Extending MKM Formats at the Statement Level

Fulya Horozal, Michael Kohlhase, and Florian Rabe

Computer Science, Jacobs University Bremen, Germany http://kwarc.info

Abstract. Successful representation and markup languages find a good balance between giving the user freedom of expression, enforcing the fundamental semantic invariants of the modeling framework, and allowing machine support for the underlying semantic structures. MKM formats maintain strong invariants while trying to be foundationally unconstrained, which makes the induced design problem particularly challenging.

In this situation, it is standard practice to define a minimal core language together with a scripting/macro facility for syntactic extensions that map into the core language. In practice, such extension facilities are either fully unconstrained (making invariants and machine support difficult) or limited to the object level (keeping the statement and theory levels fixed).

In this paper we develop a general methodology for extending MKM representation formats at the statement level. We show the utility (and indeed necessity) of statement-level extension by redesigning the OMDoc format into a minimal, regular core language (strict OMDoc) and an extension (pragmatic OMDoc) that maps into strict OMDoc.

1 Introduction

The development of representation languages for mathematical knowledge is one of the central concerns of the MKM community. After all, practical mathematical knowledge management consists in the manipulation of expressions in such languages. To be successful, MKM representation formats must balance multiple concerns. A format should be expressive and flexible (for depth and ease of modeling), foundationally unconstrained (for coverage), regular and minimal (for ease of implementation), and modular and web-transparent (for scalability). Finally, the format should be elegant, feel natural to mathematicians, and be easy to read and write. Needless to say that this set of requirements is overconstrained so that the design problem for MKM representation formats lies in relaxing some of the constraints to achieve a global optimum.

In languages for formalized mathematics, it is standard practice to define a minimal core language that is extended by macros, functions, or notations. For example, Isabelle [Pau94] provides a rich language of notations, abbreviations, syntax and printing translations, and a number of definitional forms. In narrative formats for mathematics, for instance, the TEX/LATEX format – arguably the most commonly used format for representing mathematical knowledge – goes a similar way, only that the core language is given by the TEX layout primitives and the translation is realized by macro expansion and is fully under user control. This extensibility led to the profusion of user-defined LATEX document classes and packages that has made TEX/LATEX so successful.

However, the fully unconstrained nature of the extensibility makes ensuring invariants and machine support very difficult, and thus this approach is not immediately applicable to content markup formats. There, MathML3 [ABC+10] is a good example of the state of the art. It specifies a core language called "strict content MathML" that is equivalent to OpenMath [BCC+04b] and "full content MathML". The first subset uses a minimal set of elements representing the meaning of a mathematical expression in a uniform, regular structure, while the second one tries to strike a pragmatic balance between verbosity and formality. The meaning of non-strict expressions is given by a fixed translation: the "strict content MathML translation" specified in section 4.6 of the MathML3 recommendation [ABC+10].

This language design has the advantage that only a small, regular sublanguage has to be given a mathematical meaning, but a larger vocabulary that is more intuitive to practitioners of the field can be used for actual representation. Moreover, semantic services like validation only need to be implemented for the strict subset and can be extended to the pragmatic language by translation. Ultimately, a representation format might even have multiple pragmatic front-ends geared towards different audiences. These are semantically interoperable by construction.

The work reported in this paper comes from an ongoing language design effort, where we want to redesign our OMDoc format [Koh06] into a minimal, regular core language (*strict OMDoc 2*) and an extension layer (*pragmatic OMDoc 2*) whose semantics is given by a "pragmatic-to-strict" ($\mathcal{P}2\mathcal{S}$) translation. While this problem is well-understood for mathematical *objects*, extension frameworks *at the statement level* seem to be restricted to the non-semantic case, e.g. the `amsthm` package for LATEX.

Languages for mathematics commonly permit a variety of pragmatic statements, e.g., implicit or case-based definitions, type definitions, theorems, or proof schemata. But representation frameworks for such languages do not include a generic mechanism that permits introducing arbitrary pragmatic statements — instead, a fixed set is built into the format. Among logical frameworks, Twelf/LF [PS99,HHP93] permits two statements: defined and undefined constants. Isabelle [Pau94] and Coq [BC04] permit much larger, but still fixed sets that include, for example, recursive case-based function definitions. Content markup formats like OMDoc permit similar fixed sets.

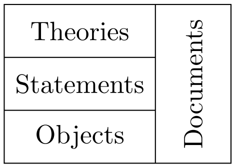
A large set of statements is desirable in a representation format in order to model the flexibility of individual languages. A large *fixed* set on the other hand is unsatisfactory because it is difficult to give a theoretical justification for fixing any specific set of statements. Moreover, it is often difficult to define the semantics of a built-in statement in a foundationally unconstrained representation format because many pragmatic statement are only meaningful under certain foundational assumptions.

