Reaping the Benefits of Modularization in Flexiformal Mathematics by GF-based AST Transformations

— *Extended Version* —

Josefin Kelber[1], Michael Kohlhase[2], Jan Frederik Schaefer[2], and Marcel Schütz[2]

[1] Technical University of Munich (TUM), Germany
[2] Computer Science, FAU Erlangen Nürnberg, Germany

Abstract. Flexiformal documents – i.e. documents with embedded semantic annotations that make some aspects of their content machine-actionable – can be instrumented to make interaction with the underlying knowledge more efficient and effective. Fostering such interactions via semantic services has proven very successful in university education, but the practical applicability is limited by the cost of flexiformalization. A method for lowering (flexi)-formalization costs is to use modular representations to profit from enhanced source sharing and induced (generated) contents. In fully formal environments this is well-understood and implemented in many systems.
In this paper we show that many of the formal techniques carry over to the informal setting, if we parse (rigorous) natural language with a semantically optimized grammar and work on abstract syntax trees instead of formulae.
We present *i)* a set of use cases for generating learning material to be used in an educational setting – concretely in the field of theoretical computer science –, *ii)* a GF grammar that allows to syntactically analyze the underlying language fragment, *iii)* a set of AST-to-AST simplifications that can be used to fine-tune the wording of the generated content and adapt it to the scientific (especially mathematical) jargon, and *iv)* a prototypical implementation that shows the technique in action.

Keywords: Flexiformal Mathematics · FTML · Math Language · Theory Morphisms · GF · NLP

1 Introduction

Scientific knowledge is highly inter-related, with relations ranging from the mappings that underlie application of theoretical results – variables in the theory are mapped to objects in the application domain which obey the theory assumptions – to inter-theory interpretations and refinements, which allow to import insights,

constructions, methods, and arguments from the source theory into the target theory (recontextualization).

In modular representation formats, such relations can be made explicit as first-class citizens – called theory morphisms, views, realizations, or just interpretations – which can be inspected, composed, applied, and reasoned with. In fully formal representation formats, this is a very well-understood process, and most formal mathematical libraries utilize the synergies induced by theory morphisms.

In flexiformal representation formats, reaping the synergies is much more difficult: instead of definition expansion and possibly simplification in formulae, we need to manipulate informal technical language. In [KS24] we have shown that transporting quiz problems given in mathematical vernacular is possible in principle using a simple template-based approach: given sufficient (but still natural) semantic annotations of the source quiz problems, flexiformal theory morphisms have the necessary information (and language snippets) to assemble relatively natural-looking quiz problems in the context of the target theory. Of course, the template-based approach only allows very limited simplifications of the generated language, which can also contain grammatical problems like failure of agreement or inflection, but more generally loses much of the conciseness we are used to in mathematical language.

Contribution In this paper we want to lift some of these limitations: instead of composing semantically annotated LaTeX snippets via templates, we parse the natural language with the Grammatical Framework (GF [Ran04; GF]), do the language composition at the level of GF abstract syntax trees, and leave the linearization of these to the GF system to guarantee grammatical correctness of the result. This had been anticipated in [KS24], but needed a semantic grammar for mathematics, which we have developed for this paper.

A side effect of this move from LaTeX strings to abstract syntax trees (essentially λ-terms in LF [Pfe01]) is that we can do more complex replacement and simplification operations and thus model some of the more advanced fine-tuning of mathematical notions when processing mathematical language.

As a running example, we will recontextualize the definition of a path in graph theory to the context of automata theory. With sufficient training, a human reader can easily recognize how automata correspond to graphs, but students in theoretical computer science courses often struggle with this and may benefit from an explicit recontextualization or generated fine-grained explanations of the relationship.

Context & Generality Our work is developed in the context of ALeA [Ber+23], a symbolic learning support system. While ALeA provides the main motivation for the work reported here, the particular setup is secondary. The main assumptions are that the underlying knowledge representation admits theories and theory morphism and that the presentation of the knowledge is flexiformal – i.e. based on rigorous natural language with formulae – see [Koh13] for details and intuitions.

For instance, the Isabelle/Isar universe would be a likely candidate where the methods presented here would be applicable.

Overview In Section 2, we discuss flexiformal recontextualization and explore relevant phenomena using a running example. Then we discuss in Section 3 an implementation that shows the feasibility of our approach. Section 4 concludes the paper and discusses future work. The appendix (Section A; not part of the submission) contains a collection of rewriting rules. If accepted, the published version will reference an updated list on arXiv.

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 the input of Dennis Müller in the development of suitable representations flexiformal theory morphisms in STEX and FLAMS.

2 Flexiformal Recontextualization

Our goal with flexiformal recontextualization is to automatically translate learning materials about concepts from one scientific theory into the context of another theory to support students learning those concepts.

2.1 Example: Paths in Graphs and Automata

As a motivating example, we consider the notion of a path in a finite automaton, which is often used in the literature on automata theory, for instance:

- "state 6 is reached by the path $1 \to 2 \to 3 \to 6$" [HMU07, p. 91],
- "show that if there exists a path in $\mathcal{A}$ that leads from a state of $\mathcal{C}_1$ to a state of $\mathcal{C}_2$, ..." [ST09, p. 178],
- "the path starts or stops at state $k+1$ but doesn't go through any state numbered higher than k" [Mar03, p. 115].

These three examples have in common that the notion of a path in an automaton is not defined in the text preceding those quotes. Instead, the notion of a path in a *graph* – or, more precisely, in an *edge-labeled multidigraph*[3]. For a human reader who is sufficiently well trained in mathematics, it does not impose any difficulties to recognize (the transition system underlying) an automaton as a graph and apply the notion of a path in a graph to the setting of automata. However, for some students this conceptual leap is not obvious, and they would benefit from further explanation. Active documents allow us to show supplementary information on demand, e.g. by hovering over the term *"path"* in the above sentences. In this hover, we want to provide

[3] For the sake of brevity we use "graph" as a synonym for "edge-labeled multidigraph" in the following.

1) the **original definition** of a *"path in a graph"*
2) the **recontextualization morphism**, i.e. an explanation how an automaton induces a graph,
3) the **recontextualization** of the original definition, i.e. the definition of a *"path in an automaton"*.

Figure 1 shows what such a pop-up note could look like for non-deterministic finite automata (NFAs).

In a graph, a **path** is a finite sequence $e_1, \ldots, e_n$ of edges with $t(e_i) = s(e_{i+1})$ for all $1 \leq i < n$.
An NFA $\langle Q, \Sigma, \delta, q_0, F\rangle$ admits a graph $\langle V, E, s, t, L, l\rangle$, where $V := Q$, $E := \{t \mid t \text{ is a transition}\}$, $s := \pi_1$, $t := \pi_3$, $L := \Sigma$, and $l := \pi_2$.
Thus, in an NFA, a **path** is a finite sequence $t_1, \ldots, t_n$ of transitions $t_i := \langle q_i, c_i, {q'}_i\rangle$ with ${q'}_i = q_{i+1}$ for all $1 \leq i < n$.

Fig. 1. Example hover, showing the original definition, the recontextualization morphism, and the recontextualized definition. Here, $s(e)$ and $t(e)$ denote the source and target, resp., of an edge e, and π_i denotes projection to the i-th component of a tuple (in an active document, the reader can find this out by hovering over *"s"*, *"t"* and *"π"*).

Manually authoring the recontextualizations would be tedious and error-prone, as a simple back-of-the-envelope calculation illustrates: if we have 10 graph concepts (e.g. *path*, *cycle*, *loop*, *connectivity*, ...) that have to be recontextualized to 15 different structures (e.g. NFAs, DFAs, RDF triple collections, implication graphs, ...), we would need i) 10 original definitions, ii) 15 recontextualization morphisms, and iii) 150 recontextualized definitions. And, of course, graphs are just one of many theories that can be recontextualized. Other examples include, e.g., logical systems, constraint satisfaction problems and optimization problems.

2.2 Recontextualization as a Pushout

In [KS24] we discussed that recontextualization corresponds to a pushout in a theory graph, which, in principle, allows us to generate recontextualizations automatically: Suppose we have a theory of graphs that provides a definition of the concept of a path, a theory of automata and a theory morphism between them which interprets the theory of graphs in the context of the theory of automata. Then we can translate the definition of a path in a graph along the morphism to get an appropriate definition in the context of automata, as illustrated in Figure 2. This translation can be regarded as the pushout of the definition of a path in a graph along that morphism. Computing such pushouts is well-understood in a fully formal setting, where we essentially apply the substitution specified by the morphism. However, our flexiformal setting is more challenging because some-

times the substitutions have to be applied to (semantically annotated) natural language.

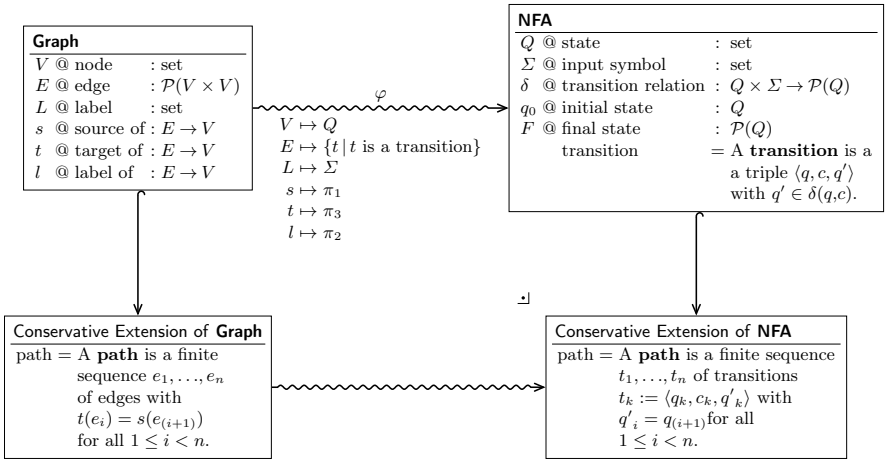


Fig. 2. Pushout for paths in graphs and finite automata. An entry of the form [⟨notation⟩ "@"] ⟨name⟩ (":" ⟨type⟩ | "=" ⟨definition⟩) is a declaration of a symbol with name ⟨name⟩, optional symbolic notation ⟨notation⟩ and either type ⟨type⟩ or definitional expression ⟨definition⟩. If the symbol denotes a set, we follow the convention to name the symbol after the elements of that set.

