Making PVS Accessible to Generic Services by Interpretation in a Universal Format

Michael Kohlhase[1], Dennis Müller[1], Sam Owre[2], Florian Rabe[3]

[1] Computer Science, FAU Erlangen-Nürnberg
[2] SRI Palo Alto
[3] Computer Science, Jacobs University Bremen

Abstract. PVS is one of the most powerful proof assistant systems, and its libraries of formalized mathematics are among the most comprehensive albeit underappreciated ones. A characteristic feature of PVS is the use of a very rich mathematical and logical foundation, including e.g., record types and undecidable subtyping, and a deep integration of decision procedures. That makes it particularly difficult to develop integrations of PVS with other tools such as translations from PVS to other reasoning tools or library management periphery.
This paper presents a translation of PVS and its libraries to the OMDoc/MMT framework that preserves the logical semantics and notations but is independent of the system itself. OMDoc and MMT are a framework for formal knowledge that abstracts from logical foundations and concrete syntax to provide a universal representation format for formal libraries and interface layer for machine support. That allows instantiating generic tool support for the PVS library and relating it to libraries of other systems.

1 Introduction

Motivation One of the most critical bottlenecks in the field of interactive theorem proving is the lack of interoperability between proof assistants and related tools. This leads to a duplication of efforts: both formalizations and auxiliary tool support (e.g., for automated proving, library management user interfaces) cannot be easily shared in other systems.

In both areas, previous work has shown significant potential for knowledge sharing. Regarding formalizations, library translations such as [KW10; OS06; KS10] have been used to transport theorems across systems, and alignments have been used to match corresponding declarations in different libraries [GK14]. Regarding tool support, Isabelle's sledgehammer component [MP08] provides a generic way to integrate different automation tools. Dedukti [BCH12] has been used as an independent proof checker for various proof assistant libraries. Premise selection has emerged as an important generic service, e.g., based on machine-learning [KU15]: a single tool can be used to predict axioms for any proof assistant – provided the language and library are available in a simple universal format such that it can be plugged into the selection algorithms.

Unfortunately, the latter point – the universal format – is often prohibitively expensive for many interesting applications. Firstly, it is extremely difficult to design a universal format that strikes a good trade-off between simplicity and universality. And secondly, even in the presence of such a format, it is difficult to implement the export of a library into the universal format. Here it is important to realize that any export attempt is doomed that uses a custom parser or type checker for the library – only the internal data structures maintained by the proof assistant are informative enough for most use cases. Consequently, only expert developers can perform this step, of which each proof assistant community only has very few.

In previous work, the authors have developed such a universal format [Koh06; RK13; KR16] for formal knowledge: OMDoc is an XML language geared towards making formula structure and context dependencies explicit while remaining independent of the underlying logical formalism. We also built a strong implementation – the MMT system – and a number of generic services, e.g., [Rab14a; KŞ06]. We have already successfully applied our approach to Mizar in [Ian+13] and HOL Light in [KR14]. In both cases, we systematically (i) manually defined the logic of the proof assistant in a logical framework, and (ii) instrumented the proof assistant to export its libraries. The OMDoc/MMT language provides the semantics that ties together the three involved levels (logical framework, logic, and library) and the implementation provides a uniform high-level API for further processing. Critically, the exports systematically avoid any (deep) encoding of logical features. That is important so that further processing can work with the exact same structure apparent to a user of the proof assistant.

Contribution We apply our approach to PVS [ORS92]: we present a definition of the PVS logic in OMDoc/MMT and an export feature for PVS libraries. We exemplify the latter by exporting the Nasa Library [Lan16b], the largest and most important library of PVS.

Finally, we present several applications that instantiate MMT-level services for PVS libraries. Notably, even though the export itself is our main contribution, these applications immediately yield added-value for PVS users. Firstly, we instantiate the generic library management facilities for browsing both the content and theory graphs of PVS libraries. Our most advanced application instantiates MathWebSearch [KŞ06], a substitution-tree–based search engine, for PVS libraries. Here users enter search queries and see search results inside PVS, and MathWebSearch performs the actual search; neither tool is aware of the respective other, and MMT provides the high-level interface that allows semantics-aware mediation between these tools.

Related Work The Logosphere project [Pfe+03] already aimed at a similar export from PVS. Both the definition of the PVS logic and the export of the library turned out to be too difficult at the time.

Independent of our work, Frederic Gilbert is pursuing a very similar export of PVS into Dedukti [BCH12] that appears to be unpublished as of this writing. Its primary interest is the independent verification of PVS libraries. An interesting difference to our approach is that Dedukti is a fixed, simple logical framework that requires a non-trivial (deep) encoding of some advanced PVS features (e.g., predicate subtyping); as we discuss in Sect. 3, our approach uses a more complex, adaptive logical framework that allows for translations of the PVS library without such encodings.

Hypatheon [Lan16a] provides a capability for indexing PVS theories and making them searchable via a GUI client. It renders proof-side assistance by finding suitable lemmas within PVS libraries. It also can retrieve other declarations and allows to view the full theories that contain them.

