30th International Conference on Types for Proofs and Programs

TYPES 2024 – Abstracts

Patrick Bahr and Rasmus Ejlers Møgelberg (eds.)

Copenhagen, Denmark, 10-14 June 2024

Preface

This volume contains the abstracts of the talks accepted for presentation at the 30th International Conference on Types for Proofs and Programs (TYPES 2024) held on 10-14 June 2024 in Copenhagen.

The TYPES meetings are a forum to present new and ongoing work in all aspects of type theory and its applications, especially in formalized and computer-assisted reasoning and computerprogramming.

The meetings from 1990 to 2008 were annual workshops of a sequence of five EU-funded networking projects. Since 2009, TYPES has been run as an independent conference series.

In response to the call for contributions, 60 abstracts were submitted. Each submission was reviewed by at least 3 program committee members. The committee decided to accept 57 abstracts. One of the accepted abstracts was withdrawn by the authors.

In addition, the conference programme included invited talks by five outstanding speakers: Brigitte Pientka (McGill University, Canada), Talia Ringer (University of Illinois at Urbana-Champaign, USA), Egbert Rijke (University of Ljubljana, Slovenia), Michael Rathjen (University of Leeds, UK), and Nicola Gambino (University of Manchester, UK). The invited talks by Michael Rathjen and Nicola Gambino were part of a special session in memory of Peter Aczel.

May 30, 2024
Copenhagen

Patrick Bahr
Rasmus Ejlers Møgelberg

Contents

Invited talks

Cocon: A Type-Theoretic Framework for Meta-Programming

Brigitte Pientka
McGill University, Montreal, Canada

Abstract

Meta-programming is the art of writing programs that produce or manipulate other programs. This allows programmers to automate error-prone or repetitive tasks, and exploit domain-specific knowledge to customize the generated code. Hence, meta-programming is widely used in a range of technologies: from cryptographic message authentication in secure network protocols to supporting reflection in proof environments such as Lean.

Unfortunately, writing safe meta-programs remains very challenging and sometimes frustrating, as traditionally errors in the generated code are only detected when running it, but not at the time when code is generated. To make it easier to write and maintain meta-programs, tools that allow us to detect errors during code generation – instead of when running the generated code – are essential.

This talk revisits Cocon, a framework for certified meta-programming. Cocon is a Martin-Löf dependent type theory for defining logics and proofs that allows us to represent domain-specific languages (DSL) within the logical framework LF and in addition write recursive programs and proofs about those DSLs [3] using pattern matching. It is a two-level type theory where Martin-Löf type theory sits on top of the logical framework LF and supports a recursor over (contextual) LF objects. As a consequence, we can embed into LF STLC or System F, etc. and then write programs about those encodings using Cocon itself. This means Cocon can serve as a target for compiling meta-programming systems –from compiling meta-programming with STLC to System F. Moreover, Cocon supports writing an evaluator for each of these sub-languages. This also allows us to reflect back our encoded sub-language and evaluate their encodings using Cocon's evaluation strategy.

I will conclude with highlighting more recent research directions and challenges (see [2] and [1]) that build on the core ideas of Cocon and aim to support meta-programming and intensional code analysis directly in System F and Martin-Löf type theory.

References

1 Jason Z. S. Hu and Brigitte Pientka. Layered modal type theory - where meta-programming meets intensional analysis. In *ESOP (1)*, volume 14576 of *Lecture Notes in Computer Science*, pages 52–82. Springer, 2024.

2 Junyoung Jang, Samuel Gélineau, Stefan Monnier, and Brigitte Pientka. Moebius: Metaprogramming using contextual types – the stage where system f can pattern match on itself. *Proc. ACM Program. Lang. (PACMPL)*, (POPL), 2022.

3 Brigitte Pientka, Andreas Abel, Francisco Ferreira, David Thibodeau, and Rebecca Zucchini. A type theory for defining logics and proofs. In *34th IEEE/ ACM Symposium on Logic in Computer Science (LICS'19)*, pages 1–13. IEEE Computer Society, 2019.

Bridging Neural and Symbolic Proof Automation

Talia Ringer

May 8, 2024

Abstract

Proof assistants like Coq, Lean, and Isabelle/HOL have shown great promise for helping people write formal proofs about both programs and mathematics. Proof automation aims to make that easier, whether through full automation or through interactive user assistance. Traditional proof automation based on symbolic AI, logic, and programming languages theory is highly predictable and understandable, but lacks flexibility and is challenging for non-experts to extend. In contrast, neural proof automation based on machine learning and natural language processing can be unpredictable and confusing, but shines in its flexibility and extensibility for non-experts. This talk will describe how these different kinds of proof automation come together to make writing formal proofs easier, and what this means for the future of formal proof.

Concrete Univalent Mathematics

Egbert Rijke

Type theory is a formal system widely used in most modern proof assistants. The language of type is convenient and powerful, and closely matches the way mathematicians express themselves conceptually. There are, however, some phenomena in type theory that some mathematicians are uneasy about. In particular, the identity type has intricacies that many mathematicians rather squash out by assuming axiom K, which asserts that equality is always a proposition. There is, however, much to learn about type theory if we resist our temptation to assume this axiom.

Types naturally come with the structure of a higher groupoid, where the identifications are the 1-cells, the identifications between identifications are 2-cells, and so on. By this observation we can also say that a pointed connected type is a higher group. More precisely, it is the classifying type of a higher group. Indeed, identifications can be concatenated, inverted, there is a unit identification called reflexivity, and these satisfy the laws of a higher group. Here we take the point of view that a higher group should be the space of symmetries of some object. In a pointed connected type, the object of which we consider the symmetries is the base point and the symmetries of that object are its self-identifications. This is an important observation: symmetries are identifications of an object with itself, and pointed connected types are concrete manifestations of (higher) groups as spaces of symmetry.

From this point of view we can develop all of group theory and higher group theory. A group action is simply a type family over the classifying type of the group. Group homomorphisms are simply pointed maps, and so on. In my lecture I will showcase how to interpret some of the most important concepts of group theory, with concrete examples.

On relating type theories and set theories

Michael Rathjen

A major turning point in constructive mathematics was Bishop's publication of *Foundations of Constructive Analysis* in 1967. The early 1970s saw the development of several foundational frameworks for constructive mathematics that were to no small extent galvanized by Bishop's work, notably Myhill's *Constructive Set Theory*, Feferman's *Explicit Mathematics* and Martin-Löf's *Type Theory*. Myhill wanted to single out principles undergirding Bishop's mathematics with the additional aim of making "*the process of formalization completely trivial, as it is in the classical case*". Albeit constructivism and set theory are sometimes depicted as antipodes, Peter Aczel showed that Myhill's intuitionistic set theory has a canonical interpretation in Martin-Löf type theory. He also found several new set-theoretic principles (especially choice principles) validated by this interpretation.

This talk will describe the set theory resulting from Aczel's interpretation in MLTT. It will also consider how the interpretation panes out when the interpreting theory is taken to be *homotopy type theory*, and what kind of set theory one obtains if one allows certain *omniscience principles* (as Bishop called them) to reign.

Logic in Dependent Type Theory

Nicola Gambino

Since around 1998, Peter Aczel became deeply interested in understanding and relating the different ways in which logic can be treated in type theory (such as the propositions-as-types paradigm or the propositions-as-elements-of-Prop idea). The motivation came from multiple angles, including experience with computer-assisted proof-checking, the type-theoretic intepretation of Constructive Set Theory, developments in Categorical Logic, and the desire to understand better fully impredicative logical systems.

This work eventually led to the proof of numerous important results and the introduction of Russell-Prawitz modalities and of logic-enriched type theories. In this talk, I will survey this part of Peter Aczel's research and its influence on subsequent developments, such as Homotopy Type Theory.

Session 3: Polarized Logic (and Formalisations and Probability Theory)

Faithful interpretations of LJT and LJQ into polarized logic

José Espírito Santo[1], Ralph Matthes[2], and Luís Pinto[1]

[1] Centro de Matemática, Universidade do Minho, Portugal
[2] IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France

Abstract

LJT and LJQ are well-known focused sequent calculi with a long history in proof theory and a connection with call-by-name and call-by-value computation, respectively. We are revisiting and problematizing a faithful interpretation of LJT into the focused sequent calculus LJP for polarized logic. Faithfulness allows for the inheritance of a solution space representation developed for LJP and the reuse of known results concerning decision problems related to inhabitation in LJP. Moreover, we are describing work in progress on a faithful interpretation of LJQ into LJP. This result was hinted before in the focusing literature, but we are aiming at a full treatment (the proofs for the fragment with implication and disjunction are settled) technically relying on the use of proof terms. The use of proof terms brings the perspective of inhabitation of simple types by the normal forms of a call-by-value language.

The authors developed for the implicational fragment of intuitionistic logic a "coinductive approach" to proof search [8, 7]. The guiding idea of the approach is to represent the entire search space of proofs for a given sequent as a single proof term. This requires extending the concept of proof term in two directions: choice points are added to represent choices found in the search process in the application of proof rules; and, since naive proof search can run into cycles, we adopt a coinductive interpretation of proof terms, so that they may represent non-wellfounded trees of locally correct applications of proof rules. The obtained expressions serve for a precise mathematical specification of decision problems related to proof search. Algorithms for these decision problems are written in an alternative, equivalent, inductively defined syntax, where cycles are represented by formal fixed-point operators.

Later this approach was extended by the present authors to polarized intuitionistic logic [6]. Formulas of this logic have one of two polarities, positive or negative, already at the level of atomic formulas, written a^+ or a^-. Positive formulas further include disjunctions $P \vee P'$, absurdity $\bot$, and negative formulas turned positive by the action of a polarity shift: $\downarrow N$. Negative formulas further include conjunctions $N \wedge N'$, implications $P \supset N$, and positive formulas turned negative $\uparrow P$. The proof system considered for this logic was a minor variant of the cut-free, focused sequent calculus $\lambda^{\pm}_{G}$, developed by the first author in [5]. Here we will call this minor variant LJP.

One advantage of studying proof search in LJP (and in polarized logics in general [11]) is that, indirectly and simultaneously, we may study proof search of other proof systems in LJP as soon as, both, we are able to soundly embed them into LJP, and such embeddings are *faithful*, a property that permits the reading back of results on the represented proof systems. This possibility has already been explored in [6] for (a minor variant of) the sequent calculus LJT [9], for which a faithful interpretation into LJP was provided. The interpretation is based on a negative polarization $(\_)^\star$ of the formulas of intuitionistic logic (in other words, $A^\star$ is a negative LJP formula for any intuitionistic formula A, notably with $(A \supset B)^\star = (\downarrow A^\star) \supset B^\star$). Faithfulness allows to decide, for instance, the existence of proofs of a given sequent in LJT by deciding the interpretation of the problem in LJP.

In this contributed talk we want to report on work in progress regarding a faithful interpretation of LJQ into LJP. System LJQ is a well-known focused sequent calculus with a long

history in proof theory and a connection with call-by-value computation [3, 4] (just as LJT is connected to call-by-name computation). The interpretation is a positive polarization $(\_)^{\mathfrak{p}}$, defined both at the level of formulas and at the level of proof terms. For formulas, this polarization is defined simultaneously with an auxiliary positive polarization $(\_)^{\bar{\mathfrak{p}}}$. In the target system LJP, only positive atoms are used for these formula translations.

Given a formula A, whereas $(\_)^{\mathfrak{p}}$ is used to interpret the subformulas occurring positively in A, the auxiliary positive polarisation $(\_)^{\bar{\mathfrak{p}}}$ is used for negative subformula occurrences. Specifically (for the treated fragment – implication and disjunction):

$$\begin{array}{lcllcllcl} a^{\mathfrak{p}} & = & a^+ & (A \supset B)^{\mathfrak{p}} & = & \downarrow (A^{\bar{\mathfrak{p}}} \supset \uparrow B^{\mathfrak{p}}) & (A \vee B)^{\mathfrak{p}} & = & A^{\mathfrak{p}} \vee B^{\mathfrak{p}} \\ a^{\bar{\mathfrak{p}}} & = & a^+ & (A \supset B)^{\bar{\mathfrak{p}}} & = & \downarrow (A^{\mathfrak{p}} \supset \uparrow B^{\bar{\mathfrak{p}}}) & (A \vee B)^{\bar{\mathfrak{p}}} & = & \downarrow\uparrow (A^{\bar{\mathfrak{p}}} \vee B^{\bar{\mathfrak{p}}}) \end{array}$$

Note that, if we erased the double shift in the $(\_)^{\bar{\mathfrak{p}}}$-translation of disjunction (or if we omitted disjunction altogether), translations $(\_)^{\bar{\mathfrak{p}}}$ and $(\_)^{\mathfrak{p}}$ would be the same. The $(\_)^{\bar{\mathfrak{p}}}$-translation is used to block the "asynchronous" (i. e., automatic) inversion of disjunctions in the left-hand side of LJP sequents, which is an effect we do not want to see in the output of our translation, as it is not observed in LJQ.

Soundness of the positive polarization holds as follows:

$$\frac{\Gamma \vdash [v : A]}{\Gamma^{\mathfrak{l}} \vdash [v^{\mathfrak{p}} : A^{\mathfrak{p}}]} \qquad \frac{\Gamma \vdash t : A}{\Gamma^{\mathfrak{l}} \vdash t^{\mathfrak{p}} : A^{\mathfrak{p}}}$$

where v (resp. t) stands for a value (resp. a proof term) of LJQ, and $(\_)^{\mathfrak{l}}$ is a *left formula* of LJP (a positive atom or a negative formula) given thus: $a^{\mathfrak{l}} = a^+$ and $A^{\mathfrak{l}} = N$ if $A^{\bar{\mathfrak{p}}} = \downarrow N$. The proof of faithfulness is obtained with the help of the obvious *forgetful map*, and, in fact, it allows to conclude faithfulness at the level of provability, but notably it also delivers a bijection at the level of proofs. Hence, analogously to LJT, faithfulness of the positive translation allows, for instance, to decide the existence of proofs of a given LJQ sequent by deciding in LJP the translation of the given LJQ sequent, through the composition of the recursive functions that calculate the finitary representation of the full solution space (cf. [6, Definition 7]) and then recurse on the structure of this representation (cf. [6, Section 4]).

It is known that LJT faces the problem of trivial unfolding of solutions of proof search in the presence of disjunction, a problem associated to the redundancy of *focused inhabitants* w. r. t. the *canonical* ones [12]. Whereas in LJT this problem only shows up when considering disjunction, in LJQ trivial unfolding of solutions of proof search problems is already experienced in dealing with implication (think for example of two distinct atoms a, b and of the sequent $x : a \supset b, y : a \vdash b$ for which bottom-up proof search can repeatedly apply $x : a \supset b$—thanks to the presence of $y : a$ in the context, generating multiple duplicates $z_1 : b, \ldots, z_n : b$, before finishing by picking up one of these duplicates). Overcoming this problem is required to address the question of the finiteness of the solution space, and is part of the ongoing work. We will explain why our treatment in [6] is not satisfactory. In this sense, the analysis of the translations of the connectives beyond implication is ongoing work for both LJT and LJQ. While they are easily accommodated in LJP as far as soundness is concerned, even faithfulness for LJT only allows for an analysis of the number of focused inhabitants.

Our work shows that LJP is a unifying framework, like LJF [11], call-by-push-value [10], or the $\lambda!$-calculus [2]. The problem of existence of call-by-value inhabitants (of intersection types) has been addressed in the recent work [1], making use of the $\lambda!$-calculus. A translation of LJQ into the focused sequent calculus LJF, with a left mode and a right mode, is already found in [11], but it is not used for the study of inhabitation.

Acknowledgements: We are thankful for the feedback we got from the anonymous reviewers. In fact, triggered by one remark, we became aware that we do not need to factor a positive translation of LJQ into LJP through a negative one, in particular not for atoms. As a side effect, we were able to simplify our analysis.

References

[1] Victor Arrial, Giulio Guerrieri, and Delia Kesner. Quantitative inhabitation for different lambda calculi in a unifying framework. *Proc. ACM Program. Lang.*, 7(POPL):1483–1513, 2023.

[2] Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. *Inf. Comput.*, 293:105047, 2023.

[3] Roy Dyckhoff and Stéphane Lengrand. LJQ: A strongly focused calculus for intuitionistic logic. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, volume 3988 of *Lecture Notes in Computer Science*, pages 173–185, 2006.

[4] Roy Dyckhoff and Stéphane Lengrand. Call-by-value lambda-calculus and LJQ. *J. Log. Comput.*, 17(6):1109–1134, 2007.

[5] José Espírito Santo. The polarized λ-calculus. In Vivek Nigam and Mário Florido, editors, *11th Workshop on Logical and Semantic Frameworks with Applications, LSFA 2016, Porto, Portugal, January 1, 2016*, volume 332 of *Electronic Notes in Theoretical Computer Science*, pages 149–168. Elsevier, 2016.

[6] José Espírito Santo, Ralph Matthes, and Luís Pinto. Coinductive proof search for polarized logic with applications to full intuitionistic propositional logic. In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*, volume 188 of *LIPIcs*, pages 4:1–4:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[7] José Espírito Santo, Ralph Matthes, and Luís Pinto. A coinductive approach to proof search through typed lambda-calculi. *Ann. Pure Appl. Log.*, 172(10):103026, 2021.

[8] José Espírito Santo, Ralph Matthes, and Luís Pinto. Inhabitation in simply-typed lambda-calculus through a lambda-calculus for proof search. *Mathematical Structures in Computer Science*, 29:1092–1124, 2019. Also found at HAL through https://hal.archives-ouvertes.fr/hal-02360678v1.

[9] Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes.* Ph.D. thesis, University Paris 7, January 1995.

[10] Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *High. Order Symb. Comput.*, 19(4):377–414, 2006.

[11] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logic. *Theor. Comput. Sci.*, 410:4747–4768, 2009.

[12] Gabriel Scherer and Didier Rémy. Which simple types have a unique inhabitant? In Kathleen Fisher and John H. Reppy, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 243–255. ACM, 2015.

Yet another formal theory of probabilities (with an application to random sampling)

Reynald Affeldt[1], Alessandro Bruni[2], Pierre Roux[3], and Takafumi Saikawa[4]

[1] National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan
[2] IT University of Copenhagen, Denmark
[3] ONERA/DTIS, Université de Toulouse, France
[4] Nagoya University, Japan

Abstract

There are already several formalizations of probability theory in the Coq proof assistant with applications to mathematics, information theory, and programming languages. They have been developed independently, do not cover the same ground, and a substantial effort is required to make them inter-operate. In this presentation, we report about an on-going effort in Coq to port and generalize a library about finite probabilities to a more generic formalization of real analysis called MathComp-Analysis. This gives us an opportunity to generalize results about convexity and probability and to enrich the library of probability inequalities. We explain our process of formalization and apply the resulting library to an original formalization of random sampling.

An overview of formalization of probabilities in Coq We know of several formalizations of probabilities in Coq[1] . InfoTheo is a formalization of finite probabilities that has been used to formalize information theory, error-correcting codes, and robust statistics (e.g., [5, 9]). Discrete probabilities has been formalized in coq-proba [18] and used to reason about programs (e.g., [10]). FormalML contains advanced theorems on probability theory [19,20]. On the other hand, the MathComp-Analysis library, built on top of the Mathematical Components library [14], provides a rich formalization of measure theory and Lebesgue integral [2, 13]. In particular, MathComp-Analysis has been used to formalize probabilistic programming [3, 17].

Porting convexity results from InfoTheo to MathComp-Analysis We learn from InfoTheo that dealing with probabilities benefits from having a theory of *convex spaces*, to represent, among others, convex functions [6, Sect. 3.3]. A convex space is a mathematical structure with an operator written `a <| p |> b` (where `p` is a real number between 0 and 1) that expresses convex combination and a few axioms about this operator (skewed commutativity, quasi-associativity, etc.). Convex spaces are advantageously formalized using Hierarchy-Builder [8], a tool to build hierarchies of mathematical structures, see [12, `convex.v`]. The operator for convex combination is better handled with a dedicated type for real numbers between 0 and 1 (to represent the `p` in `a <| p |> b`), and InfoTheo provides such a specific type. On the other hand, MathComp-Analysis also had theories for positive and non-negative real numbers (i.e., real numbers in $]0, +\infty[$ and $[0, +\infty[$). We figured out that real numbers in $[0, 1]$ can be handled similarly, thus providing a type `{i01 R}` to write convexity statements [1, `convex.v`], e.g.:

```
Definition convex_function (R : realType) (D : set R) (f : R -> R) :=
  forall t : {i01 R}, {in D &, forall (x y : R), f (x <| t |> y) <= f x <| t |> f y}.
```

Using convex spaces and convex functions from MathComp-Analysis, we have been able to port results from InfoTheo such as the convexity of the exponential function [1, `hoelder.v`]:

```
Lemma convex_powR p : 1 <= p -> convex_function `[0, +oo[ (fun x : R => powR x p).
```

We are also planning to port more related results from InfoTheo such as conical spaces [4, Sect. 4].

Basic definitions of probability theory in MathComp-Analysis Probability measures come from basic definitions about measure theory. A measure μ satisfies the following: $\mu(\emptyset) = 0$, $0 \leq \mu(A)$ for any A, and σ-additivity: $\mu(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i)$ for countably many pairwise disjoint A_i's [1, `measure.v`]. A probability measure extends a measure with the following interface (giving rise to a type `probability T R`):

[1] It should be noted that other proof assistants also provide substantial accounts of probability theory (in particular in Isabelle/HOL [7, 11] and Lean).

```
HB.mixin Record isProbability d (T : measurableType d) (R : realType) (P : set T -> \bar R) :=
  { probability_setT : P setT = 1 }. (* setT is the full set *)
```

The Lebesgue integral (noted `\int[mu]_(x in A) f x` [2, Sect. 6.4]) is used to formalize the notions of expectation, covariance, and variance [1, `probability.v`], e.g., for the expectation (noted `'E_P[X]`):

```
Definition expectation d (T : measurableType d) (R : realType) (P : probability T R)
  (X : T -> R) := \int[P]_w (X w)%:E. (* %:E turns real numbers into extended real numbers *)
```

Random variables are essentially measurable functions (noted `{mfun T >-> R}`). Like in InfoTheo, the probability measure `P` of the underlying space is encoded as a phantom type:

```
Definition random_variable d (T : measurableType d) (R : realType) (P : probability T R) :=
  {mfun T >-> R}.
Notation "{ 'RV' P >-> R }" := (@random_variable _ _ R P).
```

This way, when we write `{RV P >-> R}` for the type of a random variable, we understand that the underlying sample space is the one corresponding to the probability measure `P`.

We use Hierarchy-Builder and the cardinality theory in MathComp-Analysis [1, `cardinality.v`] to extend the mathematical structure of random variables to the one of discrete random variables:

```
HB.mixin Record MeasurableFun_isDiscrete d (T : measurableType d) (R : realType)
  (X : T -> R) of @MeasurableFun d T R X := { countable_range : countable (range X) }.
```

Let `{dRV P >-> R}` be the type of discrete random variables. From a discrete random variable `X` we can derive a function `dRV_enum` to enumerate the values a_k it takes and a function `enum_prob` to enumerate the weigths c_k so that the distribution $P_{\mathtt{X}}$ of `X` can be written as a countable sum of Dirac measures $\sum_k c_k \delta_{a_k}$, eventually recovering the fact that the expectation of `X` is $\sum_k c_k a_k$ (using the properties of the Lebesgue integral):

```
Lemma distribution_dRV A : measurable A ->
  distribution P X A = \sum_(k <oo) enum_prob X k * \d_(dRV_enum X k) A. (* \d_ is for δ *)
```

The last bit of our basic setting of probability theory in MathComp-Analysis consists of the definition of L^p spaces. For that purpose, we prove Hölder's inequality:

```
Lemma hoelder (f g : T -> R) (p q : R) : measurable_fun setT f -> measurable_fun setT g ->
  0 < p -> 0 < q -> p^-1 + q^-1 = 1 ->
    'N_1 [f \* g] <= 'N_p [f] * 'N_q [g]. (* \* is the pointwise multiplication *)
```

The notation `'N_p[f]` denotes the L^p norm of `f`. This theorem relies on the formalization of convexity mentioned above. Cauchy-Schwarz's inequality is widely used in probability theory and is just a special case of Hölder's where $p = q = 2$. Furthermore, Hölder's inequality can be used to prove Minkowski's inequality:

```
Lemma minkowski f g p : measurable_fun setT f -> measurable_fun setT g -> 1 <= p ->
  'N_p%:E[f \+ g] <= 'N_p%:E[f] + 'N_p%:E[g]. (* \+ is the pointwise addition *)
```

This lemma shows that L^p spaces are normed vector spaces.

Recent and current work We further extend the above setup with fundamental inequalities such as Markov's, Chernoff's, Chebyshev's, and Cantelli's, etc. We are now working on defining precisely L^p spaces with MathComp's generic quotients. Our development has already been used in the verification of worst-case failure probability of real-time systems [15]. We have been tackling the formalization of a sampling theorem [16, Theorem 3.1] which requires formalizing notions of random trials (including the notion of independence) and makes use of Chernoff's bound:

```
Theorem sampling (X_ : seq {RV P >-> R}) (theta delta p : R) :
  let n := size X_ in let X' x := ((\sum_(Xi in X_) Xi) x) / n%:R in is_bernoulli_trial X_ n ->
  0 < p <= 1 -> 0 < delta <= 1 -> 0 < theta < p -> 0 < n -> 3 / theta^+2 * ln(2 / delta) <= n%:R
  -> P [set i | `| X' i - p | <= theta] >= 1 - delta%:E.
```

References

[1] R. Affeldt, Y. Bertot, A. Bruni, C. Cohen, M. Kerjean, A. Mahboubi, D. Rouhling, P. Roux, K. Sakaguchi, Z. Stone, P.-Y. Strub, and L. Théry. MathComp-Analysis: Mathematical components compliant analysis library. https://github.com/math-comp/analysis, 2024. Since 2017. Version 1.1.0.

[2] R. Affeldt and C. Cohen. Measure construction by extension in dependent type theory with application to integration. *J. Autom. Reason.*, 67(3):28, 2023.

[3] R. Affeldt, C. Cohen, and A. Saito. Semantics of probabilistic programs using s-finite kernels in Coq. In *12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2023), Boston, MA, USA, January 16–17, 2023*, pages 3–16. ACM, 2023.

[4] R. Affeldt, J. Garrigue, and T. Saikawa. Formal adventures in convex and conical spaces. In *13th Conference on Intelligent Computer Mathematics (CICM 2020), Bertinoro, Forli, Italy, July 26–31, 2020*, volume 12236 of *Lecture Notes in Artificial Intelligence*, pages 23–38. Springer, Jul 2020.

[5] R. Affeldt, J. Garrigue, and T. Saikawa. A library for formalization of linear error-correcting codes. *J. Autom. Reason.*, 64(6):1123–1164, 2020.

[6] R. Affeldt, J. Garrigue, and T. Saikawa. Reasoning with conditional probabilities and joint distributions in Coq. *Computer Software*, 37(3):79–95, 2020.

[7] J. Avigad, J. Hölzl, and L. Serafin. A formally verified proof of the central limit theorem. *J. Autom. Reason.*, 59(4):389–423, 2017.

[8] C. Cohen, K. Sakaguchi, and E. Tassi. Hierarchy builder: Algebraic hierarchies made easy in Coq with Elpi (system description). In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020), June 29–July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 34:1–34:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. https://hal.inria.fr/hal-02478907/document.

[9] I. Daukantas, A. Bruni, and C. Schürmann. Trimming data sets: a verified algorithm for robust mean estimation. In *23rd International Symposium on Principles and Practice of Declarative Programming (PPDP 2021), Tallinn, Estonia, September 6–8, 2021*, pages 17:1–17:9. ACM, 2021.

[10] S. O. Gregersen, A. Aguirre, P. Haselwarter, J. Tassarotti, and L. Birkedal. Asynchronous probabilistic couplings in higher-order separation logic, 2023.

[11] J. Hölzl. *Construction and stochastic applications of measure spaces in higher-order logic*. PhD thesis, Technical University Munich, 2013.

[12] Infotheo. Infotheo: A Coq formalization of information theory and linear error-correcting codes. https://github.com/affeldt-aist/infotheo, 2018. Authors: Reynald Affeldt, Manabu Hagiwara, Jonas Sénizergues, Jacques Garrigue, Kazuhiko Sakaguchi, Taku Asai, Takafumi Saikawa, and Naruomi Obata. Last stable release: 0.7.0 (2024).

[13] Y. Ishiguro and R. Affeldt. The Radon-Nikodým theorem and the Lebesgue-Stieltjes measure in Coq. *Computer Software*, 41(2), 2024.

[14] A. Mahboubi and E. Tassi. *Mathematical Components*. Zenodo, Jan 2021. https://zenodo.org/record/7118596.

[15] F. Markovic, P. Roux, S. Bozhko, A. V. Papadopoulos, and B. B. Brandenburg. CTA: A correlation-tolerant analysis of the deadline-failure probability of dependent tasks. In *IEEE Real-Time Systems Symposium (RTSS 2023), Taipei, Taiwan, December 5–8, 2023*, pages 317–330. IEEE, 2023.

[16] S. Rajani. Applications of chernoff bounds. http://math.uchicago.edu/~may/REU2019/REUPapers/Rajani.pdf, 2019. The University of Chicago Mathematics REU 2019.

[17] A. Saito and R. Affeldt. Experimenting with an intrinsically-typed probabilistic programming language in Coq. In *21st Asian Symposium on Programming Languages and Systems (APLAS 2023), Taipei, Taiwan, November 26–29, 2023*, volume 14405 of *Lecture Notes in Computer Science*, pages 182–202. Springer, 2023.

[18] J. Tassarotti. A probability theory library for the Coq theorem prover. https://github.com/jtassarotti/coq-proba, 2023. Since 2020.

[19] The FormalML development team. FormalML: Formalization of machine learning theory with applications to program synthesis. https://github.com/IBM/FormalML, 2023. Since 2019.

[20] K. Vajjha, A. Shinnar, B. M. Trager, V. Pestun, and N. Fulton. CertRL: formalizing convergence proofs for value and policy iteration in Coq. In *10th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2021), Virtual Event, Denmark, January 17–19, 2021*, pages 18–31. ACM, 2021.

Session 4: Constructive Mathematics

A zoo of continuity properties in constructive type theory

Martin Baillon[1], Yannick Forster[1], Assia Mahboubi[12], Pierre-Marie Pédrot[1], and Matthieu Piquerez[1]

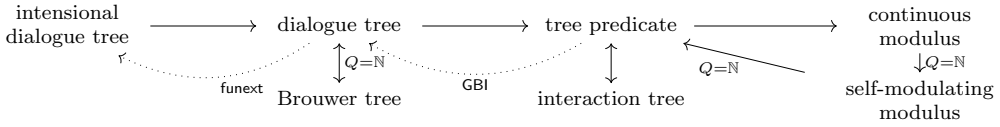
[1] Inria, France
[2] Vrije Universiteit Amsterdam, The Netherlands

Abstract

We analyse continuity properties in dependent type theory, from the perspective of constructive reverse mathematics. All our results are mechanised in the Coq proof assistant. [2]

Continuity principles stating that *all* functions are continuous play a central role in some schools of constructive mathematics. Intuitively, a function is continuous if finite amounts of output only rely on finite amounts of input. We use type theory as foundation of constructive mathematics to formalise continuity properties making this intuition precise, in order to analyse their strength and the essential differences in the continuity principles they lead to. Most results in the present formalised note are known for foundations like HA or HA^ω or are folklore, but often have been studied only at type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$. We give a first unified, formalised account, generalise as much as possible from $\mathbb{N}$, and use no strong choice axioms such as countable choice.

We consider three main forms of continuity: First, one derived from the usual definition in topology, making explicit a modulus of continuity as the list $L : \mathbb{L}Q$ of queries to the input that matter [21]. Secondly, Escardó's eloquent continuity [9] based on inductive computation trees modeled in two different forms: as dialogue trees and Brouwer trees, both also considered by Ghani, Hancock, and Pattinson [13, 14]. Thirdly, a variant of van Oosten's sequential continuity [22], which makes the computation explicit in form of a computation tree in two different forms: modeled as a predicate and modeled as a coinductive interaction tree [23]. The following diagram sums up the connections, where dotted arrows indicate a need of axioms.



Modulus continuity A function $F : (Q \to A) \to R$ has modulus $L : \mathbb{L}Q$ at $f : Q \to A$ if for g with $\forall x \in L.\ fx = gx$, we have $Ff = Fg$. F is *modulus-continuous* if for any f a modulus exists, and has a *modulus of continuity function* $M : (Q \to A) \to \mathbb{L}Q$ if for all f, Mf is a modulus at f.

Eloquent continuity We define an inductive type of *dialogue trees* $\mathbb{D}$ with leafs η labelled by a result R and internal nodes β labelled by a query $q : Q$ and branching via answers in A. We also introduce the novel notion of *intensional eloquent continuity* via the inductive predicate $\mathbb{D}_\mathsf{i}$, over type $(Q \to A) \to R$, the *intensional dialogue tree* vertex on the diagram.

$$\frac{r : R}{\eta r : \mathbb{D}} \qquad \frac{q : Q \qquad k : A \to \mathbb{D}}{\beta q k : \mathbb{D}} \qquad \frac{r : R}{\eta_\mathsf{i} r : \mathbb{D}_\mathsf{i}(\lambda f.\ r)} \qquad \frac{q : Q \qquad k : A \to (Q \to A) \to R \qquad H : \forall a : A.\ \mathbb{D}_\mathsf{i}(ka)}{\beta_\mathsf{i} q k H : \mathbb{D}_\mathsf{i}(\lambda f.\ k(fq)f)}$$

We define an evaluation function $\partial : (Q \to A) \to \mathbb{D} \to R$ as $\partial f(\eta o) := o$ and $\partial f(\beta q k) := \partial f(k(fq))$ A function F is *eloquent* if there exists $d : \mathbb{D}$ such that $\forall f : Q \to A.\ Ff = \partial f d$. A function F is *intensionally eloquent* if $\mathbb{D}_\mathsf{i} F$.

