Extending OpenMath with Sequences

Fulya Horozal, Michael Kohlhase, Florian Rabe

Computer Science, Jacobs University, Bremen

Abstract. Sequences play a great role in mathematical communication. In mathematical notation, we use sequence ellipsis (...) to denote "obvious" sequences like $1, 2, \ldots, 7$, and in conceptualizations sequence constructors like $(i^2+1)_{i \in \mathbb{N}}$. Furthermore, sequences have a prominent role as argument sequences of flexary functions. While the former cases can adequately be represented and reasoned about as domain objects in OpenMath and MathML, argument sequences are at the language level, and can only be represented, but not reasoned with, since the necessary (sequence-schematic) axioms cannot be represented in the language.
In this paper, we present a new content dictionary for domain-level sequences, and the syntax and semantics of a language extension of OpenMath and (content) MathML by (language-level) sequence variables and sequence constructors that restore reasoning without resorting to the un-natural encoding of argument sequences as domain sequence arguments. To make the extended language conservative over standard OpenMath and MathML, we give a meaning-preserving (but structure-violating) translation.

1 Introduction: The Problem

OpenMath is a content-oriented representation format for mathematical formulae. OpenMath objects exist as constants (called symbols), variables, function application, and binding expressions (and less importantly for our purposes — numbers, strings,...). OpenMath extends traditional formal systems in three ways:

i) symbols are identified by a base URI, a content dictionary name, and a local name relative to the content dictionary (CD).
ii) binding expressions are parametric in the binder (symbol), and
iii) function application and binding are **flexary**, i.e., we can apply a function to an argument sequence of arbitrary length, and we can use a binder to bind a variable sequence of arbitrary length.

All of these extensions were made to allow capturing representational practices in mathematics more closely and flexibly than traditional formal systems could. Consider for instance the addition function as an example; Mathematicians routinely write $a+b+c+d$, which OpenMath allows to write as $@(f, a, b, c, d)$[1]. Tra-

[1] Instead of using the XML or binary encoding, we will use an informal one where symbols and variables are represented as letters, applications as $@(F, A_1, \ldots, A_n)$ and bindings as $\beta(B, x_1, \ldots, x_m, A_1, \ldots, A_n)$.

ditional formal systems only allow functions of a fixed arity (we say that application is **fixary** in such systems), so addition is either taken as a binary, associative function $+_2$, so that $1 + 2 + 3 + 4$ is represented e.g. as $@(+_2, @(+_2, 1, 2), @(+_2, 3, 4))$, or addition is taken to be a function $+_{\mathsf{l}}$ from lists to objects and $1 + 2 + 3 + 4$ is represented as $@(+_{\mathsf{l}}, @(\mathsf{l}, 1, 2, 3, 4))$, where l is the list constructor.

Note that the restriction to fixary application is bought by additional levels of representation — that can however be hidden from the mathematician's view via presentation systems (e.g. with [KMR08]). In list representation, which seems close to what we want, we have tacitly used flexary symbols already since l is a flexary function here. In a pure fixary system, we have to represent $1 + 2 + 3 + 4$ as $@(+_{\mathsf{l}}, @(\mathsf{c}, 1, @(\mathsf{c}, 2, @(\mathsf{c}, 3, @(\mathsf{c}, 4, \mathsf{n})))))$, where c and n are the fixary list constructors "cons" and "nil". Clearly this loses the representational immediacy we are striving for in OpenMath.

Another example that has been discussed on the OpenMath mailing list is the case of the "multirelation" constructor that can be used to model aggregated relations like $a \in S \subseteq T$, which are commonly used in mathematical discourse. The preferred solution made heavy use of flexary applications: it was proposed to establish a flexary function symbol m for a "multirelation", so that the formula above can be represented as $@(\mathsf{m}, a, \in, S, \subseteq, T)$. So far so good, but in the content dictionary for m one would like to express that $a \in S \subseteq T$ is just an aggregated form of the two statements $a \in S$ and $S \subseteq T$. To do so in general, one would like to express the recursive equations

$$\begin{aligned} \forall r, x, y.\mathsf{m}(x, r, y) &= r(x, y) \\ \forall x, r, y, t.\mathsf{m}(x, r, y, t) &= r(x, y) \wedge \mathsf{m}(y, t) \end{aligned} \tag{1}$$

This can be (almost) encoded in two FMPs (Formal Mathematical Properties; see [Bus+04, Ch. 4]):

$$\beta(\forall, x, r, y, @(=, @(\mathsf{m}, x, r, y), @(r, x, y)))$$

$$\beta(\forall, x, r, y, t, @(=, @(\mathsf{m}, x, r, y, t), @(\wedge, @(r, x, y), @(\mathsf{m}, y, t)))) \tag{2}$$

Indeed the two FMPs allow to decompose our example $a \in S \subseteq T$, but not the longer one $x < a \in S \subseteq T$, which is encoded as $@(\mathsf{m}, x, <, a, \in, S, \subseteq, T)$. The second equation does not apply, since we would have to match the argument sequence $\in, S, \subseteq, T$ to the variable t. Clearly, any general form that gives the multirelation constructor a meaning would need to match a universally quantified variable against an argument sequence. Similar problems appear, when we want to relate flexary addition to binary addition ($+_2$ which we can conveniently define via the usual recursive equations from zero and the successor function on the natural numbers in the example above) or even to $+_{\mathsf{l}}$.

