# Towards Knowledge Management for HOL Light

Cezary Kaliszyk and Florian Rabe

[1] University of Innsbruck, Austria
[2] Jacobs University, Bremen, Germany

**Abstract.** The libraries of deduction systems are growing constantly, so much that knowledge management concerns are becoming increasingly urgent to address. However, due to time constraints and legacy design choices, there is barely any deduction system that can keep up with the MKM state of the art. HOL Light in particular was designed as a lightweight deduction system that systematically relegates most MKM aspects to external solutions — not even the list of theorems is stored by the HOL Light kernel.
We make the first and hardest step towards knowledge management for HOL Light: We provide a representation of the HOL Light library in a standard MKM format that preserves the logical semantics and notations but is independent of the system itself. This provides an interface layer at which independent MKM applications can be developed. Moreover, we develop two such applications as examples. We employ the MMT system and its interactive web browser to view and navigate the library. And we use the MathWebSearch system to obtain a search engine for it.

## 1 Introduction and Related Work

Deduction systems such as type checkers, proof assistants, or theorem provers have initially focused on soundness and efficiency. Consequently, many follow the LCF approach which amounts to a kernel that implements the logic, a theorem prover that produces input for the kernel, and a read-eval-print loop for user interaction. But over time formalization projects like [Hal05,GAA+13] have reached larger scales that call for more sophisticated *knowledge management* (KM) support.

However, it has proved non-trivial to retrofit KM support to an existing system. Moreover, even where it is possible, developers' resources are usually stretched already by improving and maintaining the kernel, the proof assistant, and performing actual formalizations. Therefore, a long-standing goal of MKM has been to provide generic KM support for deduction systems in a parametric way. This can create a valuable separation of concerns between deduction and KM systems.

We follow up on this by combining one of each: the HOL Light [Har96] proof assistant and the Mmt KM system [Rab13].

HOL Light implements the HOL logic [Pit93], which is a variant of Church's simple type theory extended by shallow polymorphism. Polymorphism is achieved by adding arbitrary type operators (to allow the construction of compound

types), and (implicitly universally quantified) type variables. HOL Light's implementation follows the LCF tradition: the kernel defines a private ML type for each of the HOL constructs. This means that the only way a user can construct HOL objects is with the help of exported kernel functions, which only allow the building of valid types, correctly-typed terms and provable theorems. The proofs of the theorems are *ephemeral*: they are only known at theorem construction time. To this end, the HOL Light kernel maintains lists (ML references) of defined types, constants and axioms.

HOL Light does not use its own syntax for the structure of proofs and libraries, instead inheriting that of OCaml (only changing OCaml's capitalization with the help of a camlp5 module). The only parser that HOL Light defines is for the object level (inner syntax). The only library management feature is a special function used to obtain the list of all theorems from the OCaml toplevel [HZ], which is also used to offer an internal search mechanism based on term matching. This means that the HOL Light library is hard to browse and search, especially for inexperienced users.

On the other hand, this lightness makes it relatively easy to export the basic structures manipulated internally by HOL Light [Wie09], which can be combined with library information gathered by patching its internal functions and processing developments multiple times [KK13].

Mmt [RK13] is a generic type theory and knowledge management system. It combines the general syntax of OpenMath [BCC+04] and OMDoc [Koh06] with the formal semantics of logical frameworks like LF [HHP93] and Isabelle [Pau94]. The Mmt system [Rab13] implements both logical algorithms such as module system and type reconstruction as well as KM support such as IDE and change management.

Mmt systematically avoids commitment to specific type systems or logics and focuses on using open interfaces and extensible algorithms. Thus, Mmt-based implementations are maximally reusable, and Mmt is a prime candidate for providing external KM support at extremely low cost.

Following our experiences with the representation of the Mizar library in Mmt [IKRU13], we know that the export of a library in an KM-friendly interchange format is the biggest bottleneck: it is so much work to get a complete and hack-free export, that in practice none exist. Therefore a lot of interesting KM research is blocked and cannot be applied. Our export proceeds in two steps. Firstly, we define HOL Light as a theory $H$ of LF-theory, which in turn is implemented within the Mmt framework. Secondly, we implement an exporter as a part of the HOL Light system that writes the HOL Light library as a set of Mmt theories that extend $H$.

Our export includes type and constant definitions, theorems, and notations. For proofs, we use a simplified export, which stores only dependencies of a theorem but not the structure of the proof. This is reasonable because most of the structure has already been eliminated at the OCaml level anyway before the kernel is involved.

The exported theories are ready-to-use for KM systems. Apart from the fact that we skipped the proofs, they formally type-check within LF. And the OMDoc-based concrete syntax as well as the high level API of Mmt make it easy to access them.