Overview We briefly recap PVS in Section 2. Then we describe the definition of the PVS logic in our framework in 3 and of the PVS library in Section 4. Building on this, we present our applications in Section 5. Section 6 concludes the paper.

2 Preliminaries

PVS [ORS92] is a verification system, combining language expressiveness with automated tools. The language is based on higher-order logic, and is strongly typed. The language includes types and terms such as: numbers, records, tuples, functions, quantifiers, and recursive definitions. Full predicate subtypes are supported, which makes type checking undecidable; PVS generates type obligations (TCCs) as artefacts of type checking. For example, division is defined such that the second argument is nonzero, where nonzero is defined:

```
nonzero_real: TYPE = {r: real | r /= 0}
```

Note that functions in PVS are total; partiality is only supported via subtyping. Dependent types for records, tuples, and function types is also supported.

Beyond this, the PVS language has structural subtypes (i.e., a record that adds new fields to a given record), dependent types for record, tuple, and functions, recursive and co-recursive datatypes, inductive and co-inductive definitions, theory interpretations, and theories as parameters, conversions, and judgements that provide control over the generation of proof obligations. Specifications are given as collections of parameterized theories, which consist of declarations and formulas, and are organized by means of imports.

The PVS prover is interactive, but with a large amount of automation built in. It is closely integrated with the type checker, and features a combination of decision procedures, BDDs, automatic simplification, rewriting, and induction. There are also rules for ground evaluation, random test case generation, model checking, and predicate abstraction. The prover may be extended with user-defined proof strategies.

PVS has been used as a platform for integration. It has a rich API, making it relatively easy to add new proof rules and integrate with other systems. Examples of this include the model checker, Duration Calculus, MONA, Maple, Ag, and Yices. The system is normally used through a customized Emacs interface, though it is possible to run it standalone (PVSio does this), and there is a preliminary XML-RPC server that will allow for more flexible interactions. PVS is open source, and is available at `http://pvs.csl.sri.com`.

As a running example, the PVS theory defining *equivalence closures* in its original syntax is given in Figure 1.

```
EquivalenceClosure[T : TYPE] : THEORY
BEGIN
  R, S : VAR PRED[[T, T]]
  x, y : VAR T
  % Higher order definition of equivalence relation closure.
  EquivClos(R) : equivalence[T] =
    { (x, y) | FORALL(S : equivalence[T]) : subset?(R, S) IMPLIES S(x, y) }
  EquivClosSuperset : LEMMA
    subset?(R, EquivClos(R))
  EquivClosMonotone : LEMMA
    subset?(R, S) IMPLIES subset?(EquivClos(R), EquivClos(S))
  ...
END EquivalenceClosure
```

Fig. 1. The PVS Prelude in the MathHub Browser

3 Defining the PVS Logic in a Logical Framework

Defining the PVS logic in a logical framework is a significant challenge. Therefore, we start by giving an overview of the difficulties before describing our approach.

Difficulties A logical frameworks like LF [HHP93], Dedukti [BCH12], or λ-Prolog [MN86] tends to admit very elegant definitions for a certain class of logics, but definitions can get awkward quickly if logics fall outside that fragment.

This often boils down to the question of shallow vs. deep encodings. The former represents a logic feature (e.g., subtyping) in terms of a corresponding framework feature, whereas the latter applies a logic encoding to remove the feature (e.g., encode subtyping in a logical framework without subtyping by using a subtyping predicate and coercion functions). Deep encodings have two disadvantages: (i) They destroy structure of the original formalization, often in a way that is not easily invertible and blows up the complexity of library translations. (ii) They require the library translation to apply non-trivial and error-prone steps that become part of the trusted code base. In fact, multiple logical frameworks (including Dedukti) were specifically designed to have a richer logical framework that allows for more logics to be defined elegantly.

Even if we completely ignore the proof theory (and thus the use of decision procedures) entirely, PVS is particularly challenging in this regard. The sequel describes the most important challenges:

The PVS typing relation is undecidable (due to predicate subtyping: selecting a sub-type of α by giving a predicate p as in $\{x \in \alpha | p(x)\}$. Thus, a shallow encoding is impossible in any framework with a decidable typing relation. The most elegant solution is to design a new framework that allows for undecidable typing and then use a shallow encoding.

PVS uses anonymous record types (like in SML) as a primitive feature. This includes record subtyping and a complex variant of record/product/function updates. A deep encoding of anonymous record types is extremely awkward: the simplest encoding would be to introduce a new named product type for every occurrence of a record type in the library. Even then it is virtually impossible to formalize an axiom like "two records are equal if they agree up to reordering of fields" without using a Turing-complete programming language. Therefore, the most feasible option again is to design a new framework that has anonymous record types as a primitive.

PVS uses several types of built-in literals, namely arbitrary-precision integers, rational numbers, and strings. Not every logical framework provides exactly the same set of built-in types and operations on them.

PVS allows for (co)inductive types. While these are relatively well-understood by now, most logical frameworks do not support them. And even if they do, they are unlikely to match the idiosyncrasies of PVS such as declaring a predicate subtype for every constructor. Again it is ultimately more promising to mimic PVS's idiosyncrasies in the logical framework so that we can use a shallow encoding.