In this paper we present a general formalism for adding new pragmatic statement forms to our OMDoc format; we have picked OMDoc for familiarity and foundation-independence; any other foundational format can be extended similarly. Consider for instance the pragmatic statement of an "implicit definition", which defines a mathematical object by describing it so accurately, that there is only one object that fits this description. For instance, the exponential function exp is defined as the (unique) solution of the differential equation $f = f'$ with $f(0) = 1$. This form of definition is extensively used in practical mathematics, so pragmatic OMDoc should offer an infrastructure for it, whereas strict OMDoc only offers "simple definitions" of the form $c := d$, where c is a new symbol and d any object. In our extension framework, the $\mathcal{P}2\mathcal{S}$ translation provides the semantics of the implicit definition in terms of the strict definition $exp := \iota f.(f' = f \wedge f(0) = 1)$, where ι is a "definite description operator": Given an expression A with free variable x, such that there is a unique x that makes A valid, $\iota x.A$ returns that x, otherwise $\iota x.A$ is undefined.

Note that the semantics of an implicit definition requires a definite description operator. While most areas of mathematics at least implicitly assume its existence, it should not be required in general because that would prevent the representation of systems without one. Therefore, we make these requirements explicit in a special theory that defines the new pragmatic statement and its strict semantics. This theory must be imported in order for implicit definitions to become available. Using our extension language, we can recover a large number of existing pragmatic statements as definable special cases, including many existing ones of OMDoc. Thus, when representing formal languages in OMDoc, authors have full control what pragmatic statements to permit and can define new ones in terms of existing ones.

In the next section, we will recap those parts of OMDoc that are needed in this paper. In Section 3, we define our extension language, and in Section 4, we look at particular extensions that are motivated by mathematical practice. Finally, in Section 5, we will address the question of extending the concrete syntax with pragmatic features as well.

2 MMT/OMDoc



OMDoc is a comprehensive content-based format for representing mathematical knowledge and documents. It represents mathematical knowledge at three levels: mathematical formulae at the *object level*, symbol declarations, definitions, notation definitions, axioms, theorems, and proofs at the *statement level*, and finally modular scopes at the *theory level*. Moreover, it adds an infrastructure for representing functional aspects of *mathematical documents* at the content markup level. OMDoc

1.2 has been successfully used as a representational basis in applications ranging from theorem prover interfaces, via knowledge based up to eLearning systems. To allow this diversity of applications, the format has acquired a large, interconnected set of language constructs motivated by coverage and user familiarity (i.e., by pragmatic concerns) and not by minimality and orthogonality of language primitives (strict concerns).

To reconcile these language design issues for OMDoc 2, we want to separate the format into a *strict* core language and a *pragmatic* extension layer that is elaborated into strict OMDoc via a "pragmatic-to-strict" ($\mathcal{P}2\mathcal{S}$) translation.

For strict OMDoc we employ the foundation-independent, syntactically minimal MMT framework (see below). For pragmatic OMDoc, we aim at a language that is feature-complete with respect to OMDoc 1.2 [Koh06], but incorporates language features from other MKM formats, most notably from Isabelle/Isar [Wen99], PVS [ORS92], and Mizar [TB85].

The MMT language was emerged from a complete redesign of the formal core[1] of OMDoc focusing on foundation-independence, scalability, modularity, while maintaining coverage of formal systems. The MMT language is described in [RK11] and implemented in [Rab08].

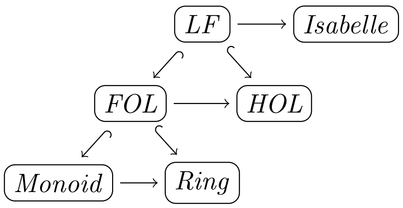


Fig. 1. An MMT Theory Graph

MMT uses **theories** as a single primitive to represent formal systems such as logical frameworks, logics, or theories. These form theory graphs such as the one on the left, where single arrows → denote theory translations and hooked arrows ↪ denote the meta-theory relation between two theories. The theory *FOL* for first-order logic is the meta-theory for *Monoid* and *Ring*. And the theory *LF* for the logical framework LF [HHP93] is the meta-theory of *FOL* and *HOL* for higher-order logic. In general, we describe the theories with meta-theory M as M**-theories**. The importance of meta-theories in MMT is that the syntax and semantics of M induces the syntax and semantics of all M-theories. For example, if the syntax and semantics are fixed for LF, they determine those of FOL and $Monoid$.

At the statement level, MMT uses **constant** declarations as a single primitive to represent all OMDoc statement declarations. These are differentiated by the type system of the respective meta-theory. In particular, the Curry-Howard correspondence is used to represent axioms and theorems as plain constants (with special types).

In Figure 2, we show a small fragment of the MMT grammar that we need in the remainder of this paper. Meta-symbols of the BNF format are given in color.

[1] We are currently working on adding an informal (natural language) representation and a non-trivial (strict) document level to MMT, their lack does not restrict the results reported in this paper.