We exemplify and exploit this infrastructure by obtaining two major KM services for HOL Light. Firstly, we use a native Mmt service: an interactive library browser based on HTML+presentation MathML. Secondly, we apply a third-party service: the MathWebSearch [KŞ06] search engine.

Our work flow and infrastructure is not only interesting per se but also serves as a first test case for future multi- and cross-library knowledge management services. We discuss applying such ideas on the KM level in Section 5.

*Related Work* Several previous exports of the HOL Light library have focused on translations to other proof assistants. All such approaches use a patched kernel, that records the applications of all kernel theorem creation steps. [OS06] and [KW10] automatically patch the sources, replacing the applications of `prove` by recording of the theorem and its name. This has been further augmented in [KK13], which obtains the list of top-level theorems from the OCaml toplevel. [OAA13] has additionally focused on exporting the proof structure of HOL Light proofs (to the extent it can be recovered), by patching tactics.

These approaches are orthogonal to ours: We focus on exporting the knowledge present in the library, keeping the proof structure transparent. In fact, this is the reason why we do not export the low-level proof structure at all (except for the calls to extension principles and the dependency information). We believe that future work can combine our export with the high level but more brittle exports à la [OAA13].

The OpenTheory project [Hur09] similarly patches the HOL Light kernel in order to export theorems with proofs in the OpenTheory format. By manually annotating the library files, users can group theorems into theories. This refactoring is crucial for reuse across systems because it permits making dependencies between theories explicit so that they can be abstracted.

Several knowledge management tools have been developed for HOL Light. [TKUG13a] export a HOL Light development into the Agora Wiki system. It heuristically HTMLizes/Wikifies the OCaml sources with anchors and references. The focus of the work is to present the informal Flyspeck text alongside with the formalization. Deconstructing compound tactics can be performed with the help of Tactician [AA12], and viewing recorded intermediate steps can be done with Proviola [TGMW10]. Online editing of HOL Light proof scripts has been added [TKUG13b]. While the browsing capability is similar to the one presented here, Agora does not look into the inner syntax of HOL Light and displays almost unmodified original OCaml files (only folding of proofs is added). The heuristic HTMLization is able to pick up a number of HOL constants, but even some of the basic ones, like conjunction, are not linked to their definitions.

None of the above systems focus on connecting proof assistants to arbitrary MKM systems. Moreover, none of them used a logical framework to formally define and thus document the HOL Light logic, instead focusing on an import that would match it with the intended target system.

## 2 Exporting the HOL Light Library

### 2.1 Defining the HOL Light Logic

Mmt itself does not force the choice of a particular logical framework: Individual frameworks are defined as Mmt plugins that provide the typing rules. We define the logic of HOL Light declaratively in the logical framework LF. LF is a dependently-typed $\lambda$-calculus, and we will use the notations `[x:A]t : {x:A}B` for $\lambda x : A.t : \Pi x : A.B$.

By using a logical framework, we can formalize and reason about the relation between different logics. This is a crucial step towards obtaining verified logic translations like in [NSM01], where the method of translation guarantees the correctness of the translated theorems. It also lets us reuse generic algorithms that depend on awareness of the logical primitives of HOL Light. These include parsing, which we will use for search queries, and type inference, which we use as an interactive feature in the browser.

HOL Light prides itself in a very small and simple to understand kernel, and its embedding in LF documents this nicely. Our definition is conceptually straightforward is mainly notable for systematically following HOL Light down to the choice of identifier names.

The three basic OCaml datatypes to represent types, terms and theorems correspond to three LF types. The OCaml type `holtype` becomes an LF type, and LF variables of this type naturally represent HOL type variables. Initially two primitive type constructors are present – `bool` and `fun`:

```
holtype : type
bool : holtype
fun  : holtype → holtype → holtype                    "1 ⇒ 2"
```

Here, we also use Mmt to define appropriate notations for all operators, e.g., `1 ⇒ 2` introduces the usual infix notation.

Next the type of HOL terms of a given type together with application and abstraction operators (denoted `'` and $\lambda$ respectively) are introduced. LF variables of the term type represent bound variables, or implicitly universally quantified free variables. The only primitive constant of HOL Light (equality) is also specified here:

```
term  : holtype → type
Abs   : {A,B} (term A → term B) → term (A ⇒ B)          "λ 3"
Comb  : {A,B} term (A ⇒ B) → term A → term B          "3 ' 4"
equal : {A} term A ⇒ (A ⇒ bool)                       "2 = 3"
```

Note that the MMT notation language does not distinguish between LF and HOL arguments. For example, in the notation for application, `3` and `4` refer to the two HOL arguments; the two LF arguments `A` and `B` in positions `1` and `2` are defined to be implicit by not mentioning them. This is a novel feature that we get back to in Sect. 3.1.

The shallow polymorphism of HOL LIGHT corresponds exactly to the LF function space, which we can see in the declaration of equality: It takes one LF argument `A` for the HOL type and two HOL arguments of type `A`.