PVS uses a module system that, while not terribly complex, does not align perfectly with the modularity primitives of existing logical frameworks. Concretely, theories are parametric, and a theory may import the same theory multiple times with different parameters, in which case any occurrence of an imported symbol is ambiguous. Simple deep encodings can duplicate the multiply-imported symbols or treat them as functions that are applied to the parameters. Both options seem feasible at first but ultimately do not scale well – already the PVS Prelude (the small library of PVS built-ins) causes difficulties. This led us (contrary to our original plans) to mimic PVS-style parametric imports in the logical framework as well to allow for a shallow encoding.

A Flexible Logical Framework We have extensively investigated definitions of PVS in logical frameworks, going back more than 10 years when a first (unpublished) attempt to define PVS in LF was made by Schürmann as part of an ongoing collaboration. In the end, all of the above-mentioned difficulties pointed us in the same direction: the logical framework must adapt to the complexity of PVS – any attempt to adapt PVS to an existing logical framework (by designing an appropriate deep encoding) is likely to be doomed. This negative result is in itself a notable contribution of this paper. It is likely to apply also to similarly complex object logics such as Coq.

Instead, we have spent several years developing the MMT framework. It is bourn out of the tradition of logical frameworks but systematically allows future extensions of the framework. Its main strength is that such extensions, e.g., the features needed for PVS, can be added at extremely low cost: whereas most logical frameworks would require reimplementing most parts starting from the kernel, MMT allows plugging in language features as a routine part of daily development.

It is beyond the scope of this paper to present the architecture of MMT, and we only sketch the extension pathways most critical for PVS.

Firstly, MMT expressions are generic syntax trees including variable binding and labeling [Rab14b]. Besides constants (global identifiers) and bound variables (local identifiers), the leaves of the syntax tree may also be arbitrary literals [Rab15]. An MMT theory defining the language T declares one constant for each expression constructor of T. For example, the MMT theory for LF declares 4 symbols for `type`, λ, Π, and application.

Secondly, all algorithms of the MMT kernel – including parsing, type reconstruction, and computation – are rule-based [Rab17]. Each rule is an object in the underlying programming language that can be generated from a declarative formulation or (in the general case) implemented directly. In either case, the current context determines which rules are used. For example, the MMT theory for LF declares three parsing rules, ten typing/equality rules, and one computation rule, which are sufficient to recover type reconstruction for LF.

Thirdly, MMT allows for *derived* declarations [Ian17]. Each derived declaration indicates the language *feature* that defines its semantics. And the individual features can be easily implemented by MMT plugins, usually by elaborating derived declarations to more primitive ones. For example, we can declare the feature of inductive types, as a derived declaration containing the constructors in its body. Notably, while elaboration defines the meaning of the derived declaration, many MMT algorithms can work with the unelaborated version, e.g., by supplying appropriate induction rules to the type reconstruction algorithm.

Defining PVS To define the language of PVS in MMT, we carried out two steps.

Firstly, we designed a logical framework that extends LF with three features: anonymous record types, predicate subtypes, and imports of multiple instances of the same parametric theory. We use MMT to build this framework modularly. LF (which already existed in MMT) and each of the three new features are defined in a separate MMT theory, each including a few constants and rules for them. Finally, we import all of them to obtain the new logical framework LFX. Then we use LFX to define the MMT theory for PVS. The sequel lists the constants and rules in this theory.

We begin with a definition of PVS's higher-order logic using only LF features. This includes dependent product and function types[4], classical booleans, and the usual formula constructors (see Figure 2). This is novel in how exactly it mirrors

[4] Contrary to typical dependently-typed languages, PVS does not allow declaring dependent *base* types, but predicate subtyping can be used to introduce types that

```
tp : type
expr : tp  type # 1 prec −1
tpjudg : {A} expr A  tp  type # 2 : 3

pvspi : {A} (expr A  tp)  tp #  2
fun_type : tp  tp  tp = [A,B]  [x: expr A] B # 1  2
pvsapply : {A,f : expr A  tp} expr ( f)  {a:expr A} expr (f a) #  3 ( 4 ) prec −1000015
lambda : {A,f : expr A  tp} ({a:expr A}expr (f a))  expr ( f) #  3
```

Fig. 2. Some Basic Typing related Symbols for PVS

the syntax of PVS (e.g., PVS allows multiple aliases for primitive constants) but requires no special Mmt features.

```
setsub : {A} (expr (A  bool))  tp #  1 | 2
rule rules?SetsubRule
```

```
object SetsubRule extends ComputationRule(PVSTheory.expr.path) {
  def apply(check: CheckingCallback)(tm: Term, covered: Boolean)
    (implicit stack: Stack, history: History): Option[Term]
   = tm match {
    case expr(PVSTheory.setsub(tp,prop)) =>
      Some(LFX.Subtyping.predsubtp(expr(tp),proof("internal_judgment",
        Lambda(doName,expr(tp),pvsapply(prop,OMV(doName),expr(tp),bool.term)._1))))
    case _ => None
  }
}
```

Fig. 3. PVS-Style Predicate Subtyping in MMT and the Corresponding Rule

We declare three constants for the three types of built-in literals together with Mmt rules for parsing and typing them. Using the new framework features, we give a shallow encoding of predicate subtyping (see Figure 3 for the new typing rule), a shallow definition of anonymous record types, as well as new declarations for PVS-style inductive and co-inductive types.