Modules	$G ::= (\texttt{theory}\ T = \{\Sigma\})^*$
Theories	$\Sigma ::= \cdot \mid \Sigma,\, c[:E][=E] \mid \texttt{meta}\ T$
Contexts	$\Gamma ::= \cdot \mid \Gamma,\, x[:E]$
Expressions	$E ::= x \mid c \mid E\,E^+ \mid E\,\Gamma.\,E$

Fig. 2. MMT Grammar

The module level of MMT introduces *theory declarations* $\texttt{theory}\ T = \{\Sigma\}$. Theories Σ contain *constant declarations* $c[:E_1][=E_2]$ that introduce named atomic expressions c with optional type E_1 or definition E_2. Moreover, each theory may declare its meta-theory T via $\texttt{meta}\,T$.

MMT expressions are a fragment of OpenMath [BCC+04a] objects, for which we introduce a short syntax. They are formed from variables x, constants c, applications $E\,E_1\ \ldots\ E_n$ of functions E to a sequence of arguments E_i, and bindings $E_1\,\Gamma.\,E_2$ that use a binder E_1, a context Γ of bound variables, and a scope E_2. Contexts Γ consist of variables $x[:E]$ that can optionally attribute a type E.

The semantics of MMT is given in terms of *foundations* for the upper-most meta-theories. Foundations define in particular the typing relation between expressions, in which MMT is parametric. For example, the foundation for *LF* induces the type-checking relation for all theories with meta-theory *LF*.

Example 1 (MMT-Theories). Below we give an MMT theory *Propositions*, which will serve as the meta-theory of several logics introduced in this paper. It introduces all symbols needed to declare logical connectives and inference rules of a logic. The syntax and semantics of this theory are defined in terms of type theory, e.g., the logical framework LF [HHP93].

$\texttt{type}$, $\to$, and *lam* are untyped constants representing the primitives of type theory. $\texttt{type}$ represents the universe of all types, $\to$ constructs function types $\alpha \to \beta$, and *lam* represents the λ-binder. o is the type of logical formulas and *proof* is a constant that assigns to each logical formula $F : o$ the type *proof* F of its proof.

$$\begin{array}{l} \texttt{theory}\ \mathit{Propositions}\ = \{ \\ \quad \texttt{type} \\ \quad \to \\ \quad \mathit{lam} \\ \quad o \qquad : \texttt{type} \\ \quad \mathit{proof} : o \to \texttt{type} \\ \} \end{array}$$

3 A Framework for Language Extensions

We will now define our extension language (EL). It provides a syntactic means to define pragmatic language features and their semantics in terms of strict OMDoc.

Syntax EL adds two primitive declarations to MMT theories: *extension declarations* and *pragmatic declarations*:

$$\boxed{\begin{array}{rl} \varSigma ::= & \varSigma,\ \texttt{extension}\ e = \varPhi \\ | & \varSigma,\ \texttt{pragmatic}\ c : \varphi \end{array}}$$

Extension declarations `extension` $e = \varPhi$ introduce a new declaration schema e that is described by $\varPhi$. Intuitively, $\varPhi$ is a function that takes some arguments and returns a list of declarations, which define the strict semantics of the declaration scheme.

Pragmatic declarations `pragmatic` $c : \varphi$ introduce new declarations that make use of a previously declared extension. Intuitively, φ applies an extension e a sequence of arguments and evaluates to the returned list of declarations. Thus, $c : \varphi$ serves as a pragmatic abbreviation of a list of strict declarations.

The key notion in both cases is that of *theory families*. They represent collections of theories by specifying their common syntactic shape. Intuitively, theory families arise by putting a λ-calculus on top of theory fragments $\varSigma$:

$$\boxed{\begin{array}{lrcl} \text{Theory Families} & \varPhi & ::= & \{\varSigma\} \mid \lambda x : E.\varPhi \\ & \varphi & ::= & e \mid \varPhi\, E \end{array}}$$

We group theory families into two non-terminal symbols as shown above: $\varPhi$ is formed from theory fragments $\{\varSigma\}$ and λ-abstraction $\lambda x : E.\varPhi$. And φ is formed from references to previously declared extension e and applications of parametric theory families to arguments E. This has the advantage that both $\varPhi$ and φ have a very simple shape.

Example 2 (Extension Declarations). In Figure 3 we give the theory *Assertion*, which declares extensions for axiom and theorem declarations. Their semantics is defined in terms of the Curry-Howard representation of strict OMDoc.

Both extensions take a logical formula $F : o$ as a parameter. The extension *axiom* permits pragmatic declarations of the form $c : \mathit{axiom}\, F$. These abbreviate MMT constant declarations of the form $c : \mathit{proof}\ F$.