In [KS24], we described a template-based approach for flexiformal recontextualization, which poses certain limitations regarding both the grammatical correctness of the translation result and how well it can be simplified to semantically equivalent but more concise expressions. For instance, consider the following definition of a path in a graph as shown in Figure 2:

> A **path** is a finite sequence $e_1, \ldots, e_n$ of edges with $t(e_i) = s(e_{i+1})$ for all $1 \leq i < n$.

The translation mechanism from [KS24] simply substitutes all components of the theory of graphs in the definiens of *"path"* with their respective values under the morphism φ, which is relatively straightforward if the relevant components (like the word *"edges"*) are semantically annotated. We obtain the following definition of a path in an automaton by applying that mechanism:

> A **path** is a finite sequence $e_1, \ldots, e_n$ of elements of $\{t \mid t \text{ is a transition}\}$ with $\pi_3(e_i) = \pi_1(e_{i+1})$ for all $1 \leq i < n$.

Semantically, we conflate nouns – like *"edge"* – with the set of all elements the noun stands for – in this case *"E"*. Linguistically, this requires us to insert

"element(s) of" when substituting a noun with a set. Getting the correct form of the noun (*"element"* vs *"elements"*) in a template-based approach is difficult, and support in our previous work [KS24] was limited to specific cases.

2.3 Polishing the Result

While the result of the translation is technically correct, it is not as concise and clear as a human author would write it. Although statistical methods, like delegating the rewriting task to an LLM, might succeed on a wider range of sentences than the approach we are presenting, those methods cannot guarantee that meaning will be preserved – which would be problematic e.g. in an educational setting. Instead, we model the simplification process as a sequence of meaning preserving rewriting operations.

To get a better intuition, let us simplify the (recontextualized) definition of a path in an automaton from the previous section step by step:

1. **Comprehension term reduction:** As a first refactoring step towards the intended expression we reduce the comprehension term expression *"elements of* $\{t \mid t$ *is a transition}"* to *"transitions"*:

   > A **path** is a finite sequence $e_1, \ldots, e_n$ of transitions with $\pi_3(e_i) = \pi_1(e_{i+1})$ for all $1 \leq i < n$.

2. **Structure expansion:** Then we expand the noun phrase *"finite sequence* $e_1, \ldots, e_n$ *of transitions"* by appending a variable definition *"*$e_k := \langle q_k, c_k, {q'}_k \rangle$*"*:

   > A **path** is a finite sequence $e_1, \ldots, e_n$ of transitions $e_k := \langle q_k, c_k, {q'}_k \rangle$ with $\pi_3(e_i) = \pi_1(e_{i+1})$ for all $1 \leq i < n$.

3. **Variable expansion:** With the variable definition introduced in the last step, we can expand their later occurrences (*"*e_i*"* and *"*e_{i+1}*"*):

   > A **path** is a finite sequence $e_1, \ldots, e_n$ of transitions $e_k := \langle q_k, c_k, {q'}_k \rangle$ with $\pi_3(\langle q_i, c_i, {q'}_i \rangle) = \pi_1(\langle q_{i+1}, c_{i+1}, {q'}_{i+1} \rangle)$ for all $1 \leq i < n$.

4. **Projection reduction:** Now that the arguments of the projections are triples we can evaluate the projections:

   > A **path** is a finite sequence $e_1, \ldots, e_n$ of transitions $e_k := \langle q_k, c_k, {q'}_k \rangle$ with ${q'}_i = q_{i+1}$ for all $1 \leq i < n$.

5. **Variable renaming (optional):** As a last step we rename the variables e_j to t_j:

   > A **path** is a finite sequence $t_1, \ldots, t_n$ of transitions $t_k := \langle q_k, c_k, {q'}_k \rangle$ with ${q'}_i = q_{i+1}$ for all $1 \leq i < n$.

Note that steps 2–3 are motivated by the desire to eliminate the projections in step 4 – otherwise, they would make the result less concise.

During our research, we have collected a growing set of basic rewriting rules in Section A that modify natural language expressions in a way that preserves their meaning with the ultimate goal of making them more concise and natural. Collecting and implementing these rules is ongoing work.

For instance, to perform the first rewriting step from our example above, we can apply rule *Element-NP CTR* from subsection A.6, namely

$$\frac{\text{“}\textit{element}\ [x]\ \textit{of}\ \{y \mid y\ \textit{is a}\ \langle\textit{noun phrase}\rangle\}\text{”}}{\text{“}\langle\textit{noun phrase}\rangle\ [x]\text{”}}\ ,$$

to the expression *“elements of* $\{t \mid t$ *is a transition*$\}$*”* which yields the expression *“transitions”*. Note that we have to respect the grammatical number of the noun *“element”* in the input expression which determines the grammatical number of the noun phrase (*“transition”*) in the output expression. In our implementation, these rules act on abstract syntax trees (see subsection 3.2 and subsection 3.3).

Other rules are more complicated and involve information that can not be obtained from the syntactical structure of the expression alone. For instance the *structure expansion* step in our example above requires to query the definiens of *“transition”* from a knowledge base. In even more extreme cases, a rewriting rule can demand a logical inference to be carried out before it can be applied. For instance, rule $\iota_\in$ *Conversion* from subsection A.7, namely

$$\frac{\vdash |A| = 1 \qquad \vdash a \in A \qquad \text{“}b \in A\text{”}}{\text{“}b = a\text{”}}\ ,$$

allows to convert an expression of the form *“*$b \in A$*”* to *“*$b = a$*”* under the constraint that it can be inferred that A consists of exactly one element a. While implementing this rule in its full generality may not be feasible, we can cover special cases that frequently occur in practice.

2.4 A Flexiformal Recontextualization Pipeline

In summary, we can think of flexiformal recontextualization as a two-step process: substitution application and simplification. The substitution is based on the recontextualization morphism, but we can use the same process to apply other substitutions. For example, in this paper, we will sometimes apply substitutions derived from definitions to perform definition expansion. Simplification is based on rewriting rules that maintain the meaning of the expression, but the application of those rules has to be carefully controlled.

While substitution and simplification are well-understood in a fully formal setting, dealing with the natural language in a flexiformal setting is more challenging. Semantic annotations can link parts of the natural language to the symbols they refer to, which helps identify where to apply the substitutions.

3 Implementation

We have implemented – on a prototype level – flexiformal substitution and simplification in the context of ALeA, using a rule-based approach to ensure correctness and trustworthiness of the generated content. The relevant documents are parsed with a natural language grammar into abstract syntax trees (ASTs). The ASTs are then processed and transformed with a set of hand-written rules, and the results are linearized into strings.

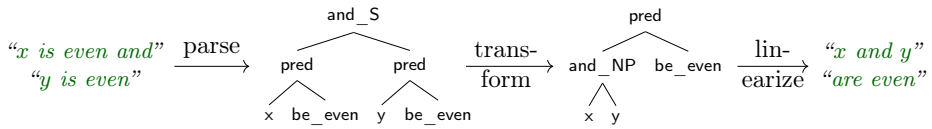
3.1 Context: The ALeA Ecosystem

Our work is implemented in the sTeX/rusteX/FTML/FLAMS/ALeA ecosystem, where technical knowledge is represented in sTeX [MK22], a variant of LaTeX that supports the annotation of the semantic structures underlying the text content. The rusteX system transforms sTeX sources into FTML, an extension of HTML5 that preserves all the semantic annotations from sTeX. FTML in turn can be processed by the FLAMS system that serves the HTML5 content and integrates knowledge management services based on the annotations into it – turning the content into active documents. The ALeA system [Ber+23] in turn pairs up the active document facilities with learner competency modeling in an adaptive learning assistant that has been used for helping more than 500 students/semester master Math, AI, and CS courses at FAU over the last two years.

Our prototype is based on FTML because processing sTeX documents, which can contain arbitrary LaTeX, is very difficult. To be precise, we use an FTML-inspired HTML extension because FTML was still in flux during development. We will adapt our implementation to real FTML once the FTML format is stable.

3.2 A Grammar for Flexiformal Mathematics

For parsing and linearization, we use GF, the Grammatical Framework [Ran04], which was designed for (multi-lingual) natural language processing. GF separates between the abstract syntax of the language and the concrete syntax that describes how abstract syntax trees (ASTs) are represented as strings. The concrete syntax has powerful mechanisms like record types and tables to deal with the peculiarities of natural language such as inflection and agreement. The abstract grammar usually abstracts away from these details, which lets us manipulate the ASTs without worrying about such "low-level" aspects of natural language. To illustrate this, consider the following minimal example where we parse a sentence, transform the AST by an aggregation rule, and then linearize the result into a new sentence string:



Here, the information whether we should use *"is even"* or *"are even"* in the final sentence is not explicitly present in the AST. Instead, the concrete syntax rule for pred must specify how the correct verb form is chosen depending on the number of the subject.

