Contextual Algebraic Theories: Generic Boilerplate beyond Abstraction (Extended Abstract)

Andreas Nuyts
imec-DistriNet, KU Leuven
Belgium

Abstract

Menkar, a proof-assistant for multimodal dependent type theory (MTT), got stuck among other things under the weight of boilerplate code. Allais et al. [1] have developed a very effective generic programming technique for dealing with boilerplate in second-order multisorted algebraic theories (SOMATs) – simple type systems where contexts are lists of types – and Fiore and Szamozvancev [12] provide a categorical/algebraic foundation for this technique. We generalize SOMATs as far as our imagination and the techniques used allow, and propose contextual multisorted algebraic theories (CMATs) as a central concept in a generic programming technique which aims to also support multimodal simple type theory (MSTT), dual-context systems and systems with context exponentiation and an amazing right adjoint.

1 Motivation

1.1 What makes a mature type system

If developers of type systems and functional programming languages have any hopes for their languages to be adopted, they will find themselves trying to check off as many of the following properties and features as possible:

Admissibility of substitution Substitution of well-typed terms should be definable as a metatheoretic operation.

Decidability of typing With the right amount of user-provided annotations, checking that a program is well-typed should be automatizable.

Soundness The empty type should be uninhabited. This proves the unprovability of contradictions in dependently typed systems, but is also a basic sanity check for simple type systems. Soundness can be proven with a model in which the empty type is indeed empty.

Canonicity We can prove similar sanity properties for other types than the empty one, e.g. that every closed boolean is equal to either true or false. Canonicity is often proved using a gluing argument [11, 14, 17].

Decidability of equality The conversion rule of type theory, which allows us to cast $a : A$ to $a : B$ without further ado if $A = B$, implies that we cannot have a type-checking algorithm without an algorithm for checking type equality. In dependent type-systems, checking the type equality $P(t_1) \stackrel{?}{=} P(t_2)$ relies in turn on an algorithm for checking equality of (open) terms $t_1 \stackrel{?}{=} t_2$.

Normalization One way of deciding equality is by comparing both hands' normal forms for syntactic equality (or α-equivalence). In the case of purely functional programs, normalization is also what will the programs result. An a priori unsafe normalization algorithm can be implemented straightforwardly by evaluating in a value model [1, §7.7]; termination of the algorithm can be proven again using a gluing argument [3, 11, 13, 17].

Interpretation If a language has side-effects, then we do not only want to normalize but also run the program.

Compilation Instead of directly running a program, we can instead compile it to an existing language.

Desugaring This approach is especially appealing if the existing language is a subset of the current one.

Pretty-printing For human-readable error messages.

Some of the listed items, such as soundness, are entirely in the domain of metatheory and are often carried out solely on paper. Others, such as substitution or a type-checker, are essential parts of an implementation. Normalization, which may be proven by gluing a presheaf model over renamings [3, 11, 13, 17], is heavy on the theoretical side but can also play an essential role in the code base of an implementation [10, 16]. What all items have in common is that they can be formalized using a dependently typed proof assistant as a metatheory, and that doing so involves great amounts of tedious boilerplate.

1.2 Weeding boilerplate

During the implementation of Menkar [25, 27], a presently incomplete proof-assistant for multimodal dependent type theory (MTT [15]), I have become keenly aware of the boilerplate issue. Menkar contains a scope-checker, a type-checker, an equality-checker, a metavariable solver and a weak-head-normalizer, which all manually traverse all possible syntax constructors, leading to a matrix of tedious code indexed by a program component and a syntax constructor,[1] and small changes often affect an entire row or column of this matrix.

Allais et al. [1] greatly alleviate this issue for a class of non-modal simply-typed languages by using generic programming. They define a universe in which these languages can

Conference'17, July 2017, Washington, DC, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] Renaming and substitution were implemented using generics [24].

be encoded and, by induction on that universe, implement in a very economical way: a renaming and a substitution operation, raw syntax and a scope-checker, a bidirectional type-checker/inferencer with elaboration for the STLC, an unsafe normalization algorithm, a desugaring operation for let-expressions and a pretty-printer. Where possible, these components are almost always implemented generically.

