Sikkel: Multimode Simple Type Theory as an Agda Library

Joris Ceulemans, Andreas Nuyts, and Dominique Devriese

imec-DistriNet, Department of Computer Science, KU Leuven, Belgium

Many variants of type theory extend a basic theory with additional primitives or properties to enable constructions or proofs that would be harder or impossible to do in the original theory. Examples of such extensions include guarded recursion [7, 13], parametricity [2, 3, 4, 19, 18, 8], univalence [5, 9, 11], directed type theory [21, 22] and nominal reasoning [20]. The question addressed in this abstract is how these extended systems can be used in existing dependently typed programming languages like Agda or Coq. We present Sikkel[1]: an Agda library that allows users to work in a class of extended (simple) type theories called multimode or multimodal type theories [16, 13], which are parametrized by a mode theory that specifies new primitive type constructors called modalities (but non-modal primitives can be easily added to Sikkel as well). Such modalities are not only useful in the object theory, but they also allow for an elegant translation of Sikkel programs to Agda programs. Fig. 1 shows Sikkel's different components, which will be discussed in the following sections.

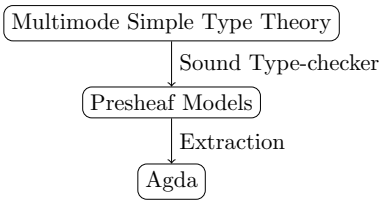

Figure 1: Sikkel's architecture.

This abstract is based on a full paper, which was presented at MSFP 2022 [10].

Multimode Simple Type Theory (MSTT) The upper layer of Fig. 1 represents MSTT, which is essentially Gratzer et al.'s MTT [13] restricted to simple types.[2] Our implementation of MSTT is parametrized by a ModeTheory record, which specifies an Agda type ModeExpr of modes, a type family ModalityExpr of modalities (indexed by two modes), as well as a unit modality, modality composition and a type of 2-cells (i.e. a mode theory is basically a strict 2-category). All MSTT contexts, types and terms live at a certain mode and are represented in Sikkel as values of data types CtxExpr, TyExpr and TmExpr indexed by a mode. Sikkel's MSTT syntax is extrinsically typed, so TmExpr is not indexed by a context or type. There is built-in support for booleans, natural numbers, function types and product types. Every modality μ from mode m to n gives rise to a modal type former, transforming a type T at mode m to $\langle\ \mu\ \mid\ T\ \rangle$ at mode n, and a lock (left adjoint) on contexts, transforming a context Γ at mode n to $\Gamma\ ,\mathsf{lock}\langle\ \mu\ \rangle$ at mode m. Variables are represented by strings[3] and always appear in the context under a modality. The following are 3 of the more interesting MSTT typing rules.

$$\frac{\alpha \in \mu \Rightarrow \mathsf{locks}(\Delta) \qquad x \notin \Delta}{\Gamma\ ,\ \mu\ \mid\ x \in T\ ,\Delta \vdash \mathsf{var}\ x\ \alpha : T}\ \textsc{Var} \qquad \frac{\Gamma\ ,\ \mu\ \mid\ x \in T \vdash s : S}{\Gamma \vdash \mathsf{lam}[\ \mu\ \mid\ x \in T\ ]\ s : \langle\ \mu\ \mid\ T\ \rangle \Rrightarrow S}\ \textsc{Lam} \qquad \frac{\Gamma\ ,\mathsf{lock}\langle\ \mu\ \rangle \vdash t : T}{\Gamma \vdash \mathsf{mod}\langle\ \mu\ \rangle\ t : \langle\ \mu\ \mid\ T\ \rangle}\ \textsc{ModIntro}$$

A variable x that is in the context under modality μ can be used to construct a term, as long as there is a 2-cell from μ to the composite of all modalities that appear in locks to the right of x (Var). In many cases, these two modalities are the same so we can replace α with the trivial 2-cell which we abbreviate as svar x. Lambda abstraction produces a modal function by extending the context with a variable under the same modality (Lam). The introduction rule for modalities (ModIntro) shows that we can construct a term of type $\langle\ \mu\ \mid\ T\ \rangle$ whenever we have a term of type T in the context locked with μ.

[1] Available at https://github.com/JorisCeulemans/sikkel/releases/tag/v1.0.