The GF community has developed the **Resource Grammar Library** (**RGL**) [GFR], which implements the basic syntax and morphology of many languages. While the RGL is intended as a library for developing application-specific grammars, we have also experimented with using it directly for parsing (see [Kel24] for more details). With some modifications to the RGL, e.g. to support mathematical language and the semantic markup of STEX, we were able to parse rougly 14 % of the English definitions in SMGloM (a flexiformal glossary that acts as a domain model for semantic annotations in STEX/ALeA). Coverage was limited, among other reasons, because of missing words and abbreviations in the lexicon, problems with pre-processing the STEX files, and missing support for some sentence structures. While many of these issues could be improved, there are more fundamental problems with directly using the RGL: The grammar has a high degree of syntactic ambiguity, which results in many possible ASTs (often in the thousands, but sometimes many more because ambiguities multiply out). This makes the proper treatment of ambiguities (see subsection 3.4) impractical. More importantly, implementing substitution and simplification for RGL ASTs is cumbersome and error-prone.

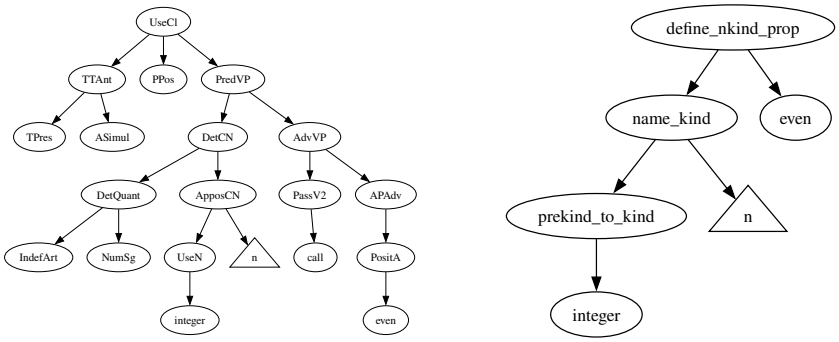


Fig. 3. ASTs for the phrase *"an integer n is called even"*. *Left:* Syntactic AST from an extended RGL grammar. *Right:* A more semantic AST from our application grammar. The formula *"n"* is actually encoded as a MathML tree, which was omitted for conciseness.

The better approach is to develop a new grammar that uses the RGL as a mere library. More concretely, we can implement a completely new abstract syntax that is more suitable for our application, and use the RGL only for the concrete syntax. In particular, we can design the abstract syntax to be more semantic, which makes processing the ASTs much easier. For example, consider the phrase *"an integer n is called even"* (typically followed by *"iff ..."*). In this

case, syntactic details (e.g. that *"be called even"* is a verb phrase in passive voice) are not relevant for further processing. The relevant information is that the property of being *"even"* is defined for integers, and that we use *"n"* as a variable. In our grammar, we refer to words like *"integer"* as Kinds. A Kind with a variable is a NamedKind. So for the phrase above, we have a rule define_nkind_prop that combines a NamedKind and a Property into a definition core. Figure 3 visualizes the ASTs for the example phrase.

There are other ways to express the same meaning, e.g. *"an integer n is said to be even"* or *"an integer n is even"*. Syntactically, they are rather different (e.g. the latter one does not use passive voice), but for our processing we basically want to treat them the same. GF supports variants, which allows us to use the define_nkind_prop rule for all three cases. In that case, the AST would not carry information about the original variant, and the linearization could use a different variant. While this should not result in ungrammatical or semantically incorrect sentences, it may result in less natural output. To avoid this, we use separate rules for each variant and use a consistent naming scheme (e.g. define_nkind_v1 and define_nkind_v2), so that we can ignore but preserve variant information throughout the recontextualization pipeline.

In our pipeline, we cannot ignore the HTML structure of the input document. For example, some semantic annotations, like definienda, are encoded via tags with specific attributes. Furthermore, formulae are encoded in MathML. To handle this, we include HTML tags in the abstract syntax trees. As a consequence, HTML tags cannot occur in arbitrary places, but rather must match the structure of the abstract syntax. For example, *"an even integer"* cannot be parsed, but *"an even integer"* can be parsed. Our experience is that the HTML structure usually matches the grammatical structure in practice.

3.3 Substitution and Simplification

In a formal setting, substitution replaces a symbol with a term. In our flexiformal setting, this basically corresponds to replacing a subtree that corresponds to a specific symbol with a different subtree. For example, in the example in Section 2, we replaced *"edges"* with *"elements of $\{t \mid t \text{ is a transition}\}$"*.

In practice, the flexiformal substitution is a bit more complex. Consider, e.g. the sentence

> Let m be a positive integer.

and a substitution given by the definition

> An integer n is called positive iff $n > 0$.

The output is

> Let m be a integer m such that $m > 0$.

In this case, the definiendum of *"positive"*, *"$\cdot > 0$"*, has a hole that must be filled with *"m"*. The AST of *"$m > 0$"* is then attached via a *"such that"* clause and the *"positive"* subtree is removed from the original AST. Figure 4 visualizes both the input and output ASTs.

Note that substitution in ASTs inherits a form of subject reduction from the underlying logical framework of GF: (grammatically) well-typed substitution into (grammatically) well-typed ASTs gives (grammatically) well-typed ASTs – in fact they need to be well-typed if we want to linearize them with GF. If the grammatical types of a substitution do not match, like *"positive"* and *"$\cdot > 0$"* in the example above, we have to insert "glue language" (e.g. *"... such that ..."*) to ensure well-typedness.

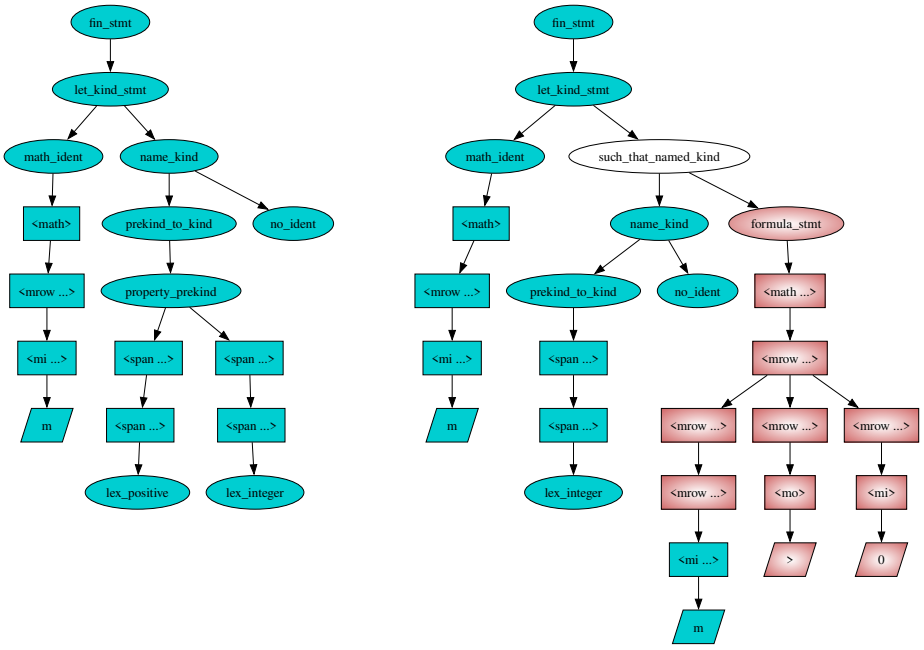


Fig. 4. Example of a substitution. *Left:* Input AST (*"Let m be a positive integer"*). *Right:* Output AST (*"Let m be a positive integer such that $m > 0$"*). The nodes from the input AST are colored in turquoise, while the nodes from the definiendum are red. The node shapes indicate the type of the node (grammar node/HTML tag/MathML literal).

After substitution, ASTs are simplified with custom rules. Section A contains a collection of rewriting rules that came up during our experiments, but only a few of them have been implemented in our prototype. Applying rewrit-

ing rules does not necessarily result in simplification. For example, in the running example in subsection 2.3, we had a structure expansion step (introducing *"$e_k := \langle q_k, c_k, q'_k \rangle$"*), so that we can do projection reduction in a later step. In our implementation, rewriting rules are triggered by the "desire" for a specific simplification. For example, the structure expansion in the running example gets triggered by the "desire" to apply projection reduction.

It appears that getting significant simplification coverage is a non-trivial task. At the moment, implementing new rewriting rules is complicated by the fact that semantic information is encoded in the ASTs in the attributes of HTML nodes, which makes the processing more cumbersome – especially for MathML nodes where the syntactic/presentational structure generally does not match the semantic structure well. We are planning to improve this by modelling the semantic structure more directly in the ASTs.

3.4 Handling Ambiguity

Sometimes, multiple ASTs correspond to the same string. In this case, GF can generate all possible ASTs. This gives us a basic tool for handling ambiguity: if our grammar is sufficiently broad to cover the correct reading, we know that one of the ASTs will be the correct one. Our grammar/ASTs are designed to be somewhat semantic (see subsection 3.2). As a consequence, this mechanism does not just handle classical structural ambiguity, but also more semantic ambiguity. A simple example for this is lexical ambiguity (which is less relevant for us though, as discussed below). Lexical ambiguity can be handled by introducing multiple grammar rules for an ambiguous word, like the word *"even"*, which can refer to an integer being divisible by two or a function with the property $f(x) = f(-x)$. By making two rules that linearize to *"even"*, e.g. even_function and even_integer, we would get two different ASTs for the same string – one for each meaning.