The extension *theorem* additionally takes a parameter $D : \mathit{proof}\ F$, which is a proof of F. It permits pragmatic declarations of the form $c : \mathit{theorem}\, F\, D$. These abbreviate MMT constant declarations of the form $c : \mathit{proof}\ F = D$.

```
theory Assertion = {
   meta Propositions
   extension axiom = λF : o. {
       c : proof F
   }
   extension theorem = λF : o. λD : proof F. {
       c : proof F = D
   }
}
```

Fig. 3. An MMT Theory with Extension Declarations

Any MMT theory may introduce extension declarations. However, pragmatic declarations are only legal if the extension that is used has been declared in the meta-theory:

Definition 1 (Legal Extension Declarations). *We say that an extension declaration* `extension` $e = \lambda x_1 : E_1. \ldots \lambda x_n : E_n. \{\Sigma\}$ *is* ***legal*** *in an MMT theory T, if the declarations $x_1 : E_1, \ldots, x_n : E_n$ and Σ are well-formed in T.*

This includes the case where Σ contains pragmatic declarations.

Definition 2 (Legal Pragmatic Declarations). *We say that a pragmatic declaration* `pragmatic` $c : e\, E_1 \ldots E_n$ *is* ***legal*** *in an MMT theory T if there is a declaration* `extension` $e = \lambda x_1 : E'_1. \ldots \lambda x_n : E'_n. \{\Sigma\}$ *in the meta-theory of T and each E_i has type E'_i.*

Here the typing relation is the one provided by the MMT foundation.

Semantics Extension declarations do not have a semantics as such because the extension declared in M only govern what pragmatic declarations are legal in M-theories. In particular, contrary to the constant declarations in M, a model of M does not interpret the extension declarations.

The semantics of pragmatic declarations is given by elaborating them into strict declarations:

Definition 3 (Pragmatic-to-Strict Translation $\mathcal{P}2\mathcal{S}$). *A legal pragmatic declaration* `pragmatic` $c : e\ E_1 \ldots E_n$ *is translated to a list of strict constant declarations*

$$c.d_1 : \gamma(F_1) = \gamma(D_1), \ldots, c.d_m : \gamma(F_m) = \gamma(D_m)$$

where γ substitutes every x_i with E_i and every d_j with $c.d_j$ if we have

$$\texttt{extension}\ e = \lambda x_1 : E'_1. \ldots \lambda x_n : E'_n. \{d_1 : F_1 = D_1, \ldots, d_m : F_m = D_m\}$$

and every expression E_i has type E'_i.

Example 3. Consider the following MMT theories in Figure 4: *HOL* includes the MMT theory *Propositions* and declares a constant i as the type of individuals. It adds the usual logical connectives and quantifiers – here we only present truth (*true*) and the universal quantifier ($\forall$) – and introduces equality ($\doteq$) on expressions of type α. Then it includes *Assertion*. This gives *HOL* access to the extensions *axiom* and *theorem*.

Commutativity uses *HOL* as its meta-theory and declares a constant $\circ$ that takes two individuals as arguments and returns an individual. It adds a pragmatic declaration named *comm* that declares the commutativity axiom for $\circ$ using the axiom extension from *HOL*.

Commutativity′ is obtained by elaborating *Commutativity* according to Definition 3.

```
theory HOL = {
   meta Propositions
   i    : type
   true : o
   ⋮
   ∀    : (α → o) → o
   ≐    : α → α → o
   include Assertion
}
```

```
theory Commutativity = {
   meta HOL
   ∘                 : i → i → i
   pragmatic comm : axiom ∀x : i. ∀y : i. x ∘ y ≐ y ∘ x
}

theory Commutativity′ = {
   meta HOL
   ∘        : i → i → i
   comm.c : proof ∀x : i. ∀y : i. x ∘ y ≐ y ∘ x
}
```

Fig. 4. A $\mathcal{P}2\mathcal{S}$ Translation Example

4 Representing Extension Principles

Formal mathematical developments can be classified based on whether they follow the axiomatic or the definitional method. The former is common for logics where theories declare primitive constants and axioms. The latter is common for foundations of mathematics where a fixed theory (the foundation) is extended only by defined constants and theorems. In MMT, both the logic and the foundation are represented as a meta-theory M, and the main difference is that the definitional method does not permit undefined constants in M-theories.

However, this treatment does not capture conservative extension principles: These are meta-theorems that establish that certain extensions are acceptable even if they are not definitional. We can understand them as intermediates between axiomatic and definitional extensions: They may be axiomatic but are essentially as safe as definitional ones.

To make this argument precise, we use the following definition:

Definition 4. *We call the theory family $\Phi = \lambda x_1 : E'_1. \ldots \lambda x_n : E'_n. \{\Sigma\}$* ***conservative*** *for M if for every M-theory T and all $E_1 : E'_1, \ldots, E_n : E'_n$, every model of T can be extended to a model of $T, \gamma(\Sigma)$, where γ substitutes every x_i with E_i.*

An extension declaration `extension` *$e = \Phi$ is called* ***derived*** *if all constant declarations in Σ have a definiens; otherwise, it is called* ***primitive****.*

Primitive extension declarations correspond to axiom declarations because they postulate that certain extensions of M are legal. The proof that they are indeed conservative is a meta-argument that must be carried out as a part of the proof that M is an adequate MMT representation of the represented formalism. Similarly, derived extension declarations correspond to theorem declarations because their conservativity follows from that of the primitive ones. More precisely: If all primitive extension principles in M are conservative, then so are all derived ones.

