# Diagram Operators in a Logical Framework

Navid Roux Florian Rabe

FAU Erlangen-Nuremberg
Germany

Computer Science Dept.
Faculty of Engineering
Friedrich-Alexander-Universität
Erlangen-Nuremberg, Germany

`navid.roux@fau.de` `florian.rabe@fau.de`

Module systems for logical frameworks allow building logics and calculi component-wise from small self-contained building blocks. While this enables a high degree of reuse, we still observed many situations where a module system was not sufficient to eliminate redundancy. One particular source of redundancy are module-level syntactic transformations that are entirely formulaic and therefore automatable, but also too complex to capture in typical module systems. In order to overcome this, we introduced the concept of diagram operators and implemented it in the MMT framework (which allows implementing various LF-style logical frameworks).

In this paper, we show how we apply diagram operators to eliminate redundancy in large modular libraries of logic formalizations. This enables building many related formalizations in a systematically related way, which simplifies the formalization process and enhances its results, while massively reducing the maintenance cost.

Diagrams of theories and theory morphisms were originally developed for algebraic specification languages [9, 5] and deduction systems [4, 6, 11]. They enable both building theories modularly and capturing representing theorems. Because logical frameworks allow treating logics as theories, the method can also be used for building logics.

The biggest case study in this direction was the LATIN project [2], where we used the Twelf module system [8] to formalize syntax, proof theory, and model theory of many logical connectives separately and built logics by combining their features. This not only led to a high degree of modularity and reuse but also made the connections between the logics more visible and teachable. It also significantly increased the number of logics that could be formalized in parallel while keeping the maintenance efforts sublinear.

However, we still observed many frustrating sources of redundancy that the module system could not eliminate. For example, formalizations of quantifiers and deduction rules in typed and untyped logics can be systematically derived from each other. Yet, we still had to formalize them individually because no module system is expressive enough to capture the derivation. Keep in mind that typed quantifiers need to take an additional argument for the type, which already goes beyond the expressivity of most module systems. Moreover, while it is logically simpler to formalize typed quantifiers first and then derive untyped ones by dropping an argument, it is didactically more appealing to formalize the untyped variant first and lift it to the typed one. A similar example are the formalizations of type-theoretical features such as function and product types relative to Curry vs. Church style formalizations of type theories. Again we had to formalize both manually even though they can in principle be systematically derived from each other.

Recently, partially motivated by these problems and inspired by ideas in [3, 1], the second author generalized the idea of theory formation operators typical for module systems to diagram operators [10],

Submitted to:
LFMTP 2020

where an operation is applied not only to a single theory but to an entire diagram at once. For example, we can form the diagram of untyped quantifiers and their interrelations, and then systematically derive the corresponding diagram of typed quantifiers. Often these operators are functorial and enjoy further valuable properties such as preserving the modular structure of the theories in the input diagram.

Diagram operators have been implemented in the MMT system [7], where individual operators are implemented as self-contained objects in the underlying programming language. This gives them access to the syntax trees of the modules and allows implementing arbitrarily complex syntactic transformations on them. Every operator is declared as an MMT symbol and implemented by a computation rule. Thus, all our operators are directly usable in MMT specifications, and all our examples have been formalized that way.

We have streamlined and expanded this implementation and applied it by implementing several important operators on formalizations of logical features in LF-style logical frameworks. This includes the operators CHURCH TO CURRY and TYPIFY FOL from above as well as INDEXIFY (a more general variant of the former), PUSHOUT, and OUT-/INLINING.

Because MMT is foundation-independent per se, these diagram operators are immediately applicable to any logical framework implemented in MMT, including many LF-style frameworks.

Our efforts are part of LATIN2, a long-term effort to further improve the LATIN logic library, including a from-scratch reimplementation in MMT (many parts of which were motivated by lessons learned in LATIN). Within LATIN2 so far, application of our operators enabled reduction by 100+ LOC and, more importantly, the synchronization of knowledge of what was redundant and badly kept in sync before.

These results are extremely promising and we expect defining more operators and realizing a superlinear increase in generated formalizations in the future.

## References

[1] J. Carette & R. O'Connor (2012): *Theory Presentation Combinators*. In J. Jeuring, J. Campbell, J. Carette, G. Dos Reis, P. Sojka, M. Wenzel & V. Sorge, editors: *Intelligent Computer Mathematics*, 7362, Springer, pp. 202–215.

[2] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski & F. Rabe (2011): *Project Abstract: Logic Atlas and Integrator (LATIN)*. In J. Davenport, W. Farmer, F. Rabe & J. Urban, editors: *Intelligent Computer Mathematics*, Springer, pp. 289–291.

[3] DOL editors (2018): *The Distributed Ontology, Modeling, and Specification Language (DOL)*. Technical Report, Object Management Group. Available at `https://www.omg.org/spec/DOL/About-DOL/`.

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

[5] J. Goguen, Timothy Winkler, J. Meseguer, K. Futatsugi & J. Jouannaud (1993): *Introducing OBJ*. In J. Goguen, D. Coleman & R. Gallimore, editors: *Applications of Algebraic Specification using OBJ*, Cambridge.

[6] F. Kammüller, M. Wenzel & L. Paulson (1999): *Locales – a Sectioning Concept for Isabelle*. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin & L. Thery, editors: *Theorem Proving in Higher Order Logics*, Springer, pp. 149–166.

[7] D. Müller & F. Rabe (2019): *Rapid Prototyping Formal Systems in MMT: Case Studies*. In D. Miller & I. Scagnetto, editors: *Logical Frameworks and Meta-languages: Theory and Practice*, pp. 40–54.

[8] F. Rabe & C. Schürmann (2009): *A Practical Module System for LF*. In J. Cheney & A. Felty, editors: *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, ACM Press, pp. 40–48.

[9] D. Sannella & M. Wirsing (1983): *A Kernel Language for Algebraic Specification and Implementation*. In M. Karpinski, editor: *Fundamentals of Computation Theory*, Springer, pp. 413–427.

[10] Y. Sharoda & F. Rabe (2019): *Diagram Operators in MMT*. In C. Kaliszyk, E. Brady, A. Kohlhase & C. Sacerdoti Coen, editors: *Intelligent Computer Mathematics*, Springer, pp. 211–226.

[11] Y. Srinivas & R. Jüllig (1995): *Specware: Formal Support for Composing Software*. In B. Möller, editor: *Mathematics of Program Construction*, Springer.