Disambiguation With Semantic Annotations The semantic annotations in SᴛEX/FTML documents can help with disambiguation. For example, in a sufficiently annotated SᴛEX/FTML document, technical terms are explicitly disambiguated with the concept they refer. Assuming a sufficiently annotated document, we therefore do not have to worry about lexical ambiguity – at least for technical terms.

The SᴛEX/FTML annotations can also help with other types of ambiguity. For example, the sentence

> An integer n is even iff n is divisible by 2.

could be interpreted as a theorem or a definition. In a definition, *"even"* would be annotated as a definiendum, but in a theorem it would be annotated as a symbol reference instead.

Ensuring Correct Results Despite Ambiguity As discussed above, during parsing we can get multiple ASTs for the same string because of ambiguity. Conversely, multiple ASTs can be linearized to the same string. This allows us to have correct results despite unresolved ambiguity. Consider, for example, an input sentence that gets parsed into two different ASTs, A and B. We apply some transformations to get A' and B'. If the linearizations of A' and B' are equal, it is ultimately irrelevant whether A or B were the correct reading. After all, the final document will only contain the linearization.

Of course, the result may be ambiguous – both A' and B' are possible readings –, but this can be considered a feature: natural language, including in mathematics, is ambiguous, and restricting the output to non-ambiguous sentences would be a severe limitation. With flexiformal substitution, we can get additional ambiguity from the substitution value itself, which increases the overall number of readings further: If we have a single substitution with readings $S_1, \ldots, S_n$ and input readings $I_1, \ldots, I_m$, we get $m \cdot n$ resulting ASTs (applying S_1 to I_1, S_1 to I_2, etc.). The final result has to be compatible with all of them. One option to increase the chances of getting a compatible result (which we have not yet implemented) is to optionally ignore some rewriting rules during simplification. For example, if one particular rewriting rule is not applicable in one of the ASTs, we may still get a linearization that is compatible with all other ASTs if we skip this rewriting rule in all other ASTs as well.

If no compatible linearization can be found, it would be up to the user to select the correct string from a list of options.

In the case of the running example (see subsection 2.3), our grammar gets two different readings (after filtering; see Section 3.4) – one where the *"with"*-phrase is attached to *"sequence"*, and one where it is attached to *"edge"*. Both readings ultimately result in the same linearization.

Reducing the Number of Readings Even though our pipeline can handle ambiguities to some extent, reducing the number of readings is still desirable. This can significantly improve performance by dampening the combinatorial explosion and, as a side effect, simplify the substitution and rewriting rules as fewer cases have to be considered. There are two ways to reduce the number of readings: grammar adjustments and filtering after parsing, which come with different trade-offs.

Let us illustrate this with the example string *"finite subset of* $\mathbb{N}$*"*, which in our grammar is called a Kind. There are two ways to read it (see Figure 5): either *"finite"* modifies *"subset of* $\mathbb{N}$*"*, or *"of* $\mathbb{N}$*"* is an argument of *"finite subset"*. We can adjust the grammar to enforce the latter reading by introducing a separate syntactic category, PreKind, that can only be modified by adjectives, and then turned into a Kind that can only be modified with prepositional arguments.

As a consequence of this, it can now be ambiguous where exactly an HTML tag is in the AST. For example, in *"finite subset of* $\mathbb{N}$*"*, the tag can now either be wrapped around *"finite subset"* before or after the PreKind is turned into a Kind. To remedy this, we can simply filter out all

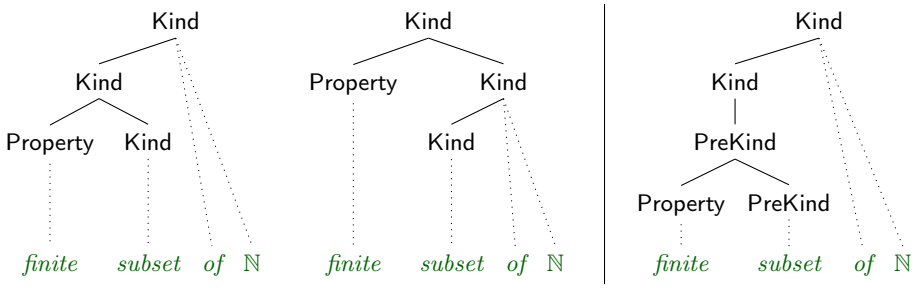


Fig. 5. Readings for an example phrase. For conciseness, syntactic categories are displayed instead of rule names. With a simple grammar, we get two different readings (*left*), but by introducing a new syntactic category, PreKind, we can enforce a single reading (*right*) at the cost of added complexity.

ASTs where an HTML tag is wrapped around the AST node that turns a PreKind into a Kind.

In general, grammar adjustments lead to a more complex grammar that may be harder to understand and maintain. Filtering, on the other hand, keeps the grammar simple, but may lead to performance problems if too many readings are first generated and then filtered out. Furthermore, filtering must be done carefully to ensure that only redundant readings are removed. Finding the right balance between the two approaches remains work in progress.

3.5 State of the Implementation

The implementation described here is our fourth prototype for exploring the feasibility of rule-based flexiformal recontextualization of learning objects. Its design was informed by the previous prototype implementations. The first prototype was presented in [KS24], and used simple templates for recontextualization of SᴛEX content. The next prototype was developed in the context of a Bachelor's thesis of one of the authors (see [Kel24]), and explored flexiformal definition expansion with a grammar based on GF's Resource Grammar Library (see subsection 3.2). To overcome some of the challenges from processing LaTeX/SᴛEX sources directly, a third prototype was developed based on the FTML format. Ultimately, it became clear that a more semantic grammar as described in subsection 3.2 was necessary to develop robust rules for substitution and simplification, which led to the implementation described in this paper.

We have implemented all rules necessary for the running example from Section 2. We can handle the whole pipeline of AST-based relocalization by substitution and polishing via rewriting with the exception of variable renaming. In constrast to the deterministic/semantic rewriting rules, variable renaming has a strong cognitive flavor, which calls for different methods and quality measures; so we leave it to future work.

At this point, every new example requires work on the grammar and the substitution/rewriting rules. While the grammar and, to an extent, the substi-

tution rules give the impression of converging, getting a reasonable coverage for the rewriting rules appears to be a much longer-term enterprise.

This shows that our overall architecture works principle; but also highlights that we need extensions in the grammar and rewriting rules to scale to realistic applications in ALeA.

We are currently refactoring the prototype into a robust foundation for further development, which will be hosted at `https://github.com/slatex/relocalization`. The repository also contains links to the prototype implementations.

4 Conclusion & Future Work

We have shown how we can extend content generation – recontextualization; technically (theory) morphism application – from the formal to the flexiformal setting by parsing the mathematical vernacular in the source into ASTs, applying AST substitutions, and then fine-tuning the conciseness of the generated language with rewriting rules. We have shown the practical feasibility of the method with a GF grammar and a set of domain-specific AST rewriting rules.

While the implementation is still at a prototype stage – it can handle our running example and a few more – we feel confident that we can scale coverage to useful levels with more development. Most importantly, we feel that the overall information architecture has stabilized, and that this is a sufficient milestone that needs to be communicated to the CICM community, so that others may become involved.

While the GF grammar has has proven easy to extend for new sentences, our "rational reconstruction" of recontextualization at the AST level has revealed a surprising extent and variety of simplifications needed at this level to account for the concise and rigorous mathematical language we see in mathematical texts written by humans. During our explorative work, we have collected a set of – mostly foundational – rewriting rules that cover cases like our running example. We believe that this set is by no means complete outside the set-theoretic foundation, and we will have to extend it considerably to scale the AST-to-AST transformation.

We conjecture that the AST rewriting can be very useful for polishing and adapting all kinds of flexiformal transformation tasks.

Note that already the current rule-set is non-confluent. Thus their application is a non-trivial search/optimization problem where the quality measure is conciseness and readability of the generated language. How to manage/implement this scalably and independently of the concrete rule set – and that has to be our goal – is an open research question.

References

[Ber+23] M. Berges, J. Betzendahl, A. Chugh, M. Kohlhase, D. Lohr, and D. Müller. "Learning Support Systems based on Mathematical Knowl-

edge Managment". In: *Intelligent Computer Mathematics (CICM) 2023*. Ed. by C. Dubois and M. Kerber. LNAI. Springer, 2023. DOI: 978-3-031-42753-4. URL: https://url.mathhub.info/CICM23ALEA.

[Dal99] H. Dalianis. "Aggregation in Natural Language Generation". In: *Computational Intelligence* 15.4 (1999), pp. 384–414. DOI: https://doi.org/10.1111/0824-7935.00099.

[GF] *GF - Grammatical Framework*. URL: http://www.grammaticalframework.org (visited on 09/27/2017).

[GFR] B. Bringert, T. Hallgren, and A. Ranta. *GF Resource Grammar Library: Synopsis*. URL: https://www.grammaticalframework.org/lib/doc/synopsis/ (visited on 03/11/2020).

[HMU07] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 3rd ed. Pearson/Addison Wesley, 2007.

[Kel24] J. Kelber. "Flexiformal Definition Expansion and Parsing Mathematical Natural Language". Bachelor's thesis. FAU Erlangen-Nürnberg, 2024.

[Koh13] M. Kohlhase. "The Flexiformalist Manifesto". In: *14th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*. Ed. by A. Voronkov, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie. Timisoara, Romania: IEEE Press, 2013, pp. 30–36. URL: https://kwarc.info/kohlhase/papers/synasc13.pdf.

