Extension Proposal: Records in Pragmatic OpenMath

Michael Kohlhase

Computer Science, Jacobs University
http://kwarc.info/kohlhase

Abstract

I propose to extend the pragmatic syntax of OpenMath by records. Record structures are utilized ubiquitously for representing objects and accessing their components in programming, and we show that this is true for mathematical practice at the informal but rigorous level as well, even though at the formal level records can be reduced to tuples or partial functions. This situation makes records an ideal case study for an OpenMath language extension (OLE) proposed in a sibling paper.

1 Introduction

OpenMath is a standard for representing the structure of mathematical objects, so that they can be exchanged between computer programs, stored in databases, or published on the worldwide web without changing their intended meaning. To achieve this goal, mathematical entities are represented as OpenMath objects, which are term structures built up as *i*) flexary function applications and *ii*) bindings from *iii*) variables and *iv*) constants . The meaning of constants – called "symbols" in OpenMath– is specified in OpenMath Content Dictionaries (CDs). The four constructors of OpenMath objects[1] can be considered the minimal set of language primitives necessary to represent mathematical objects and formal statements about them in the form of (logical) formulae. Even though it has been noted that fixary function application and λ-abstraction would also suffice, OpenMath has opted for the richer primitives so that the structure of mathematical expressions can be better preserved. With fixary applications, we would have to represent a sum "$a+c+2$" as either $@(+,a,@(+c,2))$[2] or $@(@(+a,c),2)$ instead of $@(+a,c,2)$, forcing a choice of operator nesting that is mathematically meaningless on the user. Similarly, an integral would have to be represented as $@(\int,\lambda(x,A))$ instead of $\beta(\int,x,A)$.

In this paper we want to highlight a situation, where OpenMath does not provide structure-preserving syntax for constructions used heavily in mathematical practice. I feel that this case is important enough to warrant a language primitive for this. Consider the following two examples

(1) We call a triple $\mathcal{R} := (R,+,\cdot)$ a **ring**, iff .... R is called the **base set** of $\mathcal{R}$, $+$ and $\cdot$ are called the **additive** and **multiplicative** operations of $\mathcal{R}$.

(2) We call a ring **commutative** iff its multiplicative operation is.

(3) Let $\mathcal{R} := (R,+,0,-,*,1)$ be a ring, ...

These two examples that can be found in any book on elementary algebra already show that we often deal with complex structures made up of simpler structures. Standard mathematical practice establishes a conceptual infrastructure for such structures that goes beyond the minimal requirements of using tuples for that. Two practices are salient here:

[1] OpenMath objects also include mathematically relevant primitive data types for numbers and strings, but these are irrelevant to this paper.

[2] We will use the short-hand notation $@(f,a_1,\ldots,a_n)$ for the OpenMath application of the function f to the argument sequence $a_1,\ldots,a_n$ and $\beta(s,v,A)$ for the OpenMath binding with binder s (a symbols) over bound variable x and body A.

1. Even though the first definition calls $\mathcal{R}$ a triple (it is made up of three components), standard names – we call them **accessors** for the purposes of this paper – are introduced for the the components, and these are used to refer to the components – see e.g. (2) if they are not given local names. In (1) and (1) the accessors are verbal symbols, which are usually only be used in verbalizations of mathematical objects, accessors can be used in formulae as well, e.g. the accessors π_i for the i^{th} component of a tuple.
2. The set of accessors for a given structure is variable: while (1) sees a ring as a triple, (3) represents it as a 6-tuple, including the two units and inverse for the additive operation into the mix. Mathematically, this makes sense, since units in monoids and inverses in groups are uniquely determined. The representational flexibility of treating defined functions like "unit-of" at par with the definitional ones like "base-set-of" makes working with complex structures like vector space homomorphisms (which naturally have between 13 and 35 accessors) tractable.

In programming, a similar situation has led to the development of object-oriented programming, which in turn has been analyzed/formalized in terms of record structures.

2 Records to the Rescue

Record structures are basic data structures whose components are referenced by a fixed set of names. They can be formalized as tuples with (named) projections for accessors, and they can be typed by "record types" – i.e. types that are structurally records with the same accessors – in standard ways. In our example, we could represent $\mathcal{R} = (R, +, \cdot)$ as the record $\mathcal{R} = [\text{set} = R, \text{addop} = +, \text{mulop} = \cdot]$. Component selection traditionally goes via the "dot operator": e.g. $\mathcal{R}.\text{set} = R$. For OpenMath we propose that the accessors are special symbols.

Note that record structures are syntactically similar to – but not identical with – semantic annotations in OpenMath. The latter are represented by OMATP elements in the XML encoding of OpenMath. But OMATP is only allowed as the second child of an OMATTR element, i.e. as "property lists" to an OpenMath object. The situation in content MathML is similar: the annotation−xml elements (which correspond to key/value pairs) are only allowed in semantics elements which has a content MathML expression as the first child. So even though we can represent $[\text{set} = R, \text{addop} = +, \text{mulop} = \cdot]$ as in Listing 1 we cannot use it as a first-class object, in particular, we cannot just use it as in Figure 1a – here we use the boxed record expression as a gloss for the OpenMath expression in Listing 1.

Listing 1: A record as an OpenMath OMATP

```
<OMATP>
  <OMS cd="ring" name="set"><OMV name="R"/>
  <OMS cd="ring" name="addop"><OMV name="plus"/>
  <OMS cd="ring" name="mulop"><OMV name="times"/>
</OMATP>
```

Therefore we propose to lift the OMATP symbols into the rank of an OpenMath object for records, essentially making the syntax in Figure 1a legal. We also propose specific syntax for a first-class record selection operator as in Figure 1b.

Note that we formulate our extension request as a request for a language extension in the sense of [Koh14]. That calls for a

1. pragmatic syntax (which we have specified above),

```
<OMA>
  <OMS cd="relation1" name="eq"/>
  <OMV name="R"/>
  [set = R, addop = +, mulop = ·]
</OMA>
```

(a) Proposal: OMATP as an OpenMath Object.

```
<OMSEL>
  <OMV name="R"/>
  <OMS cd="ring" name="set"/>
</OMSEL>
```

(b) Proposed Record Selection

Figure 1: Proposed Syntax Extension for Records in OpenMath

2. a set of equations – we take the standard ones: $[k_1 = v_1, \ldots, k_n = v_n].k_i = v_i$ and $[k_1 = r.k_1, \ldots, k_1 = r.k_1] = r$, if $k_1, \ldots, k_n$ are the accessors of r, and
3. a content dictionary for records in OpenMath 2 syntax together with a translation of record expressions into OpenMath 2 expressions.

As there are various ways of formalizing records, we will leave the specification of the CD to future work.

3 Conclusion

We have argued for the need of a language extension to provide first-class syntax for records in OpenMath. We have identified one possible syntax reusing the existing OMATP element providing a new OMSEL element.

References

[Koh14] Michael Kohlhase. "OpenMath Language Extensions". submitted to the OpenMath 2014 Workshop. 2014.