In the following, we will recover built-in extension statements of common representation formats as special cases of our extension declarations. We will follow a *little foundations* paradigm and state every extensions in the smallest theory in which it is meaningful. Using the MMT module system, this permits maximal reuse of extension definitions. Moreover, it documents the (often implicit) foundational assumptions of each extension.

Implicit Definitions in OMDoc Implicit definitions of OMDoc 1.2 are captured using the following derived extension declaration. If the theory *ImplicitDefinitions* in Figure 5 is included into a meta-theory M, then M-theories may use implicit definitions.

```
theory $\mathit{ImplicitDefinitions}$ $=$ {
    meta $\mathit{Propositions}$
    $\exists^!$   $: (\alpha \to o) \to o$
    $\iota$    $: (\alpha \to o) \to \alpha$
    $\iota_{ax}$ $: \mathit{proof}\ \exists^! x\, P\, x \to \mathit{proof}\ P\,(\iota\, P)$
    extension $\mathit{impldef}$ $= \lambda\alpha : \mathtt{type}.\, \lambda P : \alpha \to o.\, \lambda m : \mathit{proof}\ \exists^! x : \alpha.\, P\, x.$ {
        $c$   $: \alpha \ = \ \iota\, P$
        $c_{ax}$ $: \mathit{proof}\ \exists^! x : \alpha.\, P\, x$
    }
}
```

Fig. 5. An Extension for Implicit Definitions

Note that *ImplicitDefinitions* requires two other connectives: A description operator (ι) and a unique existential ($\exists^!$) are needed to express the meaning of an implicit definition. We deliberately assume only those two operators in order to maximize the re-usability of this theory: Using the MMT module system, any logic M in which these two operators are definable can import the theory *ImplicitDefinitions*.

More specifically, *ImplicitDefinitions* introduces the definite description operator as a new binding operator (ι), and describes its meaning by the axiom $\exists^! x\, P(x) \Rightarrow P\,(\iota\, P)$ formulated in ι_{ax} for any predicate P on α. The extension *impldef* permits pragmatic declarations of the form $f : \mathit{impldef}\ \alpha\, P\, m$, which defines f as the unique object which makes the property P valid. This leads to the well-defined condition that there is indeed such a unique object, which is discharged by the proof m. The pragmatic-to-strict translation from Section 3 translates the pragmatic declaration $f : \mathit{impldef}\ \alpha\, P\, m$ to the strict constant declarations $f.c : \alpha = \iota\, P$ and $f.c_{ax} : \mathit{proof}\ \exists^! x : \alpha\, P\, x$.

Mizar-Style Functor Definitions The Mizar language [TB85] provides a wide (but fixed) variety of special statements, most of which can be understood as conservative extension principles for first-order logic. A comprehensive list of the corresponding extension declarations can be found in [IKR11]. We will only consider one example in Figure 6.

$$
\begin{array}{l}
\mathtt{theory}\ \mathit{FunctorDefinitions}\ = \{\\
\quad \mathtt{meta}\ \mathit{Propositions}\\
\quad \wedge\ \ : o \to o \to o\\
\quad \Rightarrow : o \to o \to o\\
\quad \forall\ \ : (\alpha \to o) \to \alpha\\
\quad \exists\ \ : (\alpha \to o) \to \alpha\\
\quad \doteq\ \ : \alpha \to \alpha \to o\\
\quad \mathtt{extension}\ \mathit{functor}\ =\\
\qquad \lambda\alpha : \mathtt{type}.\, \lambda\beta : \mathtt{type}.\, \lambda \mathit{means} : \alpha \to \beta \to o.\\
\qquad \lambda \mathit{existence} : \mathit{proof}\ \forall x : \alpha.\, \exists y : \beta.\, \mathit{means}\, x\, y.\\
\qquad \lambda \mathit{uniqueness} : \mathit{proof}\ \forall x : \alpha.\, \forall y : \beta.\, \forall y' : \beta.\, \mathit{means}\, x\, y \wedge \mathit{means}\, x\, y' \Rightarrow y \doteq y'.\ \{\\
\qquad\quad f \qquad\qquad\qquad\qquad\quad : \alpha \to \beta\\
\qquad\quad \mathit{definitional\_theorem} : \mathit{proof}\ \forall x : \alpha.\, \mathit{means}\, x\, (f\, x)\\
\qquad \}\\
\}
\end{array}
$$

Fig. 6. An Extension for Mizar-Style Functor Definitions

The theory *FunctorDefinitions* describes Mizar-style implicit definition of a unary function symbol (called a *functor* in Mizar). This is different from the one above because it uses a primitive extension declaration that is well-known to be conservative. In Mizar, the axiom *definitional_theorem* is called the definitional theorem induced by the implicit definition. Using the extension *functor*, one can introduce pragmatic declarations of the form `pragmatic` c : *functor* $A\,B\,P\,E\,U$ that declare functors c from A to B that are defined by the property P where E and U discharge the induced proof obligations.