[KRS25] M. Kohlhase, F. Rabe, and M. Schütz. "Lightweight Realms". Unpublished. 2025.

[KS24] M. Kohlhase and M. Schütz. "Reusing Learning Objects via Theory Morphisms". In: *Intelligent Computer Mathematics (CICM) 2024*. Ed. by A. Kohlhase and L. Kovacz. Vol. 14960. LNAI. Springer, 2024, pp. 165–182. DOI: 10.1007/978-3-031-66997-2_10. URL: https://url.mathhub.info/relocalization/.

[Mar03] J. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill higher education. McGraw-Hill, 2003.

[MK22] D. Müller and M. Kohlhase. "sTeX3 – A LaTeX-based Ecosystem for Semantic/Active Mathematical Documents". In: 43.2 (2022). Ed. by K. Berry, pp. 197–201. URL: https://kwarc.info/people/dmueller/pubs/tug22.pdf.

[Pfe01] F. Pfenning. "Logical Frameworks". In: *Handbook of Automated Reasoning*. Ed. by A. Robinson and A. Voronkov. Vol. I and II. Elsevier Science and MIT Press, 2001.

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

[ST09] J. Sakarovitch and R. Thomas. *Elements of Automata Theory*. Cambridge University Press, 2009.

[Tao22] T. Tao. *There's more to mathematics than rigour and proofs*. 2022. URL: https://terrytao.wordpress.com/career-advice/theres-more-to-mathematics-than-rigour-and-proofs/.

A A Collection of Flexiform Rewriting Operations

During our work on flexiformal recontextualization, we have encountered a large number of rewriting operations that can help with the simplification of the output of the recontextualization pushout (see subsection 2.2 and subsection 2.3). We consider this collection a roadmap for future work.

The complexity of these operations vary from simple replacements (like $\iota_{\in}$ *Reduction* in subsection A.7) to complex rules that require inference (like $\iota_{\in}$ *Conversion* in subsection A.7) that can likely not be implemented in such generality – though support for cases that frequently occur in pratice may be feasible.

So far only the following set of those rules that are used in our running example has been implemented in our prototypes:

- *Element-NP CTR* (see subsection A.6)
- *Variable Renaming* (see subsection A.9)
- *Projection Reduction* (see subsection A.10)
- *Sequence Structure Expansion* (see subsection A.11)
- *Variable Expansion* (see subsection A.12)

As the implementation of rewriting rules in the current prototype is somewhat laborious, we leave the implementation of the remaining rules as future work after the prototypes have undergone a refactorization process (which is still work in progress at the time this paper is written) that will facilitate this implementation.

Rule Format We have developed a relatively grammar-independent, declarative rule format that allows the community to specify and contribute rules for our framework. Those rules consist of two schemas of a natural language expression indicating that any instance of the first one can be transformed to the second one in a meaning-preserving way. The rules may further be extended with a list of preconditions that must be fulfilled in order to guarantee meaning-preservingness. Those preconditions can for instance be assertions about subexpressions of the expressions to be transformed that have to be proven, typing judgements that have to be inferred or subexpressions having a definiens of a specific form. The schemas of the natural language expressions consist of

- (concrete) English words – other languages are not supported yet, but it should be relatively easy to adapt the rules accordingly,
- variables that are intended to be α-renameable
- placeholders that are annotated with linguistic categories and that are intended to be instantiated with any expression of the respective category.

Having an appropriate grammar that captures all expressions used in the rewriting rules – which we leave unspecified at this point to emphasize that we regard it as a community task to continuously develop and improve such a grammar – the expression schemas in those rules can easily be replaced by ASTs produced by the parser for such a grammar.

A.1 Definition Reduction

Consider a symbol c with a definiens δ. Replacing δ by c in a statement S does not change the meaning of S. However, when performing such a replacement we

have to be careful to properly adapt δ to the grammatical structure of S. For instance, consider the expression *"a is applicable in q"* (where a denotes an input symbol and q a state of a NFA) and the definiens *"$\delta(a,q) \neq \emptyset$"* of *"applicable"*. If we want to reduce this definiens in, e.g., *"an input symbol a with $\delta(a,q) \neq \emptyset$ for all final states q"* we have to replace *"with $\delta(a,q) \neq \emptyset$"* with, e.g., *"such that a is applicable in q"* to get the grammatically right expression *"an input symbol a such that a is applicable in q for all final states q"*. Note that we can iteratively apply further rewriting operations to this result to get a more compact expression, for instance *"an input symbol a that is applicable in all final states"*.

A.2 Variable Reduction

Expressions that contain a subexpression e of the form *"x such that"* followed by an expression e' that contains another, redundant occurence of x can often be condensed by merging e with e':

1. **Adjective Variable Reduction:**

$$\frac{\text{“}\langle \textit{noun phrase} \rangle\ x\ (\textit{such that } x \textit{ is} \mid \textit{with } x \textit{ being})\ \langle \textit{adjective phrase} \rangle\text{”}}{\text{“}\langle \textit{noun phrase} \rangle\ x\ \textit{that is}\ \langle \textit{adjective phrase} \rangle\text{”}}$$

   Example:

$$\frac{\text{“}\textit{an input symbol } a \textit{ such that } a \textit{ is applicable in } q\text{”}}{\text{“}\textit{an input symbol } a \textit{ that is applicable in } q\text{”}}$$

2. **Verb Variable Reduction:**

$$\frac{\text{“}\langle \textit{noun phrase} \rangle\ x\ \textit{such that } x\ \langle \textit{verb phrase} \rangle\text{”}}{\text{“}\langle \textit{noun phrase} \rangle\ x\ \textit{that}\ \langle \textit{verb phrase} \rangle\text{”}}$$

   Example:

$$\frac{\text{“}\textit{a prime number } p \textit{ such that } p \textit{ divides } n\text{”}}{\text{“}\textit{a prime number } p \textit{ that divides } n\text{”}}$$

3. **Atom Variable Reduction:** Let R be a symbolic atomic formula with a free variable x.

$$\frac{\text{“}\langle \textit{noun phrase} \rangle\ x\ (\textit{such that} \mid \textit{with})\ R[x]\text{”}}{\text{“}\langle \textit{noun phrase} \rangle\ R[x]\text{”}}$$

   Example:

$$\frac{\text{“}\textit{a state } q \textit{ with } q \in Q \setminus F\text{”}}{\text{“}\textit{a state } q \in Q \setminus F\text{”}}$$

Note that there are cases in which we can eliminate the variable in an adjective or verb variable reduction completely. However, such an elimination depends on the context in which the expression to be reduced occurs. For instance if we would blindly eliminate the variable X in the statement *"Consider a set X such that X is infinite."* by reducing it to *"Consider a set that is infinite."*, any reference to X that occurs in the context of that statement would suddenly be undefined.

A.3 Determiner Reduction

Expressions that involve determiners that bind one or more variables in such a way that there are multiple, redundant occurences of those variables, can be condensed to shorter expressions in some cases:

1. **Adjective Determiner Reduction:**

$$\frac{\text{“}\langle\textit{adjective}\rangle\ \langle\textit{preposition}\rangle\ x\ \textit{for}\ \langle\textit{determiner}\rangle\ \langle\textit{noun phrase}\rangle\ x\text{”}}{\text{“}\langle\textit{adjective}\rangle\ \langle\textit{preposition}\rangle\ \langle\textit{determiner}\rangle\ \langle\textit{noun phrase}\rangle\ x\text{”}}$$

   Example:

$$\frac{\text{“}\textit{applicable in } q \textit{ for a non-initial state } q\text{”}}{\text{“}\textit{applicable in a non-initial state } q\text{”}}$$

   Note that we must not eliminate *all* occurences of x in such an expression as any further references to x would become undefined then.
2. **Atom Determiner Reduction:** Let R be a symbolic atomic formula with a free variable x.

$$\frac{\text{“}\langle\textit{determiner}\rangle\ [\langle\textit{noun phrase}\rangle]\ x\ \textit{with}\ R[x]\text{”}}{\text{“}\langle\textit{determiner}\rangle\ [\langle\textit{noun phrase}\rangle]\ R[x]\text{”}}$$

   Example:

$$\frac{\text{“}\textit{those states } q \textit{ with } q \in Q \setminus F\text{”}}{\text{“}\textit{those states } q \in Q \setminus F\text{”}}$$

3. **Tuple Determiner Reduction:**

$$\frac{\text{“}\langle\textit{determiner}\rangle\ x_1, \ldots, x_n\ \textit{with}\ \langle x_1, \ldots, x_n\rangle \in X_1 \times \ldots \times X_n\text{”}}{\text{“}\langle\textit{determiner}\rangle\ \langle x_1, \ldots, x_n\rangle \in X_1 \times \ldots \times X_n\text{”}}$$

   Example:

$$\frac{\text{“}\textit{any } x, y \textit{ with } \langle x, y\rangle \in \mathbb{R} \times \mathbb{R}\text{”}}{\text{“}\textit{any } \langle x, y\rangle \in \mathbb{R} \times \mathbb{R}\text{”}}$$

A.4 Aggregation

In some cases we can use linguistic or “set-theoretic” aggregations to condense sentences, where the latter transform certain kinds of verbal expressions into corresponding symbolic expressions involving operations on sets.

