruence relation, i.e. an equivalence relation compatible with all other symbols, is necessary. This can be tested using Theorem 1. We rely on the fact that factoring a structure with a congruence by that congruence collapses this relation into the equality relation.

We reuse notation from (Ebbinghaus et al. 1994, Exercise 11.6.11) and state the following lemma. For a first-order formula ψ (with equality) over σ, we denote by ψ^* the first-order formula over $\sigma \cup \{E\}$, for a fresh binary relation symbol E, obtained by replacing all atoms $t_1 = t_2$ in ψ by $E(t_1, t_2)$, where t_1, t_2 are terms. Furthermore, let δ_E be a first-order formula axiomatizing that E is a congruence with respect to all symbols in σ. For a set Γ of first-order formulas we denote $\Gamma^* \stackrel{\text{def}}{=} \{\varphi^* \mid \varphi \in \Gamma\}$.

Lemma 1. *Let ψ be a first-order formula and Γ a finite set of first-order formulas over a vocabulary σ (with equality). Then the following are equivalent:*

(i) There exists a formula φ over σ (without equality) such that $\psi \equiv_{\mathrm{Th}(\Gamma)} \varphi$.

(ii) There exists a formula μ over σ (without E) such that $\psi^* \equiv_{\mathrm{Th}(\Gamma^* \cup \{\delta_E\})} \mu$.

Thus, whether equality is needed in ψ reduces to testing whether the symbol E is necessary in ψ^*.

Proof (of Lemma 1). We use the following fact, see (Ebbinghaus et al. 1994). For all formulas ψ, all models $\mathcal{S}$, all binary relations $E^{\mathcal{S}}$ with $(\mathcal{S}, E^{\mathcal{S}}) \models \delta_E$ it holds that $(\mathcal{S}, E^{\mathcal{S}}) \models \psi^*$ iff $\mathcal{S}/E^{\mathcal{S}} \models \psi$, where $\mathcal{S}/E^{\mathcal{S}}$ denotes the factor structure of $\mathcal{S}$ by $E^{\mathcal{S}}$.

Using this fact, one can easily show that $\psi \equiv_{\mathrm{Th}(\Gamma)} \varphi$ iff $\psi^* \equiv_{\mathrm{Th}(\Gamma^* \cup \{\delta_E\})} \varphi^*$ for any two formulas ψ, φ and set Γ of formulas. Direction (i) $\implies$ (ii) follows, since φ does not use equality by assumption, and therefore φ^* does not use E. Thus, $\varphi^* = \varphi$ is a valid choice for μ.

For (ii) $\implies$ (i), we observe that in the proof of the projective Beth definability theorem in (ten Cate and Comer 2025), the constructed formula that explicitly defines the symbol in question (in this case, E) in terms of σ does not use equality, if the background theory does not use equality, which $\mathrm{Th}(\Gamma^* \cup \{\delta_E\})$ does not. Therefore, if μ as in (ii) exists, there also exists a formula μ' equivalent to μ on $\mathrm{Th}(\Gamma^* \cup \{\delta_E\})$ that uses neither equality nor E. Thus, μ' is a valid choice for φ in (i).

The lemma statement follows. □

In a preprocessing step for ψ, all necessary symbols $\text{NECESSARY}(\psi)$ of ψ are computed relying on the described procedure. Note that this requires a first-order satisfiability test for each symbol and is therefore expensive, but this only has to be done once for each solution formula ψ.

3.2 Strategies for Explaining Non-equivalence

We now explain computational strategies for explaining non-equivalence; most using the precomputed information.

Strategies for Symbols. The following strategies identify missing symbols and candidates for fixing non-equivalence due to permuted arguments or wrong names of symbols.

S-1 Missing symbol names If a vocabulary symbol (relation, function, or constant) is necessary in ψ but does not occur in φ, then this is a blocker explaining non-equivalence.

S-2 Permuted arguments of a symbol If only φ has an atom $\mathcal{A}_\varphi$ with profile (R, v, q_φ) and only ψ has an atom with profile (R, v, q_ψ), and permuting arguments of $\mathcal{A}_\varphi$ changes q_φ into q_ψ, then modify φ by permuting the arguments. If successful (i.e., if this leads to equivalence to ψ), then this modification explains non-equivalence.