4 Translating the PVS Library

The PVS library export requires three separate developments:

depend on terms. Interestingly, this is neither weaker nor stronger than the dependent types in typical $\lambda\Pi$ calculi.

Firstly, PVS has been extended with an XML export. This is similar to the PVS LaTeX extension, which is built on the Common Lisp Pretty Printing facility. The XML export was developed in parallel with a Relax NG specification for the PVS XML files. Because PVS allows overloading of names, infers theory parameters, and automatically adds conversions, the XML generation is driven from the internal type-checked abstract syntax, rather than the parse tree. Thus the generated XML contains the fully type-checked form of a PVS specification with all overloading disambiguated. Future work on this will include the generation of XML forms for the proof trees.

```
<theory place="6049␣0␣6075␣22">
 <id>EquivalenceClosure</id>
 <const−decl place="6057␣2␣6058␣75">
   <id>EquivClos</id>
   <arg−formals>
     <binding place="6057␣12␣6057␣13">
       <id>R</id>
       <type−name>
         <id>PRED</id>
         <actuals>
           <tuple−type>
             <type−name><id>T</id></type−name>
             <type−name><id>T</id></type−name>
           </tuple−type>
         </actuals>
       </type−name>
     </binding>
   </arg−formals>
   <type−name place="6057␣17␣6057␣31">
     <id>equivalence</id>
     <actuals>
       ...
```

Fig. 4. A Part of the Function EquivalenceClosure/EquivClos in XML

Secondly, we documented the XML schema used by PVS as a set of inductive types in Scala (the programming language underlying MMT). We wrote a generic XML parser in Scala that generates a schema-specific parser from such a set of inductive types (see Figure 5 for part of the specification). That way any change to the inductive types automatically changes the parser. While seemingly a minor implementation detail, this was critical for feasibility because the XML schema changed frequently along the way.

Thirdly, we wrote an MMT plugin that parses the XML files generated by PVS and turns them into MMT content. This includes creating various generic indexes that can be used later for searching the content.

```
case class const_decl(
    named: ChainedDecl,
    arg_formals: List[bindings],
    tp: DeclaredType,
    _def: Option[Expr]
) extends Decl
```

Fig. 5. The Scala-Specification of PVS Constant Declarations Used for XML Parsing

All processing steps preserve source references, i.e., URLs that point to a location (file and line/column) in a source file (place= in Figure 4 and <link rel="...?sourceRef" in Figure 6).

```
<omdoc>
 <theory name="EquivalenceClosure"
  base="http://pvs.csl.sri.com/prelude"
  meta="http://pvs.csl.sri.com/?PVS">
  <constant name="EquivClos">
   <type>
    <om:OMOBJ>
     <om:OMA>
      <om:OMS base="http://cds.omdoc.org/urtheories" module="LambdaPi" name="apply"/>
      <om:OMS base="http://pvs.csl.sri.com/" module="PVS" name="expr"/>
      <om:OMA>
       <om:OMS base="http://cds.omdoc.org/urtheories" module="LambdaPi" name="apply"/>
       <om:OMS base="http://pvs.csl.sri.com/" module="PVS" name="pvspi"/>
        ...
        <metadata>
         <link rel="http://cds.omdoc.org/mmt?metadata?sourceRef"
           resource="prelude/pvsxml/EquivalenceClosure.xml#−1.6057.17:−1.6057.31"/>
        </metadata>
      </om:OMA>
```

Fig. 6. A Part of the Function EquivalenceClosure/EquivClos in OMDoc

The following table gives an overview of the sizes of the involved libraries and the run times of the conversion steps:

	PVS source		PVS → XML		XML → MMT	
	size	check time	result size	run time	result size	run time
Prelude	189.7kB	33s	23.5MB	11s	83.3MB	3m41s
NASA Lib	1.9MB	23m25s	387.2MB	3m11s	2.5GB	58m56s

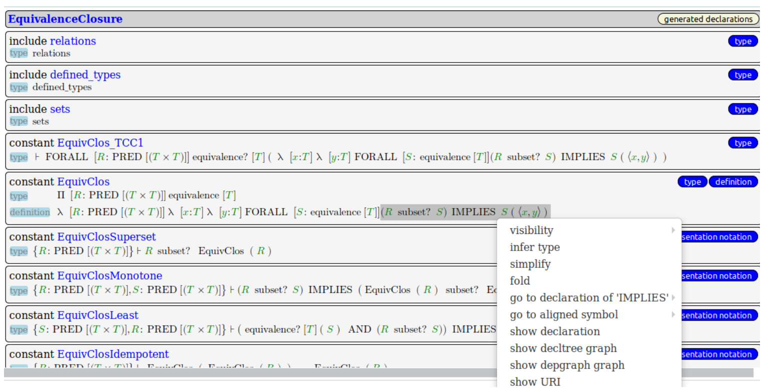
5 Applications

With the OMDoc/MMT translation of the PVS library, PVS gains access to library management facilities implemented at the OMDoc/MMT level. There are two ways two ways of taking advantage of these: publishing the converted

PVS library on MathHub or running the MathHub/Mmt toolstack locally. Both options offer similar functionality, the main difference is the intended audience: the first option is for outside users who want to access the PVS library and the latter is for PVS developers who want to develop new content or refactor the library.