Fiore and Szamozvancev [12] clarify the categorical foundations of this approach. First of all, they identify the languages encoded in Allais et al.'s inductive universe as a class of algebraic theories which they leave unnamed but which could be rightfully called free second-order multisorted algebraic theories (free SOMATs); free in the sense that they lack an equational theory. Knowing what the languages are, makes it easier to understand what are their models. Here, we emphasize that a *model* in the most general meaning of the word is just an algebra whose structure gives meaning to all syntax constructors of the language, giving rise by recursion to a unique structure-preserving interpretation map from the syntax. In fact, Fiore and Szamozvancev associate two categories of models to a single free SOMAT. One flavour of models, which we shall call **cold** models,[2] views substitution as a metatheoretic operation – i.e. not part of the language – that needs not be modelled. **Hot** models[3] on the contrary do require a semantics for substitution and provide the substitution lemma for free. Both categories have the same initial object (the same syntax), which is one way to state formally that substitution is admissible.

Allais et al.'s generic notion of *semantics* only covers cold models and in fact only very special ones involving a presheaf over renamings $\mathcal{P}$ of what they call 'values' and a non-presheaf $\mathcal{A}$ of 'computations' [12, lemma 3.2]. Terms are then modelled as functions sending value environments to computations. It is quite neat that Fiore and Szamozvancev fit Allais et al.'s work in a mathematically elegant framework whose hot models should also encompass state-of-the-art soundness and parametric models as well as glued models for canonicity and normalization.

1.3 Taking it beyond abstraction

The free SOMAT framework described above is not directly usable as a foundation for the metatheory and implementation of MTT for two reasons: (1) it is simply typed and (2) it assumes that contexts (substitutions) are lists of types (tuples of terms), so that the only context extension they can deal with is the addition of non-modal typed variables as induced by λ-abstraction. With the current work, we seek to address (2), so that we can deal with the modal introduction rule of multimodal *simple* type theory (MSTT [9]):

$$\frac{\Gamma, \text{🔒}_\mu \vdash t : T \ @\ p \qquad \mu : p \to q}{\Gamma \vdash \mathsf{mod}_\mu\, t : \langle \mu \mid T \rangle \ @\ q}$$

This rule creates for any modality μ from mode p to mode q a term of modal type $\langle \mu \mid T \rangle$ at mode q from a term of type T at mode p, where the left adjoint 🔒_μ of the modality has been applied to the context; the rule is typically modelled as the transposition operation of the dependent right adjoint (DRA) [5] that models μ. Another application of interest is a direct typing rule for the 'amazing' right adjoint $\surd$ [21] to exponentiation over a shape type (such as the path interval $\mathbb{I}$) which has applications in homotopy type theory (HoTT [31]), and ultimately its dependent generalization given by the transpension type ◊ [28] which is also relevant to internal parametricity [4, 8] and nominal type theory [30]:

$$\frac{\mathbb{I} \to \Gamma \vdash t : T}{\Gamma \vdash t^\sharp : \mathbb{I} \surd T} \qquad \frac{\Gamma, \forall (i : \mathbb{I}).\Delta \vdash t : T}{\Gamma, i : \mathbb{I}, \Delta \vdash \mathsf{merid}\, i\, t : \text{◊}\, i.T}$$

Whereas the lock operation from MSTT was still a mere context *constructor*, the introduction rule of $\surd$ really applies a *functorial operation* to the context and in the dependent case we even get a functorial operation that (just like context extension with a dependent type) is not defined on all contexts but is defined on slices over the current one.

2 Contextual Algebraic Theories

In order to encompass also modal languages such as MSTT and the amazing right adjoint, we will widen the collection of supported languages from second-order to *contextual* multisorted algebraic theories (CMATs). In this section, we discuss plain MATs, SOMATs and CMATs. We let 'XMAT' range over these three options, which have much in common.

Similar to how we fix a field before speaking about vector spaces, we will fix a set Sort of sorts before speaking about XMATs. Sorts take the role of *types* for SOMATs and more generally of *right-hand-sides* (of a judgement) for CMATs. A *free* XMAT is given by:

- For each sort σ, a set of operations $\mathrm{Op}(\sigma)$. Here, σ stands for the operation's *output* sort.
- For each operation $o \in \mathrm{Op}(\sigma)$, an *arity* $\mathrm{ar}(o)$, which is a list. The length of the list is the number of arguments the operator will take. Each element of the list is:
  (MAT) a sort, namely the sort of the argument taken,
  (SOMAT) a pair $(\bar\rho, \tau)$, where $\tau \in \mathrm{Sort}$ is the sort of the argument and $\bar\rho \in \mathrm{List\ Sort}$ is the list of sorts of all variables bound in the argument,
  (CMAT) a pair (Φ, τ), where $\tau \in \mathrm{Sort}$ is the sort of the argument and Φ is a *junctor* (see below).