Theorem 1. *Intensional eloquent continuity implies eloquent continuity. The converse is provable assuming function extensionality.*

For $Q = \mathbb{N}$, we also contribute Coq formalisations of Escardó's [10] and Sterling's [20] proofs in Agda relating Brouwer trees and dialogue trees, but omit formal definitions here.

Sequential continuity Sequential continuity has a similar intuition than eloquent continuity. However, instead of defining trees inductively, it relies on an extensional notion of tree predicates: We take any function $\tau : \mathbb{L}A \to (Q + R)$ to describe a tree We write $\mathsf{ask}\ q$ for $\mathsf{inl}\ q$ and $\mathsf{ret}\ r$ for $\mathsf{inr}\ r$. Given a tree τ, we define an operation $\hat{\tau} : (Q \to A) \to \mathbb{N} \to Q + R$ by $\hat{\tau}\ f\ (\mathsf{S}n) := \hat{\tau}\ (\lambda l.\ \tau(l +\!\!+ [fq]))\ n$ if $\tau\ [\,] = \mathsf{ask}\ q$ and $\hat{\tau}\ f\ n := \tau\ [\,]$ otherwise.

We call a tree τ well-founded if for any $f : Q \to A$ there exist n and r s.t. $\hat{\tau}\ f\ n = \mathsf{ret}\ r$. F is *sequentially continuous* if $\exists \tau : \mathbb{L}A \to Q + R. \forall f.\ \exists n.\ \hat{\tau}\ f\ n = \mathsf{ret}\ (Ff)$.

Theorem 2. *(1) Eloquent F are sequentially continuous. (2) Sequentially continuous F have a continuous modulus of continuity function. (3) Sequential continuity is equivalent to continuity with interaction trees. (4) If $Q = \mathbb{N}$, then any F with a continuous modulus of continuity function, has a self-modulating modulus of continuity function. (5) If F has a self-modulating modulus of continuity, it is sequentially continuous.*

(4) relies on a (Δ_1^0) choice operator for decidable relations on $\mathbb{N}$, definable in Coq. Steinberg, Théry, and Thies prove a similar result in Coq assuming the axiom of choice [19].

Bar induction We deal with the statement that sequentially continuous $F : (Q \to A) \to R$ are eloquent. Since this principle allows turning an extensional tree into an inductive tree, we call this principle $\mathsf{CI}(Q, A, R)$ (continuity induction). The notion is inspired by Brede and Herbelin's generalised bar induction principle $\mathsf{GBI}(Q, A)$ [5]. A bar is the (logical) complement of a well-founded tree. $\mathsf{GBI}(Q, A)$ states bars with internal nodes labeled with Q and branching over A are in fact inductively barred. They prove that $\mathsf{GBI}(\mathbb{N}, A)$ is equivalent to regular bar induction $\mathsf{BI}(A)$, where the nodes of the trees are unlabeled and still branch via A. We conjecture that this equivalence can be generalised to preserve the logical complexity of trees under mild assumption on the complexity class.

Conjecture 1. *C-$\mathsf{GBI}(\mathbb{N}, A) \leftrightarrow C$-$\mathsf{BI}(A)$, for C being some class predicate with to be determined closure properties, which in particular can be instantiated with $C := \Delta_1^0$.*

Theorem 3. *(1)* Δ_1^0-$\mathsf{GBI}(Q, A) \to \mathsf{CI}(Q, A, R)$ *(2)* $\mathsf{CI}(\mathbb{N}, A, \mathbb{L}A) \to \Delta_1^0$-$\mathsf{BI}(A)$.

Continuity principles Continuity principles stating that all (total) functions $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ are continuous appear in the literature under different names. For simplicity, we call it CP here. It is called KLS by Beeson [4] and KLST by Ishihara [15], after the Kreisel-Schoenfield-Lacombe theorem in computability theory [17], independently also proved by Ceitin (Tseitin) [6]. It is called Brouwer's continuity principle (BC) by Bauer [3]. Extensionality issues aside, for the definitions covered here we end up with two distinct ones for $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$: eloquent continuity based on inductive dialogue trees and sequential continuity based on extensional tree predicates, and thus two variants of CP. See the first author's PhD thesis for a discussion of CP in constructive type theory [1]. CP for eloquent continuity implies CP for sequential continuity. Since our results yield that an equivalence of the two for any function F exactly has the strength of BI, it is clear that BI is enough to prove that CP for sequential continuity implies CP for eloquent continuity. However, that this implication between CP principles has the strength of BI does not directly follow. In future work, we plan to analyse whether the two principles can be separated as well as the exact strength of this implication, e.g. by using model constructions à la Cohen and Rahli [7]. We want to extend the analysis to other notions of continuity, e.g. based on neighborhood functions as used e.g. by the relation of continuity to bar induction established by Kawai [16] to the so-called dynamical method from algebra [8] as well as Misselbeck-Wessel and Schuster's notion of predicates with cofinite character [18], and to partial functions, see e.g. [11, 12]. Lastly, it would be interesting to study synthetic constructions of Kleene's second (partial combinatory) algebra based on the different notions.

Acknowledgements We want to thank the Inria Gallinette team, especially Meven Lennon-Bertrand, Yann Leray, and Loïc Pujet for the initial discussions leading to this abstract. We also thank Andrej Bauer, Niklas Mück, Hugo Herbelin, Dominik Kirst, Bruno da Rocha Paiva, Vincent Rahli, Florian Steinberg, and Benno van den Berg for discussions about continuity and useful pointers. We are grateful to the members of the Types 2024 program committee for their comments and suggestions.

References

[1] Martin Baillon. *Continuity in Type Theory*. PhD thesis, Nantes University, 2023. https://gitlab.inria.fr/mbaillon/Manuscript/-/blob/main/main.pdf.

[2] Martin Baillon, Yannick Forster, Assia Mahboubi, Pierre Marie Pédrot, and Matthieu Piquerez. A zoo of continuity properties in constructive type theory. https://github.com/amahboubi/continuity-zoo, 2024.

[3] Andrej Bauer. Continuity principles and the KLST theorem. *Blog post*, 2023. URL: https://math.andrej.com/2023/07/19/continuity-principles-and-the-klst-theorem/.

[4] Michael J. Beeson. The nonderivability in intuitionistic formal systems of theorems on the continuity of effective operations. *Journal of Symbolic Logic*, 40(3):321–346, September 1975. doi:10.2307/2272158.

[5] Nuria Brede and Hugo Herbelin. On the logical structure of choice and bar induction principles. In *LICS*, pages 1–13. IEEE, 2021.

[6] Grigori Samuilovitsch Ceitin. Algorithmic operators in constructive complete separable metric spaces. *Doklady Akademii Nauk SSSR*, 128:49–52, 1959.

[7] Liron Cohen and Vincent Rahli. Realizing continuity using stateful computations. In *31st EACSL Annual Conference on Computer Science Logic, CSL 2023*, volume 252 of *LIPIcs*, 2023. doi:10.4230/LIPICS.CSL.2023.15.

[8] Michel Coste, Henri Lombardi, and Marie-Françoise Roy. Dynamical method in algebra: Effective Nullstellensätze, January 2017. arXiv:1701.05794, doi:10.48550/arXiv.1701.05794.

[9] Martín Hötzel Escardó. Continuity of Gödel's system T definable functionals via effectful forcing. In *MFPS*, volume 298 of *Electronic Notes in Theoretical Computer Science*, pages 119–141. Elsevier, 2013. doi:10.1016/j.entcs.2013.09.010.

[10] Martin Escardó and Paulo Oliva. Conversion of dialogue trees to Brouwer trees. https://www.cs.bham.ac.uk//~mhe/dialogue/dialogue-to-brouwer.agda, 2017.

[11] Yannick Forster, Dominik Kirst, and Niklas Mück. *Oracle Computability and Turing Reducibility in the Calculus of Inductive Constructions*, page 155–181. Springer Nature Singapore, 2023. doi:10.1007/978-981-99-8311-7_8.

[12] Yannick Forster, Dominik Kirst, and Niklas Mück. The Kleene-Post and Post's Theorem in the Calculus of Inductive Constructions. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.29.

[13] Neil Ghani, Peter Hancock, and Dirk Pattinson. Continuous Functions on Final Coalgebras. *Electronic Notes in Theoretical Computer Science*, 249:3–18, August 2009. doi:10.1016/j.entcs.2009.07.081.

[14] Peter Hancock, Dirk Pattinson, and Neil Ghani. Representations of Stream Processors Using Nested Fixed Points. *Logical Methods in Computer Science*, Volume 5, Issue 3, September 2009. doi:10.2168/LMCS-5(3:9)2009.

[15] Hajime Ishihara. Continuity properties in constructive mathematics. *Journal of Symbolic Logic*, 57(2):557–565, June 1992. doi:10.2307/2275292.

[16] Tatsuji Kawai. Principles of bar induction and continuity on Baire space. *Journal of Logic and Analysis*, 2019. doi:10.4115/jla.2019.11.ft3.

[17] Georg Kreisel, Daniel Lacombe, and Joseph R Shoenfield. Partial recursive functionals and effective operations. In *Constructivity in mathematics: Proceedings of the colloquium held at Amsterdam*, volume 1965, page 367, 1957.

[18] Daniel Misselbeck-Wessel and Peter Schuster. Radical theory of Scott-open filters. *Theoretical Computer Science*, 945:113677, February 2023. doi:10.1016/j.tcs.2022.12.027.

[19] Florian Steinberg, Laurent Thery, and Holger Thies. Computable analysis and notions of continuity in Coq. *Logical Methods in Computer Science*, Volume 17, Issue 2, May 2021. doi:10.23638/LMCS-17(2:16)2021.

[20] Jonathan Sterling. Higher order functions and Brouwer's thesis. *J. Funct. Program.*, 31:e11, 2021. doi:10.1017/s0956796821000095.

[21] Anne Sjerp Troelstra and Dirk van Dalen. Constructivism in mathematics. vol. i. *Studies in Logic and the Foundations of Mathematics*, 26, 1988.

[22] Jaap van Oosten. Partial combinatory algebras of functions. *Notre Dame Journal of Formal Logic*, 52(4):431–448, 2011. doi:10.1215/00294527-1499381.

[23] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. Interaction trees: representing recursive and impure programs in Coq. *Proc. ACM Program. Lang.*, 4(POPL):51:1–51:32, 2020. doi:10.1145/3371119.

The Blurred Drinker Paradox and Blurred Choice Axioms for the Downward Löwenheim-Skolem Theorem

Dominik Kirst[1,2] and Haoyi Zeng[1]

[1] Saarland University, Germany
[2] Ben-Gurion University, Israel

Abstract

In the setting of constructive reverse mathematics, we analyse the downward Löwenheim-Skolem (DLS) theorem of first-order logic, stating that every infinite model has a countable elementary submodel. Refining the well-known equivalence of the DLS theorem to the axiom of dependent choice (DC) over the classically omnipresent axiom of countable choice (CC) and law of excluded middle (LEM), our approach allows for several finer logical decompositions: Over CC, the DLS theorem is equivalent to the conjunction of DC with a newly identified principle weaker than LEM that we call the blurred drinker paradox (BDP), and without CC, the DLS theorem is equivalent to the conjunction of BDP with similarly blurred weakenings of DC and CC. Orthogonal to their connection with the DLS theorem, we also study BDP and the blurred choice axioms in their own right, for instance by showing that BDP is LEM without some contribution of Markov's principle and that blurred DC is DC without some contribution of CC. All results are stated in the Calculus of Inductive Constructions and an accompanying Coq mechanisation is available on Github.

Background The Löwenheim-Skolem theorem is a central result about first-order logic, entailing that the formalism is incapable of distinguishing different infinite cardinalities. In particular its so-called downward part, stating that every infinite model can be turned into a countably infinite model with otherwise the exact same behaviour, can be considered surprising or even paradoxical: even systems like ZF set theory, concerned with uncountably large sets like the reals or their iterated power sets, admit countable interpretations. This seeming contradiction in particular and its metamathematical relevance in general led to an investigation of the exact assumptions under which the downward Löwenheim-Skolem (DLS) theorem applies.

From the perspective of (classical) reverse mathematics [6, 10], there is a definite answer: the DLS theorem (for countable languages) is equivalent to the dependent choice axiom (DC), a weak form of the axiom of choice, stating that there is a path through every total relation [5, 8, 3]. To argue one direction, one can organise the countable submodel construction such that a single application of DC is needed. For the other direction, one uses the DLS theorem to turn a given total relation R into a countable sub-relation R', applies the classically provable countable choice (CC) to get a path f' through R', and reflects it back as a path f through R.

However, the classical answer is insufficient if one investigates the computational content of the DLS theorem, i.e. the question how effective the transformation of a model into a countable submodel really is. A more adequate answer can be obtained by switching to the perspective of *constructive* reverse mathematics [7, 4], which is concerned with the analysis of logical strength over a constructive meta-theory, i.e. in particular without the law of excluded middle (LEM), stating that $p \vee \neg p$ for all propositions p, and ideally also without CC [9]. In that setting, finer logical distinctions become visible and thus one can analyse whether the computational content of the DLS theorem is exactly the same as that of DC [1, 2] by investigating whether (1) it still follows from DC alone, without any contribution of LEM, and (2) whether it still implies the full strength of DC, without any contribution of CC. We show that neither (1) nor (2) is the case by observing that the DLS theorem requires a fragment of LEM, which we call the blurred drinker paradox, and that it implies only a similarly blurred fragment of DC.

The Blurred Drinker Paradox The usual drinker paradox states that in every bar there is a person such that everyone drinks if that person drinks. A blurred version is obtained by replacing that person by an at most countable subset, represented by a function with domain $\mathbb{N}$. We also introduce a blurring of the existence principle, the dual to the drinker paradox.

$$\mathsf{BDP}_A \;:=\; \forall P : A{\to}\mathbb{P}.\exists f : \mathbb{N}{\to}A.\,(\forall y.\, P\,(f\,y)) \to \forall x.\, P\,x$$
$$\mathsf{BEP}_A \;:=\; \forall P : A{\to}\mathbb{P}.\exists f : \mathbb{N}{\to}A.\,(\exists x.\, P\,x) \to \exists y.\, P\,(f\,y)$$

Here $\mathbb{N}$ can be replaced by other types, e.g. with $\mathbb{1}$ one recovers the usual drinker paradoxes and, in general, larger types induce weaker principles. Also, $\mathsf{BDP}_{\mathbb{N}}$ and $\mathsf{BEP}_{\mathbb{N}}$ are provable by choosing f to be the identity function. Writing BDP if BDP_A for all inhabited A (similar for other principles), we for instance obtain that the blurred drinker paradoxes decompose LEM:

Fact 1. LEM *is equivalent to the conjunction of* BDP *(or* BEP*) and Markov's principle.*

Blurred Choice Axioms Via the same blurring technique, a version of CC can be given where the outputs of choice functions f for total relations R are hidden in a countable subset.

$$\mathsf{BCC}_A \;:=\; \forall R : \mathbb{N}{\to}A{\to}\mathbb{P}.\,\mathsf{tot}(R) \to \exists f : \mathbb{N}{\to}A.\forall n.\exists m.\, R\,n\,(f\,m)$$

Therefore, BCC reduces CC to the special case $\mathsf{CC}_{\mathbb{N}}$ for relations $R : \mathbb{N}{\to}\mathbb{N}{\to}\mathbb{P}$:

Fact 2. CC *is equivalent to the conjunction of* BCC *and* $\mathsf{CC}_{\mathbb{N}}$*.*

Similarly, a blurred version of DC states that every directed relation contains a countable directed subrelation, where $R \circ f : \mathbb{N}{\to}\mathbb{N}{\to}\mathbb{P}$ is the pointwise composition of R with f.

$$\mathsf{DDC}_A \;:=\; \forall R : A{\to}A{\to}\mathbb{P}.\,\mathsf{dir}(R) \to \exists f : \mathbb{N}{\to}A.\,\mathsf{dir}(R \circ f)$$

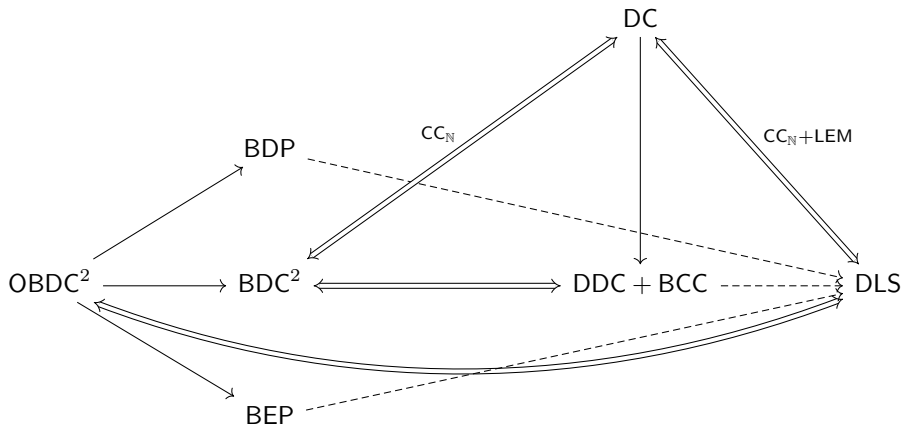
Analogous to the case of BCC, only $\mathsf{CC}_{\mathbb{N}}$ separates DDC from the full strength of DC:

Fact 3. DC *is equivalent to the conjunction of* DDC *and* $\mathsf{CC}_{\mathbb{N}}$*.*

Main Results The DLS theorem states that every first-order model $\mathcal{M}$ over a countable signature has a countable elementary submodel $\mathcal{N}$, i.e. there is an embedding $h : \mathcal{N}{\to}\mathcal{M}$ such that for every variable environment $\rho : \mathbb{N}{\to}\mathcal{N}$ and formula φ it holds that $\mathcal{N} \vDash_\rho \varphi$ iff $\mathcal{M} \vDash_{h\circ\rho} \varphi$. We obtain two logical decompositions of the DLS theorem over constructive base systems:

Theorem 1. *With* $\mathsf{CC}_{\mathbb{N}}$ *assumed,* DLS *is equivalent to the conjunction of* DC*,* BDP*, and* BEP*. Without any assumptions,* DLS *is equivalent to the conjunction of* BCC*,* DDC*,* BDP*, and* BEP*.*

The following diagram summarises these and further decompositions, where BDC^2 is a natural combination of DDC and BCC, and OBDC^2 further merges in BDP and BEP.



References

[1] Stefano Berardi, Marc Bezem, and Thierry Coquand. On the computational content of the axiom of choice. *The Journal of Symbolic Logic*, 63(2):600–622, 1998.

[2] Ulrich Berger and Paulo Oliva. Modified bar recursion and classical dependent choice. In *Logic Colloquium*, volume 1, pages 89–107, 2001.

[3] George S Boolos, John P Burgess, and Richard C Jeffrey. *Computability and logic*. Cambridge university press, 2002.

[4] Hannes Diener. Constructive reverse mathematics: Habilitationsschrift. 2018. Universität Siegen.

[5] Christian Espíndola. Löwenheim-skolem theorems and choice principles. Technical report, 2012. URL: https://www.su.se/polopoly_fs/1.229309.1426783774!/menu/standard/file/ls.pdf.

[6] Harvey M Friedman. Systems on second order arithmetic with restricted induction i, ii. *J. Symb. Logic*, 41:557–559, 1976.

[7] Hajime Ishihara. Reverse Mathematics in Bishop's Constructive Mathematics. *Philosophia Scientae*, pages 43–59, 2006. doi:10.4000/philosophiascientiae.406.

[8] Asaf Karagila. Downward löwenheim-skolem theorems and choice principles. Technical report, 2014. URL: https://karagila.org/wp-content/uploads/2012/10/Lowenheim-Skolem-and-Choice.pdf.

[9] Fred Richman. Constructive mathematics without choice. In *Reuniting the Antipodes—Constructive and Nonstandard Views of the Continuum: Symposium Proceedings, San Servolo, Venice, Italy, May 16–22, 1999*, pages 199–205. Springer, 2001.

[10] Stephen G Simpson. *Subsystems of second order arithmetic*, volume 1. Cambridge University Press, 2009.

Limited Principles of Omniscience in Constructive Type Theory

Bruno da Rocha Paiva[1], Liron Cohen[2], Yannick Forster[3], Dominik Kirst[2], and Vincent Rahli[1]

[1]University of Birmingham, UK [2]Ben-Gurion University, Israel, [3]Inria Paris, France

Abstract

The Limited Principle of Omniscience (LPO) is often enough to prove theorems of classical mathematics. LPO is an instance of the Law of Excluded Middle (LEM) which states that Σ^0_1 propositions P (i.e. existential quantification over a decidable predicate on $\mathbb{N}$) are classical (i.e. $P \vee \neg P$ holds). It implies Markov's Principle (MP), stating that Σ^0_1 propositions are stable under double negation. Several variants of MP, varying in the definition of decidability, have been introduced and used in the literature, and we have shown in previous work that two of these variants can be separated. We further show here how to separate three variants (stated over (1) a decidable predicate; (2) a Boolean-valued function; and (3) a primitive recursive Boolean-valued function), and extend those results to LPO. Furthermore, we for the first time give these separations (formalized in Agda[1]) for Martin-Löf Type Theory (MLTT), which is at the heart of many dependent type theories.

Definitions In previous work we investigated three variants of Markov's Principle (MP) [18, 4, 14, 5] and discussed [7] how to separate the the last two in constructive type theory:

$$\begin{aligned}
\mathsf{MP}_{\mathbb{P}} &:= \forall A : \mathbb{N} \to \mathbb{P}.\ (\forall n.\ An \vee \neg An) \to \quad \neg\neg(\exists n.\ An) \to (\exists n.\ An)\\
\mathsf{MP}_{\mathbb{B}} &:= \forall f : \mathbb{N} \to \mathbb{B}. \qquad\qquad\qquad\quad \neg\neg(\exists n.\ fn = \mathsf{true}) \to (\exists n.\ fn = \mathsf{true})\\
\mathsf{MP}_{\mathsf{PR}} &:= \forall f : \mathbb{N} \to \mathbb{B}.\ \mathsf{primitive\text{-}recursive}\ f \to \neg\neg(\exists n.\ fn = \mathsf{true}) \to (\exists n.\ fn = \mathsf{true})
\end{aligned}$$

We list these in order of their strength, as $\mathsf{MP}_{\mathbb{P}}$ implies $\mathsf{MP}_{\mathbb{B}}$, which in turn implies $\mathsf{MP}_{\mathsf{PR}}$. For the reverse directions, with the axiom of unique choice we can show that $\mathsf{MP}_{\mathbb{B}}$ implies $\mathsf{MP}_{\mathbb{P}}$ and under Church's Thesis $\mathsf{MP}_{\mathsf{PR}}$ implies $\mathsf{MP}_{\mathbb{B}}$.

Similarly, we can define three variants of the Limited Principle of Omniscience (LPO) [1]:

$$\begin{aligned}
\mathsf{LPO}_{\mathbb{P}} &:= \forall A : \mathbb{N} \to \mathbb{P}.\ (\forall n.\ An \vee \neg An) \to \quad (\exists n.\ An) \vee \neg(\exists n.\ An)\\
\mathsf{LPO}_{\mathbb{B}} &:= \forall f : \mathbb{N} \to \mathbb{B}. \qquad\qquad\qquad\quad (\exists n.\ fn = \mathsf{true}) \vee \neg(\exists n.\ fn = \mathsf{true})\\
\mathsf{LPO}_{\mathsf{PR}} &:= \forall f : \mathbb{N} \to \mathbb{B}.\ \mathsf{primitive\text{-}recursive}\ f \to (\exists n.\ fn = \mathsf{true}) \vee \neg(\exists n.\ fn = \mathsf{true})
\end{aligned}$$

Similarly to MP, $\mathsf{LPO}_{\mathbb{P}}$ implies $\mathsf{LPO}_{\mathbb{B}}$ and with the axiom of unique choice the converse is true. $\mathsf{LPO}_{\mathbb{B}}$ also implies $\mathsf{LPO}_{\mathsf{PR}}$ and with Church's Thesis this implication becomes an equivalence.

Notice that each variant of LPO implies its corresponding variant of MP since in general any classical proposition is double negation stable. The converse implication does not hold, as it would require that for every Σ^0_1 proposition P, the proposition $P \vee \neg P$ is also Σ^0_1. This is not the case in general due to the $\neg P$.

MLTT and $\mathrm{TT}^{\Box}_{\mathcal{C}}$ We separate the above variants of MP and LPO for MLTT [13] using: (1) a translation of MLTT to $\mathrm{TT}^{\Box}_{\mathcal{C}}$; and (2) separations of those variants for $\mathrm{TT}^{\Box}_{\mathcal{C}}$. $\mathrm{TT}^{\Box}_{\mathcal{C}}$ [3] is a family of effectful type theories parameterized by: (1) a choice operator $\mathcal{C}$, which is used to implement effectful computations; and (2) a $\Box$ modality to give meaning to effectful computations. In particular, in [7] we instantiated $\mathcal{C}$ with choice sequences [10, 19, 17, 16, 11, 20, 12] and $\Box$ with Beth coverings [21, 9, 6] to separate $\mathsf{MP}_{\mathbb{B}}$ and $\mathsf{MP}_{\mathsf{PR}}$ by exhibiting models of $\mathrm{TT}^{\Box}_{\mathcal{C}}$ that falsify

[1] github.com/vrahli/opentt/blob/77ec8765dcee83c061d567241b991a5ce261a5a8/types_lpo.lagda

$\mathsf{MP}_{\mathbb{B}}$ while satisfying $\mathsf{MP}_{\mathsf{PR}}$. A choice sequence can be seen as a reference to a list of values that can only be modified by extending it with further values. $\mathrm{TT}^{\Box}_{\mathcal{C}}$'s semantics is given in terms of a poset $\mathcal{W}$ of worlds, where a world can be seen as a list of choice sequences along with their values so far. While a given world w might not contain the n-th value of a choice sequence δ, Beth coverings allow giving meaning to δ by only requiring that choices are "eventually" available, i.e., given any infinite sequence of worlds (w.r.t. $\mathcal{W}$'s ordering relation) starting from w, there is a world in that sequence where δ's n-th choice is defined.

Separating $\mathsf{MP}_{\mathsf{PR}}$ and $\mathsf{MP}_{\mathbb{B}}$ As explained in [7], instantiating $\mathcal{C}$ with choice sequences of Booleans and $\Box$ with a Beth modality yields a model in which $\mathsf{MP}_{\mathsf{PR}}$ holds, while $\mathsf{MP}_{\mathbb{B}}$ does not.

Separating $\mathsf{MP}_{\mathbb{B}}$ and $\mathsf{MP}_{\mathbb{P}}$ To separate $\mathsf{MP}_{\mathbb{B}}$ and $\mathsf{MP}_{\mathbb{P}}$, we again instantiate $\Box$ with a Beth modality, but $\mathcal{C}$ with choice sequences of propositions, i.e. the empty $\mathbb{0}$ and unit $\mathbb{1}$ types. As a result, using a similar argument to the one used to negate $\mathsf{MP}_{\mathbb{B}}$ in [7] using Boolean choices, we now obtain that $\neg\mathsf{MP}_{\mathbb{P}}$ holds in this model. To see that $\mathsf{MP}_{\mathbb{B}}$, while not holding with Boolean choice sequences, holds with propositional choice sequences, we must show that terms that compute to Booleans cannot make use of a proposition in an essential way. This is done by way of a bisimulation on $\mathrm{TT}^{\Box}_{\mathcal{C}}$ terms which features congruence rules, as well as a rule relating the terms $\mathbb{0}$ and $\mathbb{1}$. Note that this result has appeared together with the contributions of [7] as [2].

Separating $\mathsf{LPO}_{\mathsf{PR}}$ and $\mathsf{LPO}_{\mathbb{B}}$ For the purposes of LPO, and in particular to falsify $\mathsf{LPO}_{\mathbb{B}}$, we once again require a Beth modality for instantiating $\Box$, and choice sequences for instantiating $\mathcal{C}$. In this setup, to negate $\mathsf{LPO}_{\mathbb{B}}$ we must show it does not hold in any extension w_1 of the current world w. To prove that $\mathsf{LPO}_{\mathbb{B}}$ holding at w_1 leads to a contradiction, it is enough to show that it does so in some extension w_2 of w_1. We pick w_2 to be w_1 extended with a currently empty choice sequence δ, which inhabits $\mathbb{N} \to \mathbb{B}$, and instantiate $\mathsf{LPO}_{\mathbb{B}}$ with δ. We must then prove that either of $(\exists n.\ \delta\ n)$ or $\neg(\exists n.\ \delta\ n)$ holding at w_2 leads to a contradiction. Assuming that $(\exists n.\ \delta\ n)$ holds at w_2 we obtain a contradiction by showing that there is a path from w_2 where δ's entries are always false. Assuming that $\neg(\exists n.\ \delta\ n)$ holds at w_2, i.e. $\exists n.\ \delta\ n$ does not hold at any extension of w_2, we obtain a contradiction by showing that there is an extension w_3 of w_2 where an entry of δ is set to true (for $\neg A$ to hold at w_0, it must be that A does not hold at any extension of w_0). The same reasoning does not work for $\mathsf{LPO}_{\mathsf{PR}}$. To see why, recall that primitive recursive functions are encoded by (pure) natural numbers. As a result, any f which is primitive recursive must be equal to some f_{pure} which does not use choice sequences at all, and the model can prove LPO for such f_{pure}, assuming LPO in the metatheory.

Separating $\mathsf{LPO}_{\mathbb{B}}$ and $\mathsf{LPO}_{\mathbb{P}}$ As for MP, $\mathsf{LPO}_{\mathbb{B}}$ and $\mathsf{LPO}_{\mathbb{P}}$ can be separated by instantiating $\mathcal{C}$ with choice sequences of propositions ($\mathbb{1}$ and $\mathbb{0}$) instead of Booleans (true and false).

Choice Sequences vs. References While [3] uses both choice sequences and references to a single Boolean to falsify LEM, the same cannot be done to falsify $\mathsf{MP}_{\mathbb{B}}$ or $\mathsf{LPO}_{\mathbb{B}}$. In this development, a reference to a single Boolean can be modified in further extensions of a world. Crucially, at any point, a reference can be made immutable, fixing its value in all future worlds, allowing us to falsify LEM similarly to the proof sketch of $\neg\mathsf{LPO}_{\mathbb{B}}$ (where the immutable choice true is used to obtain a contradiction from $\neg(\exists n.\ \delta\ n)$). References can be used to falsify LEM because in that case there is no need to prove that they inhabit a type that comes with a dependent elimination principle. As explained in [15], observational effects and unrestricted dependent elimination cannot coexist. For references, this manifests as the fact that they do not inhabit $\mathbb{B}$. As opposed to choice sequences whose entries are fixed once generated, reference cells' contents can change, precluding them from inhabiting types that come with dependent elimination principles, and therefore from using them to falsify $\mathsf{MP}_{\mathbb{B}}$ or $\mathsf{LPO}_{\mathbb{B}}$.

Concluding Remarks The above discussion naturally leads to two main questions: (1) is it possible to falsify $\mathsf{MP}_{\mathbb{B}}$ and $\mathsf{LPO}_{\mathbb{B}}$ using other forms of effectful computations than choice

sequences; and (2) what effectful computations that could be used to separate MP and LPO. Herbelin gives such a separation based on an exception mechanism [8].

References

[1] E. Bishop. *Foundations of constructive analysis*, volume 60. McGraw-Hill New York, 1967.

[2] Liron Cohen, Yannick Forster, Dominik Kirst, Bruno da Rocha Paiva, and Vincent Rahli. Separating Markov's Principles. In *Thirty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. ACM, 2024. doi:10.1145/3661814.3662104.

[3] Liron Cohen and Vincent Rahli. Constructing unprejudiced extensional type theories with choices via modalities. In *FSCD*, volume 228 of *LIPIcs*, pages 10:1–10:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FSCD.2022.10.

[4] Thierry Coquand and Bassel Mannaa. The independence of markov's principle in type theory. *Log. Methods Comput. Sci.*, 13(3), 2017. doi:10.23638/LMCS-13(3:10)2017.

[5] Hannes Diener. Constructive Reverse Mathematics. *arXiv:1804.05495 [math]*, 2020. arXiv:1804.05495.

[6] Michael A. E. Dummett. *Elements of Intuitionism*. Clarendon Press, second edition, 2000.

[7] Yannick Forster, Dominik Kirst, Bruno da Rocha Paiva, and Vincent Rahli. Markov's principles in constructive type theory. Presented at Types 2023, 2023.

[8] Hugo Herbelin. An intuitionistic logic that proves Markov's principle. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*. IEEE, July 2010. doi:10.1109/lics.2010.49.

[9] Verena Huber-Dyson and Georg Kreisel. *Analysis of Beth's semantic construction of intuitionistic logic*. Stanford University. Applied Mathematics and Statistics Laboratories, 1961.

[10] Stephen C. Kleene and Richard E. Vesley. *The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions*. North-Holland Publishing Company, 1965.

[11] Georg Kreisel and Anne S. Troelstra. Formal systems for some branches of intuitionistic analysis. *Annals of Mathematical Logic*, 1(3):229–387, 1970. URL: http://www.sciencedirect.com/science/article/pii/000348437090001X, doi:http://dx.doi.org/10.1016/0003-4843(70)90001-X.

[12] Joan R. Moschovakis. An intuitionistic theory of lawlike, choice and lawless sequences. In *Logic Colloquium'90: ASL Summer Meeting in Helsinki*, pages 191–209. Association for Symbolic Logic, 1993.

[13] Per Martin-Löf (notes by Giovanni Sambin). *Intuitionistic type theory*, volume 9. Bibliopolis Naples, 1984.