It seems like a serious shortcoming in the design of OpenMath and a serious impediment to adoption in mathematics that we cannot write down the necessary meaning equations as FMPs. In CMPs (Commented Mathematical Properties; again see [Bus+04, Ch. 4]) where we can use the flexibility of natural language, we can of course express the equations above. However, just resorting to CMPs would break the symmetry between CMPs and FMPs that is arguably assumed in the OpenMath 2 standard.

In this paper, we present a solution to this problem by extending OpenMath with first-class sequences. Before we present our solution, we will discuss in Sect. 2 why alternative encodings will not suffice to solve the problem. In Sect. 3, we present our extension with syntax and semantics and discuss the special role of sequence variables. In Sect. 4, we give an encoding of the extended language into standard OpenMath and content MathML wrt. the content dictionary in Appendix A. Sect. 5 concludes the paper and Sect. 6 discusses future work.

2 An Unsatisfactory Solution

Using the fixary list encoding (i.e. with "cons" and "nil"), we can indeed encode equation (1) as an OpenMath object:

$$\beta(\forall, x, r, y, t, @(=, @(\mathsf{m}, @(\mathsf{c}, x, @(\mathsf{c}, r, @(\mathsf{c}, y, t)))), @(\wedge, @(r, x, y), @(\mathsf{m}, @(\mathsf{c}, y, t))))) \tag{3}$$

Here, t is a "list variable", which matches the tail

$$@(\mathsf{c}, \in, @(\mathsf{c}, S, @(\mathsf{c}, \subseteq, @(\mathsf{c}, T, \mathsf{n}))))$$

of the argument list

$$@(\mathsf{m}, @(\mathsf{c}, x, @(\mathsf{c}, <, @(\mathsf{c}, a, @(\mathsf{c}, \in, @(\mathsf{c}, S, @(\mathsf{c}, \subseteq, @(\mathsf{c}, T, \mathsf{n}))))))))$$

which encodes the multirelation $x < a \in S \subseteq T$ in our example.

Note that this treatment can be generalized to cover all cases of meaning formulae for flexary functions; but at what cost:

C1. we add a significant amount of representational infrastructure that can only be considered as an artefact of the absence of a proper handling of argument sequences.

C2. we can no longer use the *feature of flexary functions*, but treat it like a *bug* instead by relegating it to "informal math".

C3. when we formalize informally given mathematics, we have to break formula representations, which puts up additional hurdles to realizing one of the foundational principles of mathematics: "if needed, all rigorous mathematics, could be made formal".

C4. It is natural to expect that we can to pass sequences to any flexary symbol as arguments.. Using lists, each symbol declaration has to explicitly permit a list to be passed as an arguments.

C5. we seriously impede our ability to talk about lists, if we use lists to encode language structures.

We might be able to alleviate the situation in **C1.** by using the primitive (i.e., flexary) list constructor l from above together with more advanced functions, e.g., the append function app in our situation.

$$\beta(\forall, x, r, y, t, @(=, @(\mathsf{m}, @(\mathsf{app}, @(\mathsf{l}, x, r, y), t)), @(\wedge, @(r, x, y), @(\mathsf{m}, @(\mathsf{c}, y, t))))) \tag{4}$$

However, the gain is not so pronounced as (4) shows (though it will be larger for more complicated meaning equations), and the meaning equation for `app` would be difficult (probably impossible) to express in an FMP.

C5 can be alleviated by having two CDs for lists: one for "argument lists", and one for "lists as mathematical objects". But that would put "argument lists" much more into the realm of a language feature.

3 Sequences in OpenMath

We propose an extension to the OpenMath language inspired by the language-level sequences of [KB04] and [Wol02]. A central idea is to introduce a new type of basic OpenMath object: sequences of objects.

We present the abstract syntax of our proposed extension in Sect.3.1, give its semantics in Sect.3.2 and the corresponding concrete syntax in Sect.3.3.

