Strict syntax of type theory via alpha-normalisation

Viktor Bense, Ambrus Kaposi, and Szumi Xie

Eötvös Loránd University, Budapest, Hungary, {qils07|akaposi|szumi}@inf.elte.hu

The main difficulty of using the well-typed quotiented syntax of type theory in formalisations is the so-called transport hell: the equality $(\mathsf{app}\,t\,u)[\gamma] = \mathsf{app}\,(t[\gamma])\,(u[\gamma])$ does not make sense because $t[\gamma]$ is not of a function type, but a substituted type $(A \Rightarrow B)[\gamma]$, and we need another equation on types (namely $(A \Rightarrow B)[\gamma] = (A[\gamma]) \Rightarrow (B[\gamma])$) to make it well-typed. Hence a transport will appear on the subterm $(t[\gamma])$, and it makes it difficult to use this equation: whenever we want to use it, we need to make sure that the transport is in the right place: we need to apply several equations about moving the transport in and out of the term (e.g. if $A = B = \mathbb{N}$, these equations imply that all transports disappear). Workarounds for this problem include the following.

(i) We do not use well-typed quotiented syntax, only unquotiented (but maybe well-scoped) syntax: now we can define substitution recursively on the preterms and we prove separately that it preserves typing, and so on; the most complete formalisation of normalisation proofs for type theory use this technique [2, 1]: the level of abstraction is low, hence the construction is very tedious, but with some proof automation it is not that bad.

(ii) We avoid indexing terms by their types, that is, we use natural models [5] or contextual categories (as in the formalisation of the initiality conjecture by Brunerie and de Boer [7]); another way to describe this approach is to move from the generalised algebraic [8] presentation of type theory towards its essentially algebraic [11] presentation; this makes it harder to read the definitions, we need more operations and equations, but all the transports disappear.

(iii) We make the well-typed quotiented syntax stricter using some dirty hacks such as shallow embedding [14] or rewrite rules [10]: now both the equalities $(A \Rightarrow B)[\gamma] = (A[\gamma]) \Rightarrow (B[\gamma])$ and $(\mathsf{app}\,t\,u)[\gamma] = \mathsf{app}\,(t[\gamma])\,(u[\gamma])$ are definitional, so there are no transports. These methods are good to computer check pen-and-paper proofs (as done for canonicity in [14]), but this does not provide an implementation. If the metatheory is intensional type theory, then the normalisation proof also gives a normalisation algorithm, but then we do not have access to the above dirty hacks.

(iv) Work in the internal setting of higher-order abstract syntax [6]: substitutions are modelled by metatheoretic function space, so all equations on substitutions are definitional. But the proof is in an internal language, and a separate (metatheoretic) step is needed to turn it into a proof about the real syntax (by which we mean the initial model).

(v) We can bite the bullet and fight through transport hell, resulting in formalisations with lots of transport boilerplate, e.g. [3, 4].

In this talk we propose another workaround which is an improved version of the quotient-inductive-inductive-recursive type approach of [12]. We have partial implementations of the method described below for simple type theory and type theory in (Cubical) Agda.

Just like in (iii), we make the syntax stricter, but now we don't extend the metatheory, we use methods available inside ordinary intensional type theory. The (weak) syntax is a quotient inductive-inductive type [13] definable in Cubical Agda [15]. We define α-normal forms for types and terms as types and terms that do not include substitutions. For example, for a type theory with Π types and U, α-normal forms are given by the following inductive families (we don't write universe indices for readability;

both families are propositionally truncated using equality constructors). α-normal forms are indexed by context, types and terms of the weak syntax.

$$
\begin{aligned}
&\mathsf{NTy}^\alpha &&: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{hProp}\\
&\mathsf{Nf}^\alpha &&: (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\,\Gamma) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{hProp}\\
&\Pi^\alpha &&: \mathsf{NTy}^\alpha\,\Gamma\,A \to \mathsf{NTy}^\alpha\,(\Gamma \triangleright A)\,B \to \mathsf{NTy}^\alpha\,\Gamma\,(\Pi\,A\,B)\\
&\mathsf{U}^\alpha &&: \mathsf{NTy}^\alpha\,\Gamma\,\mathsf{U}\\
&\mathsf{El}^\alpha &&: \mathsf{Nf}^\alpha\,\Gamma\,\mathsf{U}\,a \to \mathsf{NTy}^\alpha\,\Gamma\,(\mathsf{El}\,a)\\
&\mathsf{lam}^\alpha &&: \mathsf{Nf}^\alpha\,(\Gamma \triangleright A)\,B\,b \to \mathsf{Nf}^\alpha\,\Gamma\,(\Pi\,A\,B)\,(\mathsf{lam}\,b)\\
&\mathsf{app}^\alpha &&: \mathsf{Nf}^\alpha\,\Gamma\,(\Pi\,A\,B)\,t \to \mathsf{Nf}^\alpha\,\Gamma\,A\,a \to \mathsf{Nf}^\alpha\,\Gamma\,(B[\mathsf{id},a])\,(\mathsf{app}\,t\,a)\\
&\mathsf{c}^\alpha &&: \mathsf{NTy}^\alpha\,\Gamma\,A \to \mathsf{Nf}^\alpha\,\Gamma\,\mathsf{U}\,(\mathsf{c}\,A)
\end{aligned}
$$