[14] Pierre-Marie Pédrot and Nicolas Tabareau. Failure is not an option. In *European Symposium on Programming*, pages 245–271. Springer, 2018. doi:10.1007/978-3-319-89884-1_9.

[15] Pierre-Marie Pédrot and Nicolas Tabareau. The fire triangle: how to mix substitution, dependent elimination, and effects. *Proc. ACM Program. Lang.*, 4(POPL):58:1–58:28, 2020. doi:10.1145/3371126.

[16] Anne S. Troelstra. *Choice sequences: a chapter of intuitionistic mathematics*. Clarendon Press Oxford, 1977.

[17] Anne S. Troelstra. Choice sequences and informal rigour. *Synthese*, 62(2):217–227, 1985.

[18] Anne S. Troelstra and Dirk van Dalen. Constructivism in mathematics. vol. i. *Studies in Logic and the Foundations of Mathematics*, 26, 1988.

[19] Mark van Atten and Dirk van Dalen. Arguments for the continuity principle. *Bulletin of Symbolic Logic*, 8(3):329–347, 2002. URL: http://www.math.ucla.edu/~asl/bsl/0803/0803-001.ps.

[20] Wim Veldman. Understanding and using Brouwer's continuity principle. In *Reuniting the Antipodes — Constructive and Nonstandard Views of the Continuum*, volume 306 of *Synthese Library*, pages

285–302. Springer Netherlands, 2001. URL: http://dx.doi.org/10.1007/978-94-015-9757-9_24, doi:10.1007/978-94-015-9757-9_24.

[21] Beth E. W. Semantic construction of intuitionistic logic. *Journal of Symbolic Logic*, 22(4):363–365, 1957.

Post's Problem and the Priority Method in CIC

Haoyi Zeng[1], Yannick Forster[2], and Dominik Kirst[3]

[1] Saarland University, Germany
[2] Inria Paris, France
[3] Ben-Gurion University, Israel

Abstract

We describe our formalisation of a solution to Post's Problem using the priority method in synthetic computability theory. Compared to a usual, analytic approach employing explicit models of computation, a synthetic approach axiomatically considers all functions $\mathbb{N}\to\mathbb{N}$ to be computable. We work in the Calculus of Inductive Constructions and mechanise all proofs in the Coq proof assistant.

Background Posed by Emil Post in 1944 [16], Post's problem asks whether there are semi-decidable, yet undecidable predicates that are not Turing-reducible from the halting problem. Post's problem has been a crucial open question driving research in computability theory until a breakthrough came with the positive solution by Friedberg and Muchnik [9, 14] in 1956/57. They introduced independently what is now known as the priority method, in order to show that there exist two semi-decidable, Turing-reduction incomparable degrees. The priority method has since become a cornerstone in the field of computability theory, essential for exploring and understanding the structure of computability degrees [12, 13, 20].

Today, virtually every textbook on computability theory (e.g. [23, 18, 22, 15]) discusses Post's problem and the use of the priority method. From the perspective of machine-checked proofs, the interactive theorem proving community has successfully formalised cutting-edge mathematics in several proof assistants, however, formalising computability theory remains a challenge. A main intricacy is the use of models of computation for formal proofs [8], due to the level of uninteresting details involved that stay invisible on paper.

A solution is proposed by *synthetic* computability [17, 1], which exploits constructive mathematics as its foundation. In a synthetic approach to computability theory, *every* function is considered computable. For instance, the decidability of a predicate $P : X \to \mathbb{P}$ is now defined as $\exists f : X \to \mathbb{B}.\ P\ x \leftrightarrow f\ x = \mathsf{true}$, which eliminates the need to show f computable in a model.

Since the Calculus of Inductive Constructions (CIC) is a constructive system where the law of excluded middle stays consistent even when assuming axioms for synthetic computability, it is natural to ask questions of constructive reverse analysis. The formalisation and constructive analysis of synthetic computability have received attention in recent years, encompassing the study of many-one reduction, Post's theorem, the arithmetic hierarchy, and Turing computability [3, 4, 7, 5, 11]. In 2021, Andrej Bauer has posed the challenge to "give a synthetic proof of Friedberg-Mucnik theorem" [2].

Due to the historic importance of Post's problem, we consider the successful completion of this challenge a milestone in synthetic computability and the formalisation of computability theory. Notably, the necessity for the priority method will play a crucial role in the further development of machine-checked synthetic computability.

Low Simple Predicate Soare's solution to Post's problem [21] constructs a so-called low simple predicate directly, rather than proving the full Friedberg-Muchnik theorem constructing two incomparable predicates. Since a synthetic notion of simple predicates has been defined in previous work [4], we here focus on the aspect of lowness. A predicate P is *low* if its Turing jump P' is reducible to the halting problem H, ie. $P' \preceq_T H$.

This leaves us with three key questions: 1) What is the simplest technique to establish that a predicate is reducible to the halting problem? 2) How can we formalise the priority method synthetically? 3) How can we instantiate the method to obtain a low simple predicate? We address these questions within the framework of synthetic computability, following Soare [23].

1) Limit computablility We conjecture that the simplest way to give a reduction to the halting problem in our case is by using the notion of limit computability [19, 10]. It is equivalent to being reducible to the halting problem, but easier to establish. We use the notion of the characteristic relation $\hat{P} : X \to \mathbb{B} \to \mathbb{P}$ of a predicate P, where $\hat{P}\ x\ \mathsf{true} \leftrightarrow P\ x$ and $\hat{P}\ x\ \mathsf{false} \leftrightarrow \neg P\ x$. We call $P : X \to \mathbb{P}$ *limit-computable* if there is a function $f : X \to \mathbb{N} \to \mathbb{B}$ such that

$$\forall x\ b.\ \hat{P}(x, b) \leftrightarrow \exists n.\ \forall m > n.\ f(x, m) = b.$$

Based on this definition and the previously developed Turing computability in synthetic computability [6], we have already verified the limit lemma, which states that a predicate P is limit-computable if and only if P is reducible to H.

This brings us to the position that we just need to prove limit computability of P' to prove lowness of the to-be-constructed simple predicate P.

2) The finite injury priority method The priority method can be used to construct semi-decidable predicates P satisfying an infinite set of requirements R_e. In order to construct P, the following inductive predicate computably binds a list L to each stage n, where then $P\ x \coloneqq \exists n\ L.\ n \rightsquigarrow L \wedge x \in L$. The predicate is parameterized by an extension $\gamma : \mathbb{N}^* \to \mathbb{N} \to \mathbb{N} \to \mathbb{P}$, which is used to determine whether an element can enter the predicate at step $n + 1$ and can recursively depend on L that fulfils $n \rightsquigarrow L$.

$$\frac{}{0 \rightsquigarrow [\,]} \qquad \frac{n \rightsquigarrow L \quad \gamma_n^L\ x}{n + 1 \rightsquigarrow x :: L} \qquad \frac{n \rightsquigarrow L \quad \forall x.\ \neg\ \gamma_n^L\ x}{n + 1 \rightsquigarrow L}$$

In this work, we consider the simplest form of the priority method, the finite injury priority method, as originally developed by Friedberg and Muchnik, which is sufficient for constructing low simple predicates. The term "finite injury" refers to the possibility that some of the requirements in R_e on P might be broken during the construction of P due to the satisfaction of others, also known as "injury". However, because the injury is finite, P will ultimately satisfy all R_e as required.

3) Instantiation to a low simple predicate To construct a low simple predicate, γ is instantiated to an appropriate extension.

We took advantage of formalised proof to achieve this goal by constructing the proof in a modular way. First, we considered an extension γ, ensuring that some requirements R_e are met, which implies P is a simple predicate.

In the second step, we observed that certain conditions in γ can be abstracted to any convergent function ω. According to Soare's construction, we refined the results in oracle computability, providing a suitable definition to concretize this function ω, so that some requirements N_e will be finitely injured at some stages, but are eventually satisfied, which entail that this predicate is low.

Future work Currently, on top of a typical formulation of Church's thesis for synthetic computability [1], we assume the law of excluded middle for all proofs, i.e. full classical logic. As a next step, we plan to weaken this assumptions as much as possible, e.g. by restricting the logical complexity of propositions in axioms, and then perform a constructive reverse analysis.

Acknowledgements We would like to thank Gert Smolka for the discussions, and the anonymous reviewers for their helpful feedback.

References

[1] Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006. doi:10.1016/j.entcs.2005.11.049.

[2] Andrej Bauer. Synthetic mathematics with an excursion into computability theory (slide set). *University of Wisconsin Logic seminar*, 2020. URL: http://math.andrej.com/asset/data/madison-synthetic-computability-talk.pdf.

[3] Yannick Forster. Computability in constructive type theory. 2021. doi:10.22028/D291-35758.

[4] Yannick Forster and Felix Jahn. Constructive and synthetic reducibility degrees: Post's problem for many-one and truth-table reducibility in Coq. In *CSL 2023-31st EACSL Annual Conference on Computer Science Logic*, 2023. doi:10.4230/LIPIcs.CSL.2023.21.

[5] Yannick Forster and Dominik Kirst. Synthetic Turing reducibility in constructive type theory. 28th International Conference on Types for Proofs and Programs (TYPES 2022), 2022. URL: https://types22.inria.fr/files/2022/06/TYPES_2022_paper_64.pdf.

[6] Yannick Forster, Dominik Kirst, and Niklas Mück. Oracle computability and Turing reducibility in the calculus of inductive constructions. In Chung-Kil Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 155–181. Springer, 2023. doi:10.1007/978-981-99-8311-7_8.

[7] Yannick Forster, Dominik Kirst, and Niklas Mück. The Kleene-Post and Post's theorem in the calculus of inductive constructions. 2024. doi:10.4230/LIPIcs.CSL.2024.29.

[8] Yannick Forster, Fabian Kunze, and Maximilian Wuttke. Verified programming of Turing machines in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 114–128, 2020. doi:10.1145/3372885.3373816.

[9] Richard M Friedberg. Two recursively enumerable sets of incomparable degrees of unsolvability (solution of Post's problem, 1944). *Proceedings of the National Academy of Sciences*, 43(2):236–238, 1957. doi:10.1073/pnas.43.2.236.

[10] E Mark Gold. Limiting recursion. *The Journal of Symbolic Logic*, 30(1):28–48, 1965. doi:10.2307/2270580.

[11] Dominik Kirst, Yannick Forster, and Niklas Mück. Synthetic Versions of the Kleene-Post and Post's Theorem. 28th International Conference on Types for Proofs and Programs (TYPES 2022), 2022. URL: https://types22.inria.fr/files/2022/06/TYPES_2022_paper_65.pdf.

[12] Alistair H Lachlan. The priority method for the construction of recursively enumerable sets. In *Cambridge Summer School in Mathematical Logic: Held in Cambridge/England, August 1–21, 1971*, pages 299–310. Springer, 2006. doi:10.1007/BFb0066779.

[13] Manuel Lerman and Robert Soare. *d*-simple sets, small sets, and degree classes. *Pacific Journal of Mathematics*, 87(1):135–155, 1980. doi:10.2140/pjm.1980.87.135.

[14] Albert A Muchnik. On the unsolvability of the problem of reducibility in the theory of algorithms. In *Dokl. Akad. Nauk SSSR*, volume 108, pages 194–197, 1956.

[15] Piergiorgio Odifreddi. *Classical recursion theory: The theory of functions and sets of natural numbers*. Elsevier, 1992. doi:10.2307/2274492.

[16] Emil L Post. Recursively enumerable sets of positive integers and their decision problems. 1944. doi:10.1090/s0002-9904-1944-08111-1.

[17] Fred Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48(3):797–803, 1983. doi:10.2307/2273473.

[18] Hartley Rogers Jr. *Theory of recursive functions and effective computability*. MIT press, 1987. doi:10.2307/2271523.

[19] Joseph R Shoenfield. On degrees of unsolvability. *Annals of mathematics*, 69(3):644–653, 1959. doi:10.2307/1970028.

[20] Stephen G Simpson. Degrees of unsolvability: a survey of results. In *Studies in Logic and the Foundations of Mathematics*, volume 90, pages 631–652. Elsevier, 1977. doi:10.1016/S0049-237X(08)71117-0.

[21] Robert I Soare. The infinite injury priority method1. *The Journal of Symbolic Logic*, 41(2):513–530, 1976. doi:10.1017/S0022481200051598.

[22] Robert I Soare. Recursively enumerable sets and degrees. *Bulletin of the American Mathematical Society*, 84(6):1149–1181, 1978.

[23] Robert I Soare. Turing computability. *Theory and Applications of Computability. Springer*, 2016. doi:10.1007/978-3-642-31933-4.

Session 5: Proof Assistant Implementation

Type-Based Termination Checking in Agda

Kanstantsin Nisht[1] and Andreas Abel[2]

[1] Chalmers University of Technology, Gothenburg, Sweden
nisht@student.chalmers.se
[2] Department of Computer Science and Engineering,
Chalmers and Gothenburg University, Gothenburg, Sweden
andreas.abel@cse.gu.se

Abstract

We present a description of a type-based termination checker for the dependently-typed language Agda. The checker is based on the theory of sized types, i.e., types indexed by an abstract well-ordered type of sizes. It uses a variant of strongly-normalizing higher-order polymorphic lambda calculus with pattern and copattern matching as the semantic foundation. The checker is implemented for Agda, where it is used to certify termination of recursive functions and productivity of corecursive functions.

1 Introduction

Termination is an important problem in computer science. In dependently-typed programming languages, termination checking is used to ensure that the whole process of type checking is terminating. If such languages double as logics—as for instance in the proof assistants Agda, Lean, and Rocq[1]—termination is even required for consistency.

Currently, termination checkers take a conservative approach and allow only recursive calls with an argument that is *structurally* smaller than some function parameter. However, this approach is limited. Consider the following functions written in Agda:

```
data Nat : Set where
  zero : Nat
  suc : Nat → Nat

minus : Nat → Nat → Nat
minus zero y = zero
minus x zero = x
minus (suc x) (suc y) = minus x y

div : Nat → Nat → Nat
div zero y = zero
div (suc x) y = suc (div (minus x y) y)
```

Function div implements division of x by $y+1$. This function is not accepted by the termination checker of Agda, because minus x y is not structurally smaller than x. One workaround for this is the use of sized types, but they complicate the theory of Agda with subtyping, and they require any function that uses div to also use sized types.

[1] The proof assistant formerly known as Coq.

Another area of improvement here is the productivity checking of coinductive definitions, which currently relies on guardedness condition (Coquand, 1993) or explicit later-modalities (Nakano, 2000).

2 Contribution

We present System $F_\omega^{\mathrm{cop,SCT}}$, which is a modification of System F_ω^{cop} (Abel and Pientka, 2016). System $F_\omega^{\mathrm{cop,SCT}}$ is a polymorphic higher-order lambda calculus with pattern and copattern matching, which uses well-founded flavor of sized types and the size-change principle (Jones et al., 2001) to ensure strong normalization. We provide a proof of strong normalization that is based on Girard-Tait reducibility candidates (Girard, 1971) combined with well-founded induction on size-change call graphs over syntactic size annotations interpreted as ordinals.

We also present an algorithm for inference of size annotations for System $F_\omega^{\mathrm{cop,SCT}}$ and prove its soundness (Nisht, 2024). Our algorithm has linear time complexity in the size of the syntax.

As an engineering contribution, we provide an implementation of the specified algorithm for Agda (as of the moment of writing, it exists in the form of pull request). Our algorithm shows acceptable performance and relative independence from other features of Agda. It works for both inductive and coinductive definitions in a unified way.

3 Related and Future Work

Our work is based on Abel and Pientka (2016), which is a further development of Abel (2006), extending it with patterns and copatterns. Our theoretical development contributes the use of the size-change principle (Jones et al., 2001) in this system, which allows to remove the syntactic notion of *termination measure*, which results in lighter syntax of the system.

In contrast to other developments of higher-order extensions of the size-change principle (Wahlstedt, 2007), our system features type-based relation between terms, thus extending the existing syntax-based comparisons.

The problem of implicit type-based termination checker was attempted to be solved for Rocq (Chan et al., 2023), but the conclusion there was that sized typing is not practical: the earlier works on sized types (Barthe et al., 2005) described a complicated size algebra, and the corresponding algorithm of termination checking was too complex to be used in mature proof assistants. We use a simpler structure on sized types, which allows us to process size dependencies faster, although we did not manage to prove the completeness of our algorithm at the moment.

The main direction of future work is to extend our theory to dependent types, as it is the biggest limitation in expressivity of our termination checker in Agda.

References

A. Abel. *Type-based termination: A Polymorphic Lambda-Calculus with Sized Higher-Order Types*. PhD thesis, Ludwig-Maximilians-Universität München, 2006.

A. Abel and B. Pientka. Well-founded recursion with copatterns and sized types. *Journal of Functional Programming*, 26:e2, 2016. doi:10.1017/S0956796816000022.

G. Barthe, B. Grégoire, and F. Pastawski. Practical inference for type-based termination in a polymorphic setting. In P. Urzyczyn, editor, *Typed Lambda Calculi and Applications, 7th International Conference, TLCA 2005, Nara, Japan, April 21-23, 2005, Proceedings*, volume 3461 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2005. doi:10.1007/11417170_7.

J. Chan, Y. Li, and W. J. Bowman. Is sized typing for Coq practical? *Journal of Functional Programming*, 33:e1, 2023. doi:10.1017/S0956796822000120.

T. Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs, International Workshop TYPES'93, Nijmegen, The Netherlands, May 24-28, 1993, Selected Papers*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78, Berlin, Heidelberg, 1993. Springer. ISBN 3540580859. doi:10.1007/3-540-58085-9_72.

J.-Y. Girard. Une extension de l'interpretation de Gödel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. In J. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. Elsevier, 1971. doi:10.1016/S0049-237X(08)70843-7.

N. D. Jones, C. S. Lee, and A. M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, page 81–92, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133367. doi:10.1145/360204.360210.

H. Nakano. A modality for recursion. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 255–266, USA, 2000. IEEE Computer Society. ISBN 0769507255. doi:10.1109/LICS.2000.855774.

K. Nisht. Type-based termination checking in Agda. Draft, 2024. URL `https://knisht.github.io/agda/msc.pdf`.

D. Wahlstedt. *Dependent Type Theory with Parameterized First-Order Data Types and Well-Founded Recursion*. Chalmers Tekniska Högskola, Sweden, 2007. URL `https://research.chalmers.se/en/publication/45188`.

Size-preserving dependent elimination

Hugo Herbelin

IRIF, CNRS, Université de Paris, Inria, France
hugo.herbelin@inria.fr

Elimination rules of positive connectives refine the context of a proof or computation with the different possible values of the object being eliminated, such as being on one side or the other of a disjunction or such as being a tuple of objects.

Sometimes, we may require that properties from the object being eliminated do not pervade in the context. A typical example is when the object being eliminated is up to a quotient, and we want the proof or computation to preserve the quotient.

Here, we consider another form of restriction intended to ensure that the elimination does not change the "size" of the proof or computation.

The motivation is multiple. First, it has been discovered in 2013 that the guard condition implemented in Coq was inconsistent with propositional extensionality and the analysis of the incompatibility revealed that transporting an expression along propositional extensionality had indeed the ability of changing the size of the expression, an observation which is reminiscent of the homotopic interpretation of type extensionality as an equivalence. A restriction of the conditions under which an elimination propagates the size was then implemented to prevent the inconsistency.

On the other side, while implementing a small-inversion-based variant [MS13] of Goguen-McBride-McKinna's and Cockx' compilation [GMM06, Coc17, Soz10] of dependent pattern-matching [Coq92], it was observed that the new criterion was too restrictive [Mar17]. Other limitations were also reported on the Coq bug tracker [Soz23].

We conjecture the existence of a compromise that exactly captures the restriction needed to preserve the compatibility with propositional extensionality while resolving the known limitations.

First, we briefly recall the main parts[1] of the guardedness check in the variant of the Calculus of Inductive Constructions implemented in Coq for fixpoints up to 2013 (i.e. up to Coq 8.4). We call *stack* any sequence $t_1 \cdot \ldots \cdot t_n$ of terms and say that a well-typed recursive expression $\mathit{fix}\, f\, (x : I) : T := c$ is *guarded in context* Γ when c is guarded for f on x in $\Gamma, f : \Pi(x : I).T, x : I$ relatively to an empty set of smaller variables and empty stack where c *is guarded for* f *on* x *in* Γ *relatively to a set* Ξ *of smaller variables and stack* π, shortly $\Gamma \mid f \mid x \mid \Xi \vdash c \mid \pi$ guarded, is defined by means of an auxiliary judgement $\Gamma \mid f \mid x \mid \Xi \vdash c$ smaller and the following inference rules[2]:

$$\frac{y \in \Xi}{\Gamma \mid f \mid x \mid \Xi \vdash y \text{ smaller}} \qquad \frac{\Gamma \mid f \mid x \mid \Xi \vdash t \text{ smaller}}{\Gamma \mid f \mid x \mid \Xi \vdash t\, u \text{ smaller}} \qquad \frac{\Gamma \mid f \mid x \mid \Xi \vdash t \text{ smaller}}{\Gamma \mid f \mid x \mid \Xi \vdash f \mid t \text{ guarded}}$$

$$\frac{\Gamma, y : T \mid f \mid x \mid \Xi, y \vdash u \mid \pi \text{ guarded} \qquad \Gamma \mid f \mid x \mid \Xi \vdash t \text{ smaller}}{\Gamma \mid f \mid x \mid \Xi \vdash \lambda(y : T).\, u \mid t \cdot \pi \text{ guarded}} \qquad \frac{\Gamma, y : T \mid f \mid x \mid \Xi \vdash u \mid \pi \text{ guarded}}{\Gamma \mid f \mid x \mid \Xi \vdash \lambda(y : T).\, u \mid t \cdot \pi \text{ guarded}}$$

$$\frac{\Gamma, y : T \mid f \mid x \mid \Xi \vdash u \mid \text{ guarded}}{\Gamma \mid f \mid x \mid \Xi \vdash \lambda(y : T).\, u \mid \text{ guarded}} \qquad \frac{\Gamma \mid f \mid x \mid \Xi \vdash t \mid u \cdot \pi \text{ guarded}}{\Gamma \mid f \mid x \mid \Xi \vdash t\, u \mid \pi \text{ guarded}}$$

$$\frac{\Gamma \mid f \mid x \mid \Xi \vdash c \mid \text{ guarded} \qquad \Gamma, \overrightarrow{y : U}, y : J\, \overrightarrow{y} \mid f \mid x \mid \Xi \vdash P \mid \text{ guarded} \qquad \Gamma, \overrightarrow{z_k : V_k} \mid f \mid x \mid \Xi, |\overrightarrow{z_k}| \vdash u_k \mid \pi \text{ guarded}}{\Gamma \mid f \mid x \mid \Xi \vdash \mathsf{match}\ c\ \mathsf{as}\ y\ \mathsf{in}\ J\ (\overrightarrow{y : U})\ \mathsf{return}\ P\ \mathsf{with}\ C_k(\overrightarrow{z_k : V_k})\ \Rightarrow u_k\ \mathsf{end} \mid \pi \text{ guarded}}$$

where, in the Coq-like-syntax-based rule for dependent elimination, J is an inductive family and $|\overrightarrow{z_k}|$ selects the subset of $\overrightarrow{z_k}$ whose type ends with an inductive type.

From Coq 8.5, the guard was modified to prevent the inconsistency with propositional extensionality [Col14]. A way to present the change is by introducing an "undefined" term $\bot$ serving to mask the parts of the recursive structure of an inductive type that should not be considered as smaller. Then,

[1] Intentionally simplified to focus on the point under concern.

[2] The role of the stack is to propagate smallerness from outside of a case analysis to the branches of the case analysis.

if T and U possibly contain $\bot$ at some places, we write $T \cap U$ for the type obtained by setting $\bot$ at every occurrence where T and U differ (or are both already $\bot$). We also write $T \leq U$ is T has $\bot$ at all occurrences where U has already $\bot$. Even though the purpose of the inference rules is not to check typing (which is assumed to be done beforehand), the rules now carry a type, leading to a judgement of the form $\Gamma \mid f \mid x : I \mid x_1 : T_1, ..., x_n : T_n \vdash c : U \mid \pi$ guarded, with π a sequence of typed terms.

The inference rule for the base case of smallerness and for dependent case analysis are then changed as follow:

$$\frac{y : U \in \Xi \qquad T \leq U}{\Gamma \mid f \mid x : T \mid \Xi \vdash y \text{ smaller}}$$

$$\frac{\begin{array}{l}\Gamma \mid f \mid x : I \mid \Xi \vdash c \text{ guarded} \\ \Gamma, \overrightarrow{y : U}, y : J\ \vec{y} \mid f \mid x \mid \Xi \vdash P \text{ guarded} \qquad \Gamma \mid f \mid x \mid \Xi, |\overrightarrow{z_k : V_k}| \vdash u_k : P[\overrightarrow{y := \bot}, y := \bot] \cap Q \mid \pi \text{ guarded}\end{array}}{\Gamma \mid f \mid x : I \mid \Xi \vdash \mathsf{match}\ c\ \mathsf{as}\ y\ \mathsf{in}\ J\ (\overrightarrow{y : U})\ \mathsf{return}\ P\ \mathsf{with}\ C_k(\overrightarrow{z_k : V_k})\ \Rightarrow u_k\ \mathsf{end} : Q \mid \pi \text{ guarded}}$$

that is, by discarding all possible contributions of the indices to guardedness in the elimination predicate. Also, the rules popping a smaller argument $t : T$ from the stack to bind it to a variable $x : U$ declares x as smaller with type $T \cap U$.

We now come to our refinement of the 2013 criterion [Her21]. The idea is not to restrict all dependencies in the indices of the family in the elimination predicate, but only the dependencies in indices that are types, or type families, that is, for t a term, we define its *size-preserving mask* $\downarrow t$ compositionally except that all occurrences of an inductive type or inductive family J in t are replaced by $\bot$, since, after all, only inductive types are able to contribute to smallerness, so it is enough to mask inductive types. Then, the rule for dependent case analysis is refined into:

$$\frac{\begin{array}{l}\Gamma \mid f \mid x : I \mid \Xi \vdash c \text{ guarded} \\ \Gamma, \overrightarrow{y : U}, y : J\ \vec{y} \mid f \mid x \mid \Xi \vdash P \text{ guarded} \\ \Gamma \mid f \mid x \mid \Xi, |\overrightarrow{z_k : V_k}| \vdash u_k : P[\overrightarrow{y :=\downarrow t \cap \downarrow v_k}, y :=\downarrow c \cap (C_k\ \vec{z_k})] \cap Q \mid \pi \text{ guarded}\end{array}}{\Gamma \mid f \mid x : I \mid \Xi \vdash \mathsf{match}\ c\ \mathsf{as}\ y\ \mathsf{in}\ J\ (\overrightarrow{y : U})\ \mathsf{return}\ P\ \mathsf{with}\ C_k(\overrightarrow{z_k : V_k}) : J\ \vec{v_k}\ \Rightarrow u_k\ \mathsf{end} : Q \mid \pi \text{ guarded}}$$

Our main conjecture is then:

Conjecture: Propositional extensionality is not refutable in the Calculus of Inductive Constructions equipped with the modified rule.

Additionally, we conjecture that the same kind of idea would support convertibility-based elimination of equality proofs $t = u$ in strict propositions without leading to a failure of normalisation in the presence of an impredicative sort [AC20]: instead of masking inductive types in the indices, our guess is that the following reduction rule, where $\Downarrow$ masks subterms of type an impredicative sort, preserves normalisation[3]:

$$\frac{t \equiv u \qquad P[z :=\Downarrow u][y :=\Downarrow e] \text{ is } \bot\text{-free}}{(\mathsf{match}\ e : t = u\ \mathsf{as}\ y\ \mathsf{in}\ \_ = z\ \mathsf{return}\ P\ \mathsf{with}\ \mathsf{refl}\ \Rightarrow v\ \mathsf{end}) \to v}$$

The motivation for our guess is that the example in [AC20] crucially relies on eliminating a proof of extensional equality, which the modified reduction rule prevents.

[3] In Coq, this rule, without our restriction, is activated by setting the option Definitional UIP.

References

[AC20] Andreas Abel and Thierry Coquand. Failure of normalization in impredicative type theory with proof-irrelevant propositional equality. *Log. Methods Comput. Sci.*, 16(2), 2020.

[Coc17] Jesper Cockx. *Dependent Pattern Matching and Proof-Relevant Unification.* PhD thesis, KU Leuven, 2017.

[Col14] Collective. Coq modified guard checker. https://github.com/coq/coq/blob/master/kernel/inductive.ml, starting with commit 35e47b6be8d9e, 2014.

[Coq92] Thierry Coquand. Pattern Matching with Dependent Types. In B. Nordström, K. Petersson, and G. Plotkin, editors, *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, 1992.

[GMM06] Healfdene Goguen, Conor McBride, and James McKinna. Eliminating dependent pattern matching. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*, pages 521–540. Springer, 2006.

[Her21] Hugo Herbelin. Optimizing the propagation of the guard condition under a match on a type with indices by discarding only the type information from the indices. https://github.com/coq/coq/pull/14359, 2021.

[Mar17] Thierry Martinez. Fixpoint guard when generalization occurs on strict subterms. https://github.com/coq/coq/issues/5530, 2017.

[MS13] Jean-François Monin and Xiaomu Shi. Handcrafted inversions made operational on operational semantics. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 2013.

[Soz10] Matthieu Sozeau. Equations: A dependent pattern-matching compiler. In *ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 419–434. Springer, 2010.

[Soz23] Matthieu Sozeau. Guard checking regression. https://github.com/coq/coq/issues/17062, 2023.

How much do System T recursors lift to dependent types?

Hugo Herbelin

IRIF, CNRS, Université de Paris, Inria, France
hugo.herbelin@inria.fr

Consider Peano numbers (on the left) and their canonical realisability predicate (on the right):

$$\frac{}{\Gamma \vdash \mathsf{zero} : \mathbb{N}} \qquad \frac{\Gamma \vdash t : \mathbb{N}}{\Gamma \vdash \mathsf{succ}\, t : \mathbb{N}} \qquad \frac{}{\Gamma \vdash \mathsf{iszero} : \mathsf{is}\mathbb{N}\, \mathsf{zero}} \qquad \frac{\Gamma \vdash p : \mathsf{is}\mathbb{N}\, t}{\Gamma \vdash \mathsf{issucc}\, t\, p : \mathsf{is}\mathbb{N}\, (\mathsf{succ}\, t)}$$

Consider the standard recursor in $\mathbb{N}$:

$$\frac{\begin{array}{ll}\Gamma, m : \mathbb{N} \vdash A : U & \Gamma \vdash u : A[m := \mathsf{zero}] \\ \Gamma \vdash t : \mathbb{N} & \Gamma, n : \mathbb{N}, a : A[m := n] \vdash v : A[n := \mathsf{succ}\, n]\end{array}}{\Gamma \vdash (\mathsf{match}\ t\ \mathsf{with}\ [\mathsf{zero} \to u \,|\, \mathsf{succ}\, n \to_a v]) : A[m := t]}$$

and the standard recursor in $\mathsf{is}\mathbb{N}$:

$$\frac{\begin{array}{ll}\Gamma, n : \mathbb{N}, y : \mathsf{is}\mathbb{N}\, n \vdash A : U & \Gamma \vdash u : A[n := \mathsf{zero}][y := \mathsf{iszero}] \\ \Gamma \vdash p : \mathsf{is}\mathbb{N}\, t & \Gamma, n : \mathbb{N}, x : \mathsf{is}\mathbb{N}\, n, a : A[n := n][y := x] \vdash v : A[n := \mathsf{succ}\, n][y := \mathsf{issucc}\, n\, x]\end{array}}{\Gamma \vdash (\mathsf{match}\ t\ \mathsf{with}\ [\mathsf{iszero} \to u \,|\, \mathsf{issucc}\, n\, x \to_a v]) : A[n := t][y := p]}$$

Consider the following (trivial) program:

$$m : \mathbb{N} \vdash (\mathsf{match}\ m\ \mathsf{with}\ [\mathsf{zero} \to m \,|\, \mathsf{succ}\, n \to_a m]) : \mathbb{N}$$

It happens that the canonically derived equivalent program in $\mathsf{is}\mathbb{N}$ is ill-typed:

$$m : \mathbb{N}, y : \mathsf{is}\mathbb{N}\, m \not\vdash (\mathsf{match}\ y\ \mathsf{with}\ [\mathsf{iszero} \to y \,|\, \mathsf{issucc}\, n\, x \to_a y]) : \mathsf{is}\mathbb{N}\, m$$

Indeed, in the case of $\mathbb{N}$, the branches do not have dependencies, so the term being matched, m, can be reused. But in $\mathsf{is}\mathbb{N}$, branches have dependent types, so the term being matched, p, cannot be reused.

We describe in the next sections two standard solutions to recove typing but each of them as drawbacks. Then, we argue that an alternative approach is to introduce a notion of as-patterns in recursors, leading to a generalisation of the typing rule for induction where not only the type but also the body can depend on the term being matched.

1 Solution 1: expanding the dependent instances

The first solution is to expand the term being matched, the way it would be evaluated in each branch:

$$m : \mathbb{N}, p : \mathsf{is}\mathbb{N}\, m \vdash (\mathsf{match}\ p\ \mathsf{with}\ [\mathsf{iszero} \to \mathsf{iszero} \,|\, \mathsf{issucc}\, n\, x \to_a \mathsf{issucc}\, n\, x]) : \mathsf{is}\mathbb{N}\, m$$

It has the expected semantics, that is, if $p \equiv \mathsf{issucc}\, t\, q$, for $\equiv$ denoting convertibility, then, treating the ill-typed term as untyped, we correctly have:

$$\mathsf{match}\ p\ \mathsf{with}\ [\mathsf{iszero} \to \mathsf{iszero} \,|\, \mathsf{issucc}\, n\, x \to_a \mathsf{issucc}\, n\, x] \equiv \mathsf{match}\ p\ \mathsf{with}\ [\mathsf{iszero} \to p \,|\, \mathsf{issucc}\, n\, x \to_a p]$$

But the behaviour is different in terms of sharing the representation of the underlying constructors in the reduction: if the 3 occurrences of p have their representation shared (as in call-by-value), then the sharing in memory is lost. In particular, in guard-based typing systems such as Coq, the information that y in the branches of our running example is of the same size as y being matched is lost.

