Reusing Learning Objects via Theory Morphisms

Michael Kohlhase[0000−0002−9859−6337] and Marcel Schütz[0000−0002−5386−5134]

Computer Science, FAU Erlangen Nürnberg, Germany

Abstract. One of the most important motivations of module systems for formal systems is that statements and objects can be re-purposed in other contexts via the inheritance pathways. In this paper, we show that this idea can be extended to informal settings using an adaptive learning assistant (ALeA) as a concrete use case.
Concretely, we concentrate on repurposing informal definitions, quiz questions, and explanations between the contexts given by different theories in a flexiformal theory graph. Using this, we can now refactor (transport backwards over a theory morphism) e.g. quiz questions like "Is the following formula A satisfiable?" to be logic-independent and utilize them in any logic via forward morphism transport.
This goes a large step towards solving the biggest practical problem in ALeA-like systems: provisioning enough targeted semantically annotated learning objects.

This paper uses SᴛᴇX3. The semantically annotated XHTML version of this paper is available at `https://url.mathhub.info/relocalization/`

1 Introduction

STEM higher education is increasingly confronted with the problem of ensuring high quality teaching in view of a growing diversity of students. As it is not uncommon that university courses are filled with hundreds of students that completed different previous study programs at different universities in different countries it becomes a nearly impossible task for teaching staff to respond to all individual learning needs of their students without the help of computer-supported learning services.

The ALeA[1] system aims to provide individualized learning support services using methods from mathematical knowledge management (MKM) to solve this problem. ALeA generates learner-specific course materials from semantically annotated document fragments and monitors learner's interactions to gain data for learner-adaptive content generation. Learning support services include "definitions-on-hover", flashcard-based training drills, competency-adaptive section preparation drills and review problems section ends ("postparation"), and learner-adaptive explanations for all content words.

[1] Adaptive Learning Assistant, an intelligent tutoring system deveoped as part of the VoLL-KI [Koh+23] project; see [Ber+23].

The ALeA system has been in active use for three semesters for multiple courses; in particular a two-semester introductory course *Artificial Intelligence* (AI) with over 500 students per semester (see [VKC]). Predictably, the main bottleneck in practice has been the provisioning of a sufficient quantity and variety of secondary course material, in particular auto-gradable training/practice/evaluation problems that are annotated with competency preconditions and learning objectives so that they can be automatically deployed for pre/postparation.

Idea: Fully Utilizing the Flexiformal Theory Graph In this paper we explore a possible solution to this problem, which also has the potential to enhance the precision of the learner-adaptive services in ALeA: To make full use of the advanced inheritance mechanisms in the MMT[2] theory graphs that underlie the semantic annotations, which in turn drive ALeA services.

Theory graphs – theories (sequences of typed constant declarations/definitions), interconnected with theory morphisms (truth-preserving mappings between theories) – employ inclusion morphisms for modular representation and object-oriented inheritance, and ALeA has used this productively for knowledge management. But the theory graph formalism also provides view morphisms for inheritance modulo re-interpretation. In the formal setting, the importance of views is that they allow to inherit formal objects like theorems between theories and thus exponentiate their usefulness; see [RK13; Far18] for details.

If we could extend this to the setting of flexiformal theory graphs underlying ALeA, then powerful morphisms can be used to generate context-specific learning interactions and even generalize learning objects such as practice problems in much the same manner that theorems are generalized in mathematics: by identifying the necessary concepts and expressing them in the most general context so that they can be inherited in all specific contexts.

Contribution. The most critical technical problem to be solved for this idea is to generalize formal theorem inheritance via views twice: (1) from the formal to an informal setting – a must in the educational setting, and (2) from theorems to arbitrary learning objects like problems, examples, or explanations. ALeA simplifies the first generalization by building on flexiformal – partial formalization up to flexible formality – representations of learning objects in STEX[3] as a basis. To the best of our knowledge there is no previous work in this direction. Thus this paper can also be seen as a further exploration of the flexiformal continuum to determine formalization primitives needed to provide advanced MKM services in practice.

Overview. Section 2 sets up a running example that we will use to motivate and explain the learning object generation (recontextualization) via views. Section 3 presents how the semantic annotations in STEX already allow the template-based

[2] Meta Meta Tool, a software system for generic knowledge management services; see [RK13; MMT].

[3] A variant of LaTeX that allows flexiformal semantic annotations; see [CICM22c].

recontextualization of various learning objects. Section 4 concludes the paper and discusses future work, in particular how to deal with the intricacies of natural language.

Acknowledgments. The work reported in this article was conducted as part of the VoLL-KI project (see `https://voll-ki.de`) funded by the German Research/Education Ministry under grant 16DHBKI089. We gratefully acknowledge LaTeX help from Dennis Müller and fruitful discussions on the natural language aspects by Jan Frederik Schaefer.

2 Motivating Example

In the AI lecture that has been the primary experimentation ground for the ALeA system we introduce propositional logic as a domain description language for rational agents. Here is what a definition could look like:

Definition 1. Let $\Sigma{:=}\{\neg, \wedge, \Rightarrow, c_1, \ldots, c_n\}$ be a propositional signature, then the **formulae** of propositional logic are $\mathcal{L}_{\mathrm{PL}^0}{:=}\mathit{wfe}(\Sigma)$.

We call $\mathcal{I}\colon \Sigma \to \bigcup_{n=0}^{\infty}(\mathbb{B}^n \to \mathbb{B})$ a **model** of propositional logic, iff $\mathcal{I}(f)\colon \mathbb{B}^k \to \mathbb{B}$ for any k-ary symbol $f \in \Sigma$ and

- $\mathcal{I}(\neg)(x) = \mathsf{T}$ iff $x = \mathsf{F}$,
- $\mathcal{I}(\wedge)(x, y) = \mathsf{T}$ iff $x = \mathsf{T}$ and $y = \mathsf{T}$, and
- $\mathcal{I}(\Rightarrow)(x, y) = \mathsf{T}$ iff $x = \mathsf{F}$ or $y = \mathsf{T}$.