2.1 Plain MATs

So a free MAT is really just a Sort-indexed container [2], giving rise to a *syntax functor* $F : \mathrm{Set}^{\mathrm{Sort}} \to \mathrm{Set}^{\mathrm{Sort}}$ on sort-indexed sets:

$$(FX)(\sigma) = \Sigma(o \in \mathrm{Op}(\sigma)).\Pi\, i.X(\mathrm{ar}(o)_i)$$

[2] They call these meta-algebras, after the possibility to also deal with metavariables, which is completely irrelevant in the current discussion.
[3] They call these Σ-monoids, where Σ specifies the SOMAT and being a monoid means modelling substitution.

The free monad F^* over this functor, which satisfies $(F^*X)(\sigma) \cong X(\sigma) \uplus (FF^*X)(\sigma)$, sends an indexed set X to the indexed set F^*X of terms of the language with metavariables taken from X. Indeed, the above fixpoint equation can be read as 'a term of sort σ is either a metavariable, or an operator applied to a tuple of appropriately sorted terms'. In particular, $F^*\varnothing$ is the syntax without metavariables.

The category of models $\mathcal{M}$ of the language is the category of algebras of the functor F, which are indexed sets A equipped with indexed functions $FA \Rightarrow A$ assigning meaning to each of the operations; or (isomorphically) the category of monad-algebras of the monad F^*, which are algebras of F^* respecting monadic unit and multiplication.

We can extend a free MAT with equality axioms, expressed as pairs of terms with metavariables $t_{1,2} \in (F^*X)(\sigma)$. Quotienting out the equational theory thus generated, yields a monad M such that MX is the indexed set of terms with metavariables from X, up to the equational theory. The models of this non-free MAT are the monad-algebras of M.

2.2 Second-order MATs (SOMATs)

To a free SOMAT we associate a cold and a hot syntax functor[4] on sets indexed not by sorts but by *judgements* $(\bar\gamma \vdash \sigma) \in \mathsf{Jud}$ consisting of a context $\bar\gamma$ (list of sorts) and a sort σ. The cold syntax functor $F_\mathrm{c} : \mathsf{Set}^{\mathsf{Jud}} \to \mathsf{Set}^{\mathsf{Jud}}$ is:

$$(F_\mathrm{c}X)(\bar\gamma \vdash \sigma) = \Sigma(o \in \mathrm{Op}(\sigma)).\Pi\, i.X(\bar\gamma +\!\!+ \bar\rho_i \vdash \tau_i),$$

where $\mathrm{ar}(o)_i = (\bar\rho_i, \tau_i)$. The algebras of this functor are the cold models from section 1.2.

The hot syntax functor F_h on $\mathsf{Set}^{\mathsf{Jud}}$ is similar but adds a substitution operation. Even though the SOMAT is free, we can generically impose an equational theory on F_h^* by asking that substitution respects identity and composition and commutes with all operations, yielding a monad F_h° whose monad-algebras are the hot models.

A free SOMAT can be extended with equality axioms, expressed as pairs of terms $t_{1,2} \in (F_\mathrm{h}^*X)([] \vdash \sigma)$. These are expressed in the hot syntax so they can refer to substitution (like the β-rule for functions in the STLC) and in the empty context so that they are meaningful in all contexts.

2.3 Contextual MATs (CMATs)

We now drop the assumption that contexts and context extensions are necessarily lists of sorts. Instead, we will not only fix a set of sorts, but also a set Ctx of contexts and a set Jun of *junctors*. These junctors[5] take the role of context extensions but can in fact be any functors on the category of contexts and substitutions. They need not even be endo: we will fix a set Mode of modes, have sets Ctx_m and Sort_m at every mode m and a set $\mathsf{Jun}(m,n)$ for any domain and codomain modes m and n. CMATs will have dedicated sorts $\mathsf{Sub}(\Gamma)$ for substitutions and $\mathsf{JHom}(\Phi,\Psi)$ for junctor morphisms, and a 2-dimensional composition structure. Judgements now take the form $(\Gamma \vdash \sigma \mathbin{@} m)$. The cold syntax functor remains quite similar:

$$(F_\mathrm{c}X)(\Gamma \vdash \sigma \mathbin{@} m) = \Sigma(o \in \mathrm{Op}(\sigma)).\Pi\, i.X(\Gamma.\Phi_i \vdash \tau_i \mathbin{@} n_i),$$

where $\mathrm{ar}(o)_i = (\Phi_i, \tau_i)$ and $\Phi_i \in \mathsf{Jun}(m, n_i)$. The hot syntax functor is similar but again adds a substitution operation, and there are now clearly a number of laws that we want to impose generically. Commutation of substitution with the CMAT operations can be stated generically thanks to the functoriality of junctors. Again, a specific CMAT can be further extended with equality axioms.