The MathHub portal [Ian+14; MH] bundles a GitLab-based repository manager with a Mmt process and various periphery system under a common, web-based user interface. We commit the exported PVS libraries as OMDoc/Mmt files into the repository [GMP] as separate libraries – currently `Prelude` and `NASA`. MathHub has been configured to make these available via the *i*) MathHub user interface, *ii*) Mmt presentation web server, *iii*) Mmt web services, and *iv*) the MathWebSearch daemon. All of these components give the user different ways of interacting with the system and PVS content. We will explore three that are directly useful for PVS users below.

The "local MathHub" (LMH) workflow installs the components except the repository manager on a docker container and runs them from the local file system – or more commonly – from a local working copy, so that the results can be synchronized with MathHub. In that case, users are able to browse the current version of the available PVS libraries including all experimental or private theories that are part of the current development.


EquivalenceClosure

include relations
type relations

include defined_types
type defined_types

include sets
type sets

constant EquivClos_TCC1
type ⊢ FORALL [R: PRED [(T × T)]] equivalence? [T] (λ [x:T] λ [y:T] FORALL [S: equivalence [T]](R subset? S) IMPLIES S (⟨x,y⟩))

constant EquivClos
type Π [R: PRED [(T × T)]] equivalence [T]
definition λ [R: PRED [(T × T)]] λ [x:T] λ [y:T] FORALL [S: equivalence [T]](R subset? S) IMPLIES S (⟨x,y⟩))

constant EquivClosSuperset
type {R: PRED [(T × T)]} ⊢ R subset? EquivClos (R)

constant EquivClosMonotone
type {R: PRED [(T × T)],S: PRED [(T × T)]} ⊢ (R subset? S) IMPLIES (EquivClos (R) subset? Ec

constant EquivClosLeast
type {S: PRED [(T × T)],R: PRED [(T × T)]} ⊢ (equivalence? [T] (S) AND (R subset? S)) IMPLIES

constant EquivClosIdempotent

Fig. 7. The PVS Prelude in the Mmt Browser

5.1 Browsing

The transformed PVS content can be browsed interactively in the document-oriented MathHub presentation pages (theories as active documents) and in the Mmt web browser (see Figure 7). Both allow interaction with the PVS content via a special Javascript framework JOBAD – the right-click menu in Figure 7 is provided by JOBAD. The Mmt web server, which specializes on formal content also provides buttons to affect visibility of types, definitions, inferred arguments, etc. as it is more oriented towards an formalization IDE than an enhanced scientific document.

The MMT instance in the LMH workflow provides the additional feature of inter-process communication between PVS and MMT as a new JOBAD menu item: the action *navigate to this declaration in connected systems*. We implemented a listener for this action that forwards the command to PVS via an XML RPC call at the default PVS port. Correspondingly, we implemented a case in the PVS server that opens the corresponding file in the PVS emacs system and navigates to the relevant line.

5.2 Graph Viewer

MathHub includes a theory graph viewer that allows interactive, web-based exploration of the OMDoc/MMT theory graphs. It builds on the `visjs` JavaScript visualization library [VJ], which uses the HTML5 canvas to layout and interact with graphs client-side in the browser.