Flexary Extensions The above two examples become substantially more powerful if they are extended to implicit definitions of functions of arbitrary arity. This is supported by our extension language by using an LF-based logical framework with term sequences and type sequences. We omit the formal details of this framework here for simplicity and refer to [Hor12] instead. We only give one example in Figure 7 that demonstrates the potential.

$$
\begin{array}{l}
\mathtt{theory}\ \mathit{CaseBasedDefinitions}\ = \{\\
\quad \mathtt{meta}\ \mathit{Propositions}\\
\quad \wedge\ \ : o^n \to o\\
\quad \vee^!\ : o^n \to o\\
\quad \Rightarrow : o \to o \to o\\
\quad \forall\ \ : (\alpha \to o) \to o\\
\quad \mathtt{extension}\ \mathit{casedef}\ = \lambda n : \mathbb{N}.\, \lambda\alpha : \mathtt{type}.\, \lambda\beta : \mathtt{type}.\, \lambda c : (\alpha \to o)^n.\\
\qquad\qquad\qquad\qquad\qquad \lambda d : (\alpha \to \beta)^n.\, \lambda\rho : \mathit{proof}\ \forall x : \alpha.\ \vee^! \Big[c_i\, x\Big]_{i=1}^{n}.\ \{\\
\qquad f\ \ : \alpha \to \beta\\
\qquad \mathit{ax} : \mathit{proof}\ \forall x : \alpha.\ \wedge \Big[c_i\, x\ \Rightarrow\ (f\, x) = (d_i\, x)\Big]_{i=1}^{n}\\
\quad \}\\
\}
\end{array}
$$

Fig. 7. An Extension for Case-Based Definitions

The theory *CaseBasedDefinitions* introduces an extension that describes the case-based definition of a unary function f from α to β that is defined using n different cases where each case is guarded by the predicate c_i together with the respective definiens d_i. Such a definition is well-defined if for all $x \in \alpha$ exactly one out of the $c_i\,x$ is true. Note that these declarations use a special sequence constructor: for example, $\left[c_i\,x\right]_{i=1}^{n}$ simplifies to the sequence $c_1\,x\,,\ldots,c_n\,x$. Moreover, $\wedge$ and $\vee^!$ are flexary connectives, i.e., they take a flexible number of arguments. In particular, $\vee^!(F_1,\ldots,F_n)$ holds if exactly one of its arguments holds.

The pragmatic declaration **pragmatic** f : *casedef* $n\,\alpha\,\beta\,c_1\ldots c_n\,d_1\ldots d_n\,\rho$ corresponds to the following function definition:

$$f(x) = \begin{cases} d_1(x) & \text{if} \quad c_1(x) \\ \vdots & \qquad \vdots \\ d_n(x) & \text{if} \quad c_n(x) \end{cases}$$

HOL-Style Type Definitions Due to the presence of λ-abstraction and a description operator in HOL [Chu40], a lot of common extension principles become derivable in HOL, in particular, implicit definitions.

But there is one primitive definition principle that is commonly accepted in HOL-based formalizations of the definitional method: A Gordon/HOL type definition [Gor88] introduces a new type that is axiomatized to be isomorphic to a subtype of an existing type. This cannot be expressed as a derivable extension because HOL does not use subtyping.

```
theory Types = {
  meta Propositions
  $\forall$ : $(\alpha \to o) \to o$
  $\exists$ : $(\alpha \to o) \to o$
  $\doteq$ : $(\alpha \to \alpha) \to o$
  extension typedef = $\lambda\alpha : \mathtt{type}.\, \lambda A : \alpha \to o.\, \lambda P : \mathit{proof}\ \exists x : \alpha.\, A\,x.$ {
     T           : type
     Rep         : $T \to \alpha$
     Abs         : $\alpha \to T$
     Rep'        : $\mathit{proof}\ \forall x : T.\, A\,(\mathit{Rep}\,x)$
     Rep_inverse : $\mathit{proof}\ \forall x : T.\, \mathit{Abs}\,(\mathit{Rep}\,x) \doteq x$
     Abs_inverse : $\mathit{proof}\ \forall x : \alpha.\, A\,x \Rightarrow \mathit{Rep}\,(\mathit{Abs}\,x) \doteq x$
  }
}
```

Fig. 8. An Extension for HOL-Style Type Definitions

The theory *Types* in Figure 8, formalizes this extension principle. Our symbol names follow the implementation of this definition principle in Isabelle/HOL [NPW02]. Pragmatic declarations of the form **pragmatic** t : *typedef* $\alpha\,A\,P$ introduce a new non-empty type t isomorphic to the predicate A over α. Since all HOL-types must be non-empty, a proof P of the non-emptiness of A must be supplied. More precisely, it is translated to the following strict constant declarations:

- $t.T$: `type` is the new type that is being defined,
- $t.Rep : t.T \to \alpha$ is an injection from the new type $t.T$ to α,
- $t.Abs : \alpha \to t.T$ is the inverse of $t.Rep$ from α to the new type $t.T$,
- $t.Rep'$ states that the property A holds for any term of type $t.T$,
- $t.Rep\_inverse$ states that the injection of any element of type $t.T$ to α and back is equal to itself,
- $t.Abs\_inverse$ states that if an element satisfies A, then injecting it to $t.T$ and back is equal to itself.