We denote the set of models with $\mathcal{K}_{\mathrm{PL}^0}$.

Here, $\neg$ is a unary term constructor, $\wedge$ and $\Rightarrow$ are binary term constructors, and $c_1, \ldots, c_n$ are propositional constants, i.e. nullary term constructors.[4] Moreover, $\mathit{wfe}(\Sigma)$ denotes the set of all well-formed terms that can be generated from Σ. Further, $\mathbb{B}$ denotes the set $\{\mathsf{T}, \mathsf{F}\}$ of truth values. Note that both $\mathcal{L}_{\mathrm{PL}^0}$ and $\mathcal{K}_{\mathrm{PL}^0}$ as well as the whole concept *propositional logic* depend on Σ which can, however, be assumed to be clear from the context – after all Definition 1 addresses human readers and thus does not necessarily have to state all such dependencies explicitly to be understandable by the intended audience.

2.1 Generalization and Exemplification

Consider the following (conservative) extension of the concept of propositional logic that comprises a definition of a satisfaction relation $\models_{\mathrm{PL}^0}$.

Definition 2. Let $\mathcal{I}$ be a model and $\mathbf{A}$ a formula of propositional logic. $\mathcal{I} \models_{\mathrm{PL}^0} \mathbf{A}$ iff $\mathcal{I}(\mathbf{A}) = \mathsf{T}$.

This allows us to regard propositional logic as an instance of the abstract concept of a logical system:

[4] Note that this is just one of many possible choices for a propositional signature.

Definition 3. A **logical system** (or simply a **logic**) is a triple $\mathcal{L}:=\langle \mathcal{L}, \mathcal{K}, \models \rangle$, where $\mathcal{L}$ is a formal language, $\mathcal{K}$ is a set, and $\models \subseteq \mathcal{K} \times \mathcal{L}$. Members of $\mathcal{L}$ are called **formulae** of $\mathcal{L}$, members of $\mathcal{K}$ **models** for $\mathcal{L}$, and $\models$ the **satisfaction relation**.

In the AI lecture we do not show students the concept of logical systems. But in more logic-oriented courses where we also teach modular logic development in MMT/OMDoc (see LBS and KRMT on [VKC]) we do – sharing many of the learning objects between courses. The fact that propositional logic is a logical system can either be presented as a theorem

Theorem 1. *Propositional logic naturally forms a logical system* $\langle \mathcal{L}_{PL^0}, \mathcal{K}_{PL^0}, \models_{PL^0} \rangle$.
Proof sketch: Indeed, the components of the abstract logical system $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ can be directly interpreted as those of $\langle \mathcal{L}_{\mathrm{PL}^0}, \mathcal{K}_{\mathrm{PL}^0}, \models_{\mathrm{PL}^0} \rangle$.

or as an example – propositional logic is an example of a logical system – with nearly the same text depending on the course context.

Example 1. Propositional logic naturally forms a logical system $\langle \mathcal{L}_{\mathrm{PL}^0}, \mathcal{K}_{\mathrm{PL}^0}, \models_{\mathrm{PL}^0} \rangle$.

Arguably both variants should be generated from the same source – after all they comprise just two different ways the assertion that propositional logic forms a logical system is *presented*; subsection 3.1 shows how to do that.

2.2 Recontextualizing Statements

In the AI lecture, where propositional logic is used as the paradigmatic example of a logic, we introduce the notion of satisfiability next:

Definition 4. Let $\mathbf{A} \in \mathcal{L}_{\mathrm{PL}^0}$ be a formula.

- A model $\mathcal{I} \in \mathcal{K}_{\mathrm{PL}^0}$ **satisfies** $\mathbf{A}$ iff $\mathcal{I} \models_{\mathrm{PL}^0} \mathbf{A}$.
- $\mathbf{A}$ is **satisfiable** iff there exists a model that satisfies $\mathbf{A}$.

But in the domain model of the ALeA system, i.e. the representation of the knowledge that the courses offered in ALeA comprise, we already have the following definition of satisfiability at the abstract logical systems level:

Definition 5. Let $\langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system and $\mathbf{A} \in \mathcal{L}$ be a formula.

- A model $\mathcal{M} \in \mathcal{K}$ **satisfies** $\mathbf{A}$ iff $\mathcal{M} \models \mathbf{A}$.
- $\mathbf{A}$ is **satisfiable** iff there exists a model that satisfies $\mathbf{A}$.

This should make Definition 4 redundant, since it is just an adaptation of Definition 5 given Theorem 1. And indeed we will see in subsection 3.2 how Definition 4 can be computed automatically from Definition 5. We think of this as recontextualizing Definition 5 in the concrete context of propositional logic.

2.3 Recontextualizing Problems

In the ALeA system we want to provide practice problems to reinforce and train application of the new concepts at the end of each section. For satisfiability in propositional logic, this might be of the following form:

Problem 2.1 (Satisfiability)
Is the formula $c_1 \wedge (c_2 \Rightarrow \neg c_1)$ satisfiable? ☑ Yes ☐ No

If a student selects the wrong answer "No", the system might give the following – admittedly very verbose – feedback:

Actually, there is a model $\mathcal{I}$ that satisfies $c_1 \wedge (c_2 \Rightarrow \neg c_1)$: it maps c_1 to T and c_2 to F. Then $\mathcal{I}(c_1 \wedge (c_2 \Rightarrow \neg c_1)) = \mathsf{T}$: Indeed, this can directly be seen by evaluating the truth table for $c_1 \wedge (c_2 \Rightarrow \neg c_1)$.

Of course, in most educational settings we need many such problems, and in the ALeA system, which automatically deploys training problems in various places, even more. Current practice is to either manage collections of variants with different formulae or to generate them automatically. In the first approach change and consistency management become a major practical concern, whereas systems for the latter tend to be specific to very narrow domains (see e.g. [ÁNP23] for a recent example).