S-3 Different relation symbol names If only φ has an atom $\mathcal{A}_\varphi$ with profile (R_φ, v, q) and only ψ has an atom with profile (R_ψ, v, q), then modify φ by replacing the symbol R in $\mathcal{A}_\varphi$ by S. If successful (i.e., if this leads to equivalence to ψ), then this modification explains non-equivalence.

S-4 Different terms Similarly to the previous case but for terms (relying on extended profiles that incorporate information for terms).

Strategies for Quantifiers. The following strategies identify different quantification patterns and different sets of free variables.

Q-1 Different quantifier prefixes If both ψ and φ are in prenex normal form and their quantifier prefixes differ, then modify φ by replacing its prefix by the prefix of ψ if this does not introduce additional free variables. If successful (i.e., if this leads to equivalence to ψ), then this modification explains non-equivalence.

Q-2 Different order of quantifiers If only φ has an atom $\mathcal{A}_\varphi$ with profile (R, v, q_φ) and only ψ has an atom with profile (R, v, q_ψ), then modify φ by replacing[1] the quantifiers corresponding to q_φ in φ by quantifiers corresponding to q_ψ (with suitable renaming of variables). If successful (i.e., if this leads to equivalence to ψ), then this modification explains non-equivalence.

Q-3 Different free variables If the set of free variables of ψ and φ differs, then this is a blocker explaining non-equivalence.

Strategies for Guarding. The following strategies identify missing and superfluous guards in φ as well as wrong guards.

G-1 Different guards If φ contains an atom $\mathcal{A}_\varphi$ with an unguarded argument position i and ψ contains an atom $\mathcal{A}_\psi$ with the same profile but position i guarded by a guard $\mathcal{G}_\psi$, then modify φ by adding a guard $\mathcal{G}_\varphi$ for position i of $\mathcal{A}_\varphi$ (if possible). Symmetrically for ψ and φ (the modification then removes guards from φ). If successful (i.e., if this leads to equivalence to ψ), then this modification explains non-equivalence.

G-2 Wrong guarding operators If φ contains a wrongly guarded atom, then modify φ by replacing $\rightarrow$ by $\wedge$ in the guard or vice versa.

Strategies for Boolean Operators. Strategies for discovering explanations for non-equivalence for Boolean operators based on modifications have been studied extensively in (Schmellenkamp, Latys, and Zeume 2023) and can be reused for first-order logic. We only present a strategy for negations that is specific for first-order logic, and one exemplary Boolean strategy concerning implication (chosen because it explains many non-equivalences in our data set).

B-1 Different negation patterns If only φ has an atom $\mathcal{A}_\varphi$ with profile (R, v_φ, q) and only ψ has an atom with profile (R, v_ψ, q) where $v_\varphi = +$ iff $v_\psi = -$ (i.e., R is positive in exactly one of the two formulas) then modify[1] negations on the path of the atom with profile (R, v_φ, q) in φ according to the corresponding path of the atom with profile (R, v_ψ, q) in ψ. If successful (i.e. if this leads to equivalence to ψ), then this modification explains non-equivalence.

B-2 Swapping Implication Modify φ by swapping the direction of an implication (if one exists). If successful (i.e. if this leads to equivalence to ψ), then this modification explains non-equivalence.

4 Evaluation

In this section, we evaluate effectiveness and efficiency of determining (non-)equivalence, finding counter examples, and computing explanations on a large educational dataset. We address the research questions stated in the introduction:

- RQ1: Can the correctness of student attempts be determined in practice for first-order logic modelling tasks?
- RQ2: In case of an incorrect attempt, can a counterexample witnessing incorrectness be provided?
- RQ3: In case of an incorrect attempt, how can highlevel explanations be computed in practice?

Our data set was collected via an educational support system over four semesters in courses on "Logic in Computer Science" at two universities with > 300 students in each course iteration. The students were tasked with modelling statements given in natural language with first-logic formulas over a given vocabulary, for which an instructor had provided a solution formula and (if applicable) a background theory. The data has been collected for 14 scenarios with in total 62 different statements to be modelled (see extended

[1] This rough description is made precise in the implementation.