HOL LIGHT theorems are sequents $F_1, \ldots, F_n \vdash F$ using a type variable context $a_1, \ldots, a_n$. In LF, we use the judgments-as-types principle and represent the assumptions as an LF context. The above sequent corresponds to the type $\{a_1, \ldots, a_n\} \vdash F_1 \to \ldots \to \vdash F_n \to \vdash F$, and the valid proofs are exactly the terms of that type. The complete set of theorem construction rules is represented as:

```
thm   : term bool → type                                        "⊢ 1"
REFL  : {A,X:term A} ⊢ X = X
TRANS : {A,X,Y,Z:term A} ⊢ X = Y → ⊢ Y = Z → ⊢ X = Z
MP    : p,q ⊢ p = q → ⊢ p → ⊢ q
BETA  : {A,B,F:term A → term B,X:term A} ⊢ (λ F)'X = (F X)
MK_COMB : {A,B, F,G:term A⇒B, X,Y:term A}
  ⊢ F = G → ⊢ X = Y → ⊢ F'X = G'Y
ABS   : {A,B, S,T:term A → term B}
  ({x: term A} ⊢ (S x) = (T x)) → ⊢ λ S = λ T
DEDUCT_ANTISYM_RULE : {p,q} (⊢ p → ⊢ q) → (⊢ q → ⊢ p) → ⊢ p = q
```

The above seven rules are the only ones needed for HOL LIGHT theorem construction steps. The three remaining ones (`INST_TYPE`, `INST` and `ASSUME`) are naturally represented by the contexts and substitutions of LF.

In [HKR12], we introduced the MMT feature of extension declarations. Moreover, in [HRK14], we introduced LFS, a variant of LF with arity polymorphism. If we apply extension declarations to LFS, we can also formalize the two extension principles of HOL LIGHT– definitions and type definitions:

```
extension definition =
   [n: nat] [A: holtype^n → holtype] [a: {T: holtype^n} term (A T)]
      c   : {T} term (A T)
      DEF : {T} ⊢ (c T) = (a T)

extension new_basic_type_definition =
   [n:nat] [A: holtype^n → holtype]
   [P: {T: holtype^n} term (A T) ⇒ bool]
   [w: {T: holtype^n} term (A T)]
   [nonempty: {T: holtype^n} ⊢ (P T) ' (w T)]
      B       : holtype^n → holtype
      abs     : {T} term (B T) ⇒ (A T)
      rep     : {T} term (A T) ⇒ (B T)
```

```
rep_abs : {T} {b: term (B T)} ⊢ (rep T) ' ((abs T) ' b) = b
abs_rep : {T} {a: term (A T)} ⊢
                  (P T) ' a = ((abs T) ' ((rep T) ' a) = a)
```

Here `nat` is the type of natural numbers. Arity polymorphism means that declarations may take a `nat` argument and use it to form different types. In our case, we use $\texttt{holtype}^{\texttt{n}}$ to represent (essentially) `n`-tuples of `holtype`.

The `extension` keyword introduces declarations of named extension principles. Such a declaration $\lambda$-binds some arguments and then returns a list of declarations. The returned list is displayed using indentation.

The extension principle named `definition` takes an `n`-ary type constructor `A` and a polymorphic term `a` of that type. It returns a new constant `c` with an axiom `DEF` that makes `c` equal to `a`. The extension principle named `new_basic_type_definition` takes an `n`-ary type constructor `A` and a polymorphic unary predicate `P` on it. It returns a new `n`-ary type constructor `B` that is axiomatized to be isomorphic to the subtype of `A` defined by `P`. It also takes a polymorphic term `w` and a proof `nonempty` that `w` witnesses the non-emptiness of the new type.

### 2.2 Exporting the HOL Light Library

In order to export the HOL Light library to Mmt we need the information about the defined types, constants, theorems, and notations. The defined types and constants are stored by the kernel and the complete list can be accessed in any HOL Light state together with the arity of types and the most general types of constants. The only mechanism for obtaining the defined theorems in HOL Light is the `update_database` mechanism [HZ]. It accesses the internal OCaml data structures to extract name-value pairs for all defined values of the type `thm` in an arbitrary OCaml state.

Typically in order to export a library the above functionalities are invoked once [KK13], at the end of a development giving a complete list of values to export. A second processing of the developments is necessary to additionally discover the order in which the theorems are defined with the help of dependencies. This has been described in detail in [KU14] together with the splitting of theorems that are large conjunctions into separate named conjuncts.

Here we additionally preserve the separation of concepts into files. This means that we need to patch the HOL Light theory loader to store a stack of names of loaded files and call the database functions at each file entry and exit. With this additional information, the second processing of the development can recover the division of concepts between files and properly order the types, constants and theorems in the files creating an ordered export list.