[2] So just like MTT, MSTT does not support substructural modal systems such as [17] or [1].

[3] The strings are resolved to de Bruijn indices when going from the upper to the middle layer in Fig. 1. MSTT does not specify an equational theory for terms or reduction of terms.

Fig. 2 shows a simple example of a modal program written in Sikkel (the symbol · is MSTT function application). It has type $\langle\ \mu \mid T \Rrightarrow S\ \rangle \Rrightarrow \langle\ \mu \mid T\ \rangle \Rrightarrow \langle\ \mu \mid S\ \rangle$ and proves that every modality satisfies the K axiom (or, in other words, is an applicative functor).

```
applicative : ModalityExpr m n → TmExpr n
applicative μ = lam[ μ | "f" ∈ T ⇛ S ]
  lam[ μ | "t" ∈ T ] mod⟨ μ ⟩ (svar "f" · svar "t")
```

Figure 2: Example of a modal program in Sikkel.

Presheaf Models & Sound Type-checker MSTT does not have interesting computational behavior in the sense that there is no reduction for terms. Sikkel's intended use is that certain MSTT terms will be interpreted as Agda terms. However, in an off-the-shelf version of Agda this is not immediately possible for many of the modal type and term formers. For this reason, there is a middle layer in Fig. 1, representing a formalization of presheaf models of type theory in Agda, which is essentially a shallow embedding of MSTT [23]. A presheaf model is parametrized by a base category, different modes will correspond to different base categories and they determine which new type and term formers are implementable in the model. Modalities are interpreted as dependent right adjunctions (DRAs) [6] and allow to transfer semantic types and terms between different base categories. Our formalization follows the general construction by Hofmann [15] and is structured as an internal Category with Families (CwF) [12]. Although MSTT is simply-typed, Sikkel's semantic layer already anticipates the addition of dependent types and semantic types may depend on variables.

The bridge between Sikkel's first two layers is a type checker which is sound by construction in the sense that it will either refute a judgment or accept it *and* interpret it in the presheaf model. In order for this to be possible, a Sikkel user implementing a new type theory must provide interpretations of modes as base categories, modalities as DRAs and two-cells as certain natural transformations. Furthermore, if any non-modal type or term formers are added, the user must specify how they should be type-checked and interpreted.

Extraction The presheaf model over the trivial base category corresponds to the standard set model of type theory. We make use of this fact to extract semantic terms in this model to the meta-level (i.e. to Agda terms), as shown at the bottom of Fig. 1.[4] However, the interpretation of some types (e.g. function types) in the model is only *isomorphic* to their intended meaning. This is why Sikkel provides a type class Extractable for semantic types, instances of which declare the intended translated Agda type and provide a way to apply the isomorphism.

Applications We implemented two applications in Sikkel: guarded recursive type theory and a restricted form of parametricity. With guarded recursion, we can write definitions of infinite streams that would not be accepted by Agda's termination checker. The extraction mechanism allows us to interpret these definitions as ordinary Agda streams. For parametricity, we demonstrate how a function definition can be interpreted with two different representations for an abstract type, and how parametricity allows us to relate the two resulting definitions.

Future Work Eventually, we plan to extend Sikkel with support for dependent types. However, the interpretation of a dependently typed syntax turns out to be a significant challenge (especially to satisfy Agda's termination checker). Furthermore, the addition of a Hofmann-Streicher universe to our presheaf model is non-trivial. Therefore, we first intend to equip Sikkel with an extensible program logic (in the same spirit as the Edinburgh Logical Framework [14]), orthogonal to the different layers in Fig. 1, which will allow a programmer to prove properties of functions written in Sikkel and translate the proofs to Agda proofs of the extracted Agda functions. We also plan to study more examples of multimode type theories as applications of Sikkel, with an ongoing exploration of nominal type theory [20].

[4] Semantic terms over other base categories must first be transferred by applying a modality.

Acknowledgements Joris Ceulemans and Andreas Nuyts hold a PhD Fellowship and a Postdoctoral Fellowship, respectively, from the Research Foundation – Flanders (FWO). This work was partially supported by a research project of the Research Foundation - Flanders (FWO).

References

[1] Andreas Abel and Jean-Philippe Bernardy. A unified view of modalities in type systems. *Proc. ACM Program. Lang.*, 4(ICFP), aug 2020. doi:10.1145/3408972.