Evaluation summary	overall attempts			
	#	%	#dst	%dst
all attempts	125970	100.0%	37159	100.0%
equivalent attempts	27655	21.95%	3019	8.12%
non-equivalent attempts	98315	78.05%	34140	91.88%
Equivalence*				
proven by Vampire (`vampire`, `casc`, `casc_sat` combined)	27654	99.996%	3018	99.97%
Non-Equivalence**				
proven by Vampire (`vampire`, `casc`, `casc_sat` combined)	98315	100.0%	34140	100.0%
counter model found	96820	98.48%	33429	97.92%
via Vampire (finite model building)	91766	93.34%	31465	92.16%
via Random models	79439	80.80%	24694	72.33%
exclusively via Vampire (finite model building)	17381	17.68%	8735	25.59%
exclusively via Random models	5054	5.14%	1964	5.75%
Explanations**				
Explanation by strategy				
S-1 Missing symbol names	25500	25.94%	8052	23.58%
S-2 Permuted arguments of a symbol	429	0.44%	126	0.37%
S-3 Different relation symbol names	594	0.60%	123	0.36%
S-4 Different terms	1016	1.03%	116	0.34%
Q-1 Different quantifier prefixes	1603	1.63%	357	1.05%
Q-2 Different order of quantifiers	1047	1.06%	300	0.88%
Q-3 Different free variables	10171	10.34%	5375	15.74%
G-1 Different guards	11407	11.60%	2038	5.97%
G-2 Wrong guarding operators	7445	7.57%	1019	2.98%
Q-1 + G-1: Different quantifier prefixes and guards	10469	10.65%	1439	4.21%
B-1 Different negation patterns	582	0.59%	110	0.32%
B-2 Swapped implications	554	0.56%	31	0.09%
≥ 1 strategy	51269	52.15%	15610	45.72%

Table 2: Summary of the evaluation. In the experiments, each Vampire invocation was given a timeout of 20 seconds. For only one of the 37,159 formula pairs Vampire did not manage to determine equivalence within this time limit (it took 156 seconds).

* Percentages relative to equivalent attempts.

** Percentages relative to non-equivalent attempts.

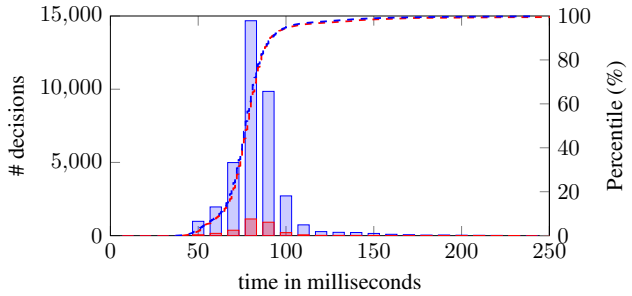

Figure 2: Time in milliseconds Vampire needed to determine whether pairs of formulas are equivalent or non-equivalent (blue) and time needed for equivalent pairs only (red). The blue and red lines show the percentiles for decisions made. Only distinct pairs of formulas are considered here.

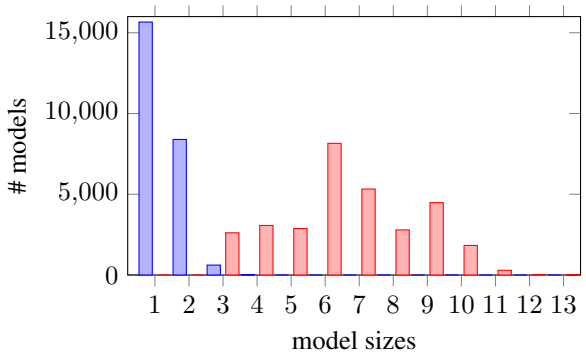

Figure 3: Number of models witnessing non-equivalence of formulas found for each universe size by Vampire (red) and number of models found for each size by the random model generator (blue).