For each HOL Light file in the export list, we produce an OMDoc [Koh06] file containing a single `theory`. This theory declares one `constant` for each exported theorem and several for each application of an extension principle. For every defined $n$-ary type constructor, an LF-constant of type $\texttt{holtype} \to \ldots \to \texttt{holtype} \to \texttt{holtype}$ is produced. For each HOL Light constant of type `A`, an LF

constant of type `term A` is produced. Similarly for each theorem asserting `F` (this includes axioms and the axioms generated by definitions and type definitions), a constant of type `thm F` is exported. In case of polymorphism, all free type variables `a` are universally quantified using {`a:holtype`}.

```
<constant name="PRE"><type>
  <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath"><om:OMA>
    <om:OMS module="LF" name="apply"></om:OMS>
    <om:OMS module="Kernel" name="term"></om:OMS>
    <om:OMA>
      <om:OMS module="LF" name="apply"></om:OMS>
      <om:OMS module="Kernel" name="fun"></om:OMS>
      <om:OMS module="nums" name="num"></om:OMS>
      <om:OMS module="nums" name="num"></om:OMS>
    </om:OMA>
  </om:OMA></om:OMOBJ>
</type></constant>
```

**Fig. 1.** OMDoc encoding of the HOL Light predecessor constant `PRE`

The writing of the particular types and terms is a straightforward transformation of the type and term structure of HOL into its corresponding Mmt/LF tree, and we show only one example. Fig. 1 shows the export of the application of the `definition` extension principle to define the constant `PRE`.

For the defined types and constants, for which HOL Light provides notations, we additionally include the information about the fixity, precedence, and delimiter symbol. For the constants, which the HOL Light parser and printer treat in a special way (like `If` and `UNIV`), mixfix notations are exported. Details are described in Section 3.1.

Proofs in our encoding of the HOL Light logic are Mmt definitions. However, we do not export the complete proof object and use the definition to store only dependency information. Then the definition is the application of a special constant (corresponding to an informal "by" operator) to all the theorems it depends on. This allows for mapping the HOL Light dependencies to Mmt dependencies directly.

The results of some extension principles are not registered at the toplevel. Many definitions are internally introduced using the Hilbert operator $\varepsilon$ in the definiens. Only the derived definitions — without $\varepsilon$ — are registered at the toplevel. Complete dependency information is available for all theorems except for such derived definitions.

The complete OMDoc created for the core library of HOL Light repository version 182 consists of 22 files with the total of 2801 Mmt constants. The constants correspond to 254 applications of extension principles and 2514 toplevel conjuncts. 95 notations are exported. The OMDoc encoding takes 75.2MB. On an IntelM 2.66 GHz CPU, it takes 6 minutes to prepare the export lists (this is

mostly computation of dependencies) and 17 seconds to export the library. The exported library is hosted at http://gl.mathhub.info/HOLLight/basic as a part of the Open Archive of Formalizations.

## 3 A Library Browser for HOL Light

### 3.1 Notations

HOL Light has no formal notation language that users could use to declare notations for new functions and type operators. But it does maintain three tables (for infix, unary prefix, and binding operators) that parser and printer use to convert between external and internal syntax. Our export represents these in terms of Mmt's general notation language. Moreover, HOL Light uses ad hoc notations for a few library symbols, most of which we can also map to Mmt notation declarations.

Below we describe a few improvements we have made to Mmt in order to better mimic and improve the notations of the HOL Light library.

*Nested Higher-Order Abstract Syntax* A major drawback of using a logical framework is the associated encoding overhead due to the higher-order abstract syntax (HOAS). This overhead does not necessarily cause performance issues, but it is much more work for theory and implementation, which is why current exports usually do not use it. This is particularly apparent when representing languages based on simple type theory like HOL Light, which already use HOAS themselves.

For example, the export of the universal quantifier yields the following LF declaration:

```
! : {A} term ((A ⇒ bool) ⇒ bool)
```

Thus, we export the formula $!x.(p\,x)$ as `! Abs [x:term A] Comb p x`, i.e., with two nested levels of binding (the LF-$\lambda$ `[]` and the HOL-$\lambda$ `Abs`), two nested levels of application (the LF-application whitespace and the HOL-application `Comb`), and two nested levels of type attributions (the LF-type attribution `:` and the HOL-type attribution `term`). Moreover, `Abs` and `Comb` take two LF-arguments for the involved HOL types, which we have not even shown here.

We now capture the complete absence of HOAS in human-facing syntax by ignoring HOAS in the Mmt notation language. For example, we export the notation of `forall` as `! V2 . 3` where `V2` represents the second argument (a bound variable), and `3` the third argument (the scope of the binding). (The unmentioned first argument is the type and remains implicit.) The fact that `!` is a unary function for LF and a (different) unary function for HOL Light is ignored and relegated to the type system.