2 Solution 2: generalizing the term being matched

Another solution is to generalise the term being matched to change its type:

$$m : \mathbb{N}, y : \mathsf{is}\mathbb{N}\, m \vdash (\mathsf{match}\ y\ \mathsf{with}\ [\mathsf{iszero} \to \lambda y^{\mathsf{is}\mathbb{N}\,\mathsf{zero}}.y \mid \mathsf{issucc}\, n\, x \to_a \lambda y^{\mathsf{is}\mathbb{N}\,(\mathsf{succ}\, n)}.y])\, y : \mathsf{is}\mathbb{N}\, m$$

which has also the expected semantics, that is, if $p \equiv \mathsf{issucc}\, t\, q$, then, treating the ill-typed term as untyped:

$$\mathsf{match}\ p\ \mathsf{with}\ [\mathsf{iszero} \to \lambda y.y \mid \mathsf{issucc}\, n\, x \to_{\lambda y.y} \mathsf{issucc}\, n\, x]\, p \equiv \mathsf{match}\ p\ \mathsf{with}\ [\mathsf{iszero} \to p \mid \mathsf{issucc}\, n\, x \to_a p]$$

This time, if all occurrences of p are shared in memory (as in call-by-value), they remain shared before and after reduction in the encoding. Also, for a guard-based typing system, if sizes are propagated across generalisations, as it is the case in Coq, the encoding preserves sizes.

However, the generalisation technically requires a stronger system than intended. For instance, if the program is primitive recursive, its encoding requires to go out of primitive recursion.

3 Solution 3: adding as-variables to recursors

In pattern-matching compilation (e.g. OCaml, Haskell, Coq, ...), it is common to use as-variables. In Coq, these as-variables were thought for long as syntactic sugar and emulated with one or the other encoding above. We argue that giving a foundational status to as-variables solves our problem. We propose the following typing rules which differ from ordinary induction only in that not only the type but also the body of branches can depend on the term being matched:

$$\frac{\begin{array}{ll}\Gamma, m : \mathbb{N} \vdash A : U & \Gamma, m : \mathbb{N} \vdash u : A[m := \mathsf{zero}] \\ \Gamma \vdash t : \mathbb{N} & \Gamma, n : \mathbb{N}, a : A[m := n], m : \mathbb{N} \vdash v : A[m := \mathsf{succ}\, n]\end{array}}{\Gamma \vdash (\mathsf{match}\ t\ \mathsf{as}\ m\ \mathsf{with} \begin{bmatrix}\mathsf{zero} & \to u \\ \mathsf{succ}\, n & \to_a v\end{bmatrix}) : A[m := t]}$$

$$\frac{\begin{array}{ll}\Gamma, n : \mathbb{N}, y : \mathsf{is}\mathbb{N}\, n \vdash A : U & \Gamma, y : \mathsf{is}\mathbb{N}\,\mathsf{zero} \vdash u : A[n := \mathsf{zero}][y := \mathsf{iszero}] \\ \Gamma \vdash p : \mathsf{is}\mathbb{N}\, t & \Gamma, n : \mathbb{N}, x : \mathsf{is}\mathbb{N}\, n, a : A[y := x], y : \mathsf{is}\mathbb{N}\,(\mathsf{succ}\, n) \vdash v : A[n := \mathsf{succ}\, n][y := \mathsf{issucc}\, n\, x]\end{array}}{\Gamma \vdash (\mathsf{match}\ t\ \mathsf{as}\ y\ \mathsf{with} \begin{bmatrix}\mathsf{iszero} & \to u \\ \mathsf{issucc}\, n\, x & \to_a v\end{bmatrix}) : A[n := t][y := p]}$$

with reduction rules:

$$\mathsf{match}\ \mathsf{zero}\ \mathsf{as}\ m\ \mathsf{with} \begin{bmatrix}\mathsf{zero} & \to u \\ \mathsf{succ}\, n & \to_a v\end{bmatrix} \quad \to \quad u[m := \mathsf{zero}]$$

$$\mathsf{match}\ (\mathsf{succ}\, t)\ \mathsf{as}\ m\ \mathsf{with} \begin{bmatrix}\mathsf{zero} & \to u \\ \mathsf{succ}\, n & \to_a v\end{bmatrix} \quad \to \quad v[n := t][m := \mathsf{succ}\, t][a := \mathsf{match}\ t\ \mathsf{as}\ m\ \mathsf{with} \begin{bmatrix}\mathsf{zero} & \to u \\ \mathsf{succ}\, n & \to_a v\end{bmatrix}]$$

$$\mathsf{match}\ \mathsf{iszero}\ \mathsf{as}\ y\ \mathsf{with} \begin{bmatrix}\mathsf{iszero} & \to u \\ \mathsf{issucc}\, n & \to_a v\end{bmatrix} \quad \to \quad u[y := \mathsf{iszero}][]$$

$$\mathsf{match}\ (\mathsf{issucc}\, t\, p)\ \mathsf{as}\ y\ \mathsf{with} \begin{bmatrix}\mathsf{iszero} & \to u \\ \mathsf{issucc}\, n\, x & \to_a v\end{bmatrix} \quad \to \quad v[n := t][x := p][y := \mathsf{issucc}\, t\, p][a := \mathsf{match}\ p\ \mathsf{as}\ y\ \mathsf{with} \begin{bmatrix}\mathsf{iszero} & \to u \\ \mathsf{issucc}\, n\, x & \to_a v\end{bmatrix}]$$

As an application, we are able using these constructions to canonically derive the realisability interpretation of terms in a way which is compatible with a guard-based typing system, as it is in Coq. We rewrite the original term into:

$$m : \mathbb{N} \vdash (\mathsf{match}\ m\ \mathsf{as}\ m'\ \mathsf{with} \begin{bmatrix}\mathsf{zero} & \to m' \\ \mathsf{succ}\, n & \to_a m'\end{bmatrix}) : \mathbb{N}$$

and canonically derive from it the following now well-typed term:

$$m : \mathbb{N}, y : \mathsf{is}\mathbb{N}\, m \vdash (\mathsf{match}\ y\ \mathsf{as}\ y'\ \mathsf{with} \begin{bmatrix}\mathsf{iszero} & \to y' \\ \mathsf{issucc}\, n\, x & \to_a y'\end{bmatrix}) : \mathsf{is}\mathbb{N}\, m$$

Assuming a guard condition that recognises an as-pattern as being of the same size as the term being matched, the guardedness of the derived term derives from the guardedness of the original term.

A Generic Translation from Case Trees to Eliminators

Kayleigh Lieverse, Lucas Escot, and Jesper Cockx

TU Delft, Netherlands

Introduction Dependently-typed languages such as Agda, Coq, Idris, and Lean allow one to guarantee correctness of a program by providing formal proofs. The type checkers of such languages elaborate the user-friendly high-level surface language to a small and fully explicit core language. This core language is in turn very difficult to manipulate by hand. Even when we work with a core language that is shown to be sound, there is still a risk that a mistake in elaboration leads us to prove the wrong thing. A lot of trust is put into this elaboration process, even though it is rarely verified. To guarantee the correctness of these elaborations we need to verify each part of the elaboration process independently. One example of the elaboration process is elaborating definitions by dependent pattern-matching [6], which provide a more user friendly, high-level interface, to the low-level construction of eliminators.

In dependent type theory, each datatype comes with an elimination principle, or eliminator, which expresses the induction principle for that datatype, and enables the definition of functions operating on this datatype. Working with eliminators by hand can quickly become unmanageable and unreadable, and for indexed datatypes we would need extra techniques like basic analysis [9] and specialization by unification [8]. Therefore, some dependently-typed languages, like Agda, provide the high-level interface of dependent pattern-matching, and their type checker has to elaborate functions defined by pattern matching back to functions defined using eliminators.

This elaboration is done in two steps. First, the function defined by dependent pattern-matching is translated into a case tree [3], where each node corresponds to a case split and each leaf corresponds to a clause of a function. Then, the case tree is translated to functions defined using eliminators. Agda only does the first step of the translation, but doesn't target eliminators. However, it is proven that this translation is always possible and that it preserves the semantics of the case tree [8] [2].

In this paper, we implement a generic, correct-by-construction translation of this second step in Agda, without the use of metaprogramming and unsafe transformations [1]. The contributions are a type-safe, correct-by construction, generic definition of case trees and an evaluation function that, given an instantiation of such a case tree and an instantiation of the telescope of function arguments, evaluates the output term of the function using only eliminators.

Sozeau [11] has presented a Coq translation from dependent pattern-matching to eliminators in the `Equations` package, that translates a given function defined by dependent pattern-matching to eliminators on a case-by-case basis. This translated function is then type checked by Coq's type checker. In this work, we provide a generic evaluation function in pure Agda code, which gives the guarantee that we can evaluate any instantiation of the generic case tree to the output term. This evaluation function is implemented in terms of generic eliminators only, even though this can only be manually enforced.

Case Trees When elaborating a function defined by dependent pattern-matching, we first translate the function to a case tree. A case tree is a tree where each node corresponds to a case split and each leaf corresponds to a clause of the function. A case split is performed on one

[1] https://github.com/klieverse/case-trees-to-eliminators

variable of a datatype `D` in the telescope of function arguments and the result is a new case tree for every constructor of `D`, where each occurrence of that variable in the telescope is replaced by that constructor. For example, for the +-function for natural numbers, we would perform a case split on the first natural number n, which would result in two new case trees: one where n maps to `zero`, and one where n maps to `suc` n, with a new fresh variable n. Then, we would fill in the clauses of the function for the leafs: we have that `zero` $+ m$ maps to m, and `suc` n $+ m$ maps to `suc` $(n + m)$:

$$[(n : \mathbb{N})(m : \mathbb{N})]\ \underline{n}\ m \begin{cases} [(m : \mathbb{N})]\ \texttt{zero}\ m \mapsto m \\ [(n : \mathbb{N})(m : \mathbb{N})]\ (\texttt{suc}\ n)\ m \mapsto \texttt{suc}\ (n + m) \end{cases}$$

In the generic representation of the case tree, we only split on variables whose type is a datatype from a fixed universe of datatype encodings [1], which allows us to identify the different constructors of that datatype, together with the fresh variables of each constructor, and to create generic functions on these datatypes (e.g. elimination principle). Using telescopes [7] to denote the function arguments, we can update the telescope for the new case tree by replacing the variable we split on by a series of fresh variables used by the constructor in the new case tree.

The clauses of a function defined by dependent pattern-matching may contain recursive calls. These calls should be replaced in the case tree by an application of the $\texttt{rec}_{\texttt{D}}$-eliminator [10], which we get by adding a type $\texttt{Below}_{\texttt{D}}$, that collects all recursive calls for the function, to the telescope of function arguments.

To deal with case splitting on indexed datatypes, we apply specialization by unification [8] on the indices from the variable we perform a case split on, together with the indices from the respective constructor. These unifiers are represented as equivalences [5]. The combination of unification rules to determine whether the unification succeeds positively or negatively have to be manually added when creating an instantiation of the generic case tree, that means that we have to handle unification manually when defining a case tree. We model four unification rules in a correct-by-construction fashion: solution, deletion, injectivity, and conflict, whilst also allowing a user to add their own rules.

Representing the unification rules for datatypes that are indexed by other datatypes is still a work in progress, as it requires higher-dimensional unification [4].

Evaluation The type signature of the evaluation function contains a telescope of function arguments Δ and a return type T that is dependent on an instantiation of that telescope. Then, if we have a case tree for a function with function arguments Δ and return type T, and an instantiation of the telescope of function arguments t, we can evaluate to the return type where t is applied to T:

```
eval : {i : ℕ}{Δ : Telescope i}{T : ⟦ Δ ⟧telD -> Set₁}
    -> CaseTree Δ T -> (t : ⟦ Δ ⟧telD) -> T t
```

To evaluate a case tree, given an instantiation of the telescope of function arguments, we perform basic analysis [9] on the datatype that a case split is performed on to match the result of the case split with the variable in the instantiation of the telescope. We recursively call the evaluation function with the remainder of the case tree until a leaf node is reached, where we then guarantee that the input instantiation of the telescope of function arguments matches that clause of the function.

References

[1] Marcin Benke, Peter Dybjer, and Patrik Jansson. Universes for generic programs and proofs in dependent type theory. *Nord. J. Comput.*, 10(4):265–289, 2003.

[2] Jesper Cockx. Dependent pattern matching and proof-relevant unification. 2017.

[3] Jesper Cockx and Andreas Abel. Elaborating dependent (co) pattern matching: No pattern left behind. *Journal of Functional Programming*, 30:e2, 2020.

[4] Jesper Cockx and Dominique Devriese. Lifting proof-relevant unification to higher dimensions. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 173–181, 2017.

[5] Jesper Cockx, Dominique Devriese, and Frank Piessens. Unifiers as equivalences: Proof-relevant unification of dependently typed data. *Acm Sigplan Notices*, 51(9):270–283, 2016.

[6] Thierry Coquand. Pattern matching with dependent types. In *Informal proceedings of Logical Frameworks*, volume 92, pages 66–79. Citeseer, 1992.

[7] Nicolas G de Bruijn. Telescopic mappings in typed lambda calculus. *Information and Computation*, 91(2):189–204, 1991.

[8] Healfdene Goguen, Conor McBride, and James McKinna. Eliminating dependent pattern matching. In *Algebra, Meaning, and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, pages 521–540. Springer, 2006.

[9] Conor McBride. Elimination with a motive. In *International Workshop on Types for Proofs and Programs*, pages 197–216. Springer, 2000.

[10] Conor McBride, Healfdene Goguen, and James McKinna. A few constructions on constructors. In *Types for Proofs and Programs: International Workshop, TYPES 2004, Jouy-en-Josas, France, December 15-18, 2004, Revised Selected Papers*, pages 186–200. Springer, 2006.

[11] Matthieu Sozeau. Equations: A dependent pattern-matching compiler. In *International Conference on Interactive Theorem Proving*, pages 419–434. Springer, 2010.

Session 6: Blockchain and Smart Contracts

Mechanizing BFT consensus protocols in Agda

Orestis Melkonian, Mauro Jaskelioff, James Chapman, and Jon Rossie

Input Output, Global (IOG)

Introduction. Consensus protocols have been around for a long time [10, 9], but there has been a surge of interest in the last decade motivated by cryptocurrency and blockchain applications, where all participants need to agree on a common order of blocks on the chain.

Consensus protocols can be permissioned or permissionless. Nakamoto-style consensus protocols are permissionless (all participants can be part of the decision procedure) while classical protocols like BFT are permissioned (a few designated ones make the decision). With the advent of proof-of-stake blockchains, permissioned protocols can be adapted to work in a blockchain setting: a committee is formed based on the stake of all participants, which makes all decisions until a new committee is designated.

BFT protocols follow a pattern of propose and vote, where a leader proposes a block and other nodes vote for it. When a block gets enough votes it gets notarized; a notarized chain (i.e. a chain of notarized blocks) is considered finalized when certain protocol-dependent conditions hold, which guarantees that all participants will agree on it.

There are several formalisations of different consensus protocols in which parts or all of the protocol and their corresponding safety and liveness properties are proved correct [13, 1, 12]. In this work, we propose an alternative *refinement*-based approach, where the formalization is divided into layers. At the most abstract layer we only model the deterministic result of the consensus protocol (a finalized chain) together with the information we need to prove that the chain is indeed finalized. The motivation for this is that we want to be able to construct and verify zero-knowledge (ZK) proofs that a given chain is notarized/final. This means that the details of *how* to get to a notarized/final chain are not important at this level of abstraction.

Of course in order to get a finalized chain we will need *a* concrete protocol, which we formalize in the next layer. It corresponds to a mechanization of the paper introducing the protocol informally. Still, one might want to explore more details than the "academic" version of the paper, e.g., to analyze and optimize performance or include the actual networking, which should again be studied in an even lower layer of abstraction.

We conduct our work in a mechanized fashion using the Agda proof assistant [11] and simultaneously develop a new compilation backend to Rust, so that our formal artifact can also be utilized for *conformance testing* against an actual blockchain in production.

Framework: an abstract view of BFT protocols. First and foremost, we stay parametric over an abstract type of transactions so that we can later instantiate that with various ledger models, but as a starting point we define everything on top of a minimal UTxO-based ledger, whose meta-theory has already been extensively studied in prior work [3, 2], with the minor extension that transactions are also allowed to register new committees. Hence, the ledger state does not only consist of the typical UTxO set of currently unspent outputs, but also the committees registered thus far.

BFT consensus relies on a subset of the participants, called the 'committee', that assigns a public key to each participant and requires a certain ratio of votes (in the form of *signed* blocks) in order to reach agreement, a.k.a. forming a *quorum*:

```
record Committee : Type where
  field ratio   : Float
        members : AssocList Pid PublicKey
```

```
quorum? : List Vote → Committee → Bool
quorum? vs com =
  com .ratio * length (com .members) < length vs
```

Block *verification* is formulated as a (labeled) transition system, which builds upon the base transaction-level transition of the ledger inherited from the underlying ledger model [6, 8].

```
data _⊢_–[_]→ᵇ_ : ℕ → Ledger × Hash → Block → Ledger × Hash → Type where
  VerifyBlock :
    • h ≡ b .block .height
    • T (quorum? (b .votes) com)
    • ∀[ v ∈ b .votes ] ∃[ pr ∈ com .members ⁉ v .pid ]
        T (verify-signature pr (v .signature) b♯)
    • _ ⊢ l –[ b .block .transactions ]→* l′
    ──────────────────────────────────────────
    h ⊢ (l , H) –[ b ]→ᵇ (l′ , (H , b♯) ♯)
```

After proving that this relation is in fact *computational*, we can extract the expected decision procedure for verifying blocks.

```
verify-block : Height → Ledger × Hash → Verifiable-Block → Maybe (Ledger × Hash)
```

Case study: the Streamlet protocol, mechanized. We have mechanized the Streamlet protocol [5]; one of the simplest in the BFT literature. As expected, the notions of **notarization** (when every block in a chain has votes from the majority) and **finalization** (when three consecutive epochs are notarized in sequence, the prefix up to the second block is necessarily *final*) are straightforward to make precise given a set of exchanged messages `ms`:

```
NotarizedBlock : Block → Type
NotarizedBlock b = length votes ≥ majority
  where
    votes = filter ((_≟ b) ∘ blockMessage) ms

NotarizedChain : Chain → Type
NotarizedChain = All NotarizedBlock
```

```
data FinalizedChain : Chain → Block → Type where
  Finalize : ∀ {ch b₁ b₂ b₃} →
    • NotarizedChain (b₃ :: b₂ :: b₁ :: ch)
    • b₃ .epoch ≡ suc (b₂ .epoch)
    • b₂ .epoch ≡ suc (b₁ .epoch)
    ──────────────────────────────
    FinalizedChain (b₂ :: b₁ :: ch) b₃
```

Moreover, the progression of the protocol can naturally be expressed as an inductive step relation; to save space we just show the expected type and one example step that records a finalized chain for a node:

```
data _–→_ : State → State → Type where
  FinalizeBlock : ∀ n ch b →
    FinalizedChain (s •messagesSoFar n) ch b
    ──────────────────────────────────────────
    s –→ finalize n ch s
```

```
s₀   –→⟨ ReceiveMessage? 𝔹 0 ⟩
s₁   –→⟨ Vote? 𝔹 [] [] ⟩
s₂   –→⟨ AdvanceEpoch ⟩
⋮    ⋮
sₙ₋₁ –→⟨ FinalizeBlock? 𝔸 [ b₆ ; b₅ ; b₂ ] b₇ ⟩
sₙ   ∎
```

Finally, we prove that all involved logical propositions are *decidable*, thus also producing an *executable specification* which can be used to run example traces as shown on the right (details omitted for brevity, proofs are automatically discharged via *proof-by-computation* [14]).

This is work in progress and we are currently closing in on a mechanized proof of *safety* (otherwise called 'consistency'): honest nodes always agree on their final chains (up to a prefix). We have not yet instantiated the general framework outlined above to the Streamlet case due to some pending decisions w.r.t. finalization, but we are confident that our framework is sufficiently equipped to cover this in principle.

Future work Next steps include formalizing other BFT protocols — in particular Simplex [4] and Jolteon [7] — with the hope that our framework sufficiently generalizes over all examples, as well as extracting executable Rust programs that can then be integrated into more lightweight methods such as simulation-based testing (particularly useful for properties that are difficult to formally prove).

References

[1] Harold Carr, Christa Jenkins, Mark Moir, Victor Cacciari Miraldo, and Lisandra Silva. Towards formal verification of HotStuff-based byzantine fault tolerant consensus in Agda. In Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez, editors, *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, volume 13260 of *Lecture Notes in Computer Science*, pages 616–635. Springer, 2022. doi:10.1007/978-3-031-06773-0_33.

[2] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Jann Müller, Michael Peyton Jones, Polina Vinogradova, and Philip Wadler. Native custom tokens in the extended UTXO model. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Applications - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part III*, volume 12478 of *Lecture Notes in Computer Science*, pages 89–111. Springer, 2020. doi:10.1007/978-3-030-61467-6_7.

[3] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. The extended UTXO model. In Matthew Bernhard, Andrea Bracciali, L. Jean Camp, Shin'ichiro Matsuo, Alana Maurushat, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers*, volume 12063 of *Lecture Notes in Computer Science*, pages 525–539. Springer, 2020. doi:10.1007/978-3-030-54455-3_37.

[4] Benjamin Y. Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV*, volume 14372 of *Lecture Notes in Computer Science*, pages 452–479. Springer, 2023. doi:10.1007/978-3-031-48624-1_17.

[5] Benjamin Y. Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 1–11. ACM, 2020. doi:10.1145/3419614.3423256.

[6] Jared Corduan, Matthias Güdemann, and Polina Vinogradova. A formal specification of the Cardano ledger. https://github.com/input-output-hk/cardano-ledger/releases/latest/download/shelley-ledger.pdf, 2019.

[7] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 296–315. Springer, 2022. doi:10.1007/978-3-031-18283-9_14.

[8] Andre Knispel, Orestis Melkonian, James Chapman, Alasdair Hill, Joosep Jääger, William DeMeo, and Ulf Norell. Formal specification of the Cardano blockchain ledger, mechanized in Agda. https://omelkonian.github.io/data/publications/cardano-ledger.pdf, 2024. under submission.

[9] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

[10] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

[11] Ulf Norell. Dependently typed programming in Agda. In *International School on Advanced Functional Programming*, pages 230–266. Springer, 2008.

[12] Vincent Rahli, Ivana Vukotic, Marcus Völp, and Paulo Jorge Esteves Veríssimo. Velisarios: Byzantine fault-tolerant protocols powered by Coq. In Amal Ahmed, editor, *Programming Languages and*

Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, volume 10801 of *Lecture Notes in Computer Science*, pages 619–650. Springer, 2018. doi:10.1007/978-3-319-89884-1_22.

[13] Søren Eller Thomsen and Bas Spitters. Formalizing Nakamoto-style proof of stake. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–15. IEEE, 2021. doi:10.1109/CSF51468.2021.00042.

[14] Paul Van Der Walt and Wouter Swierstra. Engineering proof by reflection in Agda. In *Symposium on Implementation and Application of Functional Languages*, pages 157–173. Springer, 2012.

Termination-checked Solidity-style smart contracts in Agda in the presence of Turing completeness

Fahad F. Alhabardi[1] and Anton Setzer[2]

[1,2]Swansea University, Dept. of Computer Science
[1]fahadalhabardi@gmail.com https://fahad1985lab.github.io/
[2]a.g.setzer@swansea.ac.uk http://www.cs.swan.ac.uk/~csetzer/

Abstract

This paper is a further step in extending the verification of Bitcoin Script using weakest precondition semantics in our articles [6, 1, 5] to Solidity-style smart contracts. The first step is to develop a model, which is substantially more complex than that of Bitcoin Script because smart contracts in Solidity are object-oriented. This paper extends the simple model of Solidity-style smart contracts in Agda in our article [2] to a complex model. The main addition in the complex model is that it deals with the termination problem by adding a cost per instruction (gas cost) as implemented in Ethereum, therefore execution of smart contracts passes the termination checker of Agda.

One main application of blockchain are smart contracts. Smart contracts can be defined as programs that automatically run when specific predetermined criteria are met [16, 13].

Smart contracts face several challenges, particularly in terms of security [8]. All smart contract transactions and codes are immutable once published on the blockchain network. The only way to amend the clauses of an ongoing smart contract or to withdraw it is by using functions already provided by the original contract. Thus, the developers must ensure and verify the security of the code before publishing it on the blockchain in order to avoid any errors. Errors in smart contract programs have resulted in massive financial losses [14, 15].

One formal way to specify the validity of imperative programs is Hoare logic [11]: one defines pre- and postconditions as the required conditions on the state of a program before and after execution. Hoare logic works well for guaranteeing the safety of programs, i.e. that programs work correctly when executed according to requirements. A very stringent technique can identify errors early in the development phase [12]. However, it doesn't work very well for showing that a program is secure in the presence of malicious inputs. Our solution is to use weakest preconditions of Hoare logic instead. Weakest preconditions express that the conditions are not only sufficient but also necessary for the program to end up in a state fulfilling the postcondition. An example is that certain data needs to be present in order to obtain cryptocurrency coins.

In this paper, we extend the simple model of Solidity-style smart contracts in our previous paper [2] (see as well the simulator [3]) to a complex model. In the complex model, we add gas cost. We use the gas cost to guarantee termination – each instruction costs at least one unit of gas, and once all gas allocated has been consumed, the program terminates with an out of gas error. Using this idea we succeed it showing that our implementation of the execution mechanism of programs passes Agda's termination checker.

We work directly on Solidity code rather than on its compiled Ethereum Virtual Machine (EVM) code. Therefore we cannot use gas costs associated with EVM instructions, and instead add to each high level Solidity instruction a parameter which estimates the gas costs for its execution. Therefore verification depends on good estimates for these parameters.

As in our previous simple model we have ordinary functions (corresponding to methods in the terminology of object-orientation). We encode the arguments and return values of functions

as elements of a message type, which allows as well to encode multiple arguments as single ones. In our settings functions have only one argument and one return element of this message type. Ordinary functions are given by a coalgebraic definition, which consist of a possibly unbounded sequence of basic operations such as making a transfer, looking up the balance of an account, or making recursive calls to other functions. In addition to ordinary functions, we add view functions (functions which can be modified by ordinary functions but don't call other functions). Variables are represented as view functions. They are especially useful for representing variables which have the type of a mapping, which frequently occur in Solidity code. View functions are represented as simple functions in Agda, and therefore are elements of a data type different from that of ordinary functions. Ordinary functions have instructions for updating view functions, but are not able to update ordinary functions. Therefore we keep view functions and normal functions as separate entities. (In Solidity view functions are defined as ordinary functions, but with a restriction on their code). The gas cost of ordinary functions is given by the cost of the basic instructions involved during their execution. For view functions we need in addition functions which estimate the cost for their execution.

We start by defining the data type of contracts (Contract), which includes four fields: the balance of a contract (amount), its functions (fun), its view functions (viewFunction), and the estimated gas cost for executing a view function (viewFunctionCost). The definition of Contract is as follows:

```
record Contract : Set where
  field
    amount : Amount
    fun : FunctionName → (Msg → SmartContractExec Msg)
    viewFunction : FunctionName → Msg → MsgOrError
    viewFunctionCost : FunctionName → Msg → ℕ
```

Ethereum uses a simple model of mapping addresses to their state as opposed to the UTXO model (see e.g. [9], or our article [15]) used e.g. in Bitcoin which tracks the state to previous unspent transaction outputs. We call such a mapping for brevity a ledger. Strictly speaking it is the state of a ledger – a full ledger would include its history. The execution of a smart contract function in Ethereum only depends on the current state of the ledger without its history, and function calls are executed as one atomic operation which includes all its recursive calls and updates. Therefore the correctness of a smart contract in this setting relates only to the current state of the ledger. We define therefore a ledger as a function which maps addresses to contracts: Ledger = Address→ Contract

As in the simple model, we have an execution stack, which records currently open recursive calls. The elements of the execution stack (ExecStackEl) include the following fields: the address that made the last call (lastCallAddress), the address that was called (calledAddress), continuation which determines the next execution step to be executed depending on the message returned after the call to the function has been completed, funcNameexecStackE which is the last function called and the argument of the last function call (msgexecStackEl).

```
record ExecStackEl : Set where
  field lastCallAddress calledAddress : Address
        continuation : (Msg → SmartContractExec Msg)
        costCont              : Msg → ℕ
        funcNameexecStackEl : FunctionName
        msgexecStackEl        : Msg
```

The execution stack is a list of ExecStackEl. The state of the execution (StateExecFun) include the following fields: the ledger, the execution stack (executionStack), the initial address that initiated the current sequence (initialAddr), the last called made (lastCallAddr), the address which is called (calledAddr), the current code to be executed (nextstep), the gas left (gasLeft), and two extra fields that we use with debug information: funcNameexecStackE and msgexecStackEl.

```
record StateExecFun : Set where
field ledger          : Ledger
      executionStack  : ExecutionStack
      initialAddr lastCallAddr calledAddr : Address
      nextstep        : SmartContractExec Msg
      gasLeft         : ℕ
      funNameevalState : FunctionName
      msgevalState : Msg
```

In order to state the verification conditions in Hoare logic, we define the state of the system as given by the ledger and the address making the call. Pre- and post-conditions will be defined as predicates on this state. In order to accommodate with intermediate steps in the program execution, the program will be given by the code to be executed, the execution stack and the called address. In order to have a robust definition which works as well in the simple model where programs are not guaranteed to terminate, we define a relation expressing that during execution, the program starting in a start state terminates successfully in an end state. Then, we show that the precondition is a weakest precondition for the program to end in the postcondition state. A simple example is that in order for the amount in one contract to reach a certain value, a second contract (which triggered a transfer) must have had a sufficient balance. In a follow-up paper, we will show how to formally prove this in Agda, which reveals unexpected subtleties in the precise formulation of its precondition.

Related Work. For a detailed literature review see our article [4]. Some additional work to mention is the formalisation KEVM [10] of the EVM in the K framework, which directly formalises the low level Ethereum virtual machine. Our approach works instead directly on Solidity in order to support the derivation of human readable weakest preconditions. Annenkov et. al. [7] developed a framework ConCert for extracting smart contracts from Coq, and a testing framework that allows to detect specific high level exploits. In our work we define instead a direct semantics for Solidity style contracts based on weakest preconditions. There is extensive work such as [9] from researchers, many of whom are associated with IHOK, which studies and extends the unspent transaction model (UTXO). We have studied the UTXO model used in Bitcoin in [15]. In this paper, we use the model used in Ethereum, which instead directly maps addresses to balances. Ethereum uses transaction nonces instead of UTXOs in order to prevent replay attacks.

References

[1] Fahad Alhabardi, Bogdan Lazar, and Anton Setzer. Verifying correctness of smart contracts with conditionals. In *2022 IEEE 1st Global Emerging Technology Blockchain Forum: Blockchain & Beyond (iGETblockchain)*, pages 1–6, 2022. doi:10.1109/iGETblockchain56591.2022.10087054.

[2] Fahad Alhabardi and Anton Setzer. A simple model of smart contracts in Agda, January 2023. In Abstracts for Types 2023. URL: https://types2023.webs.upv.es/TYPES2023.pdf.

[3] Fahad Alhabardi and Anton Setzer. A simulator of Solidity-style smart contracts in the theorem prover Agda, August 2023. To appear in Proceedings of ICBTA 2023, Xi'an, China. http://

www.icbta.net/. International Conference Proceedings Series by ACM (ISBN: 979-8-4007-0867-1), ACM Digital library. URL: https://csetzer.github.io/articles/alhabardiSetzerICBTA2023.pdf.

[4] Fahad Alhabardi and Anton Setzer. A model of solidity-style smart contracts in the theorem prover agda. In *2023 IEEE International Conference on Artificial Intelligence, Blockchain, and Internet of Things (AIBThings)*, pages 1–10, Mount Pleasant, MI, USA, 2023. IEEE. doi:10.1109/AIBThings58340.2023.10292478.

[5] Fahad Alhabardi, Anton Setzer, Arnold Beckmann, and Bogdan Lazar. Verification techniques for smart contracts in Agda, June 2022. In Abstracts for Types 2022. URL: https://types22.inria.fr/programme/.

[6] Fahad F. Alhabardi, Arnold Beckmann, Bogdan Lazar, and Anton Setzer. Verification of Bitcoin Script in Agda Using Weakest Preconditions for Access Control. In *27th International Conference on Types for Proofs and Programs (TYPES 2021)*, volume 239 of *LIPIcs*, pages 1:1–1:25, Dagstuhl, Germany, 2022. Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TYPES.2021.1.

[7] Annenkov, Danil and Milo, Mikkel and Nielsen, Jakob Botsch and Spitters, Bas. Extracting smart contracts tested and verified in Coq. In *CPP 2021: Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2021, page 105–121, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3437992.3439934.

[8] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A Survey of Attacks on Ethereum Smart Contracts (SoK). In *Principles of Security and Trust*, pages 164–186, Berlin, Heidelberg, 2017. Springer. doi:10.1007/978-3-662-54455-6_8.

[9] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. The Extended UTXO Model. In *Financial Cryptography and Data Security*, pages 525–539, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-54455-3_37.

[10] Hildenbrandt, Everett and Saxena, Manasvi and Rodrigues, Nishant and Zhu, Xiaoran and Daian, Philip and Guth, Dwight and Moore, Brandon and Park, Daejun and Zhang, Yi and Stefanescu, Andrei and Rosu, Grigore. KEVM: A Complete Formal Semantics of the Ethereum Virtual Machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 204–217, 2018. doi:10.1109/CSF.2018.00022.