3.1 Abstract Syntax

We will restrict attention to a core OpenMath language using variables, constants, integers, applications, bindings, and attributions. To that, we add sequences and constructors and selectors for them. In order to give and talk about our objects conveniently, we will use the abstract syntax with short-hand notations given in Fig.1. Here we mark all of our extensions in red color.

We will use the meta-variables E, E', V, V' to denote OpenMath expressions, c, c', k, k' to denote OpenMath symbols and S, S' to denote OpenMath sequences.

Expressions	E	$::=$	$x \mid c \mid n \mid @(E, S) \mid \beta(E, \Gamma, S) \mid (E\,c\,E) \mid S_E$
Sequences	S	$::=$	$\cdot \mid S, I$
Sequence items	I	$::=$	$\varsigma \mid E \mid [E]_{x/S} \mid \leq n$
Contexts	Γ	$::=$	$\cdot \mid \Gamma, x\,c\,E \ldots c\,E \mid \Gamma, \varsigma\,c\,S \ldots c\,S$

Fig. 1. Abstract OpenMath Syntax with Sequences

An OpenMath **expression** can be
- an OpenMath variable x (*i.e.*, OMV in OpenMath concrete syntax),
- an OpenMath symbol c (*i.e.*, OMS),
- an integer number n (*i.e.*, OMI),
- an application $@(E, S)$ (*i.e.*, OMA) of an operator E to a sequence S of arguments,
- a binding $\beta(E, \Gamma, S)$ (*i.e.*, OMBIND) that binds a list Γ of variables in a sequence[2] S using a binder E,

[2] OpenMath uses a single expression instead of a sequence as the scope of a binding. Our generalization picks up a separate OpenMath extension proposal made in [DK09].

– an attribution $(E\,k\,V)$ (*i.e.*, OMATTR) that attributes a key-value pair (k, V) (*i.e.*, OMATP) to the expression E[3],
– the element S_E of a sequence S at the position E whose at least first E elements are expressions[4].

A **sequence** S consists of **sequence items** I, which can be
– a sequence variable ς,
– an OpenMath expression E,
– $[E]_{x/S}$, which denotes the sequence whose element at index k arises by substituting the free variable x in E with the k-th element of S, *e.g.*, $[E(x)]_{x/E_1,\ldots,E_n}$ denotes the sequence $E(E_1), \ldots, E(E_n)$,
– $\leq n$, which denotes the sequence containing natural numbers from 1 to n.

Finally, a **context** is a list of attributed variables where we distinguish expression variables x and sequence variables ς. Note that if a content dictionary introduces, for example, an attribution key : for the type of a variable, then our abstract syntax for attributed variables yields the familiar notation $\Gamma,\, x : E$.

Example 1 (Multirelation). Recall our example, the multirelation operator m, from Sect. 1. Using sequence variables, we can give a correct encoding of equation (1).

$$\beta(\forall, x, r, y, @(=, @(\mathsf{m}, x, r, y), @(r, x, y)))$$
$$\beta(\forall, x, r, y, t, @(=, @(\mathsf{m}, x, r, y, t), @(\wedge, @(r, x, y), @(\mathsf{m}, y, t)))) \qquad (5)$$

Note that the equations (2) and (5) look the same, however the variable t in equation (5) is a sequence variable.

The sequence items $[E]_{x/S}$ and $\leq n$ together can be used to express mathematical expressions that contain ellipses. For example:

Example 2 (Sequences in Mathematics). We take the sum of elements of the sequence n^2 as our example, *i.e.*,

$$\sum_{i=1}^{n} i^2 = 1^2 + \cdots + n^2. \qquad (6)$$

1. For the right-hand side of (6),
   – first, we express the sequence $1^2, 2^2, \ldots, n^2$ using the sequence constructor $[E]_{x/S}$, where we put i^2 for E, i for x and the sequence $1, \ldots, n$ for S, *i.e.*, we have $[i^2]_{i/\leq n}$,
   – then, we apply the flexary OpenMath symbol $+$ to the sequence $1^2, 2^2, \ldots, n^2$ *i.e.*, $@(+, [i^2]_{i/\leq n})$.

[3] OpenMath permits multiple key-value pairs in an attribution but defines them to be equivalent to several nested attributions. Therefore, we can restrict attention to a single key-value pair without loss of generality.
[4] This, of course, only makes sense if E denotes a natural number.

2. The encoding of the left-hand side of (6) is straightforward: $@(@(\sum, 1, n), \beta(\lambda, i, i^2))$.
3. Finally, we put the left- and the right-hand side of (6) in an application using the symbol $=$, and we get $@(=, @(@(\sum, 1, n), \beta(\lambda, i, i^2)), @(+, [i^2]_{i/\leq n}))$.