The following rules capture such aggregation operations (see, e.g., [Dal99] for a more general elaboration of linguistic aggregation rules):

1. **And-Adjective Aggregation:**

$$\frac{\text{“}x\ \textit{is}\ \langle\textit{adjective phrase}\rangle_1\ \textit{and} \ldots \textit{and}\ x\ \textit{is}\ \langle\textit{adjective phrase}\rangle_n\text{”}}{\text{“}x\ \textit{is}\ \langle\textit{adjective phrase}\rangle_1\ \textit{and} \ldots \textit{and}\ \langle\textit{adjective phrase}\rangle_n\text{”}}$$

2. **Or-Adjective Aggregation:**

$$\frac{\text{“}x\ \textit{is}\ \langle\textit{adjective phrase}\rangle_1\ \textit{or} \ldots \textit{or}\ x\ \textit{is}\ \langle\textit{adjective phrase}\rangle_n\text{”}}{\text{“}x\ \textit{is}\ \langle\textit{adjective phrase}\rangle_1\ \textit{or} \ldots \textit{or}\ \langle\textit{adjective phrase}\rangle_n\text{”}}$$

3. **And-Verb Aggregation:**

$$\frac{\text{“}x\ \langle \textit{verb phrase} \rangle_1\ \textit{and} \ldots \textit{and}\ x\ \langle \textit{verb phrase} \rangle_n\text{”}}{\text{“}x\ \langle \textit{verb phrase} \rangle_1\ \textit{and} \ldots \textit{and}\ \langle \textit{verb phrase} \rangle_n\text{”}}$$

4. **Or-Verb Aggregation:**

$$\frac{\text{“}x\ \langle \textit{verb phrase} \rangle_1\ \textit{or} \ldots \textit{or}\ x\ \langle \textit{verb phrase} \rangle_n\text{”}}{\text{“}x\ \langle \textit{verb phrase} \rangle_1\ \textit{or} \ldots \textit{or}\ \langle \textit{verb phrase} \rangle_n\text{”}}$$

5. **Tuple Aggregation:**

$$\frac{\text{“}x_1 \in X_1\ \textit{and} \ldots \textit{and}\ x_n \in X_n\text{”}}{\text{“}\langle x_1, \ldots, x_n \rangle \in X_1 \times \ldots \times X_n\text{”}}$$

6. **And-Element Aggregation:**

$$\frac{\text{“}x \in X_1\ \textit{and} \ldots \textit{and}\ x \in X_n\text{”}}{\text{“}x \in (X_1 \cap \ldots \cap X_n)\text{”}}$$

7. **Or-Element Aggregation:**

$$\frac{\text{“}x \in X_1\ \textit{or} \ldots \textit{or}\ x \in X_n\text{”}}{\text{“}x \in (X_1 \cup \ldots \cup X_n)\text{”}}$$

A.5 Assumption Splitting

Recontextualization and rewriting operations, in particular iterated application of them, can yield deeply nested sentence structures, which can make the result difficult to read and can also cause ambiguity. Thus, when a certain nesting depth is reached, it is reasonable to split such sentences. For instance, consider the sentence *“Let f' be the derivative of a polynomial f on the real numbers with no zeroes.”*. By outsourcing all assumptions on f to a separate sentence, we can get the following: *“Let f be a polynomial on the real numbers with no zeroes. Let f' be the derivative of f.”*

In particular, this resolves the ambiguity of *“zeroes”* referring either to f or to f'. However, if the sentence to be splitted is indeed ambiguous, we have to be careful: We have to make sure that resolving the ambiguity is indeed desired and, if this is the case, which reading is the intended one.

A.6 Comprehension Term Reduction

Often, we want to reduce expressions that involve comprehension terms. A **comprehension term** (**CT**) is, in its most general form, an expression of the form $\mathcal{C} := \{R[t[y_1, \ldots, y_m]] \mid \varphi[y_1, \ldots, y_m]\}$, where R is a unary atomic formula, and t and φ are a term and a (possibly informal) formula, resp., whose free variables are among $y_1, \ldots, y_m$. We consider R trivial if no relational constraints are specified, and we consider t trivial if it is a plain variable, which lets us classify comprehension terms as

- **unguarded** if both R and t are trivial (e.g. "$\{z \mid z^n = 1$ *for some* $n \in \mathbb{N}\}$"),
- **term-guarded** if just t is non-trivial (e.g. "$\{n^2 \mid n \in \mathbb{N}\}$"),
- **relation-guarded** if just R is non-trivial (e.g. "$\{1 \leq i \leq n \mid 1/i < \varepsilon\}$"), or
- **relation-term-guarded** if both R and t are non-trivial (e.g. "$\{\langle p, n\rangle \in \mathbb{N} \times \mathbb{N} \mid p$ *is a prime divisor of* $n\}$").

We call a rewriting operation that eliminates a comprehension term from an expression a **comprehension term reduction** (**CTR**).

If we want to reduce an expression of the form "$x \in \mathcal{C}$"[4] (or, analogously, the verbalized variant "x *is/be an element of* $\mathcal{C}$") we have to distinguish several cases depending on the structure of $\mathcal{C}$. Moreover, for each case we have to take the structure of the sentence that contains the expression "$x \in \mathcal{C}$" into account: *assertion expressions*, i.e. expressions starting with an (optional) "*then*", "*hence*", "*assume*" and the like (denoted by "$\langle$*then*$\rangle$"), *declaration expressions*, i.e. sentences starting with "*let*" and the like (denoted by "$\langle$*let*$\rangle$"), *choice expressions*, i.e. sentences starting with "*choose*", "*take*", "*consider*" and the like (denoted by "$\langle$*choose*$\rangle$") or *determiner expressions*, i.e. expressions starting with a determiner. Moreover, in the case of declaration choice and quantifying expressions we can have additional trailing expressions after the comprehension term that can affect the result of the comprehension term reduction. Such trailing expressions can involve additional adjective phrases (AP), verb phrases (VP), noun phrases (NP) or proposition phrases (PP).

In the following we list reduction rules for expressions of the form $x \in \mathcal{C}$. If $\mathcal{C}$ is a unguarded CT then we can perform the following reductions:

1. **Then CTR:**

$$\frac{\text{“}\langle \textit{then}\rangle\ x \in \{y \mid \varphi[y]\}\text{”}}{\text{“}\langle \textit{then}\rangle\ \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{assume } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\}\text{”}}{\text{“}\textit{assume } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

2. **Let CTR:**

$$\frac{\text{“}\langle \textit{let}\rangle\ x \in \{y \mid \varphi[y]\}\text{”}}{\text{“}\langle \textit{let}\rangle\ x \textit{ be an element such that } \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{let } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\}\text{”}}{\text{“}\textit{let } x \textit{ be an element such that } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

Here we use the expression "*element*" as a "generic type" for x.

3. **Let-NP CTR:**

$$\frac{\text{“}\langle \textit{let}\rangle\ x \in \{y \mid \varphi[y]\} \textit{ be a } \langle \textit{noun phrase}\rangle\text{”}}{\text{“}\langle \textit{let}\rangle\ x \textit{ be a } \langle \textit{noun phrase}\rangle \textit{ such that } \varphi[x]\text{”}}$$

[4] We might also consider expressions of the form "$x_1, \ldots, x_k \in \mathcal{C}$" where more than one variable are invloved on the left-hand side, but this makes the onversion of such comprehension term expressions far more complicated and can lead to blow-ups in the resulting expressions.

Example:

$$\frac{\text{“}\textit{let } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ be a complex number}\text{”}}{\text{“}\textit{let } x \textit{ be a complex number such that } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

4. **Let-AP CTR:**

$$\frac{\text{“}\langle\textit{let}\rangle\ x \in \{y \mid \varphi[y]\} \textit{ be } \langle\textit{adjective phrase}\rangle\text{”}}{\text{“}\langle\textit{let}\rangle\ x \textit{ be } \langle\textit{adjective phrase}\rangle \textit{ such that } \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{let } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ be imaginary}\text{”}}{\text{“}\textit{let } x \textit{ be imaginary such that } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

5. **Let-PP CTR:**

$$\frac{\text{“}\langle\textit{let}\rangle\ x \in \{y \mid \varphi[y]\} \textit{ with } \langle\textit{proposition phrase}\rangle\text{”}}{\text{“}\langle\textit{let}\rangle\ x \textit{ be an element such that } \varphi[x] \textit{ and } \langle\textit{proposition phrase}\rangle\text{”}}$$

Example:

$$\frac{\text{“}\textit{let } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ with } \mathrm{Im}x > 0\text{”}}{\text{“}\textit{let } x \textit{ be an element such that } x^n = 1 \textit{ for some } n \in \mathbb{N} \textit{ and } \mathrm{Im}x > 0\text{”}}$$

Here, “$\mathrm{Im}x$” dentoes the imaginary part of a complex number x.