[2] Robert Atkey. Relational parametricity for higher kinds. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46–61, 2012. doi:10.4230/LIPIcs.CSL.2012.46.

[3] Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In *Principles of Programming Languages*, 2014. doi:10.1145/2535838.2535852.

[4] Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. A presheaf model of parametric type theory. *Electron. Notes in Theor. Comput. Sci.*, 319:67 – 82, 2015. doi:10.1016/j.entcs.2015.12.006.

[5] Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, Dagstuhl, Germany, 2014. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4628, doi:10.4230/LIPIcs.TYPES.2013.107.

[6] Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science*, 30(2):118–138, 2020. doi:10.1017/S0960129519000197.

[7] Ales Bizjak and Rasmus Ejlers Møgelberg. Denotational semantics for guarded dependent type theory. *Math. Struct. Comput. Sci.*, 30(4):342–378, 2020. doi:10.1017/S0960129520000080.

[8] Evan Cavallo and Robert Harper. Internal parametricity for cubical type theory. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, pages 13:1–13:17, 2020. doi:10.4230/LIPIcs.CSL.2020.13.

[9] Evan Cavallo, Anders Mörtberg, and Andrew W Swan. Unifying cubical models of univalent type theory. In Maribel Fernández and Anca Muscholl, editors, *Computer Science Logic (CSL 2020)*, volume 152 of *LIPIcs*, pages 14:1–14:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2020/11657, doi:10.4230/LIPIcs.CSL.2020.14.

[10] Joris Ceulemans, Andreas Nuyts, and Dominique Devriese. Sikkel: Multimode simple type theory as an Agda library. In *Proceedings of the Ninth Workshop on Mathematically Structured Functional Programming*, pages 93–112, Munich, Germany, 2022. URL: http://eptcs.web.cse.unsw.edu.au/paper.cgi?MSFP2022.5.

[11] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: A constructive interpretation of the univalence axiom. *FLAP*, 4(10):3127–3170, 2017. URL: http://www.cse.chalmers.se/~simonhu/papers/cubicaltt.pdf.

[12] Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs*, pages 120–134, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. doi:10.1007/3-540-61780-9_66.

[13] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. *Logical Methods in Computer Science*, Volume 17, Issue 3, July 2021. URL: https://lmcs.episciences.org/7713, doi:10.46298/lmcs-17(3:11)2021.

[14] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, jan 1993. doi:10.1145/138027.138060.

[15] Martin Hofmann. *Syntax and Semantics of Dependent Types*, chapter 4, pages 79–130. Cambridge University Press, 1997.

[16] Daniel R. Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science - International Symposium, LFCS 2016, Deerfield Beach, FL, USA, January 4-7, 2016. Proceedings*, volume 9537 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2016. doi:10.1007/978-3-319-27683-0_16.

[17] Daniel R. Licata, Michael Shulman, and Mitchell Riley. A fibrational framework for substructural and modal logics. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:22, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7740, doi:10.4230/LIPIcs.FSCD.2017.25.

[18] Andreas Nuyts and Dominique Devriese. Degrees of relatedness: A unified framework for parametricity, irrelevance, ad hoc polymorphism, intersections, unions and algebra in dependent type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 779–788, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209119.

[19] Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. Parametric quantifiers for dependent type theory. *Proc. ACM Program. Lang.*, 1(ICFP), August 2017. doi:10.1145/3110276.

[20] Andrew M. Pitts, Justus Matthiesen, and Jasper Derikx. A dependent type theory with abstractable names. *Electronic Notes in Theoretical Computer Science*, 312:19 – 50, 2015. Ninth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2014). URL: http://www.sciencedirect.com/science/article/pii/S1571066115000079, doi:10.1016/j.entcs.2015.04.003.

[21] E. Riehl and M. Shulman. A type theory for synthetic ∞-categories. *ArXiv e-prints*, May 2017. arXiv:1705.07442.

[22] Matthew Z. Weaver and Daniel R. Licata. A constructive model of directed univalence in bicubical sets. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 915–928. ACM, 2020. doi:10.1145/3373718.3394794.

[23] Martin Wildmoser and Tobias Nipkow. Certifying machine code safety: Shallow versus deep embedding. In *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, pages 305–320, Berlin, Heidelberg, 2004. Springer. doi:10.1007/978-3-540-30142-4_22.