Example 3 (Sequences in Proof Theory). When giving natural deduction inference rules, we deal with three kinds of flexary symbols. The rule constructor $\Longrightarrow$ takes a flexary number of premises and one conclusion. Similarly, the sequent constructor $\vdash$ takes a flexary number of formulas in the antecedent and one in the succedent. Finally, the associative operators disjunction and conjunction are flexary as well.

Using sequences, the inference rule for the introduction of flexary conjunction

$$\Theta \vdash A_1, \ldots, \Theta \vdash A_n \Longrightarrow \Theta \vdash A_1 \wedge \ldots \wedge A_n$$

can be encoded as follows

$$@(\Longrightarrow, [@(\vdash, \Theta, x)]_{x/A}, @(\vdash, \Theta, @(\wedge, A)))$$

where Θ and A are free sequence variables. The sequence of n hypotheses on the left-hand side of $\Longrightarrow$ is encoded as $[@(\vdash, \Theta, x)]_{x/A}$, where the free variable x in $@(\vdash, \Theta, x)$ is substituted for every element A_i in A.

3.2 Semantics

In order to define the semantics of our proposed OpenMath extension, we use the judgments in Fig. 2. These judgments are relative to an arbitrary fixed set of content dictionaries, which we denote as *CDs*. Most importantly, the judgment $E \rightsquigarrow_{CDs} E'$ defines the semantics of OpenMath expressions by elaborating them to expressions without sequences. The judgment $S \rightsquigarrow_{CDs} S'$ simplifies a sequence item to the concrete sequence of expressions it denotes.

Judgment	Meaning
$\vdash_{CDs} \Gamma$	Γ is a well-formed context.
$\Gamma \vdash_{CDs} E$	E is a well-formed expression over *CDs* and context Γ.
$\Gamma \vdash_{CDs} S$	S is a well-formed sequence over *CDs* and context Γ.
$\gamma \rightsquigarrow \gamma'$	γ simplifies to the context γ' over *CDs*.
$E \rightsquigarrow E'$	E simplifies to the expression E' over *CDs*.
$S \rightsquigarrow S'$	S simplifies to the sequence S' over *CDs*.

Fig. 2. Judgments

Well-formedness We follow the OpenMath principle to guarantee structural well-formedness of objects rather than semantical well-formedness. Actually, our well-formedness rules are slightly stronger: We require that well-formed OpenMath objects use only variables from a context Γ or symbols from CDs. Below we present only those rules that go beyond formalizing the usual OpenMath objects.

A well-formed context Γ contains expression variables x or sequence variables ς. The rule $\mathcal{W}_{conseq}$ allows one to add attributed sequence variables to well-formed contexts.

$$\frac{\vdash_{\mathit{CDs}} \Gamma \quad c_i \in \mathit{CDs} \quad \Gamma \vdash_{\mathit{CDs}} S_i \quad \text{for } i = 1, \ldots, n}{\vdash_{\mathit{CDs}} \Gamma, \varsigma\, c_1\, S_1\, \ldots\, c_n\, S_n} \mathcal{W}_{conseq}$$

The rule $\mathcal{W}_{seqind}$ for the well-formedness of S_E is a straightforward recursion.

$$\frac{\Gamma \vdash_{\mathit{CDs}} S \quad \Gamma \vdash_{\mathit{CDs}} E}{\Gamma \vdash_{\mathit{CDs}} S_E} \mathcal{W}_{seqind}$$

The rules for well-formed sequences are given in Fig. 3. These rules describe how to form sequences by successively appending sequence items to the empty sequence. Note that the sequence item $[E]_{x/S}$ binds the variable x in E.

$$\frac{\vdash_{\mathit{CDs}} \Gamma}{\Gamma \vdash_{\mathit{CDs}} \cdot} \mathcal{W}_{empty} \qquad \frac{\Gamma \vdash_{\mathit{CDs}} S \quad \Gamma \vdash_{\mathit{CDs}} I}{\Gamma \vdash_{\mathit{CDs}} S, I} \mathcal{W}_{seq}$$

$$\frac{\varsigma \in \Gamma}{\Gamma \vdash_{\mathit{CDs}} \varsigma} \mathcal{W}_{seqvar} \qquad \frac{\Gamma, x \vdash_{\mathit{CDs}} E \quad \Gamma \vdash_{\mathit{CDs}} S}{\Gamma \vdash_{\mathit{CDs}} [E]_{x/S}} \mathcal{W}_{seqconstr} \qquad \frac{\vdash_{\mathit{CDs}} \Gamma}{\Gamma \vdash_{\mathit{CDs}} \leq n} \mathcal{W}_{seqnat}$$

Fig. 3. Well-Formed OpenMath Sequences

Equivalence We have introduced the two new, high-level sequence constructors $\leq n$ and $[E]_{x/S}$, which in some situations can be expressed more directly as elementary sequences: OpenMath expressions with sequences can be elaborated or simplified using the equational theory of sequences. This is a straightforward recursion except for the case of S_E in Fig. 4:

S_E simplifies to an OpenMath expression if E simplifies to a natural number n and S simplifies to a sequence whose first n (or more) elements are expressions. Then S_E simplifies to the n-th element of this sequence.

