# Structuring Theories with Implicit Morphisms

Florian Rabe[1,2] and Dennis Müller[2]

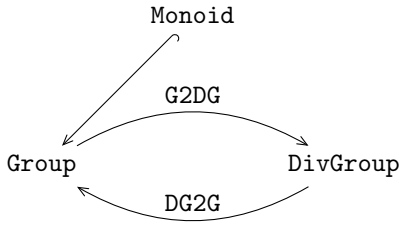
[1] LRI Paris
[2] FAU Erlangen-Nuremberg

**Abstract.** We introduce *implicit* morphisms as a concept in formal systems based on theories and theory morphisms. The idea is that there may be at most one implicit morphism from a theory $S$ to a theory $T$, and if $S$-expressions are used in $T$ their semantics is obtained by automatically inserting the implicit morphism. The practical appeal of implicit morphisms is that they hit the sweet-spot of being extremely simple to understand and implement while significantly helping with structuring large collections of theories.

Theory morphisms have proved an essential tool for managing collections of theories in logics and related formal systems. They can be used to structure theories and build large theories modularly from small components or to relate different theories to each other [SW83,AHMS99,FGT92]. Areas in which tools based on theories and theory morphisms have been developed include specification [GWM+93,MML07], rewriting [CELM96], theorem proving [FGT93], and knowledge representation [RK13].

These systems usually use a logic $L$ for the fine-granular formalization of domain knowledge, and a diagram $D$ in the category of $L$-theories and $L$-morphisms for the high-level structure of large bodies of knowledge. This diagram is generated by all theories/morphisms defined, induced, or referenced in a user's development.

For example, a user might reference an existing theory `Monoid`, define a new theory `Group` that extends `Monoid`, define a theory `DivGroup` (providing an alternative formulation of groups based on the division operation), and then define two theory morphisms `G2DG` : `Group` $\leftrightarrow$ `DivGroup` : `DG2G` that witness an isomorphism between these theories. This would result in the diagram on the right. Note that we use the syntactic direction for the arrows, e.g., an arrow $m : S \to T$ states that any $S$-expression $E$ (e.g., a sort, term, formula, or proof) can be translated to an $T$-expression $m(E)$. Crucially, $m(-)$ preserves typing and provability.




The key idea behind implicit morphisms is very simple: We maintain an additional diagram $I$, which is commutative subdiagram of $D$ and whose morphisms we call *implicit*. The condition of commutativity guarantees that $I$ has at most one morphism $i$ from theory $S$ to theory $T$, in which case we write $S \overset{*}{\hookrightarrow} T$.

Commutativity makes the following language extension well-defined: if $S \overset{*}{\hookrightarrow} T$, then any identifier $c$ that is visible to $S$ may also be used in $T$-expressions; and if $c$ is used in a $T$-expression, the semantics of $c$ is $i(c)$ where $i$ is the uniquely determined implicit morphism $i : S \to T$.

Despite their simplicity, the practical implications of implicit morphism are huge. For example, in the diagram above, we may choose to label `G2DG` implicit. Immediately, every abbreviation or theorem that we have formulated in the theory `Group` becomes available for use in `DivGroup` without any syntactic overhead. We can even label `DG2G` implicit as well if we prove the isomorphism property to ensure that $I$ remains commutative, thus capturing the mathematical intuition that `Group` and `DivGroup` are just different formalizations of the same concept. While these morphisms must be labeled manually, any inclusion morphism like the one from `Monoid` to `Group` is implicit automatically.

In fact, this principle works so well that we have refactored our MMT system (our long-standing implementation) in such a way that implicit morphisms are now more primitive than inclusion morphisms. The semantics of inclusion morphisms is obtained by saying that inclusions are implicit morphisms that map all identifiers to themselves. Even the fundamental property that a theory may reference its own identifiers is now just a consequence of the fact that all identity morphisms are implicit. Therefore, surprisingly, adding implicit morphisms deep in the MMT kernel has made its design more elegant.

## References


AHMS99. S. Autexier, D. Hutter, H. Mantel, and A. Schairer. Towards an Evolutionary Formal Software-Development Using CASL. In D. Bert, C. Choppy, and P. Mosses, editors, *WADT*, volume 1827 of *Lecture Notes in Computer Science*, pages 73–88. Springer, 1999.

CELM96. M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proceedings of the First International Workshop on Rewriting Logic*, volume 4, pages 65–89, 1996.

FGT92. W. Farmer, J. Guttman, and F. Thayer. Little Theories. In D. Kapur, editor, *Conference on Automated Deduction*, pages 467–581, 1992.

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

GWM+93. J. Goguen, Timothy Winkler, J. Meseguer, K. Futatsugi, and J. Jouannaud. Introducing OBJ. In J. Goguen, D. Coleman, and R. Gallimore, editors, *Applications of Algebraic Specification using OBJ*. Cambridge, 1993.

MML07. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *Tools and Algorithms for the Construction and Analysis of Systems 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522, 2007.

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

SW83. D. Sannella and M. Wirsing. A Kernel Language for Algebraic Specification and Implementation. In M. Karpinski, editor, *Fundamentals of Computation Theory*, pages 413–427. Springer, 1983.