We have added a generic component to Mmt that uses these no-HOAS notations and systematically converts expressions between the three nesting levels of no, single, and double HOAS.

*Unicode Notations* HOL Light notations use keyboard-friendly multi-character delimiters such as /\ and ==> instead of $\wedge$ and $\Longrightarrow$. We export all notations using the corresponding Unicode character, which is far superior for browsing.

However, to retain the ability to parse HOL Light expressions with the Mmt parser, we have added an optional lexing rule in Mmt that converts multi- into single-character delimiters.

*2-Dimensional Notations* It is desirable to use MathML's 2-dimensional display (e.g., subscripts and fractions) for browsing. Therefore, we have extended Mmt with a feature that permits declaring an optional second notation for every constant. If present, it is used for presentation only.

These notations amount to a text syntax for a small subset of MathML. Specifically, they are of the form

$$N ::= (arg \mid var \mid delim \mid implarg)^* \mid N \circ N \mid (N) \qquad \circ ::= \_ \mid \_\_ \mid \hat{} \mid \hat{}\hat{} \mid /$$

1-dimensional notations $N$ consist of a sequence of *arg*ument positions (e.g., `3` above), *var*iable positions (e.g, `V2` above), and *delim*iters (e.g., `!` and `.` above). 2-dimensional notations may now additionally use subscript _, superscript ^, underscript __ and overscript ^^ as well as fraction /.

Moreover, we use *implarg* to explicitly position implicit arguments, which can be switched on or off interactively in the browser. For example, for the same-cardinality operator `=_c` that relates two sets of type `A ⇒ bool`, we use the notation `2 = _ c ^ I1 3`, which results in the display $s =_c^A t$ (with $A$ being initially hidden).

Where possible, Mmt constructs 2-dimensional notations automatically (e.g., HOL Light's division `a/b` is rendered as a fraction). But in most cases, they have to be added manually because they are not a part of HOL Light. Our export provides nine such 2-dimentional notations for division, power (including powersets) and cardinality compararison operations. An example rendering is shown in Fig. 2.

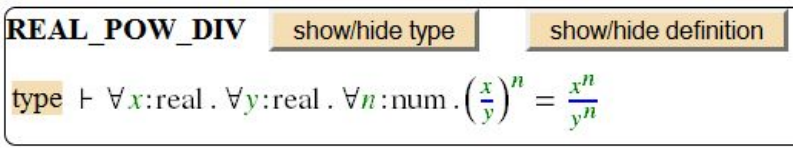


**Fig. 2.** Rendering Using 2-Dimensional Notations

### 3.2 Interactive HTML

Mmt can produce static HTML pages, as well as serve these interactively. Fig. 3 shows an overview of the appearance of the HOL Light library in the interactive

web browser. A navigation bar on the left shows the files, and the main area shows the rendering of the respective declarations.

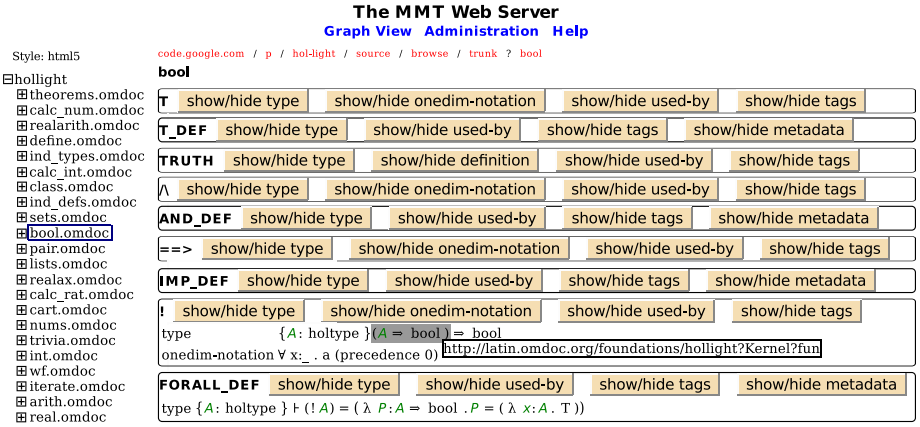


**Fig. 3.** The HOL Light Theory Browser

All formulas are presented by converting double-HOAS content MathML into no-HOAS presentation MathML using Mmt's notation-based presentation engine. `mrow` elements are used to mark up the content structure and JavaScript to ensure that only full subterms can be selected. Every presentation element carries special attributes that identify which subterm it presents. This can be seen as a form of parallel markup between the presentation in the client and the content on the server.

For example, Fig. 4 uses this to infer the type of a subterm. Here, the parallel markup identifies the selected subterm in the double-HOAS representation so that the generic type inference of Mmt can be used. After type inference, the result is rendered as no-HOAS presentation again.