The simplification of sequences is defined by induction on the sequence items. The rules are given in Fig. 4. Sequence variables simplify to themselves. $\leq n$ simplifies to a sequence of natural numbers up to n starting from 1.

In the rule $\mathcal{S}^2_{seqconstr}$, we write $E(x)$ to emphasize that x is free in E, and we write $E(E')$ for the result of substituting E' for x in E. The rules $\mathcal{S}^1_{seqconstr}$ and $\mathcal{S}^2_{seqconstr}$ together give the semantics of $[E]_{x/S}$.

$$\frac{S \rightsquigarrow E_1, \ldots, E_n, I, \ldots, I \quad E \rightsquigarrow n}{S_E \rightsquigarrow E_n}\,\mathcal{S}_{seqind}$$

$$\frac{}{\cdot \rightsquigarrow \cdot}\,\mathcal{S}_{empty} \qquad \frac{S \rightsquigarrow S' \quad I \rightsquigarrow I_1, \ldots, I_n}{S, I \rightsquigarrow S', I_1, \ldots, I_n}\,\mathcal{S}_{seq} \qquad \frac{}{\varsigma \rightsquigarrow \varsigma}\,\mathcal{S}_{seqvar}$$

$$\frac{S \rightsquigarrow I_1, \ldots, I_n \quad E \rightsquigarrow E'}{[E]_{x/S} \rightsquigarrow [E']_{x/I_1}, \ldots, [E']_{x/I_n}}\,\mathcal{S}^1_{seqconstr} \qquad \frac{E_1 \rightsquigarrow E_1' \quad E_2 \rightsquigarrow E_2'}{[E_1(x)]_{x/E_2} \rightsquigarrow E_1'(E_2')}\,\mathcal{S}^2_{seqconstr}$$

$$\frac{}{\leq n \rightsquigarrow 1, \ldots, n}\,\mathcal{S}_{seqnat}$$

Fig. 4. Simplification of OpenMath Sequences

The simplification of contexts is straightforward recursion into the expressions and sequences that occur in the attributions.

Note that these rules are language-level equivalence rules much like the rules for α-renaming of bound variables in binders expressed in the OpenMath standard and the MathML 3 recommendation. In particular, even though these rules are written as simplification rules[5], this does not clash with the OpenMath principle that *content representation formats should not prejudice any built-in simplification* effective in OpenMath and MathML.

3.3 Concrete Syntax

In this section, we present the concrete OpenMath and Content MathML syntax we devise for sequences. We stay as close to the existing OpenMath and Content MathML sytax as possible.

Sequence Variables Similar to OpenMath variables, the concrete syntax for sequence variables is the tag <**OMSV** name="..."/> with an attribute for the name of the sequence variables. For example, the sequence variable ς is written as <**OMSV** name="ς"/>. In the Content MathML syntax, we write <**si**>ς</**si**>.

[5] A sign that the introduced theory of sequences is computationally trivial and can easily be integrated into OpenMath and content MathML implementations.

n-th Element of a Sequence We introduce the tag <**OMNTH**>...</**OMNTH**> for the element S_E of a sequence at index E. For example, the third element of the sequence $2, 4, 6$ is illustrated in Fig. 5, where the index number 3 comes before the sequence itself.

```
<OMNTH>               <nth>
  <OMI>3</OMI>          <cn>3</cn>
  <OMI>2</OMI>          <cn>2</cn>
  <OMI>4</OMI>          <cn>4</cn>
  <OMI>6</OMI>          <cn>6</cn>
</OMNTH>              </nth>
```

Fig. 5. An Example of S_E in the OpenMath and Content MathML Sytax

Sequences of Naturals We introduce the tag <**OMNATS**>...</**OMNATS**> for the sequence $\leq n$ of natural numbers upto n. For example, ≤ 50 is written as <**OMNATS**><**OMI**>50</**OMI**></**OMNATS**> in OpenMath and as <**nats**><**cn**>50</**cn**></**nats**> in Content MathML.