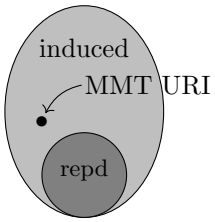
3 Recontextualizing Learning Objects

We now show how the learning objects from the running example from Section 2 can be **recontextualized**, i.e automatically adapted to a new conceptual context.

In ALeA, all learning objects – entities that may be used for learning, education or training – are represented in sTeX [CICM22c]: a variant of LaTeX that allows flexiformal semantic annotation and the generation of semantic HTML [CICM22b] that can be presented and adapted to learners' competences and preferences based on a learner model – see [Ber+23] for details. The underlying domain model is structured as a theory graph whose nodes are theories – sequences of formal declarations for domain constants and objects – and the edges are theory morphisms – truth-preserving mappings between the (formal) languages induced by the declarations. Theory morphisms come in three forms: inclusions for object-oriented inheritance, structures for named inheritance, and views as interpretations between pre-existing theories, where the focus of this paper is on the latter; see [RK13; Far18] for details.

In any case, morphisms in theory graphs induce recontextualized objects (the light gray area in the diagram below on the left) from the represented theory graph (the dark gray area). In the Mmt system [MMT], which provides knowledge management services to ALeA, a recontextualized object is identified by an MMT URI from which (together with the represented theory graph) its content can be recovered automatically. For instance, a learning object s (e.g. a

definition) over a theory S induces a recontextualized learning object $\psi(s)$ over a theory T via a view $\psi : S \rightsquigarrow T$. If S has MMT URI `g?S`, where `g` is the base URI of the document that contains S and s has MMT URI `g?S?s`, then $\psi(s)$ has the MMT URI $[\![\psi]\!]$`?g?S?s`, where $[\![\psi]\!]$ denotes the MMT URI of ψ; see [RK13] for details.



All of this is well-understood for formal MMT content. In this section we show how we can extend this to flexiformal STEX/iMMT content, where iMMT is an extension of the MMT format for representing flexiformal knowledge for which STEX acts as an interface, to make recontextualization work in the ALEA system. For now, we concentrate on how to combine STEX fragments during recontextualization, and leave the aspect of dealing with the intricacies of natural language to future work.

The STEX sources for this can be found at `https://gl.mathhub.info/Papers/24-cicm-views-in-alea/-/tree/main/source/corpus`.

3.1 Presenting Views

In Section 2 we presented propositional logic as a concrete instance of a logical system. Here we have a closer look at how the abstract notion of logical system and of propositional logic give rise to two flexiformal iMMT theories together with a view that models the "is a more concrete instance of" relation between these definitions.

theory: Logical System
include: Formal Language
$\mathcal{L}$: formal language
$\mathcal{K}$: set
$\models : \mathcal{P}(\mathcal{K} \times \mathcal{L})$

Fig. 1. Theory L

Definition 3 is based on the iMMT theory L in Fig. 1, this has three declarations of the form $c : \tau = \delta$, where c is a constant name, τ a type, and δ a definiens – the latter two are optional.

theory: Propositional Logic
include: Formal Language
include: Boolean Model
$\mathcal{L}_{\mathrm{PL}^0} = \mathit{wfe}(\Sigma)$
$\mathcal{K}_{\mathrm{PL}^0} = \{\mathcal{I} \colon \Sigma \to \bigcup_{k=0}^{\infty} \mathbb{B}^k \to \mathbb{B} \mid \mathcal{I} \text{ is a model of } \Sigma\}$
$\models_{\mathrm{PL}^0} = \mathcal{I} \models_{\mathrm{PL}^0} A$ iff $\mathcal{I}(A) = \mathsf{T}$

Fig. 2. Theory P

Definition 1 yields the iMMT theory P in Fig. 2. Again, we have three declarations, this time without types, but with definientia. The first definiens (as well as the types in Fig. 1) is fully formal, whereas the other two definientia are flexiformal as they comprise fragment of mathematical vernacular (indicated by a gray background).[5]

Now, interpreting the constants from L in the "obvious" way in P (i.e. $\mathcal{L}$ as $\mathcal{L}_{\mathrm{PL}^0}$, $\mathcal{K}$ as $\mathcal{K}_{\mathrm{PL}^0}$, and $\models$ as $\models_{\mathrm{PL}^0}$) we get the desired view $\varphi : L \rightsquigarrow P$ as depicted in Fig. 3[6].

[5] Note that in iMMT types can be flexiformal as well.

[6] We implicitly assume φ to be the identity on the types *formal language* and *set* as well as on the type constructors $\mathcal{P}$ and $\times$.

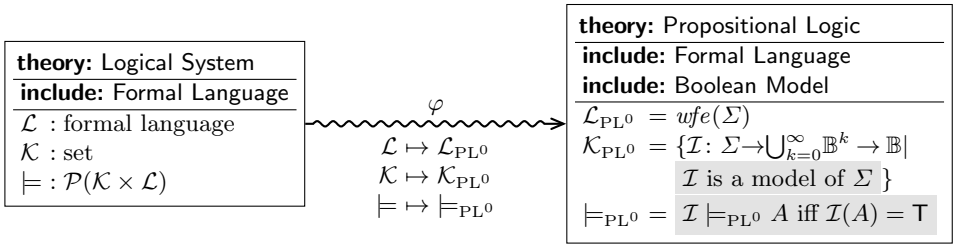


Fig. 3. View $\varphi : L \rightsquigarrow P$

The morphism condition in MMT is that the mapping φ is well-typed, i.e. if any constant c in L has type τ then $\varphi(c)$ has type $\varphi(\tau)$, which – in a formal setting – can be automatically established by the MMT type-checker. In iMMT, we have to be a bit more careful: in an assignment $c \mapsto A$ in φ both the type τ as well as the definiens A can be partially informal, so that the type-checker cannot establish well-typedness. Thus in the case of τ or A being partially informal we assume a **morphism obligation** of the form A is a $\varphi(\tau)$ per assignment. In a nutshell, partially informal assignments in views induce (informal) morphism obligations, whereas formal assignments induce (formal) morphism conditions that can be discharged automatically.[7]

We do not show the proofs for the morphism obligations from the view in Fig. 3 – $\mathcal{L}_{\mathrm{PL}^0}$ is a formal language and $\mathcal{K}_{\mathrm{PL}^0}$ is a set – since they are trivial and deemed below the level that needs to be explained.