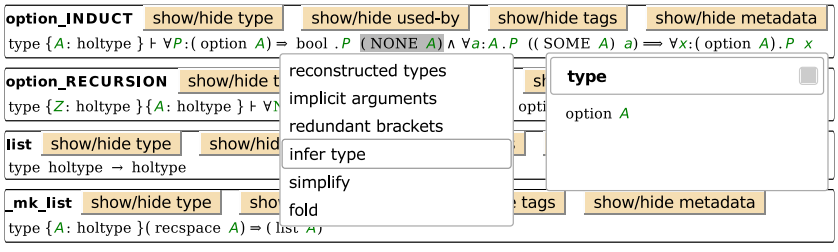


**Fig. 4.** Type Inference

Fig. 5 shows an example of the generic MMT parser in action. The exported notations are used to mimic HOL LIGHT's concrete input syntax and the result is rendered as presentation MathML. This is a very useful feature to integrate into a theory browser because it permits writing example expressions to better understand the browsed theory.

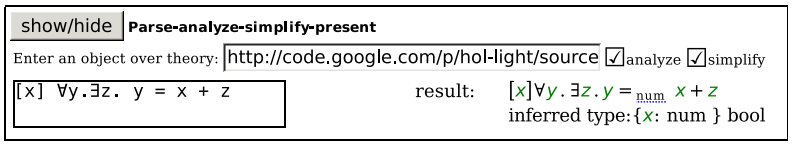


**Fig. 5.** Parser

We also added a dependency-based navigation interface that is very convenient when working with a large library. Incoming dependencies (which theorems were used) are already explicit in our export and are shown as the definition of a theorem. Outgoing dependencies (where is this theorem used) are computed by MMT and added explicitly into the HTML. Both carry cross-references for navigation. This can be seen in Fig. 6.

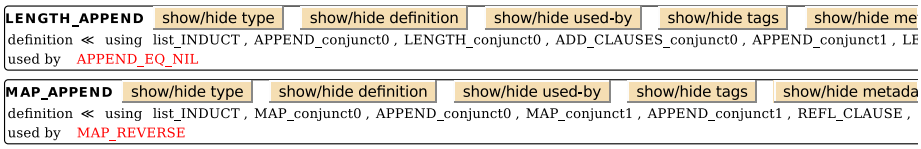


**Fig. 6.** Outgoing and Incoming Dependencies

## 4 Searching the HOL Light Library

MMT already includes a build tool for easily exporting content in different formats. The latter include building a term index that can be read by the MathWebSearch system [KŞ06], which then constructs a substitution tree index of all terms in the library.

We have already used this for Mizar in [IKRU13], and we have now substantially expanded MMT with the ability to also query MathWebSearch. This is harder than it sounds because it requires constructing queries by parsing user input and infer all implicit arguments (or replace them with additional query variables). We have also added a corresponding frontend to the library browser.

Our new MMT search interface also allows queries based on other aspects than substitution, in particular we can use regular expression queries on identifiers. This is a seemingly minor feature that goes a long way in large libraries where users typically develop sophisticated naming schemes for theorems. For example the Coq-developments in [GAA+13] append letters to theorem names as a kind of tagging mechanism.

Fig. 7 shows an example query. We search for expressions `x MOD p = y MOD p` for arbitrary `x,y,p`, and we filter the results to be from a theory whose name contains `arith`.

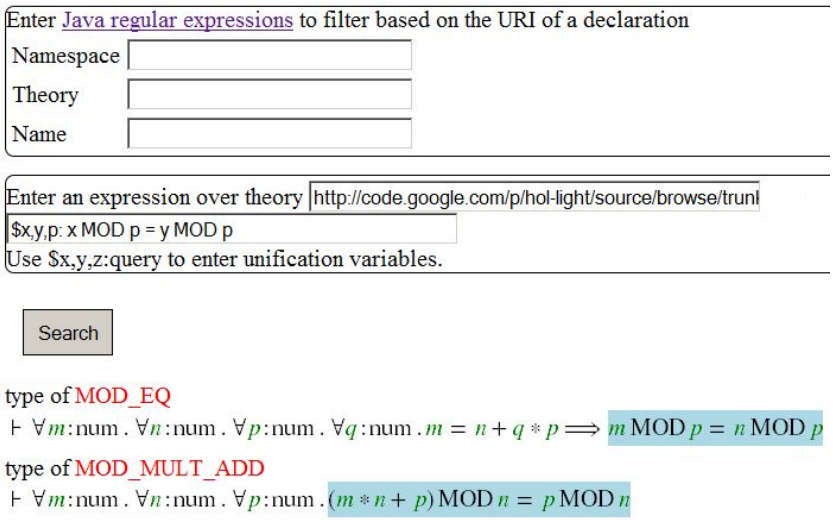


**Fig. 7.** Search

It is straightforward to add other search criteria, such as filtering to theorems whose proof uses a certain theorem, or only returning results that match on toplevel.