Sequence Constructor $[E]_{x/S}$ We introduce the tag <**OMSEQ**>...</**OMSEQ**> for the sequence $[E]_{x/S}$. It is wrapped around the binding $\lambda x. E(x)$ followed by the sequence S. For example, $[x^2]_{x/\leq 50}$, which constructs the sequence $1^2, 2^2, \ldots, 50^2$, is written as follows.

```
<OMSEQ>                                     <seq>
  <OMBIND>                                    <bind>
    <OMS cd="lc" name="lambda"/>                <csymbol cd="lc">lambda</csymbol>
    <OMBVAR>                                    <bvar>
      <OMV name="x"/>                             <ci>x</ci>
    </OMBVAR>                                   </bvar>
    <OMA>                                       <apply>
      <OMS cd="arith1" name="power"/>             <csymbol cd="arith1">power</csymbol>
      <OMI>2</OMI>                                <cn>2</cn>
      <OMV name="x"/>                             <ci>x</ci>
    </OMA>                                      </apply>
  </OMBIND>                                   </bind>
  <OMNTH>                                     <nats>
    <OMI>50</OMI>                             <cn>50</cn>
  <OMNTH>                                     </nats>
</OMSEQ>                                    </seq>
```

Fig. 6. An Example of $[E]_{x/S}$ in the OpenMath and Content MathML Sytax

4 Encoding Sequences in OpenMath

In this section, we encode sequences in terms of plain OpenMath expressions by introducing a special content dictionary (named `seqs.ocd`, see Appendix A). This content dictionary contains symbols with the following names: `sequence`, `nats`, `seq`, `nth` and `substituent`. `sequence` is a flexary operator, `nats` is unary, and `nth` binary, `seq` is a binder binding one variable to which we attribute an object with the semantic-attribution `substituent`. The content dictionary for these symbols and examples of how they are used can be found in Appendix A. We are also using the symbol `type` from the content dictionary `sts.ocd`.

By "plain OpenMath", we mean the fragment of our syntax without sequences and sequence variables. More precisely, we still permit the sequences S of the form $E_1, \ldots, E_n$ in $@(E, S)$.

We define the encoding of contexts, sequences, and expressions by induction on our syntax:

$$
\begin{aligned}
\overline{\cdot} &= \cdot \\
\overline{\Gamma,\, x\, c_1\, E_1\, \ldots\, c_n\, E_n} &= \overline{\Gamma},\, x\, c_1\, \overline{E_1}\, \ldots\, c_n\, \overline{E_n} \\
\overline{\Gamma,\, \varsigma\, c_1\, S_1\, \ldots\, c_n\, S_n} &= \overline{\Gamma},\, \varsigma\, \mathtt{type}\, \mathtt{sequence}\, c_1\, \overline{S_1}\, \ldots\, c_n\, \overline{S_n} \\[1ex]
\overline{I_1,\, \ldots\, I_n} &= @(\mathtt{sequence}, \overline{I_1},\, \ldots,\, \overline{I_n}) \\
\overline{\varsigma} &= \varsigma \\
\overline{[E]_{x/S}} &= \beta(\mathtt{seq}, x\, \mathtt{substituent}\, \overline{S}, \overline{E}) \\
\overline{\leq n} &= @(\mathtt{nats}, n) \\[1ex]
\overline{x} &= x \\
\overline{c} &= c \\
\overline{@(E, I_1, \ldots, I_n)} &= @(\overline{E}, \overline{I_1}, \ldots, \overline{I_n}) \\
\overline{\beta(E, \gamma, S)} &= \beta(\overline{E}, \overline{\gamma}, \overline{S}) \\
\overline{(E\, k\, V)} &= (\overline{E}\, k\, \overline{V}) \\
\overline{S_E} &= @(\mathtt{nth}, \overline{E}, \overline{S})
\end{aligned}
$$

Note that this encoding turns sequence variables into "normal" variables that are attributed by the key-value pair (`type`, `sequence`). Moreover, note that when encoding $@(E, S)$, the argument sequence S is encoded as an OpenMath argument sequence, i.e., without wrapping it in `sequence`. However, `sequence` is necessary when sequences occur in other positions.

5 Conclusion

We have motivated and formalized a proposal for the extension of OpenMath with language-level sequences. We suggest considering it during the development process towards OpenMath 3.

We have implemented our language and the operations on it within the OpenMath API that is part of the MMT system [Rab].

In future work, we will design a role system for our language. It is often desirable to restrict attention to some set of well-typed objects. For example, we can ascribe an arity to an operator and permit only OMA objects with the correct number of arguments. These arities should permit flexary operators and binders. OPENMATH's role system is implementing the simplest possible case of such an arity system — but it provides only two arities for operators: nullary (the role `constant`) and flexary (the role `application`). Based on [RK09], we will design a more fine-grained role system that supports both fixary and flexary roles both for operators and for binders.

6 Future Work

We have already discussed the importance of flexary representations of operators like the set and list constructors or associative binary operators. For example, in the latter case, typically, after proving associativity a new flexary operator is introduced that returns the sum of arbitrarily many arguments, e.g., in

$$@(+, a_1, \ldots, a_n) := @(+, @(+, a_1, \ldots, a_{n-1}), a_n) \qquad \text{for } n \geq 3.$$

As we have seen, by building sequences into the language, we can represent such formulas elegantly.