With these morphism obligations, we can already see how the view φ from Fig. 3 can be presented as an example or a theorem automatically. Consider the code snippets in Fig. 4 that comprise the STEX sources of Theorem 1 and Example 1.
This code can directly be generated from the components φ and the STEX templates shown in Fig. 5. Here, where the parenthesized expressions below refer to the respective components of φ,

1. $\langle$sc$\rangle$ denotes the source concept (logical system),
2. $\langle$tc$\rangle$ denotes the target concept $\varphi(\langle$sc$\rangle)$ (propositional logic),
3. $\langle$ss$\rangle$ denotes the source structure $(\langle \mathcal{L}, \mathcal{K}, \models \rangle)$,
4. $\langle$ts$\rangle$ denotes the target structure $\varphi(\langle$ss$\rangle)$ $(\langle \mathcal{L}_{\mathrm{PL}^0}, \mathcal{K}_{\mathrm{PL}^0}, \models_{\mathrm{PL}^0} \rangle)$.

Instantiating the macros $\langle$sc$\rangle$, $\langle$tc$\rangle$, $\langle$ss$\rangle$ and $\langle$tc$\rangle$ in the two templates yields the STEX code shown in Fig. 4.

[7] Note that there is a subtle difference between *informal views* and *formal views with informal assignments*: The latter consist of a (formal) listing of assignments of the form $c \mapsto A$ in which the expression A might be (partially) informal. The former, on the other hand, is characterized by a lack of such a formal structure and comprises natural language descriptions of such assignments instead.

In the same vein, *formal theories* are characterized by a (formal) listing of constant declarations of the form $c : \tau = \delta$, where the type τ and the definiens δ might be (partially) informal. In contrast, *informal theories* comprise natural language descriptions of such declarations instead.

```
\begin{sassertion}[style=theorem]
  \Sn{propositional logic} naturally forms a \sn{logical system}
  $\mathstruct{\plang,\pmods,\psat!}$.
\end{sassertion}
\begin{spfsketch}
  Indeed, the components of $\mathstruct{\plang,\pmods,\psat!}$
  can be directly interpreted as those of the abstract
  \sn{logical system} $\mathstruct{\flang,\mods,\sat!}$.
\end{spfsketch}
```

```
\begin{sexample}[for=logical system]
  \Sn{propositional logic} naturally forms a \sn{logical system}
  $\mathstruct{\plang,\pmods,\psat!}$.
\end{sexample}
```

```
\begin{sassertion}[style=theorem]
  \Sn{⟨tc⟩} naturally forms a \sn{⟨sc⟩} $⟨ts⟩$.
\end{sassertion}
\begin{spfsketch}
  Indeed, the components of $⟨ts⟩$
  can be directly interpreted as those of the
  abstract \sn{⟨sc⟩} $⟨ss⟩$.
\end{spfsketch}
```

```
\begin{sexample}[for=⟨sc⟩]
  \Sn{⟨tc⟩} naturally forms a \sn{⟨sc⟩} $⟨ts⟩$.
\end{sexample}
```

Fig. 5. LaTeX Templates for Theorem 1 and Example 1

3.2 Recontextualizing Mathematical Statements

In Definition 4 from Section 2 we have seen how a logical system can be extended by a notion of satisfiability. Such a (conservative) extension of the theory L of logical systems can be modeled as an inclusion morphism $\iota : L \hookrightarrow \widetilde{L}$, where $\widetilde{L}$ is the extension of L shown in Fig. 7[8].

The two definitions of *satisfies* and *satisfiable* from $\widetilde{L}$ can directly be translated by mapping all symbols of L they depend on along φ to corresponding definitions of these concepts in the context of propositional logic. Up to variable renaming, this then yields the extension $\widetilde{P}$ of the theory P shown in Fig. 8.

Note that this "variable renaming" $\mathcal{M} \mapsto \mathcal{I}$ has only notational, but no semantic relevance and should thus be something that any consumer (e.g. student) or supplier (e.g. lecturer) of course material should be able to alter according to their own preferences. $\widetilde{P}$ – together with the inclusion morphism $P \hookrightarrow \widetilde{P}$ and the view $\widetilde{\varphi} : \widetilde{L} \rightsquigarrow P$ that extends φ by map-

theory: Satisfiability in Logical System	
include: Logical System	
satisfies	= A model $\mathcal{M} \in \mathcal{K}$ satisfies **A** iff $\mathcal{M} \models \mathbf{A}$.
satisfiable	= **A** is satisfiable iff there exists a model that satisfies **A**.

Fig. 7. Theory $\widetilde{L}$

[8] Here and below we assume A to be declared as a variable of type *formula*.

```
1 \begin{sdefinition}
2   \vardef{Avar}{\mathbf{A}}\vardef{Mvar}{\mathcal{M}}
3   %
4   Let \sdec[logsys]{$\logsys{}{}$ be a \sn{logical system} and}
5   \sdec[Avar]{$\inset\Avar\flang$ be a \sn{formula}.}
6   \begin{itemize}
7     \item \definiens[satisfies]{A \sn{model} $\inset\Mvar\modcls$
8       \definame{satisfies} $\Avar$ iff $\satrel\Mvar\Avar$}.
9     \item \definiens[satisfiable]{$\Avar$ is \definame{satisfiable}
10      iff there exists a \sn{model} that \sn{satisfies} $\Avar$}.
11  \end{itemize}
12 \end{sdefinition}
```

Fig. 6. Satisfiability for Logical Systems

ping the symbols *satisfies* and *satisfiable* defined in $\widetilde{L}$ to their translations in $\widetilde{P}$ – is in fact the pushout of the span given by ι and φ (cf. [RK13]).

theory: Satisfiability in Propositional Logic	
include: Propositional Logic	
satisfies	= A model $\mathcal{I} \in \mathcal{K}_{\mathrm{PL}^0}$ satisfies **A** iff $\mathcal{I} \models_{\mathrm{PL}^0}$ **A**.
satisfiable	= **A** is satisfiable iff there exists a model that satisfies **A**.