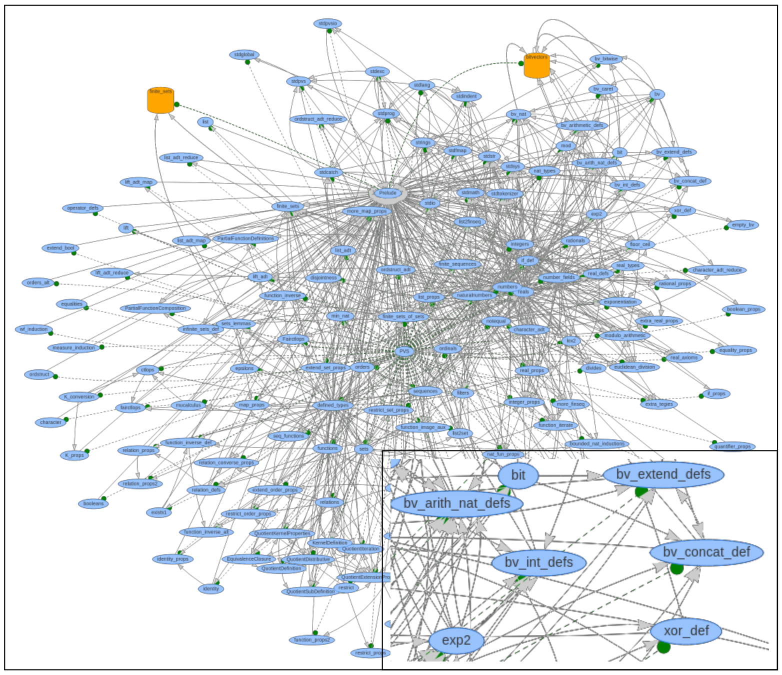


Fig. 8. The PVS Prelude in the MathHub Theory Graph viewer

PVS libraries make heavy use of theories as a structuring mechanism, which makes a graph viewer for PVS particularly attractive. Figure 8 shows the Prelude and NASA graph: in the back we see the full graph in a central-gravity layout of the theory graph induced by the PVS prelude, we have (manually) clustered the subgraphs for bit vectors and finite sets to focus on the rest. Nodes and (named) edges support the same JOBAD actions in the graph viewers as theories and structures/views do in the browsers discussed above. Most usefully, we can right-click graph nodes to navigate to the theory – in the MMT web browser or the PVS sources (via the source references).

The theory graph allows dragging nodes around to fine-tune the layout. Hovering over a node or edge triggers a preview of the theory. All nodes support the same context menu as in the browser above. Thus, it is possible to select a theory in the graph viewer and then navigate to it in the browser or (if run locally) in the PVS emacs interface.

5.3 Search

MathWebSearch [KŞ06] is an OMDoc/MMT-level formula search engine that uses query variables for subterms and first-order unification as the query language. It is developed independently, but MMT includes a plugin for generating MathWebSearch index files using its content MathML interface. Thus, any library available to MMT can be indexed and searched via MathWebSearch. Moreover, MMT includes a frontend for MathWebSearch so that search queries can be supplied in any format that MMT can understand, e.g., the XML format produced by PVS.

```
<mws:query limitmin="0" answsize="1000" totalreq="yes"
  output="xml" xmlns:m="http://www.w3.org/1998/Math/MathML"
  xmlns:mws="http://www.mathweb.org/mws/ns">
 <mws:expr>
  <apply>
   <csymbol>http://cds.omdoc.org/urtheories?LambdaPi?apply</csymbol>
   <csymbol>http://pvs.csl.sri.com/?PVS?pvsapply</csymbol>
   <mws:qvar xmlns:mws="http://www.mathweb.org/mws/ns">I1</mws:qvar>
   <mws:qvar xmlns:mws="http://www.mathweb.org/mws/ns">I2</mws:qvar>
   <csymbol>http://pvs.csl.sri.com/prelude?EquivalenceClosure?EquivClos</csymbol>
   <mws:qvar xmlns:mws="http://www.mathweb.org/mws/ns">A</mws:qvar>
  </apply>
 </mws:expr>
</mws:query>
```

Fig. 9. A Query for Applications of EquivClos

MMT exposes the search frontend both in its GUI for humans and as an HTTP service for other systems. Here we use the latter: We have added an interface feature to the PVS emacs system that allows users to enter a search query in PVS syntax. PVS parses the query, type-checks it, and converts it to XML. The XML is sent to MMT, which acts as the mediator between the proof assistant – here PVS – and library management tools – here MathWebSearch– and returns the search results to PVS.

```
[ {"lib_name" : "",
   "theory_name" : "EquivalenceClosure",
   "name" : "EquivClosMonotone",
   "component" : "type",
   "Position" : "3_2_5_5_5_5"},
  {"lib_name" : "",
   "theory_name" : "EquivalenceClosure",
   "name" : "EquivalenceCharacterization",
   "component" : "type",
   "Position" : "2_2_5_5_5_5"},
  ...
]
```

Fig. 10. A Query Result for Fig. 9

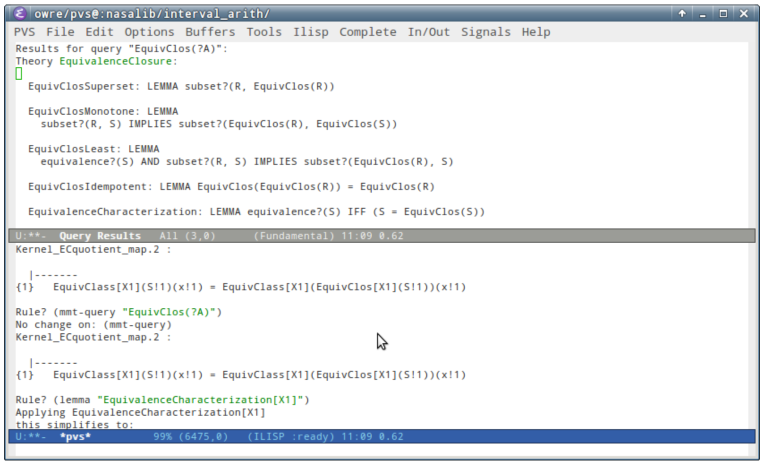
The PVS user enters the PVS query EquivClos(?A), where we have extended the PVS syntax by for query variables like ?A. After OMDoc/MMT translation, this becomes the MathWebSearch query in Figure 9 – note the additional LF-level application and the inferred query variables I1 and I1 for the domain and range types of the EquivClos function. MathWebSearch returns a JSON record with four whose start is shown in Figure 10: we see the occurrences of (instances of) EquivClos(?A) in declarations in theory EquivalenceClosure Figure 1. There are actually two more occurrences in the Prelude which currently require a separate MathWebSearch query due to theory parameters.