HOL-based proof assistants implement the type definition principle as a built-in statement. They also often provide further built-in statements for other definition principles that become derivable in the presence of type definitions, e.g., a definition principle for record types. For example, in Isabelle/HOL [NPW02], HOL is formalized in the Pure logic underlying the logical framework Isabelle [Pau94]. But because the type definition principle is not expressible in Pure, it is implemented as a primitive Isabelle feature that is only active in Isabelle/HOL.

5 Syntax Extensions and Surface Languages

Our definitions from Section 3 permit pragmatic *abstract* syntax, which is elaborated into strict abstract syntax. For human-oriented representations, it is desirable to complement this with similar extensions of pragmatic *concrete* syntax. While the pragmatic-to-strict translation at the abstract syntax level is usually non-trivial and therefore not invertible, the corresponding translation at the concrete syntax level should be compositional and bidirectional.

5.1 OMDoc Concrete Syntax for EL Declarations

First we extend OMDoc with concrete syntax that exactly mirrors the abstract syntax from Section 3. The declaration `extension` $e = \lambda x_1 : E_1. \ldots \lambda x_n : E_n. \{\Sigma\}$ is written as

```
<extension name="$e$">
  <parameter name="$x_1$">$\boxed{E_1}$</parameter>
        ⋮
  <parameter name="$x_n$">$\boxed{E_n}$</parameter>
  <theory>
    $\boxed{\Sigma}$
  </theory>
</extension>
```

Here we use the box notation $\boxed{A}$ to gloss the XML representation of an entity A given in abstract syntax.

Similarly, the pragmatic declaration `pragmatic` $c : e\, E_1 \ldots E_n$ is written as

<**pragmatic** name="c" extension="$\langle\!\langle M\rangle\!\rangle?e$">
  $\boxed{E_1} \ldots \boxed{E_n}$
</**pragmatic**>

Here $\langle\!\langle M\rangle\!\rangle$ is the meta-theory in which e is declared so that $\langle\!\langle M\rangle\!\rangle?e$ is the MMT URI of the extension.

Example 4. For the implicit definitions discussed in Section 3, we use the extension *impldef* from Figure 5, which we assume has namespace URI $\langle\!\langle U\rangle\!\rangle$. If ρ is a proof of unique existence for an f such that $f' = f \wedge f(0) = 1$, then the exponential function is defined in XML by

<**pragmatic** name="exp" extension="$\langle\!\langle U\rangle\!\rangle?ImplicitDefinitions?impldef$">
  $\boxed{\lambda f.f' = f \wedge f(0) = 1}\ \boxed{\rho}$
</**pragmatic**>

5.2 Pragmatic Surface Syntax

OMDoc is mainly a machine-oriented interoperability format, which is not intended for human consumption. Therefore, the EL-isomorphic syntax introduced is sufficient in principle – at least for the formal subset of OMDoc we have discussed so far.

OMDoc is largely written in the form of "surface languages" – domain-specific languages that can be written effectively and transformed to OMDoc in an automated process. For the formal subset of OMDoc, we use a MMT-inspired superset of the Twelf/LF [PS99,HHP93] syntax, and for informal OMDoc we use sTeX [Koh08], a semantic extension of TeX/LaTeX.

For many purposes like learning the surface language or styling OMDoc documents, **pragmatic surface syntax**, i.e., a surface syntax that is closer to the notational conventions of the respective domain, has great practical advantages. It is possible to support, i.e., generate and parse, pragmatic surface syntax by using the macro/scripting framework associated with most representation formats.

For instance, we can regain the XML syntax familiar from OMDoc 1.2 via notation definitions that transform between pragmatic elements and the corresponding OMDoc 1.2 syntax. For Twelf/LF, we would extend the module system preprocessor, and for Isabelle we would extend the SML-based syntax/parsing subsystem. We have also extended sTeXas an example of a semi-formal surface language. Here we used the macro facility of TeX as the computational engine. We conjecture that most practical surface languages for MKM can be extended similarly.

These translations proceed in two step. Firstly, pragmatic surface syntax is translated into our pragmatic MMT syntax. Our language is designed to make this step trivial: in particular, it does not have to look into the parameters used in a pragmatic surface declaration. Secondly, pragmatic MMT syntax is type-checked and, if desired, translated into strict MMT syntax. All potentially difficult semantic analysis is part of this second step. This design makes it very easy for users to introduce their own pragmatic surface syntax.