A similar effect to associative binary operators exists for binders. For example, just like for addition above, we can define a flexary universal quantifier by

$$\beta(\forall, x_1, \ldots, x_n, F) := \beta(\forall, x_1, \ldots, x_{n-1}, \beta(\forall, x_n, F)) \qquad \text{for } n \geq 2.$$

Many unary binders are commonly used in their flexary in this way, e.g., $\exists$, λ, and $\int$. A counter-example is the binder $\exists^1$ of unique existence, where $\beta(\exists^1, x, y, F)$ expresses the unique existence of a pair (x, y) such that $F(x, y)$.

The motivation of sequence variables is to exploit flexary binders in the same way as flexary operators. However, we have to be careful here:

$$\langle\!\langle \text{sequence argument} \rangle\!\rangle = \langle\!\langle \text{argument sequence} \rangle\!\rangle, \text{ but}$$
$$\langle\!\langle \text{sequence variable} \rangle\!\rangle \neq \langle\!\langle \text{variable sequence} \rangle\!\rangle$$

Here by a sequence argument, we mean the S in $@(E, S)$; and by an argument sequence, we mean the sequence $E_1, \ldots, E_n$ in $@(E, E_1, \ldots, E_n)$. Clearly, the latter is a special case of the former. Conversely, by the rules in Fig. 4, every ground sequence argument can be elaborated into an argument sequence (if all expressions E occurring in sub-objects S_E can be effectively evaluated to a natural number).

By a sequence variable, we mean the ς in $\beta(b, \varsigma, E)$; and by a variable sequence, we mean the sequence $x_1, \ldots, x_n$ in $\beta(b, x_1, \ldots, x_n, E)$. The former cannot be elaborated into the latter because it represents a variable sequence *of arbitrary length*, whereas the latter has a fixed length.

Therefore, while permitting the application of a flexary operators to a sequence argument is conservative, the binding of a sequence by a flexary does not have to be. For example, consider first-order logic with a flexary disjunction. Then we can define the predicate "E is an element of the sequence S" by

$$E \in S := @(\vee, [E = x]_{x/S}).$$

Then the axiom

$$\beta(\exists, \varsigma, \beta(\forall, x, x \in \varsigma))$$

expresses the finiteness of the universe — a property that cannot be formalized in ordinary first-order logic.

But much of the appeal of flexary operators and binders is exactly in their property that they can be introduced conservatively. Given a sequence-less base language, e.g., first-order logic, we want the meta-level representation language, i.e., OpenMath, to add sequences in a conservative way.

There are various ways to avoid the above problem. We find the most elegant to use fixed-length sequence variables. This can be achieved by attributing a sequence of types to the sequence variable. We can define $\varsigma : S$ to introduce a sequence variable whose elements are typed according to the elements of S; this fixes the length of ς to be equal to that of S. In particular, if $S \rightsquigarrow E_1, \ldots, E_n$, then $\varsigma : S$ can be elaborated to $x_1 : E_1, \ldots, x_n : E_n$. Therefore, we have

$$\langle\!\langle \text{fixed-length sequence variable} \rangle\!\rangle = \langle\!\langle \text{variable sequence} \rangle\!\rangle$$

Note that the length of a sequence variable may still be arbitrary by using a sequence variable in the type attribution, which is itself of arbitrary length, as in

$$\tau \vdash \beta(\exists, \varsigma : [U]_{x/\tau}, \beta(\forall, x, x \in \varsigma))$$

Here τ is a free sequence variable and U is the type representing the first-order universe. In first-order logic, this is a perfectly acceptable formula scheme using τ a meta-variable. But because we cannot quantify over τ, the above increase in expressivity does not occur.

References

[Bus+04] Stephen Buswell et al. *The Open Math Standard, Version 2.0*. Tech. rep. The OpenMath Society, 2004. URL: `http://www.openmath.org/standard/om20`.

[DK09] James H. Davenport and Michael Kohlhase. "Unifying Math Ontologies: A tale of two standards". In: *MKM/Calculemus Proceedings*. Ed. by Jacques Carette et al. LNAI 5625. Springer Verlag, July 2009, pp. 263–278. ISBN: 9783642026133. URL: `http://kwarc.info/kohlhase/papers/mkm09-MMLOM3.pdf`.

[KB04] Temur Kutsia and Bruno Buchberger. "Predicate Logic with Sequence Variables and Sequence Function Symbols". In: *Mathematical Knowledge Management, MKM'04*. Ed. by Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec. LNAI 3119. Springer Verlag, 2004, pp. 205–219.