Figure 11 shows a mockup of what the query might look like while doing a PVS proof. Still to be done are the definition of a query language in PVS, including the meta-variables, and sending the XML query to the MMT server. Then it needs to be put to use in actual proofs, to determine the most effective ways to use this facility.

6 Conclusion

The work reported in this paper contributes to avoiding duplication of efforts in the development of theorem proving system, their libraries, and supporting periphery systems. Concretely, we have developed a representation of the PVS logic in and an automated translation of the PVS libraries into in the OMDoc/MMT representation format.

In contrast to earlier representation and translation projects undertaken by us – e.g Mizar and HOL Light – the PVS logic is much more challenging due


owre/pvs@:nasalib/interval_arith/
PVS File Edit Options Buffers Tools Ilisp Complete In/Out Signals Help

```
Results for query "EquivClos(?A)":
Theory EquivalenceClosure:

  EquivClosSuperset: LEMMA subset?(R, EquivClos(R))

  EquivClosMonotone: LEMMA
    subset?(R, S) IMPLIES subset?(EquivClos(R), EquivClos(S))

  EquivClosLeast: LEMMA
    equivalence?(S) AND subset?(R, S) IMPLIES subset?(EquivClos(R), S)

  EquivClosIdempotent: LEMMA EquivClos(EquivClos(R)) = EquivClos(R)

  EquivalenceCharacterization: LEMMA equivalence?(S) IFF (S = EquivClos(S))

U:**-  Query Results   All (3,0)      (Fundamental) 11:09 0.62
Kernel_ECquotient_map.2 :

  |-------
{1}   EquivClass[X1](S!1)(x!1) = EquivClass[X1](EquivClos[X1](S!1))(x!1)

Rule? (mmt-query "EquivClos(?A)")
No change on: (mmt-query)
Kernel_ECquotient_map.2 :

  |-------
{1}   EquivClass[X1](S!1)(x!1) = EquivClass[X1](EquivClos[X1](S!1))(x!1)

Rule? (lemma "EquivalenceCharacterization[X1]")
Applying EquivalenceCharacterization[X1]
this simplifies to:
U:**-  *pvs*         99% (6475,0)   (ILISP :ready) 11:09 0.62
```

Fig. 11. A Mockup of the Query Result in PVS

to its highly expressive language features, which defy formalization in current logical frameworks like LF. Therefore we make use of the extensibility of the OMDoc/MMT system and implement several extensions of LF (LFX). In essence, we use the MMT system as a prototyping system for logical frameworks. Our experience with encoding the PVS logic is that critical features such as undecidable subtyping, record types, (co)inductive types and literals can naturally be expressed at this level. While our implementation certainly falls short of the maturity of established logical frameworks, it already proved very useful as a development tool. Most importantly, we use LFX to give shallow and therefore structure-preserving encodings of PVS features without having to forgo the advantages of logical frameworks.

This information architecture is essential for system interoperability. In our case we have shown that we can use the generic – i.e. system-independent – MMT tool chain for PVS. Concretely we have shown three specific, but integrated and generic periphery systems: a library browser, a theory graph viewer, and a search engine. Given the OMDoc/MMT representation of the PVS library, these directly work for PVS libraries and can be easily hooked together with the PVS system. The three systems together substitute and improve the existing functionality that was designed specifically for PVS such as the Hypatheon system [Lan16a].

But the generic periphery gives us more than just improved services, it also applies to all the other theorem proving libraries that have been translated to OMDoc/MMT: We have already mentioned HOL Light and Mizar above. But we also have experimental translations of TPS, TPTP, TPS, Focalize, Specware, IMPS, and Metamath, as well as several informal mathematical libraries including the OEIS and the SMGloM terminology base. Using flexible alignments [Kal+16] between the libraries, we can guide library developers to corresponding parts of other formalizations, approximately translate the content across libraries, or reuse notations (e.g. to show HOL Light content in a form that looks familiar to PVS users).

Acknowledgements This work has been partially funded by DFG under Grants KO 2428/13-1 and RA-18723-1. The authors gratefully acknowledge the contribution of Marcel Rupprecht, who has extended the graph viewer for this paper.

References

[BCH12] M. Boespflug, Q. Carbonneaux, and O. Hermant. "The $\lambda\Pi$-calculus modulo as a universal proof language". In: *Proceedings of PxTP2012: Proof Exchange for Theorem Proving*. Ed. by D. Pichardie and T. Weber. 2012, pp. 28–43.

[GK14] T. Gauthier and C. Kaliszyk. "Matching concepts across HOL libraries". In: *Intelligent Computer Mathematics*. Ed. by S. Watt et al. Springer, 2014, pp. 267–281.

[GMP] *MathHub PVS Git Repository*. URL: http://gl.mathhub.info/PVS (visited on 04/11/2017).

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. "A framework for defining logics". In: *Journal of the Association for Computing Machinery* 40.1 (1993), pp. 143–184.