6 Conclusion & Future Work

In this paper, we proposed a general statement-level extension mechanism for MKM formats powered by the notion of theory families. Starting with MMT as a core language, we are able to express most of the pragmatic language features of OMDoc 1.2 as instances of our new extension primitive. Moreover, we can recover extension principles employed in languages for formalized mathematics including the statements employed for conservative extensions in Isabelle/HOL and Mizar. We have also described a principle how to introduce corresponding pragmatic concrete syntax.

The elegance and utility of the extension language is enhanced by the modularity of the OMDoc 2 framework, whose meta-theories provide the natural place to declare extensions: the scoping rules of the MMT module system supply the justification and intended visibility of statement-level extensions. In our examples, the Isabelle/HOL and Mizar extensions come from their meta-logics, which are formalized in MMT.

We also expect our pragmatic syntax to be beneficial in system integration because it permit interchanging documents at the pragmatic MMT level. For example, we can translate implicit definitions of one system to those of another system even if – as is typical – the respective strict implementations are very different.

For full coverage of OMDoc 1.2, we still need to capture abstract data types and proofs; the difficulties in this endeavor lie not in the extension framework but in the design of suitable meta-logics that justify them. For OMDoc-style proofs, the $\overline{\lambda}\mu\tilde{\mu}$-calculus has been identified as suitable [ASC06], but remains to be encoded in MMT. For abstract data types we need a λ-calculus that can reflect signatures into (inductive) data types; the third author is currently working on this.

The fact that pragmatic extensions are declared in meta-theories points towards the idea that OMDoc metadata and the corresponding metadata ontologies [LK09] are actually meta-theories as well (albeit at a somewhat different level); we plan to work out this correspondence for OMDoc 2.

Finally, we observe that we can go even further and interpret the feature of definitions that is primitive in MMT as pragmatic extensions of an even more foundational system. Then definitions $c : E = E'$ become pragmatic notations for a declaration $c : E$ and an axiom $c = E'$, where $=$ is an extension symbol introduced in a meta-theory for equality. Typing can be handled similarly. This would also permit introducing other modifiers in declarations such as <: for subtype declarations.

References

ABC$^+$10. Ron Ausbrooks, Stephen Buswell, David Carlisle, Giorgi Chavchanidze, Stéphane Dalmas, Stan Devitt, Angel Diaz, Sam Dooley, Roger Hunter, Patrick Ion, Michael Kohlhase, Azzeddine Lazrek, Paul Libbrecht, Bruce Miller, Robert Miner, Murray Sargent, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 3.0. W3C Recommendation, World Wide Web Consortium (W3C), 2010.

ASC06. Serge Autexier and Claudio Sacerdoti Coen. A formal correspondence between omdoc with alternative proofs and the $\overline{\lambda}\mu\tilde{\mu}$-calculus. In Jon Borwein and William M. Farmer, editors, *Mathematical Knowledge Management (MKM)*, number 4108 in LNAI, pages 67–81. Springer Verlag, 2006.

BC04. Y. Bertot and P. Castéran. *Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

BCC$^+$04a. S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaetano, and M. Kohlhase. The Open Math Standard, Version 2.0. Technical report, The Open Math Society, 2004. See http://www.openmath.org/standard/om20.

BCC$^+$04b. Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaëtano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The OpenMath Society, 2004.

Chu40. A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5(1):56–68, 1940.

Gor88. M. Gordon. HOL: A Proof Generating System for Higher-Order Logic. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer-Academic Publishers, 1988.

HHP93. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

Hor12. F. Horozal. Logic translations with declaration patterns. https://svn.kwarc.info/repos/fhorozal/pubs/patterns.pdf, 2012.

IKR11. M. Iancu, M. Kohlhase, and F. Rabe. Translating the Mizar Mathematical Library into OMDoc format. Technical Report KWARC Report-01/11, Jacobs University Bremen, 2011.

Koh06. Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, August 2006.

Koh08. Michael Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2(2):279–304, 2008.

LK09. Christoph Lange and Michael Kohlhase. A mathematical approach to ontology authoring and documentation. In Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt, editors, *MKM/Calculemus Proceedings*, number 5625 in LNAI, pages 389–404. Springer Verlag, July 2009.

NPW02. T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer, 2002.

ORS92. S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Springer, 1992.

Pau94. L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

PS99. F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. *Lecture Notes in Computer Science*, 1632:202–206, 1999.

Rab08. F. Rabe. The MMT System, 2008. see `https://trac.kwarc.info/MMT/`.

RK11. F. Rabe and M. Kohlhase. A Scalable Module System. see `http://arxiv.org/abs/1105.0548`, 2011.

TB85. A. Trybulec and H. Blair. Computer Assisted Reasoning with MIZAR. In A. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 26–28, 1985.

Wen99. M. Wenzel. Isar - A Generic Interpretative Approach to Readable Formal Proof Documents. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics*, volume 1690, pages 167–184. Springer, 1999.