[11] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576 – 585, October 1969. doi:10.1145/363235.363259.

[12] Lamport, Leslie and Schneider, Fred B. The "Hoare Logic" of CSP, and All That. *ACM Transactions on Programming Languages and Systems*, 6(2):281–296, April 1984. doi:10.1145/2993.357247.

[13] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2976749.2978309.

[14] Zeinab Nehaï, Pierre-Yves Piriou, and Frédéric Daumas. Model-Checking of Smart Contracts. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 980–987, 2018. doi:10.1109/Cybermatics_2018.2018.00185.

[15] Anton Setzer. Modelling Bitcoin in Agda. *CoRR*, abs/1804.06398, 2018. URL: http://arxiv.org/abs/1804.06398, arXiv:1804.06398.

[16] Nick Szabo. Smart contracts: Building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought*, 18(2), 1996. URL: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.

A formal security analysis of Blockchain voting

Nikolaj Sidorenco, Laura Brædder, Lasse Letager Hansen, Eske Hoy Nielsen, and Bas Spitters

Department of Computer Science, Aarhus University, Denmark

Motivation

Digitization is facilitating many aspects of our society, and voting is no exception. However, in voting the stakes are higher, than in many other applications. Moreover, elections require trust for their outcome to be accepted.

Blockchains provide a trusted bulletin board, and as such, they have been used as part of voting protocols. Conversely, online voting also serves an important role in the blockchain ecosystem itself, for making fundamental changes to the system — so-called blockchain governance; see [6] for an overview. Online voting systems have, through time, had buggy implementations and questionable security guarantees. See [9] for a description of the many issues with blockchain voting. Two critiques of blockchain-based solutions are their security and the quality of their implementation.

We will provide a means to addressing these issues by focusing on a particular blockchain voting protocol: the Open Vote Network (OVN) [3]. The OVN allows a small group to vote anonymously; so-called boardroom voting. The OVN has been implemented as a smart-contract on the Ethereum blockchain [7], but only comes with an informal security argument. Such a smart contract implementation raises two questions; first is the protocol implemented correctly as a smart contract, and second is the protocol cryptographically secure.

A partially verified implementation of OVN is already available in ConCert [1]. This work was also presented at CoqPL'2024.

We are extending this work to a more realistic implementation with stronger guarantees and fully verifying it, including proving it cryptographically secure.

Notice that many of the tools we implement to prove the security of the OVN protocol can be reused in proving security for other bigger protocols such as ElectionGuard[1]. For instance, the parts of our verified implementation of the OVN protocol, that we can reuse are the non-interactive zero-knowledge proofs, Schnorr proofs, and a CDS construction which are made non-interactive via the Fiat-Shamir heuristic.

The Open Vote Network Protocol

The Open Vote Network(OVN) is a voting protocol based on ledgers and zero-knowledge proofs [3]. It is decentralized and, because of the properties of zero-knowledge proofs to detect any deviation from the protocol, does not require trusted parties. These proofs are publicly available in a decentralized manner through the use of a blockchain.

The protocol as described in [3], proceeds in two rounds followed by a tallying process. Knowledge of the outcome of the tally may potentially influence the voter's cast vote. To remedy this, an extra commitment round was added by [7]. Thus the protocol becomes

Round 1: Participants compute public keys, reconstruction keys, and a non-interactive zero-knowledge proof of the relation between their private and public keys. All public keys are put on the public ledger and verified by the participants.

[1] `https://www.electionguard.vote/`

Commitment phase: Participants commit to their votes and publish the commitments.
Round 2: Every participant computes and publishes their encrypted ballot and a non-interactive zero-knowledge proof of the validity of the vote.
Tallying: When each ballot is on the ledger and every zero-knowledge proof is checked to be valid, the votes can be tallied. Anyone can tally the votes by computing the product of all encrypted votes and brute-forcing the exponent.

Verifying OVN

We base our OVN implementation on the three-round Ethereum OVN implementation [7]. We chose to implement it in Hacspec [2, 8], which is a functional subset of Rust, as Rust is becoming an increasingly popular choice of smart contract language in blockchain. Some examples of blockchains based on Rust using WebAssembly are:

- Concordium, Ethereum (Rollups), Polkadot, ICP, cosmos, Near, Hyperledger Sawtooth

Hacspec is a high-assurance cryptography specification language, its main use is to specify cryptographic primitives [2, 8]. However, here we build on ongoing work extending Hacspec for smart contract specification and implementation.

The OVN implementation in Hacspec is translated [10] to both ConCert [1] and SSProve [5].

We prove cryptographic security using the SSProve framework, a framework for state separating cryptographic proofs in Coq [5]. Using the SSProve backend for Hacspec [4] we obtain an embedding of OVN in SSProve which we use to define security games showing indistinguishability of the OVN implementation and an ideal implementation. This is done in the computational model, which is more precise than the symbolic model.

We prove functional correctness using ConCert, which is a framework for smart contract verification in Coq [1]. ConCert abstractly models a blockchain as an immutable append-only ledger. Smart contracts in ConCert are modeled as state-passing functions in Coq. The blockchain model in ConCert models full execution traces allowing one to state and prove many interesting properties about smart contracts, such as trace properties, invariants over the contract's state, and interactions between contracts.

Hao et al. [3] give the following security requirements for the OVN protocol:

Maximum ballot secrecy Each ballot is a ciphertext indistinguishable from random.
Dispute-freeness The result should be publicly verifiable. Anyone can check whether all parties adhered to the protocol when casting their ballot.
Self-tallying After all ballots have been cast, anyone can compute the result.

The proof of maximum ballot secrecy in SSProve follows a 'game hopping' style of proof, where the security of the protocol is reduced to well-established cryptographic properties of hardness assumptions. The OVN security proofs rely on the decisional Diffie–Hellman (DDH) assumption.

The Self-tallying property of OVN is stated in ConCert as an invariant over the state of the contract stating that for any reachable state in a valid execution trace, it should be the case that if all followed the protocol then a tally can be computed from the public data in the contract state and that tally equals the sum of all votes.

Dispute-freeness is given directly by the use of zero-knowledge proofs in the protocol which are publicly available on the ledger. This proof has not been formalized yet.

References

[1] Danil Annenkov, Jakob Botsch Nielsen, and Bas Spitters. ConCert: A smart contract certification framework in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, page nil, January 2020.

[2] Karthikeyan Bhargavan, Franziskus Kiefer, and Pierre-Yves Strub. hacspec: Towards verifiable crypto standards. In *Security Standardisation Research - 4th International Conference, SSR 2018, Darmstadt, Germany, November 26-27, 2018, Proceedings*, pages 1–20, 2018.

[3] F. Hao, P. Ryan, and Piotr Zielinski. Anonymous voting by two-round public discussion. *IET Inf. Secur.*, 2010.

[4] Philipp G. Haselwarter, Benjamin Salling Hvass, Lasse Letager Hansen, Théo Winterhalter, Catalin Hritcu, and Bas Spitters. The Last Yard: Foundational End-to-End Verification of High-Speed Cryptography. Cryptology ePrint Archive, Paper 2023/185, 2023. `https://eprint.iacr.org/2023/185`.

[5] Philipp G. Haselwarter, Exequiel Rivas, Antoine Van Muylder, Théo Winterhalter, Carmine Abate, Nikolaj Sidorenco, Cătălin Hriţcu, Kenji Maillard, and Bas Spitters. Ssprove: A foundational framework for modular cryptographic proofs in coq. *ACM Trans. Program. Lang. Syst.*, 45(3), jul 2023.

[6] Aggelos Kiayias and Philip Lazos. SoK: Blockchain governance, 2022.

[7] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 357–375, Cham, 2017. Springer International Publishing.

[8] Denis Merigoux, Franziskus Kiefer, and Karthikeyan Bhargavan. hacspec: succinct, executable, verifiable specifications for high-assurance cryptography embedded in Rust. Technical report, Inria, March 2021.

[9] Sunoo Park, Michael Specter, Neha Narulaand, and Ronald L Rivest. Going from bad to worse: from internet voting to blockchain voting. *Journal of Cybersecurity*, 7(1), 2021.

[10] Mikkel Milo Rasmus Holdsbjerg-Larsen, Bas Spitters. A verified pipeline from a specification language to optimized, Safe Rust. CoqPL, 2022.

Session 9: Parametricity

Internal relational parametricity, without an interval

Thorsten Altenkirch[1], Ambrus Kaposi[2], Michael Shulman[3], and Elif Üsküplü[4]

[1] University of Nottingham, Nottingham, United Kingdom, txa@cs.nott.ac.uk
[2] Eötvös Loránd University, Budapest, Hungary, akaposi@inf.elte.hu
[3] University of San Diego, San Diego, CA, USA, shulman@sandiego.edu
[4] University of Southern California, Los Angeles, CA, USA, euskuplu@usc.edu

In type theories with internal parametricity, it can be proven that there is only one inhabitant of the type $\forall a.a \to a$, or that Church-encoded natural numbers support induction (for the latter we need binary parametricity). The syntactic challenge in these theories is that once there is a term witnessing internal parametricity, this term itself has to be parametric, and this results in the emergence of higher dimensional cubes. Syntax for such cubes can be given explicitly using ordered dimensions [3], named dimensions [4], or using substructural interval variables similar to cubical type theory [2, 5]. Recently, we defined a new structural type theory with internal parametricity [1] where the higher cubes are emergent, rather than explicitly built-in: there is no interval, there are no dimension variables, only some new type and term formers and several new equations. This theory featured span-based parametricity rather than relation-based parametricity, and it has a simple presheaf model without the need for Reedy-fibrancy as in [2]. The theory of Cavallo and Harper [5] is also modelled by presheaves without Reedy fibrancy, but it relies on the presence of univalence.

In this abstract we define a relation-based version of the theory of internal parametricity without an interval [1]. The theory in this abstract is justified by the same simple presheaf model, and can prove the same consequences of parametricity. For each dependent type, our new theory features an indexed heterogeneous logical relation (called bridge), and its dependencies are collected in a telescope. Telescopes come with a logical span: $\forall\bar{A}$ is the logical span for the telescope $\bar{A}$ and is equipped with two projection maps from $\forall\bar{A}$ to $\bar{A}$. $\forall\bar{A}$ is analogous to $\mathbb{I} \to \bar{A}$ in cubical type theory. Another difference from the theory [1] is the correspondence between the bridge type at the universe and relation space: there are functions in both directions, and the roundtrip when starting with a relation is identity up to a definitional isomorphism (rather than a definitional equality as in [1]). This is the price we have to pay for abandoning spans. Any model of the span-based theory ([1]) models our new relation-based theory.

We have an experimental implementation[1] of our theory in OCaml, we verified that Church-encoded natural numbers have an induction principle, and using this, we proved that they form a commutative semiring.

We describe a minimalistic version of our type theory as a second-order generalised algebraic theory (SOGAT [7, 6]). Our core theory is a standard Martin-Löf type theory featuring Π types, universes, and telescopes. It is given by the following rules, where $f : A \cong B : g$ means $f : A \to B$ and $g : B \to A$ such that $g\ (f\ a) = a$ and $f\ (g\ b) = b$.

$$\begin{array}{ll}
\mathsf{Ty} & : \mathsf{Set} \\
\mathsf{Tm} & : \mathsf{Ty} \to \mathsf{Set} \\
\Pi & : (A : \mathsf{Ty}) \to (\mathsf{Tm}\,A \to \mathsf{Ty}) \to \mathsf{Ty} \\
\mathsf{lam} & : ((a : \mathsf{Tm}\,A) \to \mathsf{Tm}\,(B\,a)) \cong \mathsf{Tm}\,(\Pi\,A\,B) : - \cdot - \\
\mathsf{U} & : \mathsf{Ty} \\
\mathsf{c} & : \mathsf{Ty} \cong \mathsf{Tm}\,\mathsf{U} : \mathsf{El}
\end{array}$$

$$\begin{array}{ll}
\mathsf{Tys} & : \mathsf{Set} \\
\mathsf{Tms} & : \mathsf{Tys} \to \mathsf{Set} \\
\diamond & : \mathsf{Tys} \\
- \triangleright - & : (\bar{A} : \mathsf{Tys}) \to (\mathsf{Tms}\,\bar{A} \to \mathsf{Ty}) \to \mathsf{Tys} \\
\mathsf{Tms}\,\diamond & = \mathbb{1} \\
\mathsf{Tms}\,(\bar{A} \triangleright A) & = (\bar{a} : \mathsf{Tms}\,\bar{A}) \times \mathsf{Tm}\,(A\,\bar{a})
\end{array}$$

[1] https://github.com/mikeshulman/narya

We have four sorts: types Ty, terms Tm, telescopic types Tys and telescopic terms Tms. Telescopic types are telescopes of Ty, and telescopic terms are just iterated (metatheoretic) Σ types of Tms. We have Π with β and η expressed as an isomorphism. Coquand universes come with code c and decode El, and we omit writing levels for convenience.

We extend this theory with the following new operations. k can stand for both 0 and 1.

$$
\begin{aligned}
\forall &: \mathsf{Tys} \to \mathsf{Tys}\\
k\_ &: (\bar{A} : \mathsf{Tys}) \to \mathsf{Tms}\,(\forall \bar{A}) \to \mathsf{Tms}\,\bar{A}\\
\mathsf{aps}\_ &: (\mathsf{Tms}\,\bar{B} \to \mathsf{Tms}\,\bar{A}) \to \mathsf{Tms}\,(\forall \bar{B}) \to \mathsf{Tms}\,(\forall \bar{A})\\
\mathsf{R}\_ &: (\bar{A} : \mathsf{Tys}) \to \mathsf{Tms}\,\bar{A} \to \mathsf{Tms}\,(\forall \bar{A})\\
\mathsf{S}\_ &: (\bar{A} : \mathsf{Tys}) \to \mathsf{Tms}\,(\forall(\forall \bar{A})) \to \mathsf{Tms}\,(\forall(\forall \bar{A}))\\
\mathsf{Br}\_ &: (A : \mathsf{Tms}\,\bar{B} \to \mathsf{Ty})(\bar{b}_2 : \mathsf{Tms}\,(\forall \bar{B})) \to \mathsf{Tm}\,(A\,(0_{\bar{B}}\,\bar{b}_2)) \to \mathsf{Tm}\,(A\,(1_{\bar{B}}\,\bar{b}_2)) \to \mathsf{Ty}\\
\mathsf{ap}\_ &: \big(a : (\bar{b}_x : \mathsf{Tms}\,\bar{B}) \to \mathsf{Tm}\,(A\,\bar{b}_x)\big)\big(\bar{b}_2 : \mathsf{Tms}\,(\forall \bar{B})\big) \to \mathsf{Tm}\,\Big(\mathsf{Br}_A\,\bar{b}_2\,\big(a\,(0_{\bar{B}}\,\bar{b}_2)\big)\,\big(a\,(1_{\bar{B}}\,\bar{b}_2)\big)\Big)\\
\mathsf{mkBr}\Pi &: \Big((a_0)(a_1)\big(a_2 : \mathsf{Tm}\,(\mathsf{Br}_A\,\bar{c}_2\,a_0, a_1)\big) \to \mathsf{Tm}\,\big(\mathsf{Br}_B\,(\bar{c}_2, a_0, a_1, a_2)\,(f_0 \cdot a_0)\,(f_1 \cdot a_1)\big)\Big) \cong\\
&\quad \mathsf{Tm}\,\big(\mathsf{Br}_{\lambda\bar{c}_x.\Pi\,(A\,\bar{c}_x)\lambda a_x.B\,(\bar{c}_x,a_x)}\,\bar{c}_2\,f_0\,f_1\big) : \lambda f_2\,a_0\,a_1\,a_2.\mathsf{ap}_{\lambda(\bar{c}_x,f_x,a_x).f_x\cdot a_x}\,(\bar{c}_2, f_0, f_1, f_2, a_0, a_1, a_2)\\
\mathsf{Gel} &: \mathsf{Tm}\,(\mathsf{El}\,a_0 \Rightarrow \mathsf{El}\,a_1 \Rightarrow \mathsf{U}) \to \mathsf{Tm}\,(\mathsf{Br}_{\mathsf{U}} * a_0\,a_1)\\
\mathsf{gel} &: \mathsf{Tm}\,(\mathsf{El}\,(R \cdot x_0 \cdot x_1)) \cong \mathsf{Tm}\,\big(\mathsf{Br}_{\lambda(*,a_x).\mathsf{El}\,a_x}\,(*, a_0, a_1, \mathsf{Gel}\,R)\,x_0\,x_1\big) : \mathsf{ungel}
\end{aligned}
$$

The main new operations are Br and ap, for computing logical relations and witnesses of logical relations (fundamental lemmas), respectively. $\forall$, k, aps, R and S are for dealing with dependencies (given as telescopes) of Br and ap: $\forall$ collects witnesses of relatedness in the telescope and the ks project out the corresponding components as prescribed by the following equations ($*$ is the inhabitant of the (meta) unit type $\mathbb{1}$).

$$
\begin{aligned}
&\forall\diamond = \diamond && \forall(\bar{B} \triangleright A) = \forall\bar{B} \triangleright \lambda\bar{b}_2.A\,(0_{\bar{B}}\,\bar{b}_2) \triangleright \lambda(\bar{b}_2, a_0).A\,(1_{\bar{B}}\,\bar{b}_2) \triangleright \lambda(\bar{b}_2, a_0, a_1).\mathsf{Br}_A\,\bar{b}_2\,a_0\,a_1\\
&k_\diamond * = * && k_{\bar{B}\triangleright A}\,(\bar{b}_2, a_0, a_1, a_2) = (k_{\bar{B}}\,\bar{b}_2, a_k)
\end{aligned}
$$

aps exresses that maps of telescopes are congruences. Everything respects aps, described as follows.

$$
\begin{aligned}
&k_{\bar{A}}\,(\mathsf{aps}_{\bar{a}}\,\bar{b}_2) = \bar{a}\,(k_{\bar{B}}\,\bar{b}_2) && \mathsf{ap}_{a\circ\bar{b}}\,\bar{c}_2 = \mathsf{ap}_a\,(\mathsf{aps}_{\bar{b}}\,\bar{c}_2)\\
&\mathsf{aps}_{\bar{a}\circ\bar{b}}\,\bar{c}_2 = \mathsf{aps}_{\bar{a}}\,(\mathsf{aps}_{\bar{b}}\,\bar{c}_2) && \mathsf{aps}_{\lambda\bar{b}_x.(\bar{a}\,\bar{b}_x, a\,\bar{b}_x)}\,\bar{b}_2 = (\mathsf{aps}_{\bar{a}}\,\bar{b}_2, a\,(0_{\bar{B}}\,\bar{b}_2), a\,(1_{\bar{B}}\,\bar{b}_2), \mathsf{ap}_a\,\bar{b}_2)\\
&\mathsf{aps}_{\lambda\bar{a}_x.\bar{a}_x}\,\bar{a}_2 = \bar{a}_2 && \mathsf{aps}_{\pi_1}\,(\bar{a}_2, a_0, a_1, a_2) = \bar{a}_2\\
&\mathsf{Br}_{A\circ\bar{b}}\,\bar{c}_2 = \mathsf{Br}_A\,(\mathsf{aps}_{\bar{b}}\,\bar{c}_2) && \mathsf{ap}_{\pi_2}\,(\bar{a}_2, a_0, a_1, a_2) = a_2
\end{aligned}
$$

R and S are reflexivity and symmetry for telescopes, their behaviour is described as follows.

$$
\begin{aligned}
&\mathsf{aps}_{\bar{b}}\,(\mathsf{R}_{\bar{A}}\,\bar{a}_x) = \mathsf{R}_{\bar{B}}\,(\bar{b}\,\bar{a}_x) && k_{\bar{A}}\,(\mathsf{R}_{\bar{A}}\,\bar{a}_x) = \bar{a}_x && \mathsf{S}_{\bar{A}}\,(\mathsf{R}_{\forall\bar{A}}\,\bar{a}_2) = \mathsf{aps}_{\mathsf{R}_{\bar{A}}}\,\bar{a}_2\\
&\mathsf{aps}_{\mathsf{aps}_{\bar{b}}}\,(\mathsf{S}_{\bar{A}}\,\bar{a}_{22}) = \mathsf{S}_{\bar{B}}\,(\mathsf{aps}_{\mathsf{aps}_{\bar{b}}}\,\bar{a}_{22}) && k_{\forall\bar{A}}\,(\mathsf{S}_{\bar{A}}\,\bar{a}_{22}) = \mathsf{aps}_{k_{\bar{A}}}\,\bar{a}_{22} && \mathsf{S}_{\bar{A}}\,(\mathsf{S}_{\bar{A}}\,\bar{a}_{22}) = \bar{a}_{22}\\
&\mathsf{S}_{\forall\bar{A}}\,(\mathsf{aps}_{\mathsf{S}_{\bar{A}}}\,(\mathsf{S}_{\forall\bar{A}}\,\bar{a}_{222})) = \mathsf{aps}_{\mathsf{S}_{\bar{A}}}\,(\mathsf{S}_{\forall\bar{A}}\,(\mathsf{aps}_{\mathsf{S}_{\bar{A}}}\,\bar{a}_{222}))
\end{aligned}
$$

Perhaps surprisingly, the above equations are enough to describe the behaviour on all higher dimensional cubes which can be built by iterating $\forall$. The operation $\mathsf{mkBr}\Pi$ lets us build a bridge at a Π type, it forms an isomorphism where the map in the other direction is definable. Gel turns a relation into a bridge at the universe, the other direction is again definable. For any relation, a witness of relatedness is isomorphic to a Br followed by Gel, this is witnessed by the gel–ungel isomorphism. This concludes the full definition of a theory of internal parametricity for our core theory with Π and U.

Example. Given an $f : (w : \mathsf{Tms}\,(\diamond \triangleright \lambda\_.\mathsf{U} \triangleright \lambda(*, a).\mathsf{El}\,a)) \to \mathsf{Tm}\,(\mathsf{El}\,(\pi_2\,(\pi_1\,w)))$, for any $a : \mathsf{Tm}\,\mathsf{U}$, $P : \mathsf{Tm}\,(\mathsf{El}\,a \Rightarrow \mathsf{U})$, $x : \mathsf{Tm}\,(\mathsf{El}\,a)$, $p : \mathsf{Tm}\,(\mathsf{El}\,(P \cdot x))$, using the unary version of theory, we have $\mathsf{ungel}\,(\mathsf{ap}_f\,(*, a, \mathsf{Gel}\,P, x, \mathsf{gel}\,p)) : \mathsf{Tm}\left(\mathsf{El}\left(P \cdot (f\,(*, a, x))\right)\right)$. Instantiating $P := \lambda y.\mathsf{Eq}\,x\,y$ and $p := \mathsf{refl}$, we obtain that f works as expected: simply returns its input x on its output (Eq is Leibniz-equality).

References

[1] Thorsten Altenkirch, Yorgo Chamoun, Ambrus Kaposi, and Michael Shulman. Internal parametricity, without an interval. *Proc. ACM Program. Lang.*, 8(POPL):2340–2369, 2024. doi:10.1145/3632920.

[2] Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. A presheaf model of parametric type theory. In Dan R. Ghica, editor, *The 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS 2015, Nijmegen, The Netherlands, June 22-25, 2015*, volume 319 of *Electronic Notes in Theoretical Computer Science*, pages 67–82. Elsevier, 2015. doi:10.1016/J.ENTCS.2015.12.006.

[3] Jean-Philippe Bernardy and Guilhem Moulin. A computational interpretation of parametricity. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 135–144. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.25.

[4] Jean-Philippe Bernardy and Guilhem Moulin. Type-theory in color. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 61–72. ACM, 2013. doi:10.1145/2500365.2500577.

[5] Evan Cavallo and Robert Harper. Internal parametricity for cubical type theory. *Log. Methods Comput. Sci.*, 17(4), 2021. doi:10.46298/LMCS-17(4:5)2021.

[6] Ambrus Kaposi and Szumi Xie. Second-order generalised algebraic theories: signatures and first-order semantics, 2024. To appear at FSCD 2024. Available: https://akaposi.github.io/sogat.pdf.

[7] Taichi Uemura. Homotopy type theory as internal languages of diagrams of ∞-logoses. 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 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSCD.2023.5.

Updates on Paranatural Category Theory

Jacob Neumann

University of Nottingham, United Kingdom
jacob.neumann@nottingham.ac.uk

It has been long-recognized[1] in the theory of functional programming languages that the concept of *parametric polymorphism* has a particular connection to the category-theoretic notion of *naturality*. For instance, Wadler [Wad89] famously noted that any System F function

$$r \quad : \quad \forall \alpha.\, \mathsf{List}(\alpha) \to \mathsf{List}(\alpha),$$

that is, a function $r_\alpha \colon \mathsf{List}(\alpha) \to \mathsf{List}(\alpha)$ which is "*polymorphic* in α", must automatically be a *natural transformation* from the List functor to itself, i.e.

$$r_Y \circ (\mathsf{map}\ f) = (\mathsf{map}\ f) \circ r_X$$

for any function $f \colon X \to Y$. However, this tight connection between parametricity and naturality breaks down for types with more complex variance; for instance, a polymorphic function $g \colon \forall \alpha.\, (\alpha \to \alpha) \to (\alpha \to \alpha)$ is not even the right *kind* of thing to be a natural transformation from the Hom functor to itself: g is only indexed by *one* type variable α, whereas Hom is a *difunctor*, a functor taking in one covariant and one contravariant argument. The notion of a *dinatural transformation* [ML78, Chapter IX] does not provide a general solution either, since dinaturals do not compose. Instead, the analogy to naturality is left there; the approach taken by Reynolds [Rey83], and consequently by most of the literature on parametric polymorphism, is to state parametricity in terms of *relations* instead of *functions*. Indeed, [HRR14] goes so far as to suggest that Reynolds's solution—"to generalize homomorphisms from functions to relations"—ought to be carried out across mathematics more broadly.

The present author sought to push back on this suggestion, on account of the more cumbersome nature of relational calculi, as well as a desire not to reinvent category theory in a relational mould (e.g. the theory of allegories [FS90]). However, to meet this challenge, defenders of function-based mathematics would need to formulate parametricity in a functional, category-theoretic way, i.e. extend the notion of naturality to mixed-variant functors so as to complete the connection above. In the preprint *Paranatural Category Theory* [Neu23], I sought to develop the category theory surrounding the most promising candidate—*strong dinatural transformations* [Mul92]—towards such a possible solution. The purpose of this talk is discuss the current status of this theory, and the difficulties that have emerged since the first draft of the preprint.

One area of focus will be the failure of the *di-Yoneda Lemma*—an analogue of the Yoneda Lemma, for difunctors and strong dinatural transformations—as originally stated. If the category of difunctors and strong dinatural transformations had such a Yoneda Lemma, then it would be possible to define exponentials in this category analogously to how they're defined in presheaf categories, thereby (potentially) bypassing some of the known issues with using strong dinatural transformations as a formalism for parametricity. I speculate on whether some restricted class of difunctors can be identified which *do* have such a Yoneda Lemma, and whether this class includes the examples important in practice.

I'll also cover some progress towards building a strong (co)end calculus. The original preprint included a development of this calculus, which generalizes the work of Awodey, et al. [AFS18]

[1] See the summary in [HRR14, Sect. 1] and the references cited there.

encoding inductive types, to cover coinductive and existential types; it also included the Yoneda-like Lemmas due to Uustalu [Uus10], connecting strong dinaturality to initial algebras/terminal coalgebras. However, the original preprint left much of the connection existing work on the (co)end calculus unexplored and several questions unanswered, which I hope to address in this talk.

References

[AFS18] Steve Awodey, Jonas Frey, and Sam Speight. Impredicative encodings of (higher) inductive types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 76–85, 2018.

[FS90] Peter J Freyd and Andre Scedrov. *Categories, allegories*. Elsevier, 1990.

[HRR14] Claudio Hermida, Uday S Reddy, and Edmund P Robinson. Logical relations and parametricity–a reynolds programme for category theory and programming languages. *Electronic Notes in Theoretical Computer Science*, 303:149–180, 2014.

[ML78] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer New York, 1978.

[Mul92] Philip S Mulry. Strong monads, algebras and fixed points. *Applications of Categories in Computer Science*, 177:202–216, 1992.

[Neu23] Jacob Neumann. Paranatural category theory. https://arxiv.org/abs/2307.09289, 2023.

[Rey83] John C Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congres*, pages 513–523, 1983.

[Uus10] Tarmo Uustalu. A note on strong dinaturality, initial algebras and uniform parameterized fixpoint operators. In *FICS*, pages 77–82, 2010.

[Wad89] Philip Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, pages 347–359, 1989.

Session 10: Models of Type Theory

Recent progress in the theory of effective Kan fibrations in simplicial sets

Benno van den Berg

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Postbus 90242, 1090 GE Amsterdam
The Netherlands
B.vandenBerg3@uva.nl

One of the most important steps in the development of homotopy type theory has been the construction, by Voevodsky, of a model of type theory with the univalence axiom in the category of simplicial sets [10]. This work builds on the classic Kan-Quillen model structure in simplicial sets.

From the very beginning people have been trying to understand how *constructive* Voevodsky's results are. Besides being of intrinsic interest, it is also relevant for the question whether these results hold relative to an arbitrary base topos. Perhaps most importantly it also asks whether one can *compute* with the univalence axiom, or any other principle that might hold in the simplicial model.

Early on, an obstruction was found by Bezem, Coquand and Parmann [4]. They observed that the classic result saying that the exponential A^B is a Kan complex whenever A and B are, is not provable constructively. We refer to this as the *BCP-obstruction*. Since Kan complexes are interpreting the types in Voevodsky's model and the exponentials are the obvious way to interpret function spaces, this blocks a direct constructive interpretation of function types in Voevodsky's model.

Indeed, the best results that we have in this direction can be found in the work of Gambino, Henry, Sattler and Szumiło [9, 6, 8]. After Henry showed that the Kan-Quillen model structure can be proven to exist constructively, these authors showed that there are basically two obstacles to obtaining a constructive account of Voevodsky's model. First, one would only have weak function types, with rules weaker than the usual ones, due to the BCP-obstruction. Secondly, to obtain a genuine model of type theory, a difficult coherence problem needs to be solved for which currently no solution is known.

In response most researchers have switched to *cubical sets*. This does not only involve changing the shapes, but also involves strengthening the notion of a Kan complex, or a Kan fibration, by adding *uniformity conditions* [3, 5]. Indeed, the usual definition of a Kan complex requires the mere existence of fillers against a class of maps, whether these are horn inclusions or open box inclusions. The other innovation is to insist that a Kan fibration comes equipped with a system of solutions which is required to satisfy certain compatibility conditions. It is in this way that one can overcome the BCP-obstruction in cubical sets.

While this has sometimes been taken to mean that cubical sets are constructively superior, matters are really not that clear. Indeed, as observed by Gambino and Sattler [7], uniformity conditions can also be used to overcome the BCP-obstruction in simplicial sets. Indeed, in their paper they define a notion of a uniform Kan complex, mirroring the cubical definition, and show that if A is a uniform Kan complex, then so is A^B for any simplicial set B.

In a book written with Eric Faber [1], we gave another solution which we call *effective Kan fibrations*, using uniformity conditions stronger than those of Gambino and Sattler. In contrast to Gambino and Sattler's notion, our definition is *local*. This means that we can show the existence of universal effective Kan fibrations, which should allow us to interpret type-theoretic universes. Indeed, the main results of our book are:

(1) Every effective Kan fibration is a Kan fibration in the usual sense, and in a classical metatheory one can show that every Kan fibration can be equipped with the structure of an effective Kan fibration.

(2) Whenever f and g are effective Kan fibrations, then so is $\Pi_f(g)$, the push forward of g along f.

(3) Being an effective Kan fibration is a local property and hence universal effective Kan fibrations exist.

The ultimate aim is to develop a constructive proof of the existence of both a model of homotopy type theory and the Kan-Quillen model structure on simplicial sets using the notion of an effective Kan fibrations. Unfortunately, this remains work in progress.

In the meantime, the speaker has obtained, often together with (former) MSc students, some further results and the purpose of this talk is to report on these. In particular, we have shown that:

1. Any simplicial group is effectively Kan ([2], jww with Freek Geerligs).
2. The effective Kan fibrations are cofibrantly generated by a countable double category ([2], jww with Freek Geerligs). Classically, this means they are the right class in algebraic weak factorisation system.
3. Whenever f is an effective Kan fibration, then W_f, the W-type associated to f is an effective Kan complex (jww with Shinichiro Tanaka).
4. A version of the Joyal-Tierney calculus works for effective Kan fibrations (jww with Eric Faber).

Since we are still working on related issues, we may have more to report in June.

References

[1] Benno van den Berg and Eric Faber. *Effective Kan fibrations in simplicial sets*. Vol. 2321. Lecture Notes in Mathematics. Springer, Cham, [2022] ©2022, pp. x+230.

[2] Benno van den Berg and Freek Geerligs. "Examples and cofibrant generation of effective Kan fibrations". arXiv:2402.10568. 2024.

[3] Marc Bezem, Thierry Coquand, and Simon Huber. "The Univalence Axiom in Cubical Sets". *J. Autom. Reasoning* 63.2 (2019), pp. 159–171.

[4] Marc Bezem, Thierry Coquand, and Erik Parmann. "Non-constructivity in Kan simplicial sets". *13th International Conference on Typed Lambda Calculi and Applications*. Vol. 38. LIPIcs. Leibniz Int. Proc. Inform. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2015, pp. 92–106.

[5] Cyril Cohen et al. "Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom". *FLAP* 4.10 (2017), pp. 3127–3170.

[6] Nicola Gambino and Simon Henry. "Towards a constructive simplicial model of univalent foundations". Accepted for publication in *Journal of the London Mathematical Society*. arXiv:1905.06281. 2019.