[Ian+13] Mihnea Iancu et al. "The Mizar Mathematical Library in OMDoc: Translation and Applications". In: *Journal of Automated Reasoning* 50.2 (2013), pp. 191–202. DOI: 10.1007/s10817-012-9271-4.

[Ian+14] Mihnea Iancu et al. "System Description: MathHub.info". In: *Intelligent Computer Mathematics 2014*. Ed. by Stephan Watt et al. LNCS 8543. Springer, 2014, pp. 431–434. ISBN: 978-3-319-08433-6. URL: http://kwarc.info/kohlhase/papers/cicm14-mathhub.pdf.

[Ian17] Mihnea Iancu. "Towards Flexiformal Mathematics". PhD thesis. Bremen, Germany: Jacobs University, 2017.

[Kal+16] Cezary Kaliszyk et al. "A Standard for Aligning Mathematical Concepts". In: *Intelligent Computer Mathematics – Work in Progress Papers*. Ed. by Michael Kohlhase et al. 2016. URL: http://kwarc.info/kohlhase/papers/cicmwip16-alignments.pdf.

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

[KR14] Cezary Kaliszyk and Florian Rabe. "Towards Knowledge Management for HOL Light". In: *Intelligent Computer Mathematics 2014*. Ed. by Stephan Watt et al. LNCS 8543. Springer, 2014, pp. 357–372. ISBN: 978-3-319-08433-6. URL: http://kwarc.info/frabe/Research/KR_hollight_14.pdf.

[KR16] M. Kohlhase and F. Rabe. "QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge". In: *Journal of Formalized Reasoning* 9.1 (2016), pp. 201–234.

[KS10] A. Krauss and A. Schropp. "A Mechanized Translation from Higher-Order Logic to Set Theory". In: *Interactive Theorem Proving*. Ed. by M. Kaufmann and L. Paulson. Springer, 2010, pp. 323–338.

[KU15] C. Kaliszyk and J. Urban. “HOL(y)Hammer: Online ATP Service for HOL Light”. In: *Mathematics in Computer Science* 9.1 (2015), pp. 5–22.

[KW10] C. Keller and B. Werner. “Importing HOL Light into Coq”. In: *Interactive Theorem Proving*. Ed. by M. Kaufmann and L. Paulson. Springer, 2010, pp. 307–322.

[KŞ06] M. Kohlhase and I. Şucan. “A Search Engine for Mathematical Formulae”. In: *Artificial Intelligence and Symbolic Computation*. Ed. by T. Ida, J. Calmet, and D. Wang. Springer, 2006, pp. 241–253.

[Lan16a] NASA Langley. *Hypatheon: A Database Capability for PVS Theories*. `https://shemesh.larc.nasa.gov/people/bld/hypatheon.html`. 2016.

[Lan16b] NASA Langley. *NASA PVS Library*. `http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html`. 2016.

[MH] *MathHub.info: Active Mathematics*. URL: `http://mathhub.info` (visited on 01/28/2014).

[MN86] D. Miller and G. Nadathur. “Higher-order logic programming”. In: *Proceedings of the Third International Conference on Logic Programming*. Ed. by E. Shapiro. Springer, 1986, pp. 448–462.

[MP08] J. Meng and L. Paulson. “Translating Higher-Order Clauses to First-Order Clauses”. In: *Journal of Automated Reasoning* 40.1 (2008), pp. 35–60.

[ORS92] S. Owre, J. Rushby, and N. Shankar. “PVS: A Prototype Verification System”. In: *11th International Conference on Automated Deduction (CADE)*. Ed. by D. Kapur. Springer, 1992, pp. 748–752.

[OS06] S. Obua and S. Skalberg. “Importing HOL into Isabelle/HOL”. In: *Automated Reasoning*. Ed. by N. Shankar and U. Furbach. Vol. 4130. Springer, 2006.

[Pfe+03] F. Pfenning et al. *The Logosphere Project*. `http://www.logosphere.org/`. 2003.

[Rab14a] F. Rabe. “A Logic-Independent IDE”. In: *Workshop on User Interfaces for Theorem Provers*. Ed. by C. Benzmüller and B. Woltzenlogel Paleo. Elsevier, 2014, pp. 48–60.

[Rab14b] Florian Rabe. “How to Identify, Translate, and Combine Logics?” In: *Journal of Logic and Computation* (2014). DOI: `10.1093/logcom/exu079`.

[Rab15] F. Rabe. “Generic Literals”. In: *Intelligent Computer Mathematics*. Ed. by M. Kerber et al. Springer, 2015, pp. 102–117.

[Rab17] F. Rabe. “A Modular Type Reconstruction Algorithm”. see `http://kwarc.info/frabe/Research/rabe_recon_17.pdf`. 2017.

[RK13] F. Rabe and M. Kohlhase. “A Scalable Module System”. In: *Information and Computation* 230.1 (2013), pp. 1–54.

[VJ] *vis.js – A dynamic, browser based visualization library*. URL: `http://visjs.org` (visited on 01/17/2017).

[Wat+14] Stephan Watt et al., eds. *Intelligent Computer Mathematics*. LNCS 8543. Springer, 2014. ISBN: 978-3-319-08433-6.