Fig. 8. Theory $\widetilde{P}$

We can now use $\widetilde{\varphi}$ to recontextualize Definition 5 to Definition 4 by transferring its annotated LaTeX sources shown in Fig. 6: All un-annotated text in it is transferred verbatimly into the target document, the source constants `\flang` ($\mathcal{L}$), `\modcls` ($\mathcal{K}$), `\satrel` ($\models$), `\logsys{}{}` ($\langle \mathcal{L}, \mathcal{K}, \models \rangle$), `satisfies` (satisfies) and `satisfiable` (satisfiable), as well as the variable declarations `\Avar` (A) and `\Mvar` ($\mathcal{M}$) are translated via $\widetilde{\varphi}$. Note that the source theory localization (the `\sdec[logsys]{...}` in line 4) is not transferred as the targets of all components of `\logsys{}{}` are *defined* symbols in $\widetilde{P}$; the local declaration in line 5 however is.

The source code of the transferred sources are shown in Fig. 9. Formatting Fig. 9 yields exactly the content of Definition 4.

```
1  \begin{sdefinition}
2    \vardef{Avar}{\mathbf{A}}\vardef{Ivar}{\mathcal{I}}
3    %
4    Let $\inset\Avar\plang$ be a \sn{proplog?formula}.
5    \begin{itemize}
6      \item \definiens[satisfies]{A \sn{proplog?model}
7        $\inset{\Ivar}{\pmods}$ \definame{satisfies} $\Avar$ iff
8        $\psat\Ivar\Avar$}.
9      \item \definiens[satisfiable]{$\Avar$ is \definame{satisfiable}
10       iff there exists a \sn{proplog?model} that
11       \sn{proplog?satisfies} $\Avar$}.
12   \end{itemize}
13 \end{sdefinition}
```

Fig. 9. Satisfiability for Propositional Logic

3.3 Recontextualizing Problems/Solutions/Feedback

As mentioned in Section 2 we not only want to be able to automatically generate concrete instances of abstract definitions which we described in the last paragraph, but also to automatically generate concrete exercises from abstract templates. Problem 2.1 in subsection 2.3, for instance, can be regarded as such a concrete instance of the following template which provides an abstract exercise about satisfiability in logical systems.

Problem 3.1
Is the formula F satisfiable? ☑ Yes ☐ No

Feedback: Actually, there is a model $\mathcal{M}$ that satisfies F: ⟨$\mathcal{M}$.definiens⟩ Then ⟨SatLogSys.conclusion⟩: ⟨SatLogSys.definiens⟩

This template depends on the (non-conservative) extension L_{ex} of $\widetilde{L}$ shown in Fig. 10 which declares two symbols F and $\mathcal{M}$ of types *formula* and *model*, respectively, and a proposition SatLogSys of type $\vdash \mathcal{M} \models F$[9] to force the answer "Yes" to be correct. We would of course also make an extension and template for the answer "No", but WLoG concentrate on the "Yes" case here.

theory: Exercise Template	
include: Logical System	
F	: formula
$\mathcal{M}$	: model
SatLogSys	: $\vdash \mathcal{M} \models F$

Fig. 10. Theory L_{ex}

Moreover, the template contains additional macros ⟨c.definiens⟩ which, for a constant c, is to be replaced by the definiens of c (or to its proof in case c is a proposition), and ⟨p.conclusion⟩ which, for a proposition p of type $\vdash\tau$, is to be replaced by its conclusion τ.

Provided with such an abstract template we can generate a concrete exercise out of it by just specifying an arbitrary view from L_{ex} to any target theory that provides appropriate "instantiations" of the symbols F and $\mathcal{M}$ as well as of the proposition SatLogSys. For instance, to generate Problem 2.1 from Section 2 out of the above template, consider the extension P_{ex} of $\widetilde{P}$ shown in Fig. 11.

If we now define an appropriate view $\psi : L_{\text{ex}} \rightsquigarrow P_{\text{ex}}$ – in our case by extending $\widetilde{\varphi}$ with the assignments $F \mapsto (c_1 \wedge (c_2 \Rightarrow \neg c_1))$, $\mathcal{M} \mapsto \mathcal{I}$, and SatLogSys $\mapsto$ SatProp – we have everything at hand to automatically generate Problem 2.1 from the above template by translating all annotated symbols via ψ and replacing the macros ⟨$\mathcal{M}$.definiens⟩, ⟨SatLogSys.conclusion⟩ and ⟨SatLogSys.definiens⟩ accordingly.

By varying the constants/propositions and their definientia/proofs in P_{ex} we get various variants of Problem 2.1 almost for free. For theories as simple as propositional logic it is even possible to generate the contents of the concrete exercise theory – in our case satisfiable formulas – automatically which reduces the amount of human interaction in the exercise generation pipeline further. But even if such an automation of generating the contents of the concrete exercise

[9] Note that we make use of the propositions-as-types paradigm here: $\vdash \mathcal{M} \models F$ should be read as the theory axiom stating that $\mathcal{M}$ satisfies F.

theory: Exercise for Propositional Logic			
include: Propositional Logic			
c_1	: formula		
c_2	: formula		
$\mathcal{I}$	: model	=	It maps c_1 to T and c_2 to F.
`SatProp`	: $\vdash \mathcal{I} \models_{\mathrm{PL}^0} (c_1 \wedge (c_2 \Rightarrow \neg c_1))$	=	As $\mathcal{I}(c_2) = \mathsf{F}$ we have $c_2 \Rightarrow \neg c_1 = \mathsf{T}$ and hence, as $\mathcal{I}(c_1) = \mathsf{T}$, we get $\mathcal{I}(c_1 \wedge (c_2 \Rightarrow \neg c_1)) = \mathsf{T}$.