[7] Nicola Gambino and Christian Sattler. "The Frobenius condition, right properness, and uniform fibrations". *J. Pure Appl. Algebra* 221.12 (2017), pp. 3027–3068.

[8] Nicola Gambino, Christian Sattler, and Karol Szumiło. "The constructive Kan-Quillen model structure: two new proofs". Accepted for publication in *The Quarterly Journal of Mathematics*.arXiv:1907.05394. 2019.

[9] Simon Henry. "A constructive account of the Kan-Quillen model structure and of Kan's Ex^{∞} functor". arXiv:1905.06160. 2019.

[10] Krzysztof Kapulkin and Peter LeFanu Lumsdaine. "The simplicial model of univalent foundations (after Voevodsky)". *J. Eur. Math. Soc. (JEMS)* 23.6 (2021), pp. 2071–2126.

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.

Coherent Categories with Families

Thorsten Altenkirch[1] and Ambrus Kaposi[2]

[1] School of Computer Science, University of Nottingham, UK
psztxa@nottingham.ac.uk
[2] Eötvös Loránd University, Budapest, Hungary
{akaposi,vetuaat}@inf.elte.hu

Categories with Families (CwF) provide a natural semantics for Type Theory so that the intrinsic syntax of Type Theory directly gives rise to an initial CwF represented as a Quotient Inductive-Inductive Type [2]. Already in loc.cit it was noted that in a Type Theory without Uniqueness of Equality (UIP) like Homotopy Type Theory (HoTT) we cannot interpret the syntax in the standard model, i.e. Sets. Instead we used a inductive-recursive universe which is not univalent to overcome this issue.

Why do we have this problem? A category in HoTT is given by a type of objects and a family of homsets which are indeed sets, i.e. 0-truncated types [1] . However, in the syntax we want contexts and types to be represented as a set, which forces us to consider strict categories (categories whose type of objects is a set). Now there is no issue if we only consider categories because they don't impose any equations on objects. However, this changes once we move to CwFs which also model types (as a presheaf over the category of contexts) and terms (as an indexed presheaf over types). Here we have equations on types to represent their functoriality and for each type-former telling us how they behave under substitutions (Beck-Chevalley conditions). If the presheaf of types is not set-truncated then these conditions are no longer propositions. We can add set-truncation to our definition of types and this is what we did in our paper [2] but this means that we cannot interpret the syntax of type theory in the standard model or other semantic models, eg. in the container model of type theory [3].

To deal with this issue we introduce the notion of a coherent CwF or 1-CwF while the set-truncated CwFs we call set CwFs or 0-CwFs. In a coherent CwF we truncate the presheaf types to groupoids (1-types). We add coherence equations so for example the pentagon law for type substitutions, and coherence equations for the substitution laws on type constructors. We can now show that the set-model and other semantic models are coherent CwFs. However, it is not clear how we can interpret the syntax of type theory which is the initial set CwF in coherent CwFs.

This leads to the main result we want to prove: a coherence theorem for 1-CwFs, namely that the initial 1-CwF is indeed a set-CwF. This is not very surprising because we haven't added any constructions which should generate higher equalities but it means that we have added enough coherences to kill any higher equalities. Given this result we know that the initial coherent CwF is a set CwF and hence we can interpret the syntax of Type Theory in any set CwF.

Our approach to prove this result is based on type normalisation. We define normal types in the initial coherent CwF as ones which are generated without using type substitutions. We can now define type-substitution recursively, not that we only need to do this for type substitutions since term substitutions take place in the set of terms. The normal types should form a set which

[1] A proposition or a (-1)-truncated type is one where all elements are equal (proof-irrelevance), a set of 0-truncated type is one whose equality is propositional, and a groupoid or a 1-type is one whose equalities are sets.

[2] This corresponds to split-type CwFs in [1]. Comprehension categories are just an equivalent representation of the same structure.

means that the coherent CwF with normal types is a set CwF and moreover it is isomorphic to the initial coherent CwF and it is also isomorphic to the initial set CwF which entails our result.

We have a proof sketch on paper but when starting to formalize in cubical agda we encountered some obstacles which may be a consequence of features lacking in cubical agda or subtle problems with our proof.

It is natural to ask why we 1-truncate types. Indeed our tentative result is a special case of [4] but this requires a quite sophisticated ∞-categorical framework and it is not clear how this result can be stated in vanilla HoTT without introducing 2-level type theory. On the other hand restricting to groupoids captures most semantic categories and indeed we know that all univalent categories can be equivalently 1-truncated.

References

[1] Benedikt Ahrens, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. Categorical structures for type theory in univalent foundations. *Logical Methods in Computer Science*, 14, 2018.

[2] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 18–29, New York, NY, USA, 2016. ACM.

[3] Thorsten Altenkirch and Ambrus Kaposi. A container model of type theory. 2021.

[4] Taichi Uemura. Normalization and coherence for ∞-type theories. *arXiv preprint arXiv:2212.11764*, 2022.

Type theory in type theory using single substitutions

Ambrus Kaposi and Szumi Xie

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

What is the syntax of Martin-Löf type theory? In this abstract, we give a minimalistic answer to this question. We aim to define the syntax of type theory only using operations that are unavoidable. We would also like to eschew boilerplate by only defining well-typed terms which are quotiented by conversion [2]. This means that type theory is a generalised algebraic theory (GAT), its syntax is a quotient inductive-inductive type (QIIT). Usual such definitions of type theory are based on categories: category with families [6], locally cartesian closed category [9, 7], contextual category [5, 4], C systems [1], etc. In this abstract we show that categories are not necessary for defining the GAT of type theory. A category-free definition of type theory is B-systems [1] which includes telescopes with complex operations and equations. In our definition we avoid these as well. In this abstract we give a tutorial-style introduction to our minimalistic definition of type theory, no prior knowledge of the metatheory of type theory is required. We introduce the operations in a naive, logical order.

We need variables in our language, so we introduce sorts of contexts, types (which depend on a context) and variables (which are in a context and have a type).

$$\mathsf{Con} : \mathsf{Set} \qquad \mathsf{Ty} : \mathsf{Con} \to \mathsf{Set} \qquad \mathsf{Var} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$$

Contexts are either empty or are built from a context extended with a type.

$$\diamond : \mathsf{Con} \qquad - \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$$

We define variables as well-typed De Bruijn indices, but to express these we need to weaken types: e.g. the zero De Bruijn index vz has a weakened type. We introduce a new sort for substitutions Sub, an instantiation operation $-[-]$ on types, and a weakening substitution p. For now, a separate sort Sub seems like an overkill because we are only using $-[\mathsf{p}]$, but it will come handy soon.

$$\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \qquad -[-] : \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta \qquad \mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$$

$$\mathsf{vz} : \mathsf{Var}\,(\Gamma \triangleright A)\,(A[\mathsf{p}]) \qquad \mathsf{vs} : \mathsf{Var}\,\Gamma\,A \to \mathsf{Var}\,(\Gamma \triangleright B)\,(A[\mathsf{p}])$$

Now we introduce Π types together with an equation on how instantiation with p acts on them. This is tricky: as Π binds a new variable in its second argument, we need a new version of the weakening substitution which leaves the last variable untouched. This is why we introduce lifting of a substitution $-^+$, and now we can state a general instantiation rule for Π which works not only for p, but arbitrary substitutions (including lifted ones).

$$\Pi : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma \qquad -^+ : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta \triangleright A[\gamma])\,(\Gamma \triangleright A)$$
$$\Pi[] : (\Pi\,A\,B)[\gamma] = \Pi\,(A[\gamma])\,(B[\gamma^+])$$

In addition to having variables, we need a sort of terms which includes variables and lambda abstraction.

$$\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \qquad \mathsf{var} : \mathsf{Var}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \qquad \mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$$

To express application, we need single substitutions as well because the argument of the function appears in the return type. In addition to p and $-^+$, $\langle - \rangle$ is the third and last operation for creating substitutions.

$$\langle - \rangle : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A) \qquad - \cdot - : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\langle a \rangle])$$

Now we would like to express the β computation rule, but for this we also need to be able to instantiate terms (in addition to types).

$$-[-] : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]) \qquad \Pi\beta : \mathsf{lam}\,t \cdot a = t[\langle a\rangle]$$

Now that we have instantiation of terms, we need to revisit all operations producing terms and provide rules on how to instantiate them: first of all, we need instantiation rules for lam and $-\cdot-$. The rule $\mathsf{lam}[]$ is well-typed because of $\Pi[]$, however $\cdot[]$ is not well-typed on its own and requires a new equation $[\langle\rangle]$.

$$\mathsf{lam}[] : (\mathsf{lam}\,t)[\gamma] = \mathsf{lam}\,(t[\gamma^+]) \quad [\langle\rangle] : A[\langle a\rangle][\gamma] = A[\gamma^+][\langle a[\gamma]\rangle] \quad \cdot[] : (t\cdot a)[\gamma] = (t[\gamma])\cdot(a[\gamma])$$

Then we need instantiation rules for variables, we list these for each possible substitution Sub: weakening of a variable increases the index by one; when instantiating with lifted substitutions and single substitutions, we have to do case distinction on the De Bruijn index whether it is zero or successor. For the latter two cases, we need type equations (named $[\mathsf{p}][^+]$ and $[\mathsf{p}][\langle\rangle]$) to typeckeck the term equations.

	$\mathsf{var}\,x[\mathsf{p}] = \mathsf{var}\,(\mathsf{vs}\,x)$	
$[\mathsf{p}][^+] : A[\mathsf{p}][\gamma^+] = A[\gamma][\mathsf{p}]$	$\mathsf{var}\,\mathsf{vz}[\gamma^+] = \mathsf{var}\,\mathsf{vz}$	$\mathsf{var}\,(\mathsf{vs}\,x)[\gamma^+] = \mathsf{var}\,x[\gamma][\mathsf{p}]$
$[\mathsf{p}][\langle\rangle] : A[\mathsf{p}][\langle a\rangle] = A$	$\mathsf{var}\,\mathsf{vz}[\langle a\rangle] = a$	$\mathsf{var}\,(\mathsf{vs}\,x)[\langle a\rangle] = \mathsf{var}\,x$

Finally, to typecheck the $\Pi\eta$ rule, we need our last equations on types.

$$[\mathsf{p}^+][\langle\mathsf{vz}\rangle] : A[\mathsf{p}^+][\langle\mathsf{var}\,\mathsf{vz}\rangle] = A \qquad \Pi\eta : t = \mathsf{lam}\,(t[\mathsf{p}] \cdot \mathsf{var}\,\mathsf{vz})$$

This concludes all the rules for type theory with Π (this type theory is actually empty because there are no base types, but is enough to illustrate our method). We summarise as follows: there are three kinds of substitutions (single weakening, single substitution, lifted substitution), we have 5 equations describing how instantiation acts on variables, and 4 equations which describe general properties of instantiation on types. The rest of the rules are specific to our single type former Π: the only extra requirement is that each operation is equipped with an instantiation rule ($\Pi[]$, $\mathsf{lam}[]$, $\cdot[]$). Perhaps surprisingly, this is enough to define the syntax: there is no need for Con and Sub to a form a category, no need for parallel substitutions, empty substitution, parallel weakenings, telescopes, or combinations of these.

When adding new type formers, we only need the rules for the type former, and an extra instantiation (naturality) rule for each operation. For example, a Coquand-universe can be added by $\mathsf{U} : \mathsf{Ty}\,\Gamma$, $\mathsf{El} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma$, $\mathsf{c} : \mathsf{Ty}\,\Gamma \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$, $\mathsf{U}\beta : \mathsf{El}\,(\mathsf{c}\,A) = A$, $\mathsf{U}\eta : \mathsf{c}\,(\mathsf{El}\,a) = a$, and three instantiation rules (note that we would need indexing U and Ty by universe levels to avoid inconsistency). In [8], we showed that any second-order generalised algebraic theory has a single substitution presentation.

In the syntax (initial model, QIIT) of the above theory, all the rules of categories with families (CwF [6]) are admissible. That is, by induction on the single substitution syntax, we can define parallel substitutions (lists of terms) which are composable and form a category; we can define instantiation by parallel substitutions for types and terms, these have the usual universal property of comprehension. The main ingredient for this construction is the notion of α-normal form: a kind of normal form which is still quotiented by $\Pi\beta$, $\Pi\eta$, but does not include explicit substitutions. If a type is in α-normal form, we know whether it is Π, U, or El of a term. If a term is in α-normal form, we know whether it is a variable, a lam, an application or a code for a type (note that any function on an α-normal type/term has to respect $\Pi\beta$, $\Pi\eta$, $\mathsf{U}\beta$, $\mathsf{U}\eta$). We prove α-normalisation (every term has a unique α-normal form) and then use induction on α-normal forms to define parallel instantiation and prove its properties.

We formalised a single substitution calculus with an infinite hierarchy of types closed under Π and U as a QIIT in Agda, proved α-normalisation, and derived all the rules for parallel substitutions (CwF

equipped with Π types and U): https://bitbucket.org/akaposi/single. In the formalisation, we use sProp-valued equality and the syntax is postulated as a QIIT with rewrite rules for its β laws.

It is clear that the rules for the single substitution calculus are all derivable from the CwF-rules. The other direction is however not true: there are more models of the single substitution calculus than the parallel substitution calculus, but the syntaxes are isomorphic. The situation is analogous to the relationship of combinatory logic and lambda calculus: their syntaxes are isomorphic, but the former has more models [3].

References

[1] Benedikt Ahrens, Jacopo Emmenegger, Paige Randall North, and Egbert Rijke. B-systems and C-systems are equivalent. *The Journal of Symbolic Logic*, page 1–9, 2023. doi:10.1017/jsl.2023.41.

[2] 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.

[3] Thorsten Altenkirch, Ambrus Kaposi, Artjoms Sinkarovs, and Tamás Végh. Combinatory logic and lambda calculus are equal, algebraically. 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 24:1–24:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.FSCD.2023.24, doi:10.4230/LIPICS.FSCD.2023.24.

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

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

[6] 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.

[7] Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and martin-löf type theories. *Math. Struct. Comput. Sci.*, 24(6), 2014. doi:10.1017/S0960129513000881.

[8] Ambrus Kaposi and Szumi Xie. Second-order generalised algebraic theories: signatures and first-order semantics, 2024. To appear at FSCD 2024. Available: https://akaposi.github.io/sogat.pdf.

[9] R. A. G. Seely. Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society*, 95(1):33, 1984.

Session 11: New Type Theories

Harmony in Duality

Henning Basold and Herman Geuvers

[1] Leiden University, h.basold@liacs.leidenuniv.nl
[2] Radboud University, herman@cs.ru.nl

In previous work [BG16], we devised a type theory that grew out of the desire for a theory of harmonious duality of inductive and coinductive types. We showed how common type formers, such as dependent sum, dependent products and equality fit into this duality. However, that type theory has the caveat that the resulting equality type mimics the identity type of Martin-Löf type theory (MLTT), meaning that it is quite weak on coinductive types. For instance, one cannot prove function extensionality, let alone element-wise equality of infinite sequences. Observational type theory (OTT) [AMS07] offers a different perspective on equality and defines equality proofs by induction on types to the effect that, for instance, function extensionality becomes provable. Later work has combined observational type theory with homotopy-theoretical ideas [CFM18], and ideas from OTT also entered cubical type theory [SAG19]. In all these theories, function extensionality and similar useful identies are provable. Unfortunately, none of these theories has a clear duality between inductive and coinductive types.

Actually, let us take the usual view on equality on a type A as an inductive type $\mathbf{Eq}_A$ with one constructor for reflexivity as introduction principle and recursion as elimination principle. In Agda, this type could be defined as on the left below, where $\top$ is the unit type.

$$\begin{array}{ll} \textbf{data } \mathbf{Eq}_A : \mathrm{A} \to \mathrm{A} \to \textbf{Ty where} & \textbf{codata } \mathbf{InEq}_A : \mathrm{A} \to \mathrm{A} \to \textbf{Ty where} \\ \quad \mathrm{refl} \ : (\mathrm{x} : \mathrm{A}) \to \top \to \mathbf{Eq}\ \mathrm{x}\ \mathrm{x} & \quad \mathrm{irefl} \ : (\mathrm{x} : \mathrm{A}) \to \mathbf{InEq}\ \mathrm{x}\ \mathrm{x} \to \bot \end{array}$$

Dualising this type, we end up with the coinductive type on the right, which is written in wishful Agda syntax. The way to read it is that irefl is an observer for the type $\mathbf{InEq}_A$, which is only enabled for elements of the type $\mathbf{InEq}_A$ x x but not for off-diagonal cases. This type looks suspiciously like the negation of equality and, indeed, one can define functions between **InEq** x y and $\neg$(**Eq** x y) by using standard recursion and corecursion principles, which shows that these types are logically equivalent. In other words, dualising equality yields something that is not always useful in constructive mathematics. That being said, this gives us a clue what to look for: a constructive version of inequality, which is of course *apartness* [TvD88; BSV02]. It turns out that apartness is naturally *inductive* because one has to present finite evidence that two things are not the same [GJ21]. For instance, two real numbers are apart if there is a rational number in between. The dual of apartness is bisimilarity [GJ21] and, by adopting the *coinduction principle*, we can identify bisimilarity and equality. Thus, one is lead to the view that equality is naturally a coinductive relation, contrary to the usual inductive view of MLTT.

In this talk, I aim to show how a type theory can be constructed around an inductive apartness relation and a coinductive equality relation. This type theory is based on simple introduction and elimination principles for type-level equality and apartness. The elimination principle for type equality is coercion, as in OTT, while that for apartness results from suitably adapting the so-called strong extensionality principle for apartness [TvD88]. Equality and apartness proofs between terms are obtained as corecursion and recursion principles on relations, essentially encoding the fibrational view on bisimilarity and apartness in the type theory [GJ21]. The term-level equality and apartness are then brought to the type-level by using the idea of explicit parameter handling that was at the foundation of our previous type theory with dual inductive-coinductive dependent types [BG16].

Explicitly, we work with judgements of the form $\Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow \mathbf{Ty}$, which says that A is a type with free variables in context Γ_1 and parameters in context Γ_2. The context and type formation rules from the previous system [BG16] are restricted to rule out equality formation, but extended with explicit equality and apartness relations on the term and type level:

$$\frac{\Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow \mathbf{Ty} \quad \Gamma_1 \vdash B : \Gamma_2 \twoheadrightarrow \mathbf{Ty}}{\Gamma \vdash A \sim B \;\; \mathbf{TyRel}} \qquad \frac{\Gamma \vdash A, B : \mathbf{Ty} \quad \Gamma \vdash s : A \quad \Gamma \vdash t : B}{\Gamma \vdash s \sim t : \mathbf{Ty}}$$

$$\frac{\Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow \mathbf{Ty} \quad \Gamma_1 \vdash B : \Gamma_2 \twoheadrightarrow \mathbf{Ty}}{\Gamma \vdash A \# B \;\; \mathbf{TyRel}} \qquad \frac{\Gamma \vdash A, B : \mathbf{Ty} \quad \Gamma \vdash s : A \quad \Gamma \vdash t : B}{\Gamma \vdash s \# t : \mathbf{Ty}}$$

To state the remaining rules, it is important to know that parameterised types have application and abstraction, akin to the simply typed λ-calculus:

$$\frac{\Gamma_1 \vdash A : (x : B, \Gamma_2) \twoheadrightarrow \mathbf{Ty} \quad \Gamma_1 \vdash s : B}{\Gamma_1 \vdash A\, s : \Gamma_2[s/x] \twoheadrightarrow \mathbf{Ty}} \qquad \frac{\Gamma_1, x : B \vdash A : \Gamma_2 \twoheadrightarrow \mathbf{Ty}}{\Gamma_1 \vdash (x).\, A : (x : B, \Gamma_2) \twoheadrightarrow \mathbf{Ty}}$$

Then we can state the rules for type-level equality, where we write $\equiv$ for definitional equality:

$$\frac{\Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow \mathbf{Ty} \quad \Gamma_1 \vdash B : \Gamma_2 \twoheadrightarrow \mathbf{Ty} \quad A \equiv B}{\Gamma \vdash \mathrm{Refl} : A \sim B}$$

$$\frac{\Gamma_1 \vdash A : (x : B, \Gamma_2) \twoheadrightarrow \mathbf{Ty} \quad \Gamma_1 \vdash s, t : B \quad \Gamma_1 \vdash p : s \sim t}{\Gamma_1 \vdash A\, p : A\, s \sim A\, t}$$

Type equality proofs can then be used, as in OTT, by means of coercion and coherence:

$$\frac{\Gamma \vdash Q : A \sim B \quad \Gamma \vdash s : A}{\Gamma \vdash s[Q\rangle : B} \qquad \frac{\Gamma \vdash Q : A \sim B \quad \Gamma \vdash s : A}{\Gamma \vdash \{s \,||\, Q\} : s \sim s[Q\rangle}$$

We then also need principles to eliminate apartness of types:

$$\frac{\Gamma \vdash A : \mathbf{Ty} \quad \Gamma \vdash B : \mathbf{Ty} \quad A \equiv B \quad \Gamma \vdash Q : A \# B \quad \Gamma \vdash C : \mathbf{Ty}}{\Gamma \vdash \#\mathrm{elim}\, Q : C}$$

$$\frac{\Gamma \vdash A : (x : B) \twoheadrightarrow \mathbf{Ty} \quad \Gamma \vdash s, t : B \quad \Gamma, y : s \# t \vdash p_1 : B \quad \Gamma, z : A\, t \vdash p_2 : B \quad \Gamma \vdash q : A\, s}{\Gamma \vdash \mathrm{extr}(y.p_1, z.p_2)\, q : B}$$

The first of these two says that type-level apartness implies the negation of definitional equality. Underlying the second rules is the strong extensionality principle $A\, s \to s \# t \vee A\, t$, which says that whenever A holds on s then any t is either apart from s or A must hold also for it [TvD88, p. 386]. The above rule combines strong extensionality with elimination of disjunction.

From these rules alone, we can already derive that $\sim$ is an equivalence relation on terms, and that $\#$ is a (pre)apartness relation. To prove concrete identities and differences, we also need introduction rules for term-level equality and apartness. In the talk, I will show how these are obtained in a standard way by mimicking lifting types with type variables to relations, just as functors are lifted to relations [HJ97]. Note that the equality and apartness relations on terms are heterogeneous, like in OTT, which is required for formulating coherence. This is convenient but can be improved upon. Unfortunately, I do not know at this point how the relation lifting can be reconciled with the cubical approach [CFM18; SAG19]. It seems that the cubical approach picks out the canonical relation lifting for equality but then we must ask what the dual of that is, if negation should be avoided. Finally, we will discuss computations for this calculus, and some steps towards proving canonicity and strong normalisation. Once that is done, we may enjoy for a while the harmonious duality of inductive-coinductive types.

References

[AMS07] Thorsten Altenkirch, Conor McBride and Wouter Swierstra. "Observational Equality, Now!" In: *Proc. of PLPV '07*. Workshop on Programming Languages Meets Program Verification. ACM, 2007, pp. 57–68. ISBN: 978-1-59593-677-6. DOI: 10.1145/1292597.1292608.

[BG16] Henning Basold and Herman Geuvers. "Type Theory Based on Dependent Inductive and Coinductive Types". In: *Proceedings of LICS '16*. Logic In Computer Science. ACM, 2016, pp. 327–336. DOI: 10.1145/2933575.2934514. arXiv: 1605.02206.

[BSV02] Douglas Bridges, Peter Schuster and Luminiţa Vîţă. "Apartness, Topology, and Uniformity: A Constructive View". In: *Mathematical Logic Quarterly* 48.S1 (2002), pp. 16–28. ISSN: 1521-3870. DOI: 10.1002/1521-3870(200210)48:1+<16::AID-MALQ16>3.0.CO;2-7.

[CFM18] James Chapman, Fredrik Nordvall Forsberg and Conor McBride. *The Box of Delights (Cubical Observational Type Theory)*. Unpublished Note. Jan. 2018. URL: https://github.com/msp-strath/platypus/blob/master/January18/doc/CubicalOTT/CubicalOTT.pdf.

[GJ21] Herman Geuvers and Bart Jacobs. "Relating Apartness and Bisimulation". In: *Logical Methods in Computer Science* Volume 17, Issue 3 (30th July 2021). ISSN: 1860-5974. DOI: 10.46298/lmcs-17(3:15)2021.

[HJ97] Claudio Hermida and Bart Jacobs. "Structural Induction and Coinduction in a Fibrational Setting". In: *Information and Computation* 145 (1997), pp. 107–152. DOI: 10.1006/inco.1998.2725.

[SAG19] Jonathan Sterling, Carlo Angiuli and Daniel Gratzer. "Cubical Syntax for Reflection-Free Extensional Equality". In: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*. Ed. by Herman Geuvers. Vol. 131. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 31:1–31:25. ISBN: 978-3-95977-107-8. DOI: 10.4230/LIPIcs.FSCD.2019.31.

[TvD88] Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in Mathematics: An Introduction, Volume II*. Studies in Logic and the Foundations of Mathematics 123. North-Holland, 1988. 384 pp. ISBN: 0-444-70358-6.

A modal deconstruction of Löb induction

Daniel Gratzer[1] and Lars Birkedal[1]

Aarhus University
Aarhus, Denmark

1 Guarded type theory

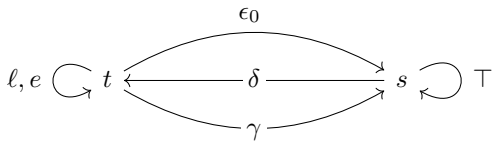
A fundamental mismatch between type theory and programming language theory is the status of recursive definitions; they are ubiquitous in programming languages and yet only allowed in limited situations in type theory. To this end, type theorists have put forward *guarded recursion* [Bir+12; Nak00] as a method for soundly and robustly including recursive definitions in type theory. Guarded recursive type theories extend type theory with a modality $\blacktriangleright$ (forming an applicative functor [MP08]) and an operator $\mathsf{loeb} : (\blacktriangleright A \to A) \to A$. The latter—Löb induction—allows users to form recursive definitions provided each recursive occurrence is "guarded" by the $\blacktriangleright$ modality. In particular, loeb unfolds like a fixed point combinator: $\mathsf{loeb}(f) = f(\mathsf{next}(\mathsf{loeb}(f)))$ where $\mathsf{next} : A \to \blacktriangleright A$ is the "return" of the applicative functor $\blacktriangleright$.

As with any type theory, the key question for guarded type theories is whether they can be shown to satisfy canonicity and normalization. Unfortunately, Gratzer and Birkedal [GB22] show that it is essentially impossible for a guarded type theory to satisfy both canonicity and normalization. Various approaches to this dichotomy have been proposed: stratified guarded type theory [GB22] and clocked cubical type theory [BGM17; KMV22] both add Löb induction to a modal type theory, but strive to limit the situations where it can unfold (essentially only on closed terms). In this work, we propose a novel solution to this problem: rather than adding and subsequently restricting Löb induction, we propose a sufficiently rich modal apparatus for guarded type theory so as to make Löb induction derivable. The resulting operator then naturally unfolds only in certain modal contexts without direct intervention.

Concretely, we instantiate and extend cubical multimodal type theory MTT [Aag+22; Gra+20] with a particular collection of modalities to make Löb induction *derivable*. The cubical aspect is fungible, but the underlying theory must be univalent: we capitalize on both univalence and the resulting good behavior of homotopy propositions in various stages of our construction. We term the resulting theory Gatsby, but we emphasize that our methodology is reasonably general and can be applied to any well-adapted modal univalent system.

2 Guarded accessible type theory: Gatsby

The starting point for Gatsby is to describe the underlying *mode theory* [LS16] used to instantiate (cubical) MTT. This is a poset-enriched category which describes the desired copies of type theory as objects and the modalities connecting them as morphisms:



Intuitively, $s = \mathbf{Set}$ and $t = \mathbf{PSh}(\omega)$ while ℓ, e are the $\blacktriangleright$ and $\blacktriangleleft$ functors [Bir+12]. The three modalities $\epsilon_0, \delta, \gamma$ represent the adjoint triple $\Pi_0 \dashv \Delta \dashv \Gamma$ between $\mathbf{PSh}(\omega)$ and $\mathbf{Set}$. The

remaining modality $\top$ is the most surprising: it encodes the operation sending a set X to $\mathbf{1}$. We further enrich this category in posets to force these modalities to interact in expected ways (e.g., $\blacktriangleleft \dashv \blacktriangleright$, $\mu_0 \circ \top \circ \nu_0 = \mu_1 \circ \top \circ \nu_1$). The crucial equation linking the $\top$ modality to the guarded apparatus is $\epsilon_0 \circ \ell = \top \circ \epsilon_0$ which captures the fact that $(\blacktriangleright X)\, 0 = \mathbf{1}$.

Unfortunately, modalities alone are insufficient to derive Löb induction:

Theorem 2.1. *There is no mode theory containing a modality μ such that $(\langle \mu \mid \bot \rangle \to \bot) \to \bot$.*[1]

In light of the above, we must add some new rule to our system to include Löb induction. Surprisingly, it suffices to add a rule governing the behavior of $\top$. More accurately, we add a rule restricting the adjoint action $-.\{\top\}$ used to specify the modal type $\langle \top \mid - \rangle$.

$$\frac{\Gamma \vdash r : \mathbf{1}.\{\top\}}{\Gamma \vdash \mathcal{J}}$$

Lemma 2.2. *The above rule implies $\langle \top \mid A \rangle \simeq \mathbf{1}$.*

We now set out to define Löb induction *relative* to the following proposition:

$$\mathsf{acc} = \left\| \textstyle\sum_{n:\mathbf{Nat}} \blacktriangleright^n \mathbf{Void} \right\|$$

In the above, we have used the more standard $\blacktriangleright$ notation rather than $\langle \ell \mid - \rangle$. The acc proposition holds when there are "finitely many steps left"; after some number of $\blacktriangleright$s are applied, the system trivializes. acc is not derivable in our system, but it does suffice to construct Löb induction and $\langle \mu \mid \mathsf{acc} \rangle$ does hold for many choices of μ:

Theorem 2.3. *If acc holds, then each $f : \blacktriangleright A \to A$ has a unique guarded fixed point.*

Theorem 2.4. *$\langle \mu \mid \mathsf{acc} \rangle$ holds if $\mu = \epsilon_0 \circ e^k$ for any $k \in \mathbb{N}$.*

The core idea of Gatsby is then to thread acc through various constructions to utilize guarded recursion in our proof. The final construction can then be placed under a modality to discharge the acc assumption and actually obtain a concrete result. To aid in this process, it is helpful to work with acc-null types; those for which $A \to (\mathsf{acc} \to A)$ is an equivalence.

Theorem 2.5. *The universe of acc-null types $\mathbf{U}_{\mathsf{acc}}$ is closed under* **Unit**, Π, Σ, $(+)$, **Bool**, **Nat**, **Id**, $\blacktriangleright$, *smaller universes of acc-null types. Furthermore, $\Box A$ is always acc-null.*

The force of the above theorem is that when working with guarded recursion, there is no need to explicitly work with acc. All types which arise in this way will be acc-null and we can therefore *produce* acc whenever we need it to use Löb induction. We summarize this as follows:

Theorem 2.6. *HoTT extended with $\blacktriangleright$, loeb, and $\Box$ can be translated into Gatsby.*

Finally, while we have not proven it, we note that our single additional rule has an obvious analog in cubical type theory whose false cofibration enjoys a similar rule. We therefore conjecture that the canonicity and normalization results for cubical type theory [Hub19; SA21] can be combined with those for MTT [Gra22] to prove these results for Gatsby.

[1] $\langle \mu \mid - \rangle$ is the MTT notation for the modal type induced by the morphism μ in the mode theory.

References

[Aag+22] Frederik Lerbjerg Aagaard, Magnus Baunsgaard Kristensen, Daniel Gratzer, and Lars Birkedal. *Unifying cubical and multimodal type theory*. 2022. arXiv: 2203.13000 [cs.LO] (cit. on p. 1).

[BGM17] Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. "The clocks are ticking: No more delays!" In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2017. DOI: 10.1109/LICS.2017.8005097. URL: http://www.itu.dk/people/mogel/papers/lics2017.pdf (cit. on p. 1).

[Bir+12] Lars Birkedal, Rasmus Møgelberg, Jan Schwinghammer, and Kristian Støvring. "First steps in synthetic guarded domain theory: step-indexing in the topos of trees". In: *Logical Methods in Computer Science* 8.4 (2012). Ed. by Patrick Baillot. DOI: 10.2168/LMCS-8(4:1)2012 (cit. on p. 1).

[Gra22] Daniel Gratzer. "Normalization for Multimodal Type Theory". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '22. Haifa, Israel: Association for Computing Machinery, 2022. ISBN: 9781450393515. DOI: 10.1145/3531130.3532398. URL: https://doi.org/10.1145/3531130.3532398 (cit. on p. 2).

[GB22] Daniel Gratzer and Lars Birkedal. "A Stratified Approach to Löb Induction". In: *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*. Ed. by Amy Felty. Vol. 228. Leibniz International Proceedings in Informatics (LIPIcs). Saarbrücken, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2022. DOI: 10.4230/LIPIcs.FSCD.2022.3. URL: https://jozefg.github.io/papers/a-stratified-approach-to-lob-induction.pdf (cit. on p. 1).

[Gra+20] Daniel Gratzer, G.A. Kavvos, Andreas Nuyts, and Lars Birkedal. "Multimodal Dependent Type Theory". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. ACM, 2020. DOI: 10.1145/3373718.3394736 (cit. on p. 1).

[Hub19] Simon Huber. "Canonicity for Cubical Type Theory". In: *Journal Automated Reasoning* 63.2 (2019), pp. 173–210. DOI: 10.1007/s10817-018-9469-1. URL: https://doi.org/10.1007/s10817-018-9469-1 (cit. on p. 2).