## 5 Conclusion and Future Work

We have demonstrated that generic MKM services – in our case: browsing and search – can be utilized for libraries of proof assistants. However, this is contingent upon a good export of the library, which requires a substantial investment.

Each export requires two parts: A modification of the proof assistant that enables exporting the high-level structure needed by the MKM services; and a representation of the logic and library in an MMT-like interchange language. Our work constitutes the second step and was enabled by previous work providing

the first step. Even though HOL Light is one the simplest proof assistants in this respect, the latter goes all the way back to [OS06] and has only become available recently.

Therefore, we believe a major value of our work is in describing the data flow and the architecture necessary to connect deduction and knowledge management systems, and to kick off future work that applies the same approach to other systems.

*An Open Archive of Formalizations* In a collaboration with Michael Kohlhase, the second author has initiated a project of collecting the libraries of proof assistants using Mmt as the common representation language. After representing Mizar [IKRU13], the present work is the second library present in the OAF. We will continue this effort and develop it into a central hub for publishing and sharing libraries.

*MathWiki* This serves as a next step towards a universal mathematical library that integrates the libraries of various proof assistants with each other and with the semi-formal proofs done by mathematicians initiated in [CK07]. The idea of wiki-style editing provided both for formal and informal text allows to identify overlaps and transport ideas. It can also be extended with human-annotated or machine-generated correspondences between concepts in formal libraries [GK14] in order to navigate across libraries. Semantic annotations, both human- and machine-generated, enhance the search capabilities, which can be further strengthened by learning-assisted automated reasoning techniques [KU14].

*Library Refactoring* The axiomatic method encapsulates definitions and theorems in the smallest theory in which they make sense. This abstraction from the prerequisites has the advantage that results can be moved easily between theories and logics, thus maximizing reuse. This is a cornerstone of the mathematical method, and it has motivated formalized mathematics early on. For example, it is at the center of IMPS [FGT93] and motivated Reynolds' seminal work [Rey83].

Yet, HOL Light and most other systems with large libraries use the definitional method that conservatively extends a single theory. We expect a rising interest in refactoring libraries according to the axiomatic method. For HOL Light, we already see the beginnings in the systematic manual refactoring of the OpenTheory project [Hur09], which can inspire and evaluate future automated refactoring methods.

As a first step for example, we can introduce a new theory for every type definition and turn all theorems derived from the extension principle into axioms. Currently this is tricky for HOL Light though because we can only tell if a theorem depends on *some* extension principle but not on which one. A clustering analysis of all theorems can provide further pointers to automated refactoring. Similarity analysis across libraries may also prove helpful to spot overlap, which is typically worthy of being refactored into a separate theory.

More generally, future work will develop logic-independent refactoring tools that can be applied generically to large definitional libraries. Exports like ours that represent libraries in universal formats while preserving the structure and semantics are the crucial prerequisite to apply such refactoring tools.

*Library Integration* The definitional method is also the reason why the integration of libraries across logics has so far been extremely difficult. Moving a definition $d$ from one library to another is not desirable if (as is typical) the target library already uses a different definition $d'$ for the same concept $c$. In those cases, we can abstract from the definitions and translate between libraries via partial theory morphisms.

Here the partiality of a morphism stems from dropping $d$, which permits translating $c$ to its counterpart. This requires a dependency analysis because dropping $d$ invalidates all results that depended on $d$. These must be dropped as well or reestablished in the target system.

Theoretically, this integration approach has been suggested in [RKS11]. Independently, its feasibility has been demonstrated in [KK13], which amounts to giving ad hoc partial morphisms that translate the HOL Light library into the Isabelle/HOL library.

Future work can systematically write and verify these morphisms in Mmt. This will yield verified integration functions even without reproving the translated theorems in the target system.

## Acknowledgments

This work has been partially supported by the Austrian Science Fund (FWF): P26201.

## References

AA12. M. Adams and D. Aspinall. Recording and refactoring HOL Light tactic proofs. In *Proceedings of the IJCAR workshop on Automated Theory Exploration*, 2012. Available at http://homepages.inf.ed.ac.uk/smaill/atxwing/atx2012_submission_9.pdf.

BCC+04. 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.

CK07. P. Corbineau and C. Kaliszyk. Cooperative repositories for formal proofs. In M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, editors, *Proc. of the 6th International Conference on Mathematical Knowledge Management (MKM'07)*, volume 4573 of *LNCS*, pages 221–234. Springer Verlag, 2007.

FGT93. W. Farmer, J. Guttman, and F. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11(2):213–248, 1993.

GAA+13. G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. L. Roux, A. Mahboubi, R. O'Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A Machine-Checked Proof of the Odd Order Theorem. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving*, pages 163–179, 2013.