Fig. 11. Theory P_{ex}

theory is not possible, we get a tremendous reduction of time that a human has to spend to create such exercises.

theory: Exercise for First-Order Logic			
include: First-Order Logic			
include: Sine			
c	: constant		
f	: unary function symbol		
R	: binary relation symbol		
$\mathfrak{M}$	: model	=	Take $\mathbb{R}$ as its underlying set and map c to π, f to sin, and R to $>$.
`SatFol`		=	As $\pi > \sin(\pi)$, both $R(c,f(c))$ and $(\forall x.R(f(x),x)) \Rightarrow \neg R(c,f(c))$ are true in $\mathfrak{M}$.

Fig. 12. Theory F_{ex}

Using compositionality of views we can, furthermore, re-use existing views between an "abstract" exercise template and a "concrete" exercise theory like ψ above by "re-instantiating" the constants/propositions in their target theories. For instance, assume that we have a theory of first-order logic F together with an appropriate view $L \rightsquigarrow F$[10]. Then, similar to the extension of P by P_{ex} we can extend F to a theory F_{ex} like the one shown in Fig. 12.

By defining a view $\chi : P_{\mathrm{ex}} \rightsquigarrow F_{\mathrm{ex}}$ via $c_1 \mapsto R(c,f(c))$, $c_2 \mapsto (\forall x.R(f(x),x))$, and $\mathcal{I} \mapsto \mathfrak{M}$, we get an instance of our abstract satisfiability exercise for first-order logic via the composition $\chi \circ \psi$ as shown in Fig. 13.

3.4 Implementation

So far we have seen how we can present views as a theorem or an example and how to recontextualize statements and problems along views. Now, we see this can be automated. An implementation of this in Haskell can be found at `https://gitos.rrze.fau.de/voll-ki/fau/system/relocalization/`

In all three generation scenarios we presuppose a representation of the respective view in STEX. In the case of the view $\varphi : L \rightsquigarrow P$ from our running example this representation has the following form:

[10] Note that such a view can automatically be extended to a view $\widetilde{L} \rightsquigarrow F$ by the pushout operation described above.

Problem 3.2
Is the formula $R(c,f(c)) \wedge ((\forall x.R(f(x),x)) \Rightarrow \neg R(c,f(c)))$ satisfiable?
☑ Yes ☐ No

Feedback: Actually, there is a model $\mathfrak{M}$ that satisfies $R(c,f(c)) \wedge ((\forall x.R(f(x),x)) \Rightarrow \neg R(c,f(c)))$: Take $\mathbb{R}$ as its underlying set and map c to π, f to sin, and R to $>$. Then $\mathfrak{M} \models_{\mathrm{PL}^1} (R(c,f(c)) \wedge ((\forall x.R(f(x),x)) \Rightarrow \neg R(c,f(c))))$: As $\pi > \sin(\pi)$, both $R(c,f(c))$ and $(\forall x.R(f(x),x)) \Rightarrow \neg R(c,f(c))$ are true in $\mathfrak{M}$.

Fig. 13. Satisfiability Exercise for First-Order Logic

```
\begin{extstructure*}{proplog}
  \begin{realization}{logsys}
    \assign{flang}{\plang}
    \assign{modcls}{\pmods}
    \assign{satrel}{\psat!}
  \end{realization}
\end{extstructure*}
```

It declares a view from L (`logsys`) to P (`proplog`) that comprises three assignments $\mathcal{L} \mapsto \mathcal{L}_{\mathrm{PL}^0}$ (`\assign{flang}{\plang}`), $\mathcal{K} \mapsto \mathcal{K}_{\mathrm{PL}^0}$ (`\assign{modcls}{\pmods}`) and $\models \mapsto \models_{\mathrm{PL}^0}$ (`\assign{satrel}{\psat!}`).

Assuming, that such a representation of the respective view lives in a sufficiently "tame" fragment of the LaTeX language, we can use standard LaTeX parsing libraries to parse it in an abstract syntax tree (AST). In our implementation we used Haskell's HaTeX library [Cas23] for that purpose. Having such an AST of the representation of a view like φ it is a straightforward task to extract its source and target concepts (`logsys` and `proplog`, respectively) as well as the components of the source and target structures (`flang`, `modcls`, `satrel` and `plang`, `pmods`, `psat!`, respectively).

In the case of representing φ as a theorem or an example, we can use standard find-and-replace procedures to extract the relevant components from such an AST and substitute the respective placeholders ($\langle$sc$\rangle$, $\langle$tc$\rangle$, $\langle$ss$\rangle$ and $\langle$ts$\rangle$) in the templates described in subsection 3.1.

In case of recontextualizing definitions like those of *satisfies* and *satisfiable* we parse their STeX sources (see Fig. 6) also in an AST and substitute all relevant components within this AST with their (previously extracted) images under φ. Moreover, we allow to specify explicit variable renamings in the user interface of our implementation which are taken into account during the substitution process. The resulting AST is then finally rendered as an STeX document, again using methods provided by the HaTeX library.

Essentially the same is done for recontextualizing problems, despite that we additionally have to run the previously described process in between to recontextualize all those definitions that are required in the recontextualized exercise but not provided by the target theory (like that of *satisfiable* in Problem 2.1).

The advantage of implementing the template instantiation process at the STeX level is that this gives us a very direct realization of the ideas presented

above, and gives us sTeX sources that can manually be further polished. The disadvantage is that the generated sTeX must still be formatted into SHTML for processing. An advantage is to implement a the same ideas at the level of SHTML, the semantically annotated HTML+MathML generated from the original sTeX– as all the structure persists in the SHTML this is analogous. This would shorten the processing pipeline and would simplify on-the-fly processing, but would not allow manual polishing.