[KMV22] Magnus Baunsgaard Kristensen, Rasmus Ejlers Møgelberg, and Andrea Vezzosi. "Greatest HITs: Higher inductive types in coinductive definitions via induction under clocks". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. New York, NY, USA: Association for Computing Machinery, 2022. DOI: 10.1145/3531130.3533359 (cit. on p. 1).

[LS16] Daniel R. Licata and Michael Shulman. "Adjoint Logic with a 2-Category of Modes". In: *Logical Foundations of Computer Science*. Ed. by Sergei Artemov and Anil Nerode. Springer International Publishing, 2016, pp. 219–235. DOI: 10.1007/978-3-319-27683-0_16 (cit. on p. 1).

[MP08] Conor McBride and Ross Paterson. "Applicative programming with effects". In: *Journal of Functional Programming* 18.1 (2008). DOI: 10.1017/S0956796807006326. URL: http://www.staff.city.ac.uk/~ross/papers/Applicative.pdf (cit. on p. 1).

[Nak00] H. Nakano. “A modality for recursion”. In: *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*. IEEE Computer Society, 2000, pp. 255–266 (cit. on p. 1).

[SA21] Jonathan Sterling and Carlo Angiuli. “Normalization for Cubical Type Theory”. In: *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS ’21. New York, NY, USA: ACM, 2021 (cit. on p. 2).

Poset Type Theory

Thierry Coquand and Jonas Höfer

University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden

Our work is motivated by a recent construction of a model of univalent type theory by Sattler (a description of which can be found in [6, 10]). This construction is done in two steps. One instance of the first step, is a cubical model based on presheaves over finite, non-empty posets (in contrast with simplicial sets based on presheaves over finite non-empty linear posets). In a second step, a new model of type theory is obtained by localizing along a lex modality. This second model validates Whitehead's principle, and dependent choice. Furthermore, it is expected to give a *good* constructive notion of homotopy types of spaces. This was not the case for previous cubical type theories and associated model structures [3, 9, 11].

We design a cubical type theory based on the first model, which can be interpreted in the second model as well. As a cubical type theory, it supports homogeneous composition and coercion between arbitrary endpoints, connections, and diagonal cofibrations. This makes it an extension of both Cartesian cubical type theory [2], and the Dedekind version of CCHM [4]. Furthermore, using the connections it is possible to define an equivariant coercion operation, that is, a coercion in n variables that is invariant under permutation of dimensions. We have written a corresponding type checker and evaluator.[1] Furthermore, we conjecture it to have good meta theoretical properties, such as normalization and decidability of type checking. Lastly, this work is a first step towards a type theory for the localized model that validates the additional reasoning principles.

Base Category, Birkhoff Duality, and Cofibrations We describe the structure of the first model which we operationalized. We make heavy use of Birkhoff duality between finite non-empty posets and finite, bounded, non-degenerate distributive lattices. Note that finite distributive lattices are exactly finitely presented distributive lattices.

Theorem 1 (Birkhoff duality [12])**.** *Let* $[1]$ *denote the two element poset as well as the two element distributive lattice. The contravariant functors* $\mathrm{Pos}(-,[1]) : \mathrm{Pos} \to \mathrm{DLat}$ *and* $\mathrm{DLat}(-,[1]) : \mathrm{DLat} \to \mathrm{Pos}$ *are adjoint equivalences.*

Our base category of cubes $\square$ is given by finite non-empty posets. We construct a cubical model of univalent type theory by starting with the standard model of extensional type theory on $\widehat{\square}$. The interval $\mathbb{I}$ is modelled, as usual, by the Yoneda embedding of $[1]$. The cofibration classifier Φ is constructed at some stage X by starting with the meet-semilattice of sieves on X generated by embeddings, and freely adding joins. This is a proper subset of the set of all stage-wise decidable sieves, for example, the sieve generated by $\{0,1\} \hookrightarrow \{0 < 1\}$ is not part of $\Phi(\{0 < 1\})$, but this choice of cofibrations yield the right model structure for the localized model as well. These sieves correspond to the ones generated by equations from the lattice point of view. Embeddings correspond exactly to subposets, and under the duality, they correspond to adding equations to a finite presentation of a lattice. Given $\mathbb{I}$ and Φ, we can extract a model of univalent type theory by standard means [5, 8], by restricting to those types which admit a coercion and homogeneous composition operation for our choices of $\mathbb{I}$ and Φ.

In the implementation, we represent all semantic objects using the lattice point of view. We exploit that, by Birkhoff duality, every finite distributive lattice can be embedded into a

[1] https://github.com/JonasHoefer/poset-type-theory

$$
\begin{array}{lll}
A, B, v, w & ::= & k \mid (\lambda_x t)\rho \mid (v, v) \mid (\lambda_z^{v,v} t)\rho \mid \mathsf{ext}\, u\, [\varphi \hookrightarrow v] \\
& & \mathsf{U} \mid \Pi\, A\, B \mid \Sigma\, A\, B \mid \mathsf{Path}\, A\, v\, v \mid \mathsf{Ext}\, v\, [\varphi \hookrightarrow (v, w, w')] \\
& & \mathsf{coe}^{r \to s}\, (\lambda_z \Pi\, A\, B)\alpha \quad \mid \mathsf{coe}^{r \to s}\, (\lambda_z \Sigma\, A\, B)\alpha \quad \mid \mathsf{coe}^{r \to s}\, (\lambda_z \mathsf{Path}\, A\, v\, v)\alpha \\
& & \mathsf{coe}^{r \to s}\, (\lambda_z \Pi\, A\, B)\alpha\, v_0 \mid \mathsf{coe}^{r \to s}\, (\lambda_z \Sigma\, A\, B)\alpha\, v_0 \mid \mathsf{coe}^{r \to s}\, (\lambda_z \mathsf{Path}\, A\, v\, v)\alpha\, v_0 \\
& & \mathsf{coe}^{r \to s}\, (\mathsf{Ext}\, A\, [\varphi \hookrightarrow (v, w, w')]) \mid \mathsf{hcomp}^{r \to s}\, (\Pi\, A\, B)\, v_0\, [\varphi \hookrightarrow (\lambda_z v)\alpha] \\
& & \mathsf{hcomp}^{r \to s}\, (\Sigma\, A\, B)\, v_0\, [\varphi \hookrightarrow (\lambda_z v)\alpha] \mid \mathsf{hcomp}^{r \to s}\, (\mathsf{Path}\, A\, v\, v)\, v_0\, [\varphi \hookrightarrow (\lambda_z v)\alpha] \\
k & ::= & kv \mid k.1 \mid k.2 \mid k\, @^{v,v}\, r \mid \mathsf{extFun}\, [\varphi \hookrightarrow w]\, k \\
& & \mathsf{coe}^{r \to s}\, (\lambda_z k)\, v_0 \mid \mathsf{hcomp}^{r \to s}\, k\, v_0\, [\varphi \hookrightarrow (\lambda_z v)\alpha] \\
\varphi, \psi & ::= & (r = s) \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid 0_\Phi \mid 1_\Phi \\
r, s & ::= & r \wedge s \mid r \vee s \mid 0_\mathbb{I} \mid 1_\mathbb{I} \mid z_i
\end{array}
$$

Figure 1: Structure of values including neutrals, relative to $\langle z_1, \ldots, z_n \mid R \rangle$.

boolean lattice, which allows us to reduce the decision problems involving $\mathbb{I}$ and Φ (that are needed for type checking and evaluation) to boolean SAT problems.

Structure of neutral values The structure of our values is similar to the one by Kovács [7]. We have two levels of closures $(\lambda_x t)\rho$ where t is a term and ρ an environment, and $(\lambda_z v)\alpha$ where v is a value and α the underlying function of a map between finitely presented lattices, given by $z_1 = r_1, \ldots, z_n = r_n$.

A description of the values is given in Figure 1. Values are always treated relative to a stage $\langle z_1, \ldots, z_n \mid R \rangle$. We uniformly require that $r \neq s$ in hcomp and coe. Furthermore, for all systems $[\varphi \hookrightarrow -]$, we represent φ as $\bigvee_i \psi_i$ where $\psi_i = \bigwedge_j (r_j = s_j) \neq 1$, and require that $\varphi \neq 1$ and that all φ_i are pairwise incomparable.

Pseudocomplement The meet semilattice generated by embeddings is closed under the pseudo-complement inherited from the subobject classifier $\Omega \in \widehat{\square}$. The pseudo-complement of a sieve generated by an inclusion of posets $X \hookrightarrow Y$ is given by the inclusion of the complement of the image. As a consequence, Φ is closed under this complement as-well. This means that for any φ we have a maximal φ^* such that $\varphi \wedge \varphi^* = 0$.

Syntactically, this can be used to avoid hcomp values with empty systems, similar to [1]. We require that $\varphi^* = 0$ for any $\mathsf{hcomp}\, A\, u_0\, [\varphi \hookrightarrow (\lambda_z u)\alpha]$. Because $-^*$ is natural, no restriction will yield an empty system. Every hcomp can be completed to one satisfying this property, by replacing the system with $[\varphi \hookrightarrow (\lambda_z u)\alpha, \varphi^* \hookrightarrow (\lambda_z u_0)]$. However, this completion operation is *not* stable under the usual reduction rules for hcomp. Hence, it cannot be delayed arbitrarily.

Future work Potential future works includes the extension of the implementation with the above technique for avoiding empty system. It should also be possible to operationalize the second model as well, obtaining a closed evaluator which allows computation involving Whitehead's principle and dependent choice. This would allow for computation of results from synthetic homotopy theory that rely on these additional axioms.

References

[1] Carlo Angiuli. *Computational Semantics of Cartesian Cubical Type Theory*. PhD thesis, Carnegie Mellon University, USA, 2021.

[2] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Robert Harper, Kuen-Bang Hou (Favonia), and Daniel R. Licata. Syntax and models of cartesian cubical type theory. *Math. Struct. Comput. Sci.*, 31(4):424–468, 2021.

[3] Evan Cavallo and Christian Sattler. Relative elegance and cartesian cubes with one connection, 2023.

[4] 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.

[5] Thierry Coquand. A survey of constructive presheaf models of univalence. *ACM SIGLOG News*, 5(3):54–65, 2018.

[6] Thierry Coquand. A new model of dependent type theory. https://www.cse.chalmers.se/~coquand/semimodel.pdf, 2023. Describes model construction by Christian Sattler.

[7] András Kovács. cctt. https://github.com/AndrasKovacs/cctt/, 2023.

[8] Ian Orton and Andrew M. Pitts. Axioms for modelling cubical type theory in a topos. *Log. Methods Comput. Sci.*, 14(4), 2018.

[9] Christian Sattler. The equivalence extension property and model structures, 2017.

[10] Christian Sattler. Why does M form an external lex operation? https://www.cse.chalmers.se/~sattler/docs/external-lex-operation-intuition.pdf, 2023.

[11] Michael Shulman. The derivator of setoids. *Cah. Topol. Géom. Différ. Catég.*, 64(1):29–96, 2023.

[12] Gavin C. Wraith. Using the generic interval. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 34(4):259–266, 1993.

Towards Quantitative Inductive Families

Yulong Huang and Jeremy Yallop

University of Cambridge, UK

We present the on-going work to extend Quantitative Type Theory (QTT) [2] with inductive families [6] whose constructors are user-annotated. We give the general scheme for defining lists with quantities, which we believe can be extended to arbitrary inductive families, subsuming instances of datatypes like dependent pairs [1, 2, 4], unit [2, 4], Boolean [2], natural numbers [1], and lists [3] scattered in recent work.

Quantitative Type Theory. Quantitative type theory extends MLTT with runtime usage annotations on variables, ranging from 0 (unused), and 1 (used linearly), to ω (used unlimitedly). Our judgements are in the form of $\Gamma \vdash M \overset{\sigma}{:} A \ ; \ \underline{m}$ (inspired by [1]), which says that M is well-typed in Γ and $\underline{m} = q_1, \cdots, q_n$ is a *quantity assignment* to variables in the context.

$$\textsc{ty-Pi}\ \frac{\Gamma \vdash A \overset{0}{:} \mathrm{Type} \ ; \ \underline{0} \qquad \Gamma, x{:}A \vdash B \overset{0}{:} \mathrm{Type} \ ; \ \underline{0}}{\Gamma \vdash \Pi x \overset{q}{:} A.\, B \overset{0}{:} \mathrm{Type} \ ; \ \underline{0}} \qquad \textsc{ty-Var}\ \frac{x : A \in \Gamma}{\Gamma \vdash x \overset{\sigma}{:} A \ ; \ \sigma_x}$$

The typing rules are standard except for quantity information. The parameter σ in the judgement (above the colon) indicates the *mode* of type checking, either 0 (where variable usage is ignored) or 1 (where variable usage is counted). Terms checked in mode 0 are runtime-irrelevant and require no resource, and the rules ensure that they will not appear at runtime. Types are runtime irrelevant, so Π is judged with $\sigma = 0$, takes erased arguments, and is assigned a vector of zero quantities. Variables are assigned σ_x, denoting σ for x and 0 for other variables (note the notation abuse here to implicitly coerce modes to quantities).

$$\textsc{ty-Lam}\ \frac{\Gamma, x{:}A \vdash M \overset{\sigma}{:} B \ ; \ \underline{m}, q}{\Gamma \vdash \lambda x \overset{q}{:} A.\, M \overset{\sigma}{:} \Pi x \overset{q}{:} A.\, B \ ; \ \underline{m}} \qquad \textsc{ty-App}\ \frac{\sigma' = 0 \Leftrightarrow (\sigma = 0 \vee q = 0) \qquad \Gamma \vdash M \overset{\sigma}{:} \Pi x \overset{q}{:} A.\, B \ ; \ \underline{m} \qquad \Gamma \vdash N \overset{\sigma'}{:} A \ ; \ \underline{n}}{\Gamma \vdash M\ N \overset{\sigma}{:} B[M/x] \ ; \ \underline{m} + q\underline{n}}$$

The lambda case is straightforward. For an application $M\ N$, M uses resources $\underline{m}$ and uses its argument q times, while N uses resources $\underline{n}$, so the total resource of the application is $\underline{m} + q\underline{n}$ (operations on quantities, like addition and multiplication, extend pointwise to assignments). The side condition ensures that erased terms cannot appear at runtime: a function accepts an erased argument ($\sigma' = 0$) only if the entire application is runtime irrelevant ($\sigma = 0$), or if the function does not use its argument at all ($q = 0$).

QTT terms are subject to the usual $\beta\eta$-equality and conversion. The quantities have a partial order of $0 \leq \omega \geq 1$. QTT supports a sub-usaging rule for over-approximating the resource usage: if $\Gamma \vdash M \overset{\sigma}{:} A \ ; \ \underline{m}$ and $\underline{m} \leq \underline{m}'$ then we can also assign $\underline{m}'$ to M.

QTT with linear lists. We extend QTT with linear lists, an instance of the general scheme for lists. Here is the inductive family signature with each constructor argument marked with a quantity (both 1 here), specifying its runtime usage:

$$\texttt{data}\ \mathbf{List}^{11}\ (A{:}\mathrm{Type}) : \mathrm{Type}\ \texttt{where}\ [\,] : \mathbf{List}^{11} A \mid \_::\_ : \Pi x \overset{1}{:} A.\, \Pi xs \overset{1}{:} \mathbf{List}^{11} A.\, \mathbf{List}^{11} A$$

This extends our type theory with a type formation rule and one introduction rule for each constructor. Type former $\mathbf{List}^{11}$ is judged with $\sigma = 0$, since "types need nothing". The introduction rules are similar to rule TY-APP, where we sum the usage of all the arguments.

$$\textsc{ty-List}\ \frac{\Gamma \vdash A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}}{\Gamma \vdash \mathbf{List}^{11}\, A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}} \qquad \textsc{ty-Nil}\ \frac{\Gamma \vdash A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}}{\Gamma \vdash [\,] \overset{\sigma}{:} \mathbf{List}^{11}\, A\ ;\ \underline{0}} \qquad \textsc{ty-Cons}\ \frac{\Gamma \vdash M \overset{\sigma}{:} A\ ;\ \underline{m} \quad \Gamma \vdash N \overset{\sigma}{:} \mathbf{List}^{11}\, A\ ;\ \underline{n}}{\Gamma \vdash M :: N \overset{\sigma}{:} \mathbf{List}^{11}\, A\ ;\ \underline{m} + \underline{n}}$$

Given a predicate P, list L, and branches M and N for the empty and non-empty cases, we get a term of type $P[L/ls]$ from the eliminator. The typing and β-reduction (omitted here) are conventional. The two branches M and N use resources $\underline{m}$ and $\underline{n}$ respectively. The last premise says that N must use the head argument x and the inductive hypothesis r exactly once, while the tail argument xs is for typing only (hence unused).

$$\textsc{ty-ElimList}\ \frac{\Gamma, ls{:}\mathbf{List}^{11}\, A \vdash P \overset{0}{:} \mathrm{Type}\ ;\ \underline{0} \quad \Gamma \vdash L \overset{\sigma}{:} \mathbf{List}^{11}\, A\ ;\ \underline{l} \quad \Gamma \vdash M \overset{\sigma}{:} P[[\,]/ls]\ ;\ \underline{m} \quad \Gamma, x{:}A, xs{:}\mathbf{List}^{11}\, A, r{:}P[xs/ls] \vdash N \overset{\sigma}{:} P[x :: xs/ls]\ ;\ \underline{n}, 1, 0, 1}{\Gamma \vdash \mathit{Elim}_{list}(P, L, M, (x, xs, r).N) \overset{\sigma}{:} P[L/ls]\ ;\ \underline{l} + (\underline{m} \sqcup \omega \underline{n})}$$

The eliminator reduces to M if L is empty, using resources $\underline{m}$. Otherwise, it evaluates N many times until it hits the base case M, using resources $\underline{m} + \omega\underline{n}$ (we do not know the exact number of recursions). We take the join of the quantities in these two cases (since we do not know which branch the eliminater reduces to) and add the resource used by L to obtain the quantity assignment for the eliminator, $\underline{l} + (\underline{m} \sqcup \omega\underline{n})$.

Lists with quantities. For any two fixed quantities p and q, we can give a inductive family signature similar to that of $\mathbf{List}^{11}$, except the constructor arguments are marked with p and q instead of 1. Again, our type theory is extended with type formation and introduction rules. The type former for $\mathbf{List}^{pq}$ is judged with $\sigma = 0$ as before. Constructor applications are treated like function applications – we sum the resource assigned to each argument multiplied by its designated usage. As in TY-APP, the side conditions say that an argument is erased only if the entire expression is runtime irrelevant or it is never used.

$$\textsc{ty-List-pq}\ \frac{\Gamma \vdash A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}}{\Gamma \vdash \mathbf{List}^{pq}\, A \overset{0}{:} \mathrm{Type}\ ;\ \underline{0}} \qquad \textsc{ty-Cons-pq}\ \frac{\sigma_1 = 0 \Leftrightarrow (\sigma = 0 \vee p = 0) \quad \sigma_2 = 0 \Leftrightarrow (\sigma = 0 \vee q = 0) \quad \Gamma \vdash M \overset{\sigma_1}{:} A\ ;\ \underline{m} \quad \Gamma \vdash N \overset{\sigma_2}{:} \mathbf{List}^{pq}\, A\ ;\ \underline{n}}{\Gamma \vdash M :: N \overset{\sigma}{:} \mathbf{List}^{pq}\, A\ ;\ p\underline{m} + q\underline{n}}$$

The eliminator for $\mathbf{List}^{pq}$ has the same structure and typing rules as in rule TY-ELIMLIST with a more general premiss for N. The list's head should be used p times in N. The tail is used q_1 times directly and q_2 times in the recursion in N, so the combined usage $q_1 + q_2$ should be no more than q, the specified usage of the tail. For example, if $p = q = 1$, we have a more general eliminator of $\mathbf{List}^{11}$ that can only use either xs or r once.

$$\textsc{ty-ElimList-pq}\ \frac{\Gamma, ls{:}\mathbf{List}^{pq}\, A \vdash P \overset{0}{:} \mathrm{Type}\ ;\ \underline{0} \quad \Gamma \vdash L \overset{\sigma}{:} \mathbf{List}^{pq}\, A\ ;\ \underline{l} \quad \Gamma \vdash M \overset{\sigma}{:} P[[\,]/ls]\ ;\ \underline{m} \quad q_1 + q_2 \le q \quad \Gamma, x{:}A, xs{:}\mathbf{List}^{pq}\, A, r{:}P[xs/ls] \vdash N \overset{\sigma}{:} P[x :: xs/ls]\ ;\ \underline{n}, p, q_1, q_2}{\Gamma \vdash \mathit{Elim}_{list}(P, L, M, (x, xs, r).N) \overset{\sigma}{:} P[L/ls]\ ;\ \underline{l} + (\underline{m} \sqcup (q_2\underline{m} + (q_2 + 1)\underline{n}))}$$

Calculation of the quantity assignment is also similar to rule TY-ELIMLIST. We join the usage of the eliminator's two cases and add the resource for creating the list. The base case M has usage m. The usage of N depends on the number of times it uses the induction hypothesis r, i.e. the value of q_2: the base case is evaluated q_2 times and the inductive case $q_2 + 1$ times, so quantity assignment for rule ELIMLIST-PQ is $\underline{l} + (\underline{m} \sqcup (q_2\underline{m} + (q_2 + 1)\underline{n}))$.

Our extension is sound because erasing quantity-related information gives the usual inductive families [6], whose soundness is well known [5]. We have shown that the extension respects QTT's syntactic properties, e.g. substitution and subject reduction.

Lemma 1.1 (Substitution)**.** *The following rule for substitution is admissible:*

$$\textsc{ty-Subst}\quad \frac{\Gamma, x{:}A \vdash M \overset{\sigma}{:} B \;;\; \underline{m}, q \qquad \Gamma \vdash N \overset{\sigma'}{:} A \;;\; \underline{n} \qquad \sigma' = 0 \Leftrightarrow q = 0}{\Gamma \vdash M[N/x] \overset{\sigma}{:} B[N/x] \;;\; \underline{m} + q\underline{n}}$$

Lemma 1.2 (Reduction)**.** *If* $\Gamma \vdash M \overset{\sigma}{:} A \;;\; \underline{m}$ *and* M *reduces to* M'*, then* $\Gamma \vdash M' \overset{\sigma}{:} A \;;\; \underline{m}$ *is derivable.*

References

[1] Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. A graded modal dependent type theory with a universe and erasure, formalized. *Proc. ACM Program. Lang.*, 7(ICFP):920–954, 2023.

[2] Robert Atkey. Syntax and semantics of quantitative type theory. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 56–65. ACM, 2018.

[3] Robert Atkey. Polynomial time and dependent types. *Proc. ACM Program. Lang.*, 8(POPL):2288–2317, 2024.

[4] Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. A graded dependent type system with a usage-aware semantics. *Proc. ACM Program. Lang.*, 5(POPL):1–32, 2021.

[5] Peter Dybjer. Inductive sets and families in martin-lof's type theory and their set-theoretic semantics. In *Logical frameworks*, pages 280–306. 1991.

[6] Peter Dybjer. Inductive families. *Formal aspects of computing*, 6:440–465, 1994.

Session 12: Formalisations and Probability Theory (and Polarized Logic)

Quasi Morphisms for Almost Full Relations

Dominique Larchey-Wendling

Université de Lorraine, CNRS, LORIA, France
dominique.larchey-wendling@loria.fr

Abstract

In Coq, we mechanize two morphisms for transferring the almost full property between relations.

The study of almost full relations [10] (constructive WQOs) mainly consists in establishing closure properties of the `af` predicate. For instance, Higman's lemma [3, 1, 8] states its closure under the homeomorphic embedding of lists, and Kruskal's theorem [4, 9], closure under the homeomorphic embedding of rose trees. The later concerns a nested type and embedding. Our former Coq constructive proof of Kruskal's tree theorem [5] suffers from being quite monolithic, a property unfortunately inherited from the pen&paper proof of which it derives [9]. In the process of a major refactoring effort aimed at modularity, removal of code duplication, and readability, we have identified two important tools to transfer `af` from one relation R to another T, i.e. to establish entailments of shape $\texttt{af}\,R \to \texttt{af}\,T$.

We present these tools independently of the context of intricate developments. The first one is simple but versatile: it is sufficient to provide a surjective relational morphism from R to T. The second one, more specialized, but instrumental in the constructive proofs of Higman/Kruskal's results [1, 9], aims at transfers of shape $\texttt{af}\,R \to \texttt{af}\,T{\uparrow}y_0$. In that case, it is sufficient to provide a quasi morphism to enable the transfer (see below). When assuming decidability of relations as in [8], a quasi morphism can be turned into a surjective relational morphism, allowing for an easy proof of transfer. In the general case, the transfer is much more involved. The two bricks that compose this tool, the FAN theorem and a combinatorial principle, can be traced back to [1], and are repeatedly inlined in [9]. However, the quasi morphism result is never stated in a general setting to be established independently, hence this abstract.

We only present the main results and the ingredients to obtain them, sticking to a somewhat informal presentation, w/o giving justifications. Strict preciseness is deferred to the available Coq artifact [7] that is both standalone, compact with less than 1k loc, commented and designed for human readability.[1] See also [6] for a presentation on how these results are used e.g. to establish Higman's lemma.

Below we write $\mathbb{P}$ for `Prop`, and we use $\texttt{rel}_1\,X \coloneqq X \to \mathbb{P}$ (resp. $\texttt{rel}_2\,X \coloneqq X \to X \to \mathbb{P}$) to represent unary (resp. binary) relations, denoting $\subseteq$ for relations inclusion. For $R : \texttt{rel}_2\,X$ and $P : \texttt{rel}_1\,X$, we write $R{\Downarrow}P : \texttt{rel}_2\,\{x \mid P\,x\}$ for the restriction of R to the subtype. We adopt the usual notations for lists: $[]$ for the empty list, :: for the cons(tructor), and $\in$ for list membership. The product embedding for lists is defined inductively as $\texttt{Forall}_2\,R : \texttt{list}\,X \to \texttt{list}\,Y \to \mathbb{P}$ by the two rules of Fig. 1.

Following [10], a binary relation $R : \texttt{rel}_2\,X$ is *almost-full* (AF) if it satisfies the predicate $\texttt{af}\,R : \mathbb{P}$ defined inductively by the two rules of Fig. 1. There, we define the *lifted relation* $R{\uparrow}a$ by $(R{\uparrow}a)\;x\;y \coloneqq R\;x\;y \vee R\;a\;x$, and we extend lifting to lists by $R{\uparrow\uparrow}[a_1;\ldots;a_n] \coloneqq R{\uparrow}a_n \ldots {\uparrow}a_1$. Intuitively, R is AF if it is bound to become a full relation, whatever sequence of liftings is applied to it. An alternative formulation uses the inductive `bar` predicate and $\texttt{good}\,R$ sequences/lists as defined in Fig. 1. For any list $l : \texttt{list}\,X$, we establish the equivalence $\texttt{af}\,(R{\uparrow\uparrow}l) \leftrightarrow \texttt{bar}\,(\texttt{good}\,R)\,l$, and in particular we get $\texttt{af}\,R \leftrightarrow \texttt{bar}\,(\texttt{good}\,R)\,[]$. This result allows for an easy application of the FAN theorem (see below).

Already in [10], monotonicity is present as a tool to transfer `af` from one relation to another, i.e. $R \subseteq S \to \texttt{af}\,R \to \texttt{af}\,T$, but R and T must share the same ground type.[2] Also mentioned in [10], one can transport `af` using a map $f : X \to Y$ with $\texttt{af\_comap} : \texttt{af}\,R \to \texttt{af}\,\big(\lambda\,x_1\,x_2, R\,(f\,x_1)\,(f\,x_2)\big)$, but this tool is quite cumbersome to use as the target `af` relation has to be put first in this restrictive shape.

[1] In this abstract, the results are `Prop`-bounded but the artifact itself is generic in `Prop`-bounded vs `Type`-bounded alternatives.

[2] Coquand's constructive version of Ramsey's theorem $\texttt{af}\,R \to \texttt{af}\,T \to \texttt{af}\,(R \cap T)$ is their main focus but we won't need it.

$$\frac{}{\texttt{Forall}_2\, R\, []\, []} \qquad \frac{R\, x\, y \qquad \texttt{Forall}_2\, R\, l\, m}{\texttt{Forall}_2\, R\, (x :: l)\, (y :: m)}$$

$$\frac{\forall x\, y,\, R\, x\, y}{\texttt{af}\, R} \qquad \frac{\forall a,\, \texttt{af}\, R{\uparrow}a}{\texttt{af}\, R}$$

$$\frac{R\, y\, x \qquad y \in l}{\texttt{good}\, R\, (x :: l)} \qquad \frac{\texttt{good}\, R\, l}{\texttt{good}\, R\, (x :: l)}$$

$$\frac{P\, l}{\texttt{bar}\, P\, l} \qquad \frac{\forall x,\, \texttt{bar}\, P\, (x :: l)}{\texttt{bar}\, P\, l}$$

Figure 1: Inductive rules for $\texttt{Forall}_2$, af, good and bar, with $R : \texttt{rel}_2\, \_\, \_$ and $P : \texttt{rel}_1\, (\texttt{list}\, \_)$.

Instead, we introduce the notion of *surjective relational morphism* to transport af from $R : \texttt{rel}_2\, X$ to $T : \texttt{rel}_2\, Y$. This is a *relational* map $f : X \to Y \to \mathbb{P}$ with the two following properties:

1. $\forall y, \exists x, f\, x\, y$ (surjective); 2. $\forall x_1\, x_2\, y_1\, y_2, f\, x_1\, y_1 \to f\, x_2\, y_2 \to R\, x_1\, x_2 \to T\, y_1\, y_2$ (morphism).

Under these assumptions we establish $\texttt{af}\, R \to \texttt{af}\, T$. This formulation is more versatile: a) there is no constraint on the shape of the target T, b) it does not restrict morphisms to total functions, hence they can be *partial*, c) but also critically, they can map to *several outputs*. For instance, the entailment $\texttt{af}\, R \to \texttt{af}\, R{\Downarrow}P$ is trivial to establish using such a morphism. But w/o some strong hypotheses on P (e.g. Booleanness), there is no surjective functional map *onto* the ground type $\{x \mid P\, x\}$ of $R{\Downarrow}P$.

We use relational morphisms extensively in this development, e.g. for short proofs of the transfer $\texttt{af}\, R{\uparrow}a \to \texttt{af}\, R{\Downarrow}(\neg R\, a)$ and the converse $\texttt{af}\, R{\Downarrow}(\neg R\, a) \to \texttt{af}\, R{\uparrow}a$. But the later requires the *decidability* of $(R\, a)$ as an additional hypothesis.[3]

We switch to the central transfer tool used in the proofs of Higman's and Kruskal's results, the notion of *quasi morphism*. It allows to establish the entailment $\texttt{af}\, R \to \texttt{af}\, T{\uparrow}y_0$ for $R : \texttt{rel}_2\, X$, $T : \texttt{rel}_2\, Y$ and $y_0 : Y$. For this, one needs the following data: a map $ev : X \to Y$ from analyses to evaluations and a predicate $E : \texttt{rel}_1\, X$ characterizing *exceptional* analyses satisfying:[4]

1. $\forall y, \texttt{fin}\, (ev^{-1}\, y)$; 2. $\forall x_1\, x_2, R\, x_1\, x_2 \to T\, (ev\, x_1)\, (ev\, x_2) \vee E\, x_1$; 3. $\forall y, (ev^{-1}\, y) \subseteq E \to T\, y_0\, y$.

where we denote $ev^{-1} y \coloneqq (\lambda x, ev\, x = y)$ and call them *analyses* of (the evaluation) y. They are assumed finitely many by Item 1; Item 2 states that ev is a morphism unless applied to exceptional analyses; and Item 3 states that y embeds y_0 when all its analyses are exceptional. One can "quickly" justify quasi morphisms by further assuming the decidability of both $T\, y_0$ and E. Indeed, in that case ev becomes a surjective relational morphism from $R{\Downarrow}(\neg E)$ to $T{\Downarrow}(\neg T\, y_0)$. Yet the statement of the quasi-morphism result carefully avoids negation, and we establish it w/o those decidability assumptions. Nonetheless in that general case, the proof uses two non-trivial tools (also mechanized in the artifact), related to the choice sequences for $ll : \texttt{list}\, (\texttt{list}\, X)$, i.e. the inhabitants of $\texttt{FAN}\, ll \coloneqq \lambda\, c, \texttt{Forall}_2\, (\cdot \in \cdot)\, c\, ll$:[5]

- the FAN theorem for inductive bars: for $P : \texttt{rel}_1\, (\texttt{list}\, X)$ *monotone*, i.e. $\forall x\, l, P\, l \to P(x :: l)$, we have $\texttt{bar}\, P\, [] \to \texttt{bar}\, (\lambda\, ll, \texttt{FAN}\, ll \subseteq P)\, []$;[6]
- a finite combinatorial principle: for $P : \texttt{rel}_1\, (\texttt{list}\, X)$, $B : \texttt{rel}_1\, X$, and $ll : \texttt{list}\, (\texttt{list}\, X)$, assuming $\forall c, \texttt{FAN}\, ll\, c \to P\, c \vee \exists x, x \in c \wedge B\, x$ (any choice sequence satisfies P or meets B), we have either $\exists c, \texttt{FAN}\, ll\, c \wedge P\, c$ (P contains a choice sequence), or $\exists l, l \in ll \wedge \forall x, x \in l \to B\, x$ (there is a list in ll which is included in B).[7]

[3] Using negations like in $\neg R\, a$ (as done in e.g. [8]) allows for equivalences between $\texttt{af}\, R$ and (inductive) well-foundedness of list expansion restricted to bad sequences, but be aware that this approach usually restricts the study to decidable relations.

[4] The analysis/evaluation terminology follows [9, page 241], and an exceptional analysis "contains a disappointing sub-tree."

[5] Intuitively, $\texttt{FAN}\, [l_1; \ldots; l_n]$ spans the (finitely many) lists $[c_1; \ldots; c_n]$ such that $c_1 \in l_1, \ldots, c_n \in l_n$.