3 Current Status and Work Plan

Current status. Currently, we have formalized free MATs, non-free MATs and their categories of models in Agda [26]. In order to quotient out the equational theory, we are using higher inductive types in Agda-cubical [32].

Work plan. The plan is to proceed by formalizing CMATs and implementing/proving a number of the following results:

- Formalize SOMATs and a translation to CMATs,
- Show that we can subsume and generalize Allais et al.'s work [1], including its applications,
- Define MSTT [9] as a CMAT and prove normalization as a hot model; combined with Allais et al.'s results, this effectively amounts to implementing MSTT as a programming language,
- Define a dual-context modal calculus [19, 29] as a CMAT and obtain similar results,
- Define a simple type system with amazing right adjoint $\surd$ [21] as a CMAT and obtain similar results.

Further ahead. The motivating use cases were multimodal *dependent* type theory (MTT [15]) and the transpension type [28]. A dependent version of the current work would be a theory of contextual *generalized* [6, 7] algebraic theories (CGATs). The framework for defining and reasoning about quotient-inductive-inductive-types (QIITs, essentially the same as GATs) using type-theoretic signatures [18, 20] seems a good starting point.

With dependent types, I believe that it may also be possible to support linear and other substructural type systems. In CMATs, junctors have to act on *all* contexts. Hence, it is not possible to remove certain resources from the context as we have no way to know that they will be there. In CGATs, junctors would have to act only on slices over the current context, where we do know that they provide these resources.

Non-instances. Type systems such as adjoint logic [22] and LSR substructural type theory [23] are not ready to be defined as either CMATs or CGATs precisely because they do not work with 'left adjoint' operations in the way MTT does: there is no most general context, functorially obtained, in which we can type-check a subterm.

[4] In fact, these can be obtained by first translating the SOMAT to a MAT.
[5] 'Junctor' is Latin for 'binder' and sounds a lot like 'functor'.

References

[1] Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. 2021. A type- and scope-safe universe of syntaxes with binding: their semantics and proofs. *J. Funct. Program.* 31 (2021), e22. https://doi.org/10.1017/S0956796820000076

[2] Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. 2015. Indexed containers. *Journal of Functional Programming* 25 (2015), e5. https://doi.org/10.1017/S095679681500009X

[3] Thorsten Altenkirch and Ambrus Kaposi. 2017. Normalisation by Evaluation for Type Theory, in Type Theory. *Log. Methods Comput. Sci.* 13, 4 (2017). https://doi.org/10.23638/LMCS-13(4:1)2017

[4] Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. 2015. A Presheaf Model of Parametric Type Theory. *Electron. Notes in Theor. Comput. Sci.* 319 (2015), 67 – 82. https://doi.org/10.1016/j.entcs.2015.12.006

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

[6] John Cartmell. 1978. *Generalised Algebraic Theories and Contextual Categories.* Ph.D. Dissertation.

[7] John Cartmell. 1986. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Logic* 32 (1986), 209–243. https://doi.org/10.1016/0168-0072(86)90053-9

[8] Evan Cavallo and Robert Harper. 2020. Internal Parametricity for Cubical Type Theory. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain.* 13:1–13:17. https://doi.org/10.4230/LIPIcs.CSL.2020.13

[9] Joris Ceulemans, Andreas Nuyts, and Dominique Devriese. 2022. Sikkel: Multimode Simple Type Theory as an Agda Library. In *MSFP.* http://eptcs.web.cse.unsw.edu.au/paper.cgi?MSFP2022.5.pdf

[10] Thierry Coquand. 1996. An algorithm for type-checking dependent types. *Science of Computer Programming* 26, 1 (1996), 167–177. https://doi.org/10.1016/0167-6423(95)00021-6

[11] Thierry Coquand. 2018. *Canonicity and normalization for Dependent Type Theory.* arXiv:1810.09367 https://arxiv.org/abs/1810.09367

[12] Marcelo Fiore and Dmitrij Szamozvancev. 2022. Formal metatheory of second-order abstract syntax. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–29. https://doi.org/10.1145/3498715

[13] Daniel Gratzer. 2021. Normalization for multimodal type theory. *CoRR* abs/2106.01414 (2021). arXiv:2106.01414 https://arxiv.org/abs/2106.01414

[14] Daniel Gratzer, Alex Kavvos, Andreas Nuyts, and Lars Birkedal. 2020. Type Theory à la Mode. (2020). https://anuyts.github.io/files/mtt-techreport.pdf Technical report.