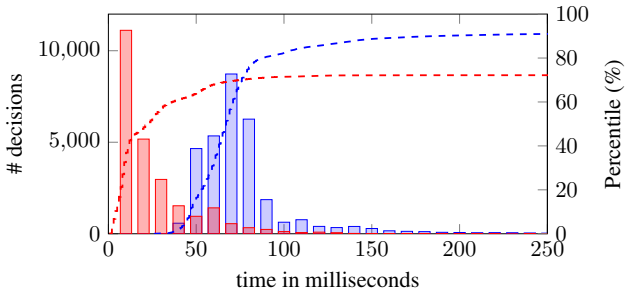

Figure 4: Time needed by Vampire's finite model builder (blue) and by the random model generator (red). The blue and red lines show the percentiles of counter examples found with respect to all distinct non-equivalent pairs of formulas.

version (Vehlken et al. 2026) for an overview as well as an example scenario).

The data set consists of 125,970 tuples $(\mathrm{id}, \Gamma, \psi, \varphi)$, containing an id, background theory Γ, solution formula ψ, and student attempt φ. In our evaluation we measure performance with respect to (i) all such tuples, and (ii) tuples with distinct solution attempts. The number of distinct solution attempts is 37,159. The data set is summarized in the first lines of Table 2.

One reason for the high number of attempts, and in particular incorrect attempts, is likely that the system provides feedback after every attempt and students have an unlimited number of attempts per exercise. Because of the large number of attempts, we used timeouts in our evaluations. Unless stated otherwise, the experiments were run with a timeout of 20 seconds.

4.1 RQ1: Determining Correctness of Student Attempts

Setting. For answering whether correctness of student attempts can be determined in practice, we evaluate (1) equivalence for all pairs of formulas ψ and φ modulo Γ with the theorem prover Vampire, and (2) measure how fast (non-)equivalence can be tested.

This and all other experiments in this paper were run on a small server (16x 2.4 GHz CPU cores, 48 GB RAM). We use version 4.8 of Vampire and the three modes `vampire`, `casc`, and `casc_sat` (Bártek et al. 2025). Preliminary experiments indicated that no single mode is optimal for all our formula pairs. Therefore, we start a *race* between the three different modes by running them in parallel in independent processes. As soon as one returns a decisive answer (indicating equivalence or non-equivalence), this answer is returned and all other processes are terminated.

Results. For all but one of 125,970 formula pairs, (non-)equivalence was determined within the 20 second timeout (see Table 2). For the remaining pair, in a post-test, equivalence was proven by Vampire in 156s. It turns out that for all except 114 pairs ($\approx 0.3\%$), (non-)equivalence could be decided within 200ms, also see Figure 2.

In summary, testing correctness of solution attempts in first-order modelling tasks is practically feasible in educational scenarios.

4.2 RQ2: Finding Counter Examples

Setting. For answering whether counter examples witnessing non-equivalence of formulas can be found in practice, we evaluate (1) for which ratio of pairs Vampire and a random model generator can find counter example models, (2) how fast, and (3) how large the generated counter models are (as a proxy for how useful counter examples are for explaining non-equivalence). We note that both methods only build finite models, so it is possible that counter examples for some of the formula pairs cannot be found this way.

The random model generator was parameterized to generate models of size 1 to 10, the probability of picking a tuple was 0.5 (cf. Section 2). For each universe size m, up to $m \cdot 1000$ models were generated and tested. Vampire was instructed to use its finite model builder via the flag `--saturation_algorithm fmb`.

Results. Counter examples were found for 98.48% of the pairs of non-equivalent formulas by at least one of the methods, with Vampire finding counter examples for 93.34% of all pairs and the random model generator finding counter examples for 80.8% of all pairs (see Table 2). Thus, Vam-

