Representing Categories of Theories in a Proof-Theoretical Logical Framework

Fulya Horozal and Florian Rabe

Computer Science, Jacobs University Bremen, Germany

Logical frameworks have been introduced as an abstract formalism for representing logics and studying their properties. We are interested in how these frameworks support *(i)* the representation of the categories of signatures or theories usually associated with logics and related declarative languages and *(ii)* the reasoning about and the formulation of algorithms on signatures and theories such as type-checking or functor application.

Model theoretical logical frameworks, such as institutions [3], work with fully abstract categories. This has the advantage that the framework is independent of the particular logic; thus theory-related concepts such as model classes can be formulated in full generality. However, this makes it difficult to provide generic tool support for reasoning and algorithms about individual theories.

Proof theoretical logical frameworks, such as LF [4], on the other hand, work with concrete theories and represent each individual theory of a logic. They provide generic tool support for individual theories. However, they do not have an abstract concept to represent the category of theories as a whole, and therefore, they cannot reason about or operate on arbitrary theories.

The Logic Atlas and Integrator (LATIN) project [1] developed a foundationally unconstrained logical framework [2], which unifies aspects of both model theoretical and proof theoretical frameworks. In LATIN, logics are represented as diagrams of theories, and translations between logics as theory morphisms between these diagrams. Below, we focus on the LF instance of the LATIN framework, but our work can be generalized to other frameworks (e.g., Isabelle [5]).

In LATIN-LF, the syntax of a logic is represented as an LF signature L and theories of the logic as inclusion morphisms $L \hookrightarrow \Sigma$, where Σ adds the declarations of the non-logical symbols to L. Thus, the category of theories of L is a subcategory of the coslice category $LF\backslash L$ containing only inclusions.

However, this category usually contains a lot more objects than needed. For instance, if $L = FOL$ represents first-order logic, then Σ should only contain declarations of function or predicate symbols. More precisely, if *term* and *form* are the types of first-order terms and formulas, then Σ should contain only declarations of the form $f : \mathit{term} \to \ldots \to \mathit{term} \to \mathit{term}$ (n-ary function symbols) and $p : \mathit{term} \to \ldots \to \mathit{term} \to \mathit{form}$ (n-ary predicate symbols).

In LATIN-LF, functors between categories of theories are represented as LF signature morphisms. More specifically, a functor from the category of theories of L to that of L' is obtained by an LF signature morphism $l : L \to L'$, and functor application is obtained by pushout along l.

However, this is not expressive enough to cover all interesting functors. For example, it cannot represent the translation from FOL to ZFC, where the first-

order declaration $f : term \to term$ is translated to a set F together with an axiom that F is a unary ZFC function on the universe.

In this work, we introduce LFP, a major extension of the LF type theory with what we call *declaration patterns*. LFP permits elegant and concise representation of categories of theories in LF. Our extension is based on the central observation that theories of virtually all logics L consist of a list of declarations each of which must conform to one of a few patterns that are fixed by L. By adding a new primitive to LF that defines such patterns, we are able to represent these categories.

More specifically, declaration patterns use parameterized signatures with a simple type theory built on top. Interestingly, we have to introduce one additional primitive concept in LFP that is technically independent but practically necessary for declaration patterns: We usually need sequences of LF objects to define the declaration patterns of a logic L even if the formulas of L do not use sequences. For example, in FOL, we would introduce the declaration patterns $func = \lambda n : Nat.\{f : term^n \to term\}$ and $pred = \lambda n : Nat.\{p : term^n \to form\}$ for n-ary function and predicate symbols respectively, where Nat is the type of natural numbers in LFP and $term^n$ denotes the sequence $term, \ldots, term$ of length n.

Declaration patterns also prove crucial for logic translations because they permit concise representations of functors between theory categories. We introduce a notion of pattern-based functors between categories of theories. These are defined by induction on the list of declarations in a theory using one case for every declaration pattern. Moreover, functors that arise from pushouts can be recovered as a special case.

LFP is implemented as part of the MMT system [6]. Our implementation can match arbitrary LF signatures against the declaration patterns defined in an LFP signature. Matching signatures can be translated along the functors described in LFP.

References

1. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, and F. Rabe. Project Abstract: Logic Atlas and Integrator (LATIN). In J. Davenport, W. Farmer, F. Rabe, and J. Urban, editors, *Intelligent Computer Mathematics*, volume 6824 of *Lecture Notes in Computer Science*, pages 287–289. Springer, 2011.
2. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, F. Rabe, and K. Sojakova. Towards Logical Frameworks in the Heterogeneous Tool Set Hets. In T. Mossakowski and H. Kreowski, editors, *Recent Trends in Algebraic Development Techniques 2010*, volume 7137 of *Lecture Notes in Computer Science*, pages 139–159. Springer, 2012.
3. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.
4. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.
5. L. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, 1994.
6. F. Rabe and M. Kohlhase. A Scalable Module System. see http://arxiv.org/abs/1105.0548, 2011.