[15] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. 2021. Multimodal Dependent Type Theory. *Logical Methods in Computer Science* Volume 17, Issue 3 (July 2021). https://doi.org/10.46298/lmcs-17(3:11)2021

[16] Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. 2019. Implementing a Modal Dependent Type Theory. *Proc. ACM Program. Lang.*, Article 107 (2019), 29 pages. https://doi.org/10.1145/3341711

[17] Ambrus Kaposi, Simon Huber, and Christian Sattler. 2019. Gluing for Type Theory. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany (LIPIcs, Vol. 131)*, Herman Geuvers (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 25:1–25:19. https://doi.org/10.4230/LIPIcs.FSCD.2019.25

[18] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. 2019. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.* 3, POPL (2019), 2:1–2:24. https://doi.org/10.1145/3290315

[19] G. A. Kavvos. 2020. Dual-Context Calculi for Modal Logic. *Log. Methods Comput. Sci.* 16, 3 (2020). https://lmcs.episciences.org/6722

[20] András Kovács. 2022. *Type-Theoretic Signatures for Algebraic Theories and Inductive Types.* Ph.D. Dissertation. Eötvös Loránd University, Budapest, Hungary. https://andraskovacs.github.io/pdfs/phdthesis_compact.pdf

[21] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. 2018. Internal Universes in Models of Homotopy Type Theory. In *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK.* 22:1–22:17. https://doi.org/10.4230/LIPIcs.FSCD.2018.22

[22] Daniel R. Licata and Michael Shulman. 2016. *Adjoint Logic with a 2-Category of Modes.* Springer International Publishing, 219–235. https://doi.org/10.1007/978-3-319-27683-0_16

[23] Daniel R. Licata, Michael Shulman, and Mitchell Riley. 2017. A Fibrational Framework for Substructural and Modal Logics. In *FSCD '17.* 25:1–25:22. https://doi.org/10.4230/LIPIcs.FSCD.2017.25

[24] José Pedro Magalhães, Atze Dijkstra, Johan Jeuring, and Andres Löh. 2010. A generic deriving mechanism for Haskell. In *Proceedings of the 3rd ACM SIGPLAN Symposium on Haskell, Haskell 2010, Baltimore, MD, USA, 30 September 2010*, Jeremy Gibbons (Ed.). ACM, 37–48. https://doi.org/10.1145/1863523.1863529

[25] Andreas Nuyts. 2019. Menkar. https://github.com/anuyts/menkar/. GitHub repository.

[26] Andreas Nuyts. 2022. Algebraic theories with contexts, in cubical Agda. https://github.com/anuyts/ctx-alg/ GitHub repository.

[27] Andreas Nuyts and Dominique Devriese. 2019. Menkar: Towards a Multimode Presheaf Proof Assistant. In *TYPES.*

[28] Andreas Nuyts and Dominique Devriese. 2021. Transpension: The Right Adjoint to the Pi-type. *CoRR* abs/2008.08533 (2021). arXiv:2008.08533 https://arxiv.org/abs/2008.08533

[29] Frank Pfenning and Rowan Davies. 2001. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science* 11, 4 (2001), 511–540. https://doi.org/10.1017/S0960129501003322

[30] Andrew M. Pitts, Justus Matthiesen, and Jasper Derikx. 2015. A Dependent Type Theory with Abstractable Names. *Electronic Notes in Theoretical Computer Science* 312 (2015), 19 – 50. https://doi.org/10.1016/j.entcs.2015.04.003 Ninth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2014).

[31] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics.* http://homotopytypetheory.org/book, IAS.

[32] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2019. Cubical Agda: a dependently typed programming language with univalence and higher inductive types. *PACMPL* 3, ICFP (2019), 87:1–87:29. https://doi.org/10.1145/3341691