[KMR08] Michael Kohlhase, Christine Müller, and Florian Rabe. "Notations for Living Mathematical Documents". In: *Intelligent Computer Mathematics.* 9th International Conference, AISC, 15th Symposium, Calculemus, 7th International Conference MKM (Birmingham, UK, July 28–Aug. 1, 2008). Ed. by Serge Autexier et al. LNAI 5144. Springer Verlag, 2008, pp. 504–519. URL: `http://omdoc.org/pubs/mkm08-notations.pdf`.

[Rab] Florian Rabe. *MMT – A Module system for Mathematical Theories.* URL: `http://trac.kwarc.info/MMT/` (visited on 08/18/2010).

[RK09] Florian Rabe and Michael Kohlhase. "A better Role System for OpenMath". In: *22nd OpenMath Workshop.* Ed. by James H. Davenport. July 2009. URL: `http://kwarc.info/kohlhase/papers/om09-roles.pdf`.

[Wol02] Stephen Wolfram. *The Mathematica Book.* Cambridge University Press, 2002.

A The Content Dictionary `seqs.ocd` for Encoding Sequence Features as Symbols

```
<CD xmlns="http://www.openmath.org/OpenMathCD">

<CDName> seq </CDName>
<CDBase>http://www.openmath.org/cd</CDBase>
<CDURL> http://www.openmath.org/cd/seqs.ocd </CDURL>

<Description>
This CD defines symbols for working with sequences.
</Description>

<CDDefinition>
<Name> sequence </Name>
<Role> Application </Role>
<Description>
The symbol to represent a sequence of n objects.
</Description>

<CMP> sequence(a,b,c) </CMP>

<FMP>
<OMOBJ xmlns="http://www.openmath.org/OpenMath" version="2.0" cdbase="http://www.
      openmath.org/cd">
  <OMA>
    <OMS cd="seqs" name="sequence"/>
    <OMV name="a"/>
    <OMV name="b"/>
    <OMV name="c"/>
  </OMA>
</OMOBJ>
</FMP>
</CDDefinition>

<CDDefinition>
<Name> nth </Name>
<Role> Application </Role>
<Description>
The symbol to retrieve the n−th element of a sequence.
</Description>
```

```
<CMP> nth(2,sequence(a,b,c)) </CMP>

<FMP>
<OMOBJ xmlns="http://www.openmath.org/OpenMath" version="2.0" cdbase="http://www.
      openmath.org/cd">
  <OMA>
    <OMS cd="seqs" name="nth"/>
    <OMI>2</OMI>
    <OMA>
      <OMS cd="seq" name="sequence"/>
      <OMV name="a"/>
      <OMV name="b"/>
      <OMV name="c"/>
    </OMA>
  </OMA>
</OMOBJ>
</FMP>
</CDDefinition>

<CDDefinition>
<Name> nats </Name>
<Role> Application </Role>
<Description>
The symbol to construct the sequence from 1 to a given natural number n.
</Description>

<CMP> nats(3) </CMP>

<FMP>
<OMOBJ xmlns="http://www.openmath.org/OpenMath" version="2.0" cdbase="http://www.
      openmath.org/cd">
  <OMA>
    <OMS cd="seqs" name="nats"/>
    <OMI>3</OMI>
  </OMA>
</OMOBJ>
</FMP>
</CDDefinition>

<CDDefinition>
<Name> substituent </Name>
<Role> Semantic−Attribution </Role>
<Description>
The symbol to attribute to a variable a (possibly list of) substituent(s).
</Description>
</CDDefinition>

<CDDefinition>
<Name> seq </Name>
<Role> Binder </Role>
<Description>
The symbol to construct a sequence from an object with a free variable and a list of objects.
</Description>

<CMP> (seq(x,sequence(1,2,3)). x^2) </CMP>

<FMP>
<OMOBJ xmlns="http://www.openmath.org/OpenMath" version="2.0" cdbase="http://www.
      openmath.org/cd">
  <OMBIND>
    <OMS cd="seqs" name="seq"/>
    <OMBVAR>
      <OMATTR>
        <OMATP>
          <OMS cd="seqs" name="substituent"/>
          <OMA>
            <OMS cd="seqs" name="sequence"/>
```

```
              <OMI>1</OMI>
              <OMI>2</OMI>
              <OMI>3</OMI>
            </OMA>
          </OMATP>
          <OMV name="x"/>
        </OMATTR>
      </OMBVAR>
      <OMA>
        <OMS cd="arith1" name="power"/>
        <OMI>2</OMI>
        <OMV name="x"/>
     </OMA>
   </OMBIND>
</OMOBJ>
</FMP>
</CDDefinition>
</CD>
```