GK14. T. Gauthier and C. Kaliszyk. Matching concepts across hol libraries. In S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors, *Proc. of the 7th Conference on Intelligent Computer Mathematics (CICM'14)*, LNCS. Springer Verlag, 2014. to appear.

Hal05. T. Hales. Introduction to the Flyspeck Project. In T. Coquand, H. Lombardi, and M. Roy, editors, *Mathematics, Algorithms, Proofs*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

Har96. J. Harrison. HOL Light: A Tutorial Introduction. In *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 265–269. Springer, 1996.

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.

HKR12. F. Horozal, M. Kohlhase, and F. Rabe. Extending MKM Formats at the Statement Level. In J. Campbell, J. Carette, G. Dos Reis, J. Jeuring, P. Sojka, V. Sorge, and M. Wenzel, editors, *Intelligent Computer Mathematics*, pages 64–79. Springer, 2012.

HRK14. F. Horozal, F. Rabe, and M. Kohlhase. Flexary Operators for Formalized Mathematics. In S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics*. Springer, 2014. to appear.

Hur09. J. Hurd. OpenTheory: Package Management for Higher Order Logic Theories. In G. D. Reis and L. Théry, editors, *Programming Languages for Mechanized Mathematics Systems*, pages 31–37. ACM, 2009.

HZ. J. Harrison and R. Zumkeller. update_database module. Part of the HOL Light distribution.

IKRU13. M. Iancu, M. Kohlhase, F. Rabe, and J. Urban. The Mizar Mathematical Library in OMDoc: Translation and Applications. *Journal of Automated Reasoning*, 50(2):191–202, 2013.

KK13. C. Kaliszyk and A. Krauss. Scalable LCF-style proof translation. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of *LNCS*, pages 51–66. Springer Verlag, 2013.

Koh06. M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer, 2006.

KŞ06. M. Kohlhase and I. Şucan. A Search Engine for Mathematical Formulae. In T. Ida, J. Calmet, and D. Wang, editors, *Artificial Intelligence and Symbolic Computation*, pages 241–253. Springer, 2006.

KU14. C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, 2014. http://dx.doi.org/10.1007/s10817-014-9303-3.

KW10. C. Keller and B. Werner. Importing HOL Light into Coq. In M. Kaufmann and L. Paulson, editors, *Interactive Theorem Proving*, pages 307–322. Springer, 2010.

NSM01. P. Naumov, M. Stehr, and J. Meseguer. The HOL/NuPRL proof translator - a practical approach to formal interoperability. In *14th International Conference on Theorem Proving in Higher Order Logics*. Springer, 2001.

OAA13. S. Obua, M. Adams, and D. Aspinall. Capturing hiproofs in hol light. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *MKM/Calculemus/DML*, volume 7961 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2013.

OS06. S. Obua and S. Skalberg. Importing HOL into Isabelle/HOL. In N. Shankar and U. Furbach, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.

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

Pit93. A. Pitts. The HOL logic. In M. J. C. Gordon and T. F. Melham, editors, *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

Rab13. F. Rabe. The MMT API: A Generic MKM System. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Intelligent Computer Mathematics*, pages 339–343. Springer, 2013.

Rey83. J. Reynolds. Types, Abstraction, and Parametric Polymorphism. In *Information Processing*, pages 513–523. North-Holland, Amsterdam, 1983.

RK13. F. Rabe and M. Kohlhase. A Scalable Module System. *Information and Computation*, 230(1):1–54, 2013.

RKS11. F. Rabe, M. Kohlhase, and C. Sacerdoti Coen. A Foundational View on Integration Problems. In J. Davenport, W. Farmer, F. Rabe, and J. Urban, editors, *Intelligent Computer Mathematics*, pages 107–122. Springer, 2011.

TGMW10. C. Tankink, H. Geuvers, J. McKinna, and F. Wiedijk. Proviola: A tool for proof re-animation. In S. Autexier, J. Calmet, D. Delahaye, P. D. F. Ion, L. Rideau, R. Rioboo, and A. P. Sexton, editors, *AISC/MKM/Calculemus*, volume 6167 of *Lecture Notes in Computer Science*, pages 440–454. Springer, 2010.

TKUG13a. C. Tankink, C. Kaliszyk, J. Urban, and H. Geuvers. Communicating formal proofs: The case of Flyspeck. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of *LNCS*, pages 451–456. Springer Verlag, 2013.

TKUG13b. C. Tankink, C. Kaliszyk, J. Urban, and H. Geuvers. Formal mathematics on display: A wiki for Flyspeck. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Proc. of the 6th Conference on Intelligent Computer Mathematics (CICM'13)*, volume 7961 of *LNCS*, pages 152–167. Springer Verlag, 2013.

Wie09. F. Wiedijk. Stateless HOL. In T. Hirschowitz, editor, *TYPES*, volume 53 of *EPTCS*, pages 47–61, 2009.