4 Conclusion & Future Work

We have presented a novel method for using the structural information encoded in theory graphs for content generation in an adaptive learning assistant: Extending the functionality of MKM operations on flexiformal theory graphs, we can recontextualize learning objects along views – morphisms that contain non-trivial language mappings. Especially for fine-grained domain models with large numbers of views this considerably increases the practical impact of learning objects. This is especially effective in the educational arena, where informal, but semantically interpretable learning objects dominate.

The work reported here is an important step towards the better utilization of explicit semantics in active document systems like ALeA, but much remains to be done!

Practical Challenges on Domain Models For the future, we will probably want to make views (and also structures) into primary learning objects and correspondingly design learning support services to help learners remember, understand, and apply them better. Arguably, mastering theory morphisms at all cognitive dimensions [AK09] is a key component in mastering STEM domains beyond mere rote learning.

Currently, the ALeA domain models do not contain many views and structures since they were insufficiently supported in the sTeX format and software ecosystem, and before the work reported in this paper, there were few incentives for adding them. Regarding views this has changed with the work reported in this paper; elaborating corresponding results for structures to increase their practical value within the sTeX ecosystem still remains future work though. Updating domain models for ALeA will be a major investment, but we expect that this will bring the learner support to a completely new level. Additionally, we expect that the investment can be recouped by the additional sharing opportunities.

Dealing with Mathematical Vernacular In our development of recontextualization we have concentrated on the logistics of assembling text fragments into a coherent – and context-adequate – narration. We have however glossed over the fact that template-based natural language generation (NLG) does not – in general – yield grammatically correct and natural-sounding language, since it cannot account for phenomena like inflection, agreement on case or number, and similar. For instance if we want to specialize a solution template of the form, e.g.,

"`there are` ⟨count⟩ `unifier ...`", the word `unifier` and the article `are` need to be adapted (singular or plural) depending to the value of the placeholder ⟨count⟩ during recontextualization.

In the educational setting we can either live with these problems – after all the alternative to course materials with grammar flaws would be *no* course materials – or we can use the generated materials as raw material to be manually (or automatically) polished. Or we can try to fix them by implementing better NLG methods.

Arguably, current (statistical and neural) NLP are not suited, since they are too imprecise and/or the underlying language models have not been trained on mathematical vernacular. We expect that using symbolic NLP methods rather than the simple fragment replacements we currently employ in flexiformal recontextualization will result in higher-quality learning objects and thus enhance the practical value of the methods reported here.

We have carried out extensive experiments with parsing mathematical vernacular in the Grammatical Framework GF [Ran04] and using it as a front-end to the MMT system (see [Sch20; SAK20]). This suggests that much of the mathematical vernacular in practical use can be analyzed syntactically and transformed into MMT/OMDoc where it can be manipulated via the methods outlined above; from this we can then generate grammatical natural language – GF is a bi-directional parsing/generation framework. Practically evaluating coverage and extending the GF grammars as well as the semantics construction views transcends the scope of this paper.

Flexiformal Definition Expansion Another direction in which the practical utility of the methods presented in this paper could be expanded is to allow automatic definition expansion during the recontextualization processes.

Recall for instance that the right-hand side of the equivalence in the definition of *satisfies* in $\widetilde{L}$, namely "$\mathcal{M} \models A$", is directly translated to "$\mathcal{I} \models_{\mathrm{PL}^0} A$" (assuming a variable renaming $\mathcal{M} \mapsto \mathcal{I}$ during that translation). We might, however, want to expand the definition of $\models_{\mathrm{PL}^0}$ within the definition of *satisfies* such that "$\mathcal{I} \models_{\mathrm{PL}^0} A$" is expanded to "$\mathcal{I}(A) = \mathsf{T}$". Similarly, we might want to expand the definition of *satisfies* in the definition of *satisfiable* such that the right-hand side of that definition becomes, e.g., there exists a model $\mathcal{I}$ such that $\mathcal{I} \models_{\mathrm{PL}^0} A$ which could further be expanded to there exists a model $\mathcal{I}$ such that $\mathcal{I}(A) = \mathsf{T}$.

If the grammatical structure of a sentence S in which an expression e is to be definition-expanded to an expression $\widetilde{e}$ is sufficiently simple, such an expansion can easily be achieved by (1) creating an abstract syntax tree for S, (2) finding all subtrees of S that represent e and replacing them with the syntax tree of $\widetilde{e}$ (with an appropriate instantiation of the variables in $\widetilde{e}$) and (3) linearizing the resulting syntax tree. Steps (1) and (3) are natural use cases of GF, while step (2) is standard (abstract syntax) tree substitution.

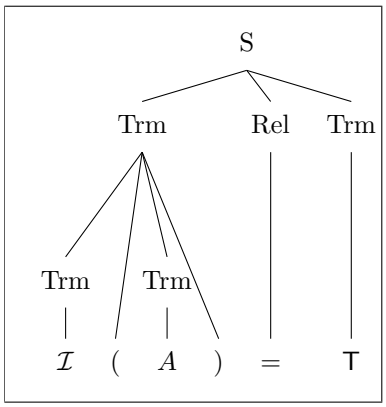


Fig. 14. Syntax Tree for the Definiens of $\models_{\mathrm{PL}^0}$

For instance, by replacing the subtree that represents "$\mathcal{I} \models_{\mathrm{PL}^0} A$" (the boxed part in Fig. 15) by the syntax tree that represents "$\mathcal{I}(A) = \mathsf{T}$" (Fig. 14) in the syntax tree for the definition of *satisfies* (Fig. 15) and linearizing the resulting tree we get the desired formulation A model $\mathcal{I} \in \mathcal{K}_{\mathrm{PL}^0}$ satisfies A iff $\mathcal{I}(A) = \mathsf{T}$.