6. **Choose CTR:**

$$\frac{\text{“}\langle\textit{choose}\rangle\ x \in \{y \mid \varphi[y]\}\text{”}}{\text{“}\langle\textit{choose}\rangle\ x \textit{ with } \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{choose an } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\}\text{”}}{\text{“}\textit{choose an } x \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

7. **Choose-NP CTR:**

$$\frac{\text{“}\langle\textit{choose}\rangle\ x \in \{y \mid \varphi[y]\} \textit{ that is a } \langle\textit{noun phrase}\rangle\text{”}}{\text{“}\langle\textit{choose}\rangle\ \langle\textit{noun phrase}\rangle\ x \textit{ with } \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{choose an } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ that is a real number}\text{”}}{\text{“}\textit{choose a real number } x \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

8. **Choose-AP CTR:**

$$\frac{\text{“}\langle\textit{choose}\rangle\ x \in \{y \mid \varphi[y]\} \textit{ that is } \langle\textit{adjective phrase}\rangle\text{”}}{\text{“}\langle\textit{choose}\rangle\ \langle\textit{adjective phrase}\rangle\ x \textit{ with } \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{choose an } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ that is imaginary}\text{”}}{\text{“}\textit{choose an imaginary } x \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

9. **Choose-VP CTR:**

$$\frac{\text{“}\langle\textit{choose}\rangle\ x \in \{y \mid \varphi[y]\}\ \textit{that}\ \langle\textit{verb phrase}\rangle\text{”}}{\text{“}\langle\textit{choose}\rangle\ x\ \textit{that}\ \langle\textit{verb phrase}\rangle\ \textit{with}\ \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{choose an } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ that solves the equation } E\text{”}}{\text{“}\textit{choose an } x \textit{ that solves the equation } E \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

10. **Choose-PP CTR:**

$$\frac{\text{“}\langle\textit{choose}\rangle\ x \in \{y \mid \varphi[y]\}\ \textit{with}\ \langle\textit{proposition phrase}\rangle\text{”}}{\text{“}\langle\textit{choose}\rangle\ x\ \textit{with}\ \varphi[x]\ \textit{and}\ \langle\textit{proposition phrase}\rangle\text{”}}$$

Example:

$$\frac{\text{“}\textit{choose an } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ with } \mathrm{Re}x > 0\text{”}}{\text{“}\textit{choose an } x \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N} \textit{ and } \mathrm{Re}x > 0\text{”}}$$

Here and in the following, “$\mathrm{Re}x$” dentoes the imaginary part of a complex number x.

11. **Determiner CTR:**

$$\frac{\text{“}\langle\textit{determiner}\rangle\ [\langle\textit{noun phrase}\rangle \mid \langle\textit{adjective phrase}\rangle]\ x \in \{y \mid \varphi[y]\}\text{”}}{\text{“}\langle\textit{determiner}\rangle\ [\langle\textit{noun phrase}\rangle \mid \langle\textit{adjective phrase}\rangle]\ x\ \textit{such that}\ \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{all complex numbers } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\}\text{”}}{\text{“}\textit{all complex numbers } x \textit{ such that } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

12. **Determiner-NP CTR:**

$$\frac{\text{“}\langle\textit{determiner}\rangle\ [\langle\textit{adjective phrase}\rangle]\ x \in \{y \mid \varphi[y]\}\ \textit{that is a}\ \langle\textit{noun phrase}\rangle\text{”}}{\text{“}\langle\textit{determiner}\rangle\ [\langle\textit{adjective phrase}\rangle]\ \langle\textit{noun phrase}\rangle\ x\ \textit{with}\ \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{all positive } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ that are real numbers}\text{”}}{\text{“}\textit{all positive real numbers } x \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

13. **Determiner-AP CTR:**

$$\frac{\begin{array}{c}\text{“}\langle\textit{determiner}\rangle\ [\langle\textit{noun phrase}\rangle \mid \langle\textit{adjective phrase}\rangle]\ x \in \{y \mid \varphi[y]\}\ \textit{that is} \\ \langle\textit{adjective phrase}\rangle\text{”}\end{array}}{\begin{array}{c}\text{“}\langle\textit{determiner}\rangle\ \langle\textit{adjective phrase}\rangle\ [\langle\textit{noun phrase}\rangle \mid \langle\textit{adjective phrase}\rangle]\ x\ \textit{with} \\ \varphi[x]\text{”}\end{array}}$$

Example:

$$\frac{\text{“}\textit{all complex numbers } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ that are imaginary}\text{”}}{\text{“}\textit{all imaginary complex numbers } x \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

14. **Determiner-VP CTR:**

$$\frac{\text{“}\langle \textit{determiner} \rangle\ [\langle \textit{noun phrase} \rangle \mid \langle \textit{adjective phrase} \rangle]\ x \in \{y \mid \varphi[y]\}\ \textit{that}\ \langle \textit{verb phrase} \rangle\text{”}}{\text{“}\langle \textit{determiner} \rangle\ [\langle \textit{noun phrase} \rangle \mid \langle \textit{adjective phrase} \rangle]\ x\ \textit{that}\ \langle \textit{verb phrase} \rangle\ \textit{with}\ \varphi[x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{all complex numbers } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ that solves the equation } E\text{”}}{\text{“}\textit{all complex numbers } x \textit{ that solves the equation } E \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N}\text{”}}$$

15. **Determiner-PP CTR:**

$$\frac{\text{“}\langle \textit{determiner} \rangle\ [\langle \textit{noun phrase} \rangle \mid \langle \textit{adjective phrase} \rangle]\ lx \in \{y \mid \varphi[y]\}\ \textit{with}\ \langle \textit{proposition phrase} \rangle\text{”}}{\text{“}\langle \textit{determiner} \rangle\ [\langle \textit{noun phrase} \rangle \mid \langle \textit{adjective phrase} \rangle]\ x\ \textit{with}\ \varphi[x]\ \textit{and}\ \langle \textit{proposition phrase} \rangle\text{”}}$$

Example:

$$\frac{\text{“}\textit{all complex numbers } x \in \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \textit{ with } \mathrm{Re}x > 0\text{”}}{\text{“}\textit{all complex numbers } x \textit{ with } x^n = 1 \textit{ for some } n \in \mathbb{N} \textit{ and } \mathrm{Re}x > 0\text{”}}$$

Similarly, we can reduce expressions of the form *“$\mathcal{C} \subseteq X$”*. Note that in this case we only need to consider the case of assertion expressions; declaration, choice and determiner expressions do usually not involve comprehension term expressions of the form *“$\mathcal{C} \subseteq X$”*. We can perform the following reductions:

16. **Subset CTR:**

$$\frac{\text{“}\langle \textit{then} \rangle\ \{y \mid \varphi[y]\} \subseteq X\text{”}}{\text{“}\langle \textit{then} \rangle\ y \in X\ \textit{for all } y\ \textit{with } \varphi[y]\text{”}}$$

Example:

$$\frac{\text{“}\textit{Assume } \{z \mid z^n = 1 \textit{ for some } n \in \mathbb{N}\} \subseteq X.\text{”}}{\text{“}\textit{Assume } z \in X \textit{ for all } z \textit{ with } z^n = 1 \textit{ for some } n \in \mathbb{N}.\text{”}}$$

If $\mathcal{C}$ is a guarded CT, we can first apply the following reduction steps to transform $\mathcal{C}$ to a unguarded CT and then apply the rules from above to the resulting expression.

17. **Term Guard Reduction:**

$$\frac{\text{“}\{t[y_1, \ldots, y_m] \mid \varphi[y_1, \ldots, y_m]\}\text{”}}{\text{“}\{z \mid \textit{there exist } y_1, \ldots, y_m \textit{ such that } z = t[y_1, \ldots, y_m] \textit{ and } \varphi[y_1, \ldots, y_m]\}\text{”}}$$

Example:

$$\frac{\text{“}\{n^2 \mid n \in \mathbb{N}\}\text{”}}{\text{“}\{z \mid \textit{there exists an } n \textit{ such that } z = n^2 \textit{ and } n \in \mathbb{N}\}\text{”}}$$

18. **Relation Guard Reduction:**

$$\frac{\text{“}\{R[y] \,|\, \varphi[y_1, \ldots, y_m]\}\text{”}}{\text{“}\{y \,|\, R[y] \text{ and } \varphi[y_1, \ldots, y_m]\}\text{”}}$$

Example:

$$\frac{\text{“}\{1 \leq i \leq n \,|\, 1/i < \varepsilon\}\text{”}}{\text{“}\{i \,|\, 1 \leq i \leq n \text{ and } 1/i < \varepsilon\}\text{”}}$$

19. **Relation-Term Guard Reduction:**

$$\frac{\text{“}\{R[t[y_1, \ldots, y_m]] \,|\, \varphi[y_1, \ldots, y_m]\}\text{”}}{\begin{array}{c}\text{“}\{z \mid \text{there exist } y_1, \ldots, y_m \text{ such that } z = t[y_1, \ldots, y_m] \text{ and } R[y] \text{ and} \\ \varphi[y_1, \ldots, y_m]\}\text{”}\end{array}}$$

Example:

$$\frac{\text{“}\{\langle p, n\rangle \in \mathbb{N} \times \mathbb{N} \,|\, p \text{ is a prime divisor of } n\}\text{”}}{\begin{array}{c}\{z \mid \text{there exist } p, n \text{ such that } z = \langle p, n\rangle \text{ and } z \in \mathbb{N} \times \mathbb{N} \text{ and } p \text{ is a prime}\text{”} \\ \text{divisor of } n\}\text{”}\end{array}}$$

In the special case of a comprehension term $\mathcal{C}$ being of the form *“*$\{y \,|\, y \text{ is a } \langle$*noun phrase*$\rangle\}$*”* we get the following additional reduction rules:

20. **Element-NP CTR:**

$$\frac{\text{“}\textit{element } [x] \textit{ of } \{y \,|\, y \textit{ is a } \langle\textit{noun phrase}\rangle\}\text{”}}{\text{“}\langle\textit{noun phrase}\rangle\ [x]\text{”}}$$

Example:

$$\frac{\text{“}\textit{element } x \textit{ of } \{z \,|\, z \textit{ is a complex number}\}\text{”}}{\text{“}\textit{complex number } z\text{”}}$$

21. **Subset-NP CTR:**

$$\frac{\text{“}\textit{subset } [x] \textit{ of } \{y \,|\, y \textit{ is a } \langle\textit{noun phrase}\rangle\}\text{”}}{\text{“}\textit{set } [x] \textit{ of } \langle\textit{noun phrase}\rangle\text{”}}$$