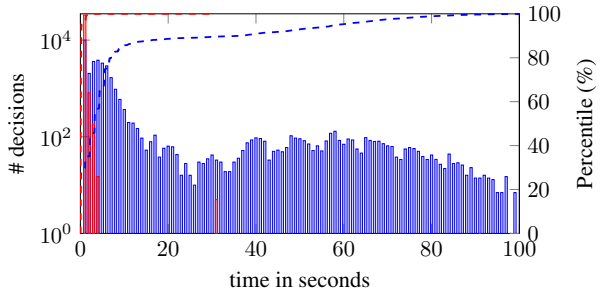

Figure 5: Time in seconds needed to run *all* strategies (sequentially) for each distinct pair of non-equivalent formulas without a cache (blue) and with a cache (red). The blue and red lines show the percentiles for decisions made. Note that the # decisions axis is in logarithmic scale. Individual Vampire calls had a timeout of 30s, but the strategies themselves did not have an explicit timeout.

pire could determine non-equivalence of all non-equivalent pairs, but not derive counter examples for a fraction of formula pairs. There are 17.68% and 5.14% pairs for which solely Vampire or the random model generator, respectively, found a counter example.

For 30,822 out of 34,140 pairs of non-equivalent formulas (90.3%), counter examples were found within 200ms, also see Figure 4.

When both approaches found a model for a pair of formulas, Vampire's model was larger by a factor of 4.0 (median) or 4.5 (average), see also Figure 4.

In summary, counter examples can be found sufficiently efficiently for most pairs of formulas to provide a first explanation of incorrect student attempts.

4.3 RQ3: Computing Highlevel Explanations

Setting. For answering whether explanations for non-equivalence can be computed in practice, we evaluate (1) which ratio of pairs can be explained by our strategies, and (2) how fast. The expectation, guided by our experience in grading, is that while a significant fraction will be explainable by our strategies, there is also a significant fraction of non-sensical solution attempts.

We implemented all strategies as well as a testing framework to apply them to the dataset. Each strategy was applied to every formula pair. In addition to our strategies from Section 3 we also used a strategy Q-1+G-1 that combines modifying differing quantifier prefixes and differing guards. This strategy was added after inspecting pairs that were unexplained by the other strategies.

For determining efficiency in practical scenarios, we ran our experiment also with a cache to store equivalence decisions for pairs of first-order formulas. The cache was initially filled with the equivalence decision of all the 37,159 pairs of formulas. After each Vampire call, the cache was extended with the resulting decisions.

Individual Vampire calls had a timeout of 30s, but the strategies themselves did not have an explicit timeout.

Results. Our strategies explain 52.15% of all formula pairs and 45.72% of distinct pairs. A detailed split for how many (total and distinct) pairs are explained by the individual strategies is provided in Table 2.

For 87% of all pairs, running all strategies sequentially took less than 10 seconds (see Figure 5, the # decisions axis is in logarithmic scale). Using a cache significantly reduces the average runtime from 9.1 seconds without using a cache, to 0.15 seconds, see Figure 5. With cache, all but 200 pairs need at most 2 seconds. This indicates that the time needed to run the strategies is dominated by the time needed to run Vampire. Thus, in addition to exploit that pairs of formulas might occur multiple times, a cache is beneficial in practice as also (different) strategies may test the same pairs of formulas multiple times.

In summary, for more than half of all student attempts, a conceptual explanation can be found. These explanations can also be found fast enough to be provided to students instantly by an educational support system.

We expect that by fine-tuning the current implementation, the ratio of pairs for which conceptual explanations can be provided can be further increased (e.g. by adding further strategies and also combining strategies). Also the time needed to compute explanations can likely be decreased by engineering the used algorithms.

5 Summary, Impact, and Future Work

We designed algorithms for explaining non-equivalence of first-order formulas and evaluated equivalence testing, finding counter models, and the methods for explaining on large educational data sets with > 100.000 pairs of first-order formulas.

Our evaluation shows that for educational tasks on first-order modelling, (1) modern theorem provers like Vampire can easily determine correctness of modelling attempts, also in the presence of background theories; (2) counter example models can easily be computed for most incorrect modelling attempts by combining Vampire's finite model builder and a random model generator; and (3) conceptual explanations for incorrectness can be computed for a large fraction of modelling attempts ($> 50\%$ of attempts in our data set). All computations for (1) – (3) are fast enough to provide feedback instantly and to scale to large cohorts.