For NTy^α and Nf^α, we define weakening and substitution with α-normal terms, this is done by recursion on α-normal forms. With the help of these, by induction on α-normal forms, we prove α-normalisation: we obtain elements of $\mathsf{isContr}\,(\mathsf{NTy}^\alpha\,\Gamma\,A)$ and $\mathsf{isContr}\,(\mathsf{Nf}^\alpha\,\Gamma\,A\,a)$ for any A and a. Note that α-normal forms are propositionally truncated, so they cannot distinguish equal terms (e.g. $\mathsf{app}\,(\mathsf{lam}\,t)\,a$ and $t[\mathsf{id},a]$). However, knowing that all terms have α-normal forms, we can redefine weakening and substitution of terms by induction on α-normal forms: as they result in singletons, we are allowed to eliminate from the propositionally truncated types (in other words, we use unique choice; Wk is the family of weakenings, NSb is the family of α-normal substitutions).

$$
\begin{aligned}
&-[-]^{\mathsf{wk}} : \mathsf{NTy}^\alpha\,\Gamma\,A \to \mathsf{Wk}\,\Delta\,\Gamma\,\gamma \to (A' : \mathsf{Ty}\,\Gamma) \times (A' = A[\gamma])\\
&-[-]^{\mathsf{wk}} : \mathsf{Nf}^\alpha\,\Gamma\,A\,a \to \mathsf{Wk}\,\Delta\,\Gamma\,\gamma \to (a' : \mathsf{Tm}\,\Gamma\,A) \times (a' = a[\gamma])\\
&-[-]^{\mathsf{sb}} : \mathsf{NTy}^\alpha\,\Gamma\,A \to \mathsf{NSb}\,\Delta\,\Gamma\,\gamma \to (A' : \mathsf{Ty}\,\Gamma) \times (A' = A[\gamma])\\
&-[-]^{\mathsf{sb}} : \mathsf{Nf}^\alpha\,\Gamma\,A\,a \to \mathsf{NSb}\,\Delta\,\Gamma\,\gamma \to (a' : \mathsf{Tm}\,\Gamma\,A) \times (a' = a[\gamma])
\end{aligned}
$$

Now we define a new model of type theory where all components are syntactic, but substitution is defined using the above defined $-[-]^{\mathsf{sb}}$ functions. This model is isomorphic to the syntax (as witnessed by the equalities in the type of $-[-]^{\mathsf{sb}}$), but the equations about substitutions (such as $(A \Rightarrow B)[\gamma] = (A[\gamma]) \Rightarrow (B[\gamma])$ and $(\mathsf{app}\,t\,u)[\gamma] = \mathsf{app}\,(t[\gamma])\,(u[\gamma])$) hold by definition. To be completely precise, we do not use a category with families (CwF [9]) based notion of type theory, but a single substitution based one where we have separate single weakening and single substitution operations: this allows us to use the above technique to strictify the substitution rules for binders and all the rules for variables. We were not able to obtain strictification of the analogous rules using a parallel substitution (CwF) based notion of type theory. All the rules of CwFs are admissible in the single substitution syntax, but as the single substitution calculus is smaller, induction on it needs fewer methods, and as several equalities are definitional in the strict syntax, there is less transport hell when defining these methods.

We formalised α-normalisation for a dependent type theory in Agda[1] and derived all the rules for CwFs using a postulated weak syntax. In Cubical Agda, we have a formalisation[2] of simple type theory using single substitution, we proved α-normalisation, and defined a strict syntax where all rules about substitutions are definitional. We are currently working on showcasing how the strict syntax simplifies a canonicity proof for simple type theory, and we plan to redo the same for dependent types. We hope that formalising normalisation for a syntax with strict substitution rules will be significantly less work than fighting transport hell directly (option (v)).

[1]https://bitbucket.org/akaposi/single

[2]https://bitbucket.org/akaposi/qiirt/src/master/STT-SSC-cubical

References

[1] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL):23:1–23:29, 2018. doi:10.1145/3158111.

[2] Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. Martin-Löf à la Coq. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, pages 230–245. ACM, 2024. doi:10.1145/3636501.3636951.

[3] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.

[4] Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for type theory, in type theory. *Log. Methods Comput. Sci.*, 13(4), 2017. doi:10.23638/LMCS-13(4:1)2017.

[5] Steve Awodey. Natural models of homotopy type theory. *Math. Struct. Comput. Sci.*, 28(2):241–286, 2018. doi:10.1017/S0960129516000268.

[6] Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. For the metatheory of type theory, internal sconing is enough. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 18:1–18:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.FSCD.2023.18, doi:10.4230/LIPICS.FSCD.2023.18.

[7] Guillaume Brunerie and Menno de Boer. Formalization of the initiality conjecture, 2020. URL: https://github.com/guillaumebrunerie/initiality.

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

[9] Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019. URL: http://arxiv.org/abs/1904.00827, arXiv:1904.00827.

[10] Jesper Cockx. Type theory unchained: Extending agda with user-defined rewrite rules. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway*, volume 175 of *LIPIcs*, pages 2:1–2:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.TYPES.2019.2, doi:10.4230/LIPICS.TYPES.2019.2.

[11] Peter Freyd. Aspects of topoi. *Bulletin of the Australian Mathematical Society*, 7(1):1–76, 1972. doi:10.1017/S0004972700044828.

[12] Ambrus Kaposi. Towards quotient inductive-inductive-recursive types. In Eduardo Hermo Reyes and Alicia Villanueva, editors, *29th International Conference on Types for Proofs and Programs (TYPES 2023)*. Valencia, 2023. URL: https://types2023.webs.upv.es/TYPES2023.pdf.

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

[14] Ambrus Kaposi, András Kovács, and Nicolai Kraus. Shallow embedding of type theory is morally correct. In Graham Hutton, editor, *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, volume 11825 of *Lecture Notes in Computer Science*, pages 329–365. Springer, 2019. doi:10.1007/978-3-030-33636-3_12.

[15] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: A dependently typed programming language with univalence and higher inductive types. *J. Funct. Program.*, 31:e8, 2021. doi:10.1017/S0956796821000034.