Example:

$$\frac{\text{“}\textit{subsets of } \{z \,|\, z \textit{ is a complex number}\}\text{”}}{\text{“}\textit{sets of complex numbers}\text{”}}$$

A.7 Iota Conversion

If an expression *“*A*”* denotes a singleton set consisting of an element a, then we can convert an expression of the form *“*$b \in A$*”* to *“*$b = a$*”*. Similarly, we can convert an expression of the form *“*$\{a\} \subseteq B$*”* to *“*$a \in B$*”*. As this is just an application of the ι operator, we call such conversions *iota conversions* (or *ι conversions*). In the most simple case, when *“*A*”* is the expression *“*$\{a\}$*”*, this yields the following reduction rules:

1. $\iota_\in$ **Reduction:**

$$\frac{\text{“}b \in \{a\}\text{”}}{\text{“}b = a\text{”}}$$

2. $\iota_\subseteq$ **Reduction:**

$$\frac{\text{“}\{a\} \subseteq B\text{”}}{\text{“}a \in B\text{”}}$$

If we allow logical inference during the conversion process, we can generalize those rules. If we can infer that A has exactly one element and that a is an element of A, then we can convert an expression of the form *“$b \in A$”* to *“$b = a$”* and *“$A \subseteq B$”* to *“$a \in B$”*:

3. $\iota_\in$ **Conversion:**

$$\frac{\begin{array}{c}\vdash |A| = 1,\\ \vdash a \in A,\\ \text{“}b \in A\text{”}\end{array}}{\text{“}b = a\text{”}}$$

4. $\iota_\subseteq$ **Conversion:**

$$\frac{\begin{array}{c}\vdash |A| = 1,\\ \vdash a \in A,\\ \text{“}A \subseteq B\text{”}\end{array}}{\text{“}a \in B\text{”}}$$

We get a slightly different variant of the above situation if we replace singleton sets by *parametrized* singleton sets, i.e. functions $f\colon A \to \mathcal{P}(B)$ whose values are singleton sets. Then, by abuse of notation, an expression of the form *“$b \in f(a)$”* can be rewritten as *“$b = f(a)$”* which yields the following conversion rule:

5. **Parametrized ι Conversion:**

$$\frac{\begin{array}{c}f\colon A \to \mathcal{P}(B),\\ \vdash \forall a \in A.\, |f(a)| = 1,\\ \text{“}b \in f(a)\text{”}\end{array}}{\text{“}b = f(a)\text{”}}$$

Note that since this conversion is based on a form of abuse of notation, we must be careful when applying it. Whereas for readers who understand that a function of the form $A \to \mathcal{P}(B)$ whose values are singleton sets is essentially the same as a function $A \to B$ it is convenient to replace expressions of the form *“$b \in f(a)$”* by *“$b = f(a)$”*, readers that lack that understanding might be confused when they encounter such an equality expression. Following [Tao22] we call the second kind of reader *rigorous* and the first one *post-rigorous* (wrt. the concept of such functions). The issue of automatically adapting the degree of rigour of flexiformal documents to the specific needs of the respective reader is discussed in more generality in [KRS25].

A.8 Provable Equality Conversion

Assume we have a theorem stating that an expression e is equal to an expression e'. Then e can be converted to e' without changing the meaning of the context it occurs in.

For instance consider a relation R for which we have defined its transitive closure, written as R^+. If we further have a theorem stating "$R^+ = \bigcup_{n\in\mathbb{N}^+} R^n$", we can reduce "$\bigcup_{n\in\mathbb{N}^+} R^n$" to "$R^+$".

A.9 Variable Renaming

When we recontextualize expressions along a theory morphism, we sometimes get results that contain uncommon variable names. For instance, it is common to name the nodes of a path in a graph $v_1, \ldots, v_n$, whereas in a path in an automaton they are usually named $q_1, \ldots, q_n$ instead. Thus at some point in the recontextualization process a variable renaming should take place.

Variable naming in mathematical vernacular is a necessarily heuristic semi-local optimization process that needs to be supported semantically. One idea towards that is to annotating variable declarations with types and, moreover, equip – via STEX annotations to be developed – type-like symbols with a list of "preferred variable names", from which we can pick, and make this part of the notation/presentation process in STEX. As a simple fallback heuristic if no such annotations are given we can, for instance, use the first letter of the verbalization of the respective type symbol as a variable name. In our prototype implementations we use this latter heuristics to rename variables whenever their types change during recontextualization.

Additionally, the user of a system that carries out a recontextualization should be able to change the notation of variables after the recontextualization process. This would allow them to choose their own favorite variable naming conventions such that they is not tied to a fixed convention determined by the system.

A.10 Projection Reduction

Let π_i be the projection from a Cartesian product $A_1 \times \ldots \times A_n$ to its i-th component A_i. Applying β-reduction to such a projection applied to an n-tuple in $A_1 \times \ldots \times A_n$ yields the following rewriting rule which we call *projection reduction*:

$$\frac{\text{“}\pi_i(\langle a_1, \ldots, a_n\rangle)\text{”}}{\text{“}a_i\text{”}}$$

A.11 Structure Expansion

To apply certain rewrtiting operations, the expressions to be rewritten must be of a specific form. Thus, in some cases it it necessary to make this form explicit

within the sentence to be rewritten such that in a further step the expression can be reduced by another operation that depends on this specific form. The following rules capture such expansion operations:

1. **Variable Structure Expansion:**

$$\frac{\text{“}\langle \textit{noun} \rangle\ x\text{”},\quad \langle \text{noun} \rangle := \text{“}A\ \langle \textit{noun} \rangle\ \textit{is a}\ \langle \textit{noun phrase} \rangle\ \langle \textit{symbolic term} \rangle\ \textit{with}\ \ldots\text{”}}{\text{“}\langle \textit{noun} \rangle\ x := \langle \textit{symbolic term} \rangle\text{”}}$$

   Example:

$$\frac{\text{“}\textit{NFA}\ \mathcal{A}\text{”},\quad \text{NFA} := \text{“}\textit{An NFA is a structure}\ \langle Q, \Sigma, \delta, q_0, F \rangle\ \textit{with the following properties:}\ \ldots\text{”}}{\text{“}\textit{NFA}\ \mathcal{A} := \langle Q, \Sigma, \delta, q_0, F \rangle\text{”}}$$

2. **Sequence Structure Expansion:**

$$\frac{\text{“}\textit{sequence}\ x_1, \ldots, x_n\ \textit{of}\ \langle \textit{noun} \rangle\text{”},\quad \langle \text{noun} \rangle := \text{“}A\ \langle \textit{noun} \rangle\ \textit{is a}\ \langle \textit{noun phrase} \rangle\ \langle \textit{symbolic term} \rangle\ \textit{with}\ \ldots\text{”}}{\text{“}\textit{sequence}\ x_1, \ldots, x_n\ \textit{of}\ \langle \textit{noun} \rangle\ x_i := \langle \textit{symbolic term} \rangle_i\text{”}}$$

   Here and in the following, “$\langle \textit{symbolic term} \rangle_i$” denotes the expression “$\langle \textit{symbolic term} \rangle$” with all its free variables indexed with a subscript “i”.

   Example:

$$\frac{\text{“}\textit{sequence}\ t_1, \ldots, t_n\ \textit{of transitions}\text{”},\quad \text{transition} := \text{“}A\ \textit{transition is a triple}\ \langle q, c, q' \rangle\ \textit{with}\ q' \in \delta(q,c).\text{”}}{\text{“}\textit{sequence}\ t_1, \ldots, t_n\ \textit{of transitions}\ t_i := \langle q_i, c_i, {q'}_i \rangle\text{”}}$$

3. **Family Structure Expansion:**

$$\frac{\text{“}\textit{family}\ (x_i)_{i \in I}\ \textit{of}\ \langle \textit{noun} \rangle\text{”},\quad \langle \text{noun} \rangle := \text{“}A\ \langle \textit{noun} \rangle\ \textit{is a}\ \langle \textit{noun phrase} \rangle\ \langle \textit{symbolic term} \rangle\ \textit{with}\ \ldots\text{”}}{\text{“}\textit{sequence}\ x_1, \ldots, x_n\ \textit{of}\ \langle \textit{noun} \rangle\ x_i := \langle \textit{symbolic term} \rangle_i\text{”}}$$

   Example:

$$\frac{\text{“}\textit{family}\ (I_j)_{j \in J}\ \textit{of open intervals}\text{”},\quad \text{open interval} := \text{“}\textit{An open interval is an interval}\ (x,y)\ \textit{with}\ x, y \in \mathbb{R}.\text{”}}{\text{“}\textit{family}\ (I_j)_{j \in J}\ \textit{of open intervals}\ I_j := (x_j, y_j,)\text{”}}$$

A.12 Variable Expansion

Certain rewriting operations depend on a specific form of the expression to be rewritten. Hence, in some cases it can become necessary to expand a variable x in an expression e to a term t that is provably equal to x which brings e in the

right form to apply a certain operation. For instance, if we have an expression e of the form "$\pi_3(x)$", where π_3 denotes the projection onto the third component of a tuple, we cannot reduce e to the third coponent of x until we know what its third component actually is. So if we can derive "$x = \langle a, b, c\rangle$" (for certain expressions a, b, and c) then we can rewrite e to "$\pi_3(\langle a, b, c\rangle)$" and, by applying another rewriting operation in a further step, eventually reduce it to "c".