Our computational strategies for finding explanations are currently being integrated into a state-of-the-art educational support system for learning logic used by > 1.000 students per year. We plan to analyse the educational data from this integration using our algorithms to better understand how students learn foundations of logics.

Acknowledgements

We thank Jean Christoph Jung for insightful discussions about Beth definability and interpolation in first-order logic. This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 448468041.

AI Declaration

Generative AI has only been used for editorial improvements in the writing of this paper.

References

Bártek, F.; Bhayat, A.; Coutelier, R.; Hajdú, M.; Hetzenberger, M.; Hozzová, P.; Kovács, L.; Rath, J.; Rawson, M.; Reger, G.; Suda, M.; Schoisswohl, J.; and Voronkov, A. 2025. The Vampire diary. In Piskac, R., and Rakamaric, Z., eds., *Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part III*, volume 15933 of *Lecture Notes in Computer Science*, 57–71. Springer.

D'Antoni, L.; Kini, D.; Alur, R.; Gulwani, S.; Viswanathan, M.; and Hartmann, B. 2015. How can automatic feedback help students construct automata? *ACM Trans. Comput.-Hum. Interact.* 22(2).

Ebbinghaus, H.-D.; Flum, J.; Thomas, W.; and Ferebee, A. S. 1994. *Mathematical logic*. Springer.

Gesellschaft für Informatik e. V. 2016. Empfehlungen für Bachelor- und Master-Programme im Studienfach Informatik an Hochschulen. gi.de.

Greenman, B.; Saarinen, S.; Nelson, T.; and Krishnamurthi, S. 2023. Little tricky logic: Misconceptions in the understanding of LTL. *Art Sci. Eng. Program.* 7(2).

Hodges, W. 1993. *Model theory*, volume 42 of *Encyclopedia of mathematics and its applications*. Cambridge University Press.

Kneisel, T.; Vehlken, F.; and Zeume, T. 2025. Logical modelling in CS education: Bridging the natural language gap. In Cristea, A. I.; Walker, E.; Lu, Y.; Santos, O. C.; and Isotani, S., eds., *Artificial Intelligence in Education - 26th International Conference, AIED 2025, Palermo, Italy, July 22-26, 2025, Proceedings, Part IV*, volume 15880 of *Lecture Notes in Computer Science*, 457–463. Springer.

Kovács, L., and Voronkov, A. 2013. First-order theorem proving and Vampire. In Sharygina, N., and Veith, H., eds., *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, 1–35. Springer.

Libkin, L. 2004. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

Reger, G.; Suda, M.; and Voronkov, A. 2016. Finding finite models in multi-sorted first-order logic. In Creignou, N., and Berre, D. L., eds., *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, 323–341. Springer.

Schmellenkamp, M.; Zeume, T.; Argo, S.; Kiefer, S.; Siems, C.; and Stebel, F. 2025. Detecting and explaining (in-) equivalence of context-free grammars. *Proceedings of the ACM on Programming Languages* 9(OOPSLA2):2954–2980.

Schmellenkamp, M.; Latys, A.; and Zeume, T. 2023. Discovering and quantifying misconceptions in formal methods using intelligent tutoring systems. In Doyle, M.; Stephenson, B.; Dorn, B.; Soh, L.; and Battestilli, L., eds., *Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 1, SIGCSE 2023, Toronto, ON, Canada, March 15-18, 2023*, 465–471. ACM.

Schmellenkamp, M.; Schmalstieg, F.; and Zeume, T. 2025. Errors and misconceptions in first-order logic modeling. In Leinonen, J., and Duran, R., eds., *Proceedings of the 25th Koli Calling International Conference on Computing Education Research, Koli Calling 2025, Koli, Finland, November 11-16, 2025*, 11:1–11:11. ACM.

Schmellenkamp, M.; Vehlken, F.; and Zeume, T. 2024. Teaching formal foundations of computer science with Iltis. *Bull. EATCS* 142.

ten Cate, B., and Comer, J. 2025. Interpolation in first-order logic.

Vehlken, F.; Zeume, T.; Carrasco Bustamante, E.; Cornély, M.; and Pradel, L. 2026. Identifying and explaining (non-)equivalence of first-order logic formulas.