However, such a simple "find-and-replace" procedure on syntax trees quickly becomes insufficient when the grammatical structure of the sentences in which a definition expansion is to be applied gets too complicated. For instance, consider the definition of *satisfiable*: A is satisfiable iff there exists a model that satisfies A. To be able to replace *satisfies* by its definiens $\mathcal{I} \models_{\mathrm{PL}^0} A$ we have to introduce a fresh variable $\mathcal{I}$ in that sentence to match the first argument of $\mathcal{I} \models_{\mathrm{PL}^0} A$ which requires a manipulation of the abstract syntax tree of that sentence that goes beyond simply substituting subtrees. As such more sophisticated methods for flexiformal definition expansion extend the scope of this paper, we will not elaborate further on this here.

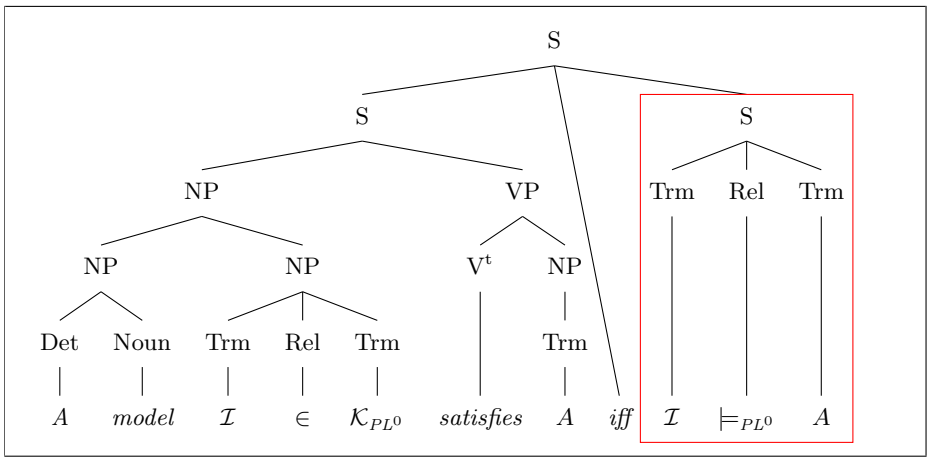


Fig. 15. Syntax Tree for the Definiens of *satisfies*

References

[AK09] L. W. Anderson and D. R. Krathwohl. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. New York: Longman, 2009.

[ÁNP23] E. Ábrahám, J. K. F. Nalbach, and V. M. Promies. "Automated Exercise Generation for Satisfiability Checking". In: *LNCS*. Vol. 13962. Springer, 2023, pp. 1–16. DOI: 10.1007/978-3-031-27534-0_1. URL: https://publications.rwth-aachen.de/record/957774.

[Ber+23] M. Berges, J. Betzendahl, A. Chugh, M. Kohlhase, D. Lohr, and D. Müller. "Learning Support Systems based on Mathematical Knowledge Managment". In: *Intelligent Computer Mathematics (CICM) 2023*. LNAI. in press. Springer, 2023. URL: https://url.mathhub.info/CICM23ALEA.

[Cas23] D. Casanueva. *HaTeX: The Haskell LaTeX library*. 2023. URL: https://hackage.haskell.org/package/HaTeX (visited on 04/05/2024).

[CICM22a] K. Buzzard and T. Kutsia, eds. *Intelligent Computer Mathematics*. Vol. 13467. LNAI. Springer, 2022.

[CICM22b] M. Kohlhase and D. Müller. "System Description: sTeX3 – A LaTeX-based Ecosystem for Semantic/Active Mathematical Documents". In: *Intelligent Computer Mathematics (CICM) 2022*. Ed. by K. Buzzard and T. Kutsia. Vol. 13467. LNAI. Springer, 2022, pp. 184–188. URL: https://kwarc.info/people/dmueller/pubs/cicm22stexsd.pdf.

[CICM22c] D. Müller and M. Kohlhase. "Injecting Formal Mathematics Into LaTeX". In: *Intelligent Computer Mathematics (CICM) 2022*. Ed. by K. Buzzard and T. Kutsia. Vol. 13467. LNAI. Springer, 2022, pp. 168–183. URL: https://kwarc.info/people/dmueller/pubs/cicm22stexmmt.pdf.

[Far18] W. M. Farmer. "A New Style of Proof for Mathematics Organized as a Network of Axiomatic Theories". 2018. URL: https://arxiv.org/abs/1806.00810.

[Koh+23] M. Kohlhase et al. "Project VoLL-KI – Learning from Learners". In: *Künstliche Intellienz* (2023). accepted.

[MMT] *MMT – Language and System for the Uniform Representation of Knowledge*. Project web site. URL: https://uniformal.github.io/ (visited on 01/15/2019).

[Ran04] A. Ranta. "Grammatical Framework — A Type-Theoretical Grammar Formalism". In: *Journal of Functional Programming* 14.2 (2004), pp. 145–189.

[RK13] F. Rabe and M. Kohlhase. "A Scalable Module System". In: *Information & Computation* 0.230 (2013), pp. 1–54. URL: https://kwarc.info/frabe/Research/mmt.pdf.

[SAK20] J. F. Schaefer, K. Amann, and M. Kohlhase. "Prototyping Controlled Mathematical Languages in Jupyter Notebooks". In: *Mathe-*

matical Software – ICMS 2020. 7th international conference. Ed. by A. M. Bigatti, J. Carette, J. H. Davenport, M. Joswig, and T. de Wolff. Vol. 12097. LNCS. Springer, 2020, pp. 406–415. URL: https://kwarc.info/kohlhase/papers/icms20-glf-jupyter.pdf.

[Sch20] J. F. Schaefer. *Implementing ForTheL in GLIF – A case study*. Master Project Report. 2020. URL: https://gl.kwarc.info/supervision/projectarchive/-/blob/master/2020/Schaefer_Frederik.pdf.

[VKC] *VoLL-KI based Courses at FAU*. URL: https://courses.voll-ki.fau.de (visited on 07/21/2023).