[6] Compared to [1, 2], this FAN theorem has a shorter proof because it avoids the explicit construction of the FAN as a list.

[7] Classically (with excluded middle and choice), the combinatorial principle is trivial and not limited to finite fans.

References

[1] Daniel Fridlender. Higman's lemma in type theory. In Eduardo Giménez and Christine Paulin-Mohring, editors, *Types for Proofs and Programs*, pages 112–133, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[2] Daniel Fridlender. An Interpretation of the Fan Theorem in Type Theory. In Thorsten Altenkirch, Bernhard Reus, and Wolfgang Naraschewski, editors, *Types for Proofs and Programs*, pages 93–105, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[3] Graham Higman. Ordering by Divisibility in Abstract Algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, 1952.

[4] Joseph B. Kruskal. Well-Quasi-Ordering, The Tree Theorem, and Vazsonyi's Conjecture. *Transactions of the American Mathematical Society*, 95(2):210–225, 1960.

[5] Dominique Larchey-Wendling. A mechanized inductive proof of Kruskal's tree theorem. https://members.loria.fr/DLarchey/files/Kruskal, 2015.

[6] Dominique Larchey-Wendling. Higman's lemma in the Almost Full library. https://github.com/DmxLarchey/Kruskal-Higman, 2024.

[7] Dominique Larchey-Wendling. Quasi Morphisms for Almost Full relations (artifact). https://github.com/DmxLarchey/Quasi-Morphisms, 2024.

[8] Helmut Schwichtenberg, Monika Seisenberger, and Franziskus Wiesnet. *Higman's Lemma and Its Computational Content*, pages 353–375. Springer International Publishing, Cham, 2016.

[9] Wim Veldman. An intuitionistic proof of Kruskal's theorem. *Archive for Mathematical Logic*, 43(2):215–264, Feb 2004.

[10] Dimitrios Vytiniotis, Thierry Coquand, and David Wahlstedt. Stop When You Are Almost-Full. In Lennart Beringer and Amy Felty, editors, *Interactive Theorem Proving*, pages 250–265, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Looking Back: A Probabilistic Inverse Perspective on Test Generation

Joachim T. Kristensen[1], Tobias Reinhard[2,3*†], and Michael K. Thomsen[1,4]

[1] Department of Informatics, University of Oslo, Oslo, Norway
{joachkr,michakt}@ifi.uio.no
[2] KU Leuven, Leuven, Belgium
[3] Technical University of Darmstadt, Darmstadt, Germany
[4] Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

1 Introduction

Software validation is hard, among others things because of the sheer size of the input space [7, 18]. Reversible computation has shown promises to mitigate some difficulties in software debugging [3, 6], but has not been applied to the wider area of software validation. To alleviate this, we propose to relax reversible computing to a combination of probabilistic [4, 14] and inverse computation [2, 9, 12]. This will create a new model that is a great candidate for mitigating the difficulties of software validation.

Imagine being able to find the cause of any error by simply calling a program's inverse on it. A significant limitation to this naïve approach is that the program needs to be injective; unfortunately, most programs do not have this property. To overcome this limitation, we characterise the inverse of a program in terms of a probabilistic program as follows:

Let $f : A \to B$ be a function, and consider two inputs $a_0 \neq a_1 \in A$ such that $f(a_0) = f(a_1)$. There exists no function $g : B \to A$ such that $g \circ f = id_A$. Instead, we consider the function $\texttt{INVERT}(f) : B \to \delta(A)$ [1] which maps each output $f(a)$ to a probability distribution over the corresponding possible inputs $\delta : A \to \texttt{Prob}$. We call INVERT(f) the *probabilistic invers* of f. Consider any output $b \in B$. The probability distribution $\delta_b = \texttt{INVERT}(f)(b)$ maps each possible input $a \in A$ to the likelihood $\delta_b(a) \in \texttt{Prob}$ of "b originating from a". Note that this function is guaranteed to exist, which cannot be said for classic inverse functions. Moreover, the latter are generally partial functions, which complicates reasoning about them. Meanwhile our probabilistic inverses are guaranteed to be total.

In conventional reversible programming languages, programs are guaranteed to be (globally) invertible by enforcing a strict syntactic discipline [8,17,19]. Programs may only be comprised of locally isomorphic parts and combinations are restricted to preserve their isomorphic properties. These restrictions do not apply to the programs we consider. Moreover, we conjecture that our probabilistic inverses can be used to reason about the quality of test generators such as `QuickCheck` [1, 10], or perhaps even to derive such generators.

2 Test Case Generation

Generating input for `QuickCheck` properties is hard for at least two reasons: (i) The search space is huge. (ii) The counterexamples usually refer to edge cases that correspond to low-

*This research is partially funded by the Research Fund KU Leuven, and by the Cybersecurity Research Program Flanders.
†This research is partially funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297.
[1] We abbreviate the space of probability distribution over A by $\delta(A) := (A \to \texttt{Prob})$.

probability events. Knowing the probability distribution allows to (i) trim the search space and (ii) identify edge cases by favouring low-probability input. To illustrate these challenges, let us consider the following expression evaluation function (cf. Appendix A for full definition):

```
eval env (Val v      ) = return v
eval env (Var x      ) = env x
eval env (Let x e0 e1) = eval (extend x (eval env e0) env) e1
```

Suppose further, that our goal is to generate expressions for testing type preservation. That is, there is another function `typeOf`, and we want to check the property:

```
prop_type_preservation e = (typeOf e == typeOf (Val (eval emptyEnv e)))
```

Randomly generating test expressions is unlikely to yield useful results. Most expressions are not closed and many do not contain any bindings at all. Considering any such expressions will not help us in testing our type preservation property. In contrast, the expressions we are interested in only make up a small fraction of the entire input space: Closed expressions with a fair number of bindings. We capture this limitation with the following predicate:

```
interesting e = closed e && numberOfVars e `elem` normalDistribution (6, 3)
```

Interesting expressions can have an arbitrary many variables. We specify the number of bindings as probability distribution: A normal distribution of 6 variables and standard derivation of 3.

3 Probability Type System

This work aims to characterise probabilistic inverses $\texttt{INVERT}(f) : B \to \delta(A)$ of non-injective programs $f : A \to B$. The main challenge in addressing this goal is to extract the inverse probability distribution $\delta \in \delta(A)$. We propose to solve this via a type system that augments types τ by distributions δ. Consequently, our typing relation has the form $e : (\tau, \delta)$. There are four main cases to consider: (i) values of base types (i.e. non-function types), (ii) built-in injective operations, (iii) non-injective operations and (iv) control flow.

Base Values and Injective Operations: Expressions of any base type do not take any input. To avoid special treatment, we extract the trivial distribution $\delta : () \mapsto 1$. Treating built-in injective operations $op : A \to B$ is similarly straightforward. For each $b \in B$, we can extract the distribution $\delta_b : A \to \texttt{Prob}$ by setting $\delta_b(a) = 1$ if $op(a) = b$ and $\delta_b(\_) = 0$ otherwise.

Non-Injective Operations: Extracting distributions for expressions involving non-injective operations like $+$ is more challenging. We propose to handle such expressions by following the structure of the AST and combining the subexpressions' distributions. The biggest challenge is finding a sound distribution composition $\delta_0 \oplus \delta_1$, satisfying the typing judgement

$$+(\tau, \delta) : \frac{\Gamma \vdash e_0 : (\tau, \delta_0) \quad \Gamma \vdash e_1 : (\tau, \delta_1)}{\Gamma \vdash e_0 + e_1 : (\tau, \delta_0 \oplus \delta_1)} \ .$$

Control Flow and Probability Bounds: We can treat branching control flow similarly to non-injective operations. The main challenge is to extract distributions for each branch and to combine them. A branch's probability is proportional to the fraction of inputs for which an execution follows said branch. To avoid computability issues, we resort to over-approximations, i.e., extracting upper bounds on the probability of each branch. This requires us to relax our notion of probability distributions and accept that the sum of all branch bounds exceeds 1. Yet, it is important to note that this will still allow us to reason about the quality of test generators.

Related Work: Previous work has defined a semantics for probabilistic programming with higher-order functions [15]. In [5], this work was extended to allow a structured way to formulate statistics with the possibility to work outside the standard measure-theoretic formalization

of probability theory. This work has so far concluded in the paradigm of exact conditioning for observations in probabilistic programs [16]. A method for automatically deriving Monte Carlo samplers from probabilistic programs [13] has applied automatic differentiation and transformation inversion, while [11] have relaxed the usual PPL design constraint to achieve a richer language model.

References

[1] Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of Haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, ICFP '00, page 268–279. ACM, 2000.

[2] Edsger W. Dijkstra. *Program Inversion*. Springer, 1979.

[3] Jakob Engblom. A review of reverse debugging. In *Proceedings of the 2012 System, Software, SoC and Silicon Debug Conference*, pages 1–6. IEEE, 2012.

[4] Mor Harchol-Balter. *Introduction to probability for computing*. Cambridge University Press, 2023.

[5] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017.

[6] James Hoey, Ivan Lanese, Naoki Nishida, Irek Ulidowski, and Germán Vidal. A case study for reversible computing: Reversible debugging of concurrent programs. *Reversible Computation: Extending Horizons of Computing: Selected Results of the COST Action IC1405 12*, pages 108–127, 2020.

[7] John Hughes. Experiences with QuickCheck: Testing the hard stuff and staying sane. In *A List of Successes That Can Change the World*, pages 169–186. Springer International Publishing, 2016.

[8] Petur Andrias Højgaard Jacobsen, Robin Kaarsgaard, and Michael Kirkedal Thomsen. CoreFun: A typed functional reversible core language. In Jarkko Kari and Irek Ulidowski, editors, *Reversible Computation*, pages 304–321. Springer International Publishing, 2018.

[9] Joachim Tilsted Kristensen, Robin Kaarsgaard, and Michael Kirkedal Thomsen. Jeopardy: An invertible functional programming language, 2022.

[10] Leonidas Lampropoulos, Diane Gallois-Wong, Cătălin Hriţcu, John Hughes, Benjamin C. Pierce, and Li-yao Xia. Beginner's luck: A language for property-based generators. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, page 114–129. ACM, 2017.

[11] Alexander K. Lew, Matin Ghavamizadeh, Martin C. Rinard, and Vikash K. Mansinghka. Probabilistic programming with stochastic probabilities. *Proc. ACM Program. Lang.*, 7(PLDI), 2023.

[12] Kazutaka Matsuda and Meng Wang. Sparcl: a language for partially-invertible computation. *Proc. ACM Program. Lang.*, 4(ICFP), 2020.

[13] David A. Roberts, Marcus Gallagher, and Thomas Taimre. Reversible jump probabilistic programming. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 634–643. PMLR, 16–18 Apr 2019.

[14] Sheldon M Ross. *Introduction to probability models*. Academic press, 2014.

[15] Sam Staton, Hongseok Yang, Frank Wood, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, page 525–534, New York, NY, USA, 2016. Association for Computing Machinery.

[16] Dario Stein and Sam Staton. Probabilistic programming with exact conditions. *J. ACM*, 71(1), feb 2024.

[17] Michael Kirkedal Thomsen and Holger Bock Axelsen. Interpretation and programming of the reversible functional language. In *Symposium on the Implementation and Application of Functional Programming Languages*, IFL '15, pages 8:1–8:13. ACM, 2016.

[18] J.A. Whittaker. What is software testing? And why is it so hard? *IEEE Software*, 17(1):70–79, 2000.

[19] Tetsuo Yokoyama and Robert Glück. A reversible programming language and its invertible self-interpreter. In *Partial Evaluation and Program Manipulation. PEPM '07*, pages 144–153. ACM, 2007.

A Full Type Preservation Example

```
data Expr
  = Val Value
  | Var Name
  | Let Name Expr Expr

data Value
  = VInt  Integer
  | VBool Bool

type Name  = String
type Error = String
type Env   = Name -> Either Error Value

emptyEnv :: Env
emptyEnv x = Left $ "unbound variable " ++ x

extend :: Eq a => a -> b -> ((a -> b) -> (a -> b))
extend x v env y = if x == y then v else env y

eval :: Env -> Expr -> Either Error Value
eval _   (Val v      ) = return v
eval env (Var x      ) = env x
eval env (Let x e0 e1) = eval (extend x (eval env e0) env) e1

closed :: Expr -> Bool
closed e = (freeVars e == [])

freeVars :: Expr -> [Name]
freeVars (Val _      ) = [ ]
freeVars (Var x      ) = [x]
freeVars (Let x e0 e1) = freeVars e0 ++ filter (/= x) (freeVars e1)

numberOfVars :: Expr -> Int
numberOfVars (Val _      ) = 0
numberOfVars (Var _      ) = 1
numberOfVars (Let _ e0 e1) = numberOfVars e0 + numberOfVars e1
```

```haskell
data Type
  = TInt
  | TBool

typeOf :: Expr -> Type
typeOf = typeOf' (const undefined)
  where
    typeOf' _ (Val (VInt  _)) = TInt
    typeOf' _ (Val (VBool _)) = TBool
    typeOf' r (Var        x ) = r x
    typeOf' r (Let x e0 e1)   = typeOf' (extend x (typeOf' r e0) r) e1

prop_type_preservation :: Expr -> Bool
prop_type_preservation e = (typeOf e == typeOf (Val (eval emptyEnv e)))

-- The goal of this research
instance Arbitrary Expr where
  arbitrary = probabilisticInverse interesting True

interesting :: Expr -> Bool
interesting e = closed e && numberOfVars e `elem` normalDistribution (6, 3)
```

Polarized Lambda-Calculus at Runtime, Dependent Types at Compile Time

András Kovács

University of Gothenburg Gothenburg, Sweden
andrask@chalmers.se

We describe a particular two-level type theory [3]:

- The object level is a polarized simply-typed calculus, with computation types (functions, computational products) and value types (inductive types, closures).
- The meta level is a standard dependent type theory.

This supports two-stage compilation, where we can use dependent types in the surface language, but only object-level constructions remain after staging. It appears to be an excellent setting for performance-focused staged programming.

The polarization lets us control closures and function arities in a fairly lightweight way. In particular, if we do not use the explicit closure type former, then all function calls can be compiled to calls and jumps to statically known code locations. Hence, we migth ask: how much can we reproduce from the abstraction tools of functional programming, without using any runtime closures? Perhaps surprisingly, closures are rarely used in an essential way.

- A *map* function for lists is meant to be inlined, and after inlining no runtime closures remain.
- Monadic binding in Haskell is a higher-order function. Without compiler optimizations, we get a big overhead from a deluge of runtime closures; but with optimizations we expect that no closures remain.

In the current work, we start to build up libraries in the mentioned two-level theory, aiming to minimize the usage of runtime closures and eliminate abstraction overheads. We want to shift work from general-purpose optimizing compilers to deterministic & extensible metaprogramming. For example, instead of expecting GHC to sufficiently inline our monadic code, we implement efficient monadic code generation ourselves.

Staged Monad Transformers

Despite recent competition from a variety of algebraic effect systems, monads and monad transformers remain the most widely used strongly-typed effect system, so it make sense for us to develop their staged flavor.

In our approach, we *don't use monads at all in the object language.* Instead, we use meta-level monads, and only convert down to the object level when runtime control dependencies force us to do so. Generally, the staged version of a monad is obtained by extending it with *code generation as an effect.* The code generation monad is the following:

$$\textbf{newtype}\ \mathsf{Gen} : \mathsf{MetaTy} \to \mathsf{MetaTy}$$
$$\textbf{newtype}\ \mathsf{Gen}\ A = \mathsf{Gen}\ (\{R : \mathsf{ObjTy}\} \to (A \to \Uparrow R) \to \Uparrow R)$$

Here, ObjTy is the universe of object-level types, MetaTy is a meta-level universe, and $\Uparrow R$ is (intuitively) the type of metaprograms which generate object expressions of type R.

This is a continuation monad where *eventually* we have to return an object-level value, so we can only "run" actions of type $\mathsf{Gen}\,(\Uparrow A)$ to extract a result with type $\Uparrow A$. However, while working in the monad, we are free to use both object-level and meta-level constructions, and introduce object-level let-binders. In general, for some monad transformer stack M, we obtain its staged version by putting a Gen at the bottom of the stack. For example, $\mathsf{State}\,(\Uparrow \mathsf{Int})$ becomes $\mathsf{StateT}\,(\Uparrow \mathsf{Int})\,\mathsf{Gen}$. In this monad in particular, we can modify an object expression in the state, and also generate object code via Gen.

Using our transformer library, we program in meta-monads, and insert conversions to and from object code at the points of dynamic dependencies (e.g. object-level function calls). Such conversions are defined in a compositional manner, by recursion on transformer stacks. Overall, this style of programming requires modest extra noise compared to Haskell, but it guarantees high-quality code output by staging.

Staged Stream Fusion

Stream fusion [1] is another application that we developed in this setting. The idea here is to give a convenient list-like interface for programming with meta-level state machines. Such machines can be turned into efficient object-level code as blocks of mutually recursive functions, without storing intermediate results in runtime lists. We demonstrate very concise solutions to two well-known problems.

First, we need to generate mutually recursive blocks. One part of the challenge is to have guaranteed well-scoping and well-typing, although the number (and types) of definitions in a mutual block is only known at staging time; see e.g. [4]. We also want to avoid any runtime overheads. We represent a mutual block as a single recursive definition with a computational product type. The polarization guarantees that this can be compiled without downstream overheads to an actual mutual block.

Second, fusion for arbitrary combinations of zipping and *concatMap* has been a long-standing challenge. Currently, the *strymonas* library [2] supports such fusion but its solution is much more complex than ours, and it also relies on mutable references in the object language. In contrast, we compile to pure object code, which also enables us to parameterize streams over arbitrary monads and embed monadic effects in stream definitions.

Our streams are tuples containing a type for an internal state, an initial state and a representation of state transitions. Moreover, the internal state is required to be a finite sum-of-products of object-level value types. Now, if such sums-of-products are closed under Σ-types, concatMap is easily definable by using a Σ-type of the family of internal states that we get from an $A \to \mathsf{Stream}\,B$ function. We construct such Σ-types from an internal *generativity axiom*, which expresses that certain metaprograms can't depend on object-level terms. This axiom does hold in the staging semantics and we can erase it during staging. From this, we construct *transient* Σ-types which end up as non-dependent product types after staging.

The ability to analyze object code has been often viewed as a desirable feature in metaprogramming. In contrast, we demonstrate a use-case for the explicit *lack* of intensional analysis, through our generativity axiom. This might be compared to *internal parametricity* statements in type theories.

References

[1] Duncan Coutts. *Stream fusion : practical shortcut fusion for coinductive sequence types*. PhD thesis, University of Oxford, UK, 2011.

[2] Tomoaki Kobayashi and Oleg Kiselyov. Complete stream fusion for software-defined radio. In Gabriele Keller and Meng Wang, editors, *Proceedings of the 2024 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation, PEPM 2024, London, UK, 16 January 2024*, pages 57–69. ACM, 2024.

[3] András Kovács. Staged compilation with two-level type theory. *Proc. ACM Program. Lang.*, 6(ICFP):540–569, 2022.

[4] Jeremy Yallop and Oleg Kiselyov. Generating mutually recursive definitions. In Manuel V. Hermenegildo and Atsushi Igarashi, editors, *Proceedings of the 2019 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM@POPL 2019, Cascais, Portugal, January 14-15, 2019*, pages 75–81. ACM, 2019.

Session 13: Algebraic Geometry and Topology

Grothendieck's Functor of Points Approach to Schemes in Type Theory – Constructivity and Size Issues

Max Zeuner[1] and Matthias Hutzler[2]

[1] Stockholm University, Stockholm, Sweden
zeuner@math.su.se

[2] Gothenburg University, Gothenburg, Sweden
matthias-hutzler@posteo.net

The field of algebraic geometry owes many of its biggest 20th century advances to Grothendieck's invention of scheme theory. Formalizing this involved and abstract notion has become somewhat of a benchmark challenge for different proof assistant communities and their libraries of formalized mathematics [Chi01, BHL+21, BPL22, Che22, ZM23].

In this talk we want to present a recent formalization of quasi-compact, quasi-separated schemes (aka qcqs-schemes, an important subclass of schemes) in `Cubical Agda` [ZH24]. Unlike previous formalization in `Lean` [BHL+21] or `Isabelle/HOL` [BPL22], we do not define schemes as locally ringed spaces, but rather as functors from the category of commutative rings to the category of sets satisfying certain properties. This approach was actually the preferred one of Grothendieck and is often taken to be more amenable for constructive mathematics.[1]

Regardless of whether one works classically or constructively, the category of functors from rings to sets is not locally small, since arrows between two such functors are natural transformations, i.e. families of functions indexed by the "big" type of all rings in a given universe. As a result, one has to address size issues and we will discuss different options to do so in constructive type theory.[2]

Schemes as Functors Classic algebraic geometry studies the solution-set of a system of polynomials $p_1(x_1, \dots, x_n) = \cdots = p_m(x_1, \dots, x_n) = 0$, where the x_i would typically take values in some algebraically closed field k, by means of the algebraic properties of the corresponding quotient ring ${}^{k[x_1,\dots,x_n]}/_{\langle p_1,\dots,p_m\rangle}$. By generalizing from quotients of polynomial rings to arbitrary commutative rings, one arrives at the notion of *affine* schemes. By then introducing a way of *gluing* together affine schemes, one obtains a general notion of schemes.

From a categorical point view, schemes are thus well-behaved colimits of affine schemes. The category of affine schemes in turn is equivalent to the opposite category of commutative rings. Fixing a universe level ℓ, we can consider the Yoneda embedding, which we will denote as $\mathsf{Sp} : \mathsf{CommRing}_\ell^{op} \hookrightarrow \mathsf{Psh}(\mathsf{CommRing}_\ell^{op})$. The presheaf category $\mathsf{Psh}(\mathsf{CommRing}_\ell^{op})$ is the free cocompletion of affine schemes and thus contains schemes as a full subcategory.

We will call elements of this presheaf category, which is of course just the functor category $\mathsf{CommRing}_\ell \to \mathsf{Set}_\ell$, $\mathbb{Z}$-functors. The goal is then to formalize the defining property of schemes, or in our case qcqs-schemes. In `Cubical Agda` this means defining a term

$$\mathsf{isQcQsScheme} : \mathbb{Z}\mathsf{Functor}_\ell \to \mathsf{hProp}_{\ell+1}$$

where $\mathsf{hProp}_{\ell+1}$ is the type of "big" h-propositions living in the successor universe.[3]

[1] See e.g. the discussion https://github.com/agda/cubical/issues/657

[2] One can also avoid size issues by restricting to schemes of finite presentation. This is the approach taken in synthetic algebraic geometry [Ble21, CCH23].

[3] The increase in the universe level seems unavoidable since the definition of scheme requires quantification over the type of (small) rings as we will see below.

The scheme-property has two components, a locality condition and the existence of an affine cover. Schemes are local in the sense that they are sheaves with respect to the Zariski topology on $\mathsf{CommRing}_\ell^{op}$. All the relevant algebraic definitions and lemmas for this part of the formalization were essentially already formalized in [ZM23].

Constructive Formalization It remains to define the notion of affine open covering. Working in the predicative type theory of `Cubical Agda` without additional resizing assumptions [Voe11], we get the slightly restricted notion of a finite cover by *compact opens*. The key tool is the classifier $\mathcal{L} : \mathbb{Z}\mathsf{Functor}_\ell$ that maps a ring A to its *Zariski lattice* $\mathcal{L}_A$, the lattice of finitely generated ideals of A modulo equality of radical ideals.[4] The equivalence class of the f.g. ideal $I = \langle f_1, ..., f_n \rangle \subseteq A$ in $\mathcal{L}_A$ is denoted by $D(f_1, ..., f_n)$ and we get the induced affine compact open of $\mathsf{Sp}(A)$, denoted $\mathsf{Sp}(A)_I$, whose B-valued points are given by

$$\mathsf{Sp}(A)_I(B) \;:=\; \Sigma_{\varphi:\mathsf{Hom}(A,B)}\; 1 \in \langle \varphi(f_1), ..., \varphi(f_n) \rangle \;\simeq\; \Sigma_{\varphi:\mathsf{Hom}(A,B)}\; D\big(\varphi(f_1), ..., \varphi(f_n)\big) \equiv D(1)$$

where $D(1)$ is the equivalence class of the "1-ideal". For a general $\mathbb{Z}$-functor X we call a natural transformation $U : X \Rightarrow \mathcal{L}$ a compact open of X. We get an induced $\mathbb{Z}$-functor $[\![U]\!]^{\mathsf{co}}$ whose A-valued points are given by $[\![U]\!]^{\mathsf{co}}(A) \;:=\; \Sigma_{x:X(A)}\; U(x) \equiv D(1)$. One can check that this is a sensible definition by observing that for a point $x : X(A)$ with $U(x) \equiv D(f_1, ..., f_n)$ one gets pullback squares

$$\begin{array}{ccccc}
\mathsf{Sp}(A)_{\langle f_1,\ldots,f_n\rangle} & \longrightarrow & [\![U]\!]^{\mathsf{co}} & \longrightarrow & \mathbf{1} \\
\downarrow & \lrcorner & \downarrow & \lrcorner & \downarrow{\scriptstyle D(1)} \\
\mathsf{Sp}(A) & \xrightarrow{\phi_x} & X & \xrightarrow{U} & \mathcal{L}
\end{array}$$

where ϕ_x corresponds to $x : X(A)$ by the Yoneda lemma. Not only do we get a direct definition of compact opens by classifying them by the internal lattice $\mathcal{L}$, we also automatically get the definition of compact open cover, as the compact opens carry a canonical distributive lattice structure. We can define a functorial qcqs-scheme to be a Zariski sheaf X such that there merely exist $U_1, ..., U_n : X \Rightarrow \mathcal{L}$, which are affine in the sense that there are rings A_i with $[\![U_i]\!]^{\mathsf{co}} \cong \mathsf{Sp}(A_i)$ naturally, and which cover X in the sense that for a ring A and $x : X(A)$ we always get $\bigvee_{i=1}^n U_i(x) \equiv D(1)$ in $\mathcal{L}_A$. Building on this definition, [ZH24] contains a fully formalized, constructive proof that compact opens of affine schemes are qcqs-schemes.

Size Issues Starting with rings living in the universe at level ℓ, morphisms of $\mathbb{Z}$-functors, i.e. natural transformations, will inevitably live in the successor universe. As a consequence, the lattice of compact opens associated to a $\mathbb{Z}$-functor is big. This means that we only get a functor $\mathsf{CompOpen} : \mathbb{Z}\mathsf{Functor}_\ell \to \mathsf{DistLattice}_{\ell+1}^{op}$. Coquand, Lombardi and Schuster give a constructive definition of qcqs-schemes as *ringed* lattices [CLS09]. For proving the two notions equivalent, one needs to get rid of this increase in the universe level at least for qcqs-schemes.

It needs to be mentioned that the standard reference by Demazure & Gabriel [DG80] actually considers $\mathbb{Z}$-functors $\mathsf{CommRing}_\ell \to \mathsf{Set}_{\ell+1}$.[5] This choice seems to somewhat mitigate size-issues when proving functorial schemes equivalent to schemes as locally ringed spaces. If time permits, we will discuss how univalence can help to *construct* a functor $\mathsf{QcQsSch}_\ell \to \mathsf{DistLattice}_\ell^{op}$ from the fact that for a qcqs-scheme X, there merely exists a small distributive lattice in $\mathsf{DistLattice}_\ell$ isomorphic to the big lattice of compact opens of X.

[4] How to formalize this lattice predicatively is described in [ZM23]. The restriction to compact opens comes from the fact that the type of ideals of a ring lives in the successor universe, so we have to restrict to f.g. ideals.

[5] They actually assume two Grothendieck universes $\mathcal{U} \subseteq \mathcal{V}$. As type theoretic universes in presheaf models are usually "lifted" from Grothendieck universes [HS97], our translation only seems natural.

References

[BHL+21] Kevin Buzzard, Chris Hughes, Kenny Lau, Amelia Livingston, Ramon Fernández Mir, and Scott Morrison. Schemes in Lean. *Experimental Mathematics*, 0(0):1–9, 2021.

[Ble21] Ingo Blechschmidt. Using the Internal Language of Toposes in Algebraic Geometry, 2021.

[BPL22] Anthony Bordg, Lawrence Paulson, and Wenda Li. Simple Type Theory is not too Simple: Grothendieck's Schemes Without Dependent Types. *Experimental Mathematics*, 0(0):1–19, 2022.

[CCH23] Felix Cherubini, Thierry Coquand, and Matthias Hutzler. A Foundation for Synthetic Algebraic Geometry, 2023.

[Che22] Tim Cherganov. Sheaf of rings on Spec R. https://github.com/UniMath/UniMath/blob/0df0949b951e198c461e16866107a239c8bc0a1e/UniMath/AlgebraicGeometry/Spec.v, 2022.

[Chi01] Laurent Chicli. Une formalisation des faisceaux et des schémas affines en théorie des types avec Coq. Technical Report RR-4216, INRIA, June 2001.

[CLS09] Thierry Coquand, Henri Lombardi, and Peter Schuster. Spectral Schemes as Ringed Lattices. *Annals of Mathematics and Artificial Intelligence*, 56(3):339–360, 2009.

[DG80] Michel Demazure and Peter Gabriel. *Introduction to algebraic geometry and algebraic groups*. Elsevier, 1980.

[HS97] Martin Hofmann and Thomas Streicher. Lifting Grothendieck Universes. https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf, 1997. Unpublished note.

[Voe11] Vladimir Voevodsky. Resizing rules – their use and semantic justification. Slides from a talk at TYPES, Bergen, 11 September, 2011.

[ZH24] Max Zeuner and Matthias Hutzler. The Functor of Points Approach to Schemes in Cubical Agda. https://arxiv.org/abs/2403.13088, 2024.

[ZM23] Max Zeuner and Anders Mörtberg. A Univalent Formalization of Constructive Affine Schemes. In Delia Kesner and Pierre-Marie Pédrot, editors, *28th International Conference on Types for Proofs and Programs (TYPES 2022)*, volume 269 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:24, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

A Constructive Cellular Approximation Theorem in HoTT

Axel Ljungström and Loïc Pujet

Stockholm University, Stockholm, Sweden

At the heart of homotopy type theory (HoTT) is the analogy between types and spaces. This permits the use of type theory as a language for algebraic topology, *i.e.* for the study of spaces and maps between spaces up to homotopy by means of algebraic invariants, such as homotopy groups [13, 1, 4] and (co)homology groups [9, 5, 3, 14, 7, 2, 6, 8, 10]. Although the methods of algebraic topology apply to very general notions of spaces, the theory is often easier to develop in the context of a more restricted and well-behaved class: CW complexes. As such, it is natural to define CW complexes in the language of HoTT, in order to obtain a notion of spaces which is easier to work with than arbitrary types.

In this work, we revisit the definition of CW complexes given by Buchholtz and Favonia [3] and develop their theory. In particular, we focus on the *cellular approximation theorem*, a cornerstone result in algebraic topology which says that arbitrary maps between CW complexes and their homotopies may be approximated by maps and homotopies which respect the cellular structure [12, chap. 10]. We give a constructive proof of the theorem which relies heavily on the (provable) principle of finite choice[1], and we discuss the extent to which the theorem can be strengthened while remaining constructive. The work we present here is intended to serve as a foundation for a larger project on the development of cellular homology with Anders Mörtberg.

In order to define CW complexes, we will need the following definition:

Definition 1 (CW skeleta)**.** *An **ordered CW skeleton** is an infinite sequence of types*

$$\varnothing = C_{-1} \xrightarrow{\iota_{-1}} C_0 \xrightarrow{\iota_0} C_1 \xrightarrow{\iota_1} \dots$$

equipped with maps $\alpha : \mathbb{S}^n \times A_n \to C_n$ where A_n is equivalent to $\mathsf{Fin}(k_n)$ for some $k_n : \mathbb{N}$ and the following square is a pushout:

$$\begin{array}{ccc} \mathbb{S}^n \times A_n & \longrightarrow & A_n \\ {\scriptstyle \alpha_n}\downarrow & \ulcorner & \downarrow \\ C_n & \xrightarrow[\iota_n]{} & C_{n+1} \end{array}$$

*An **unordered CW skeleton** is defined similarly, but each A_n is only assumed to be merely finite, i.e. for all n we have a proof of $\|A_n \simeq \mathsf{Fin}(k_n)\|_{-1}$.*

The pushout condition ensures that the $(n+1)$-skeleton C_{n+1} is obtained by attaching a finite number of n-dimensional cells to the n-skeleton C_n. In the case of an ordered CW skeleton, each type of cells is equipped with an order inherited from $\mathsf{Fin}(k_n)$, hence the name. Given a CW skeleton $C_\bullet$, we write C_∞ for the colimit of the sequence of n-skeleta, and for any n we write $\iota_\infty : C_n \to C_\infty$ for the inclusion of C_n into the colimit C_∞.

Definition 2 (CW complexes)**.** *A type X is said to be an ordered (resp. unordered) CW complex if there merely exists an ordered (resp. unordered) CW skeleton $C_\bullet$ such that X is equivalent to the colimit C_∞.*

[1] The proof has been fully formalised in `Cubical Agda`, and is available at https://github.com/loic-p/cellular/blob/main/summary.agda

A map between two CW complexes X and Y is simply a map between the underlying types. The cellular approximation theorem states that such maps may be approximated by *cellular maps*, *i.e.* sequences of maps between the n-skeleta of X and Y. In order to prove this theorem constructively for unordered CW complexes, we need to define finite cellular maps:

Definition 3 (Cellular m-maps)**.** *Given two CW skeleta $C_\bullet$ and $D_\bullet$, a cellular m-map from $C_\bullet$ to $D_\bullet$ is a finite sequence of maps $(f_n : C_n \to D_n)_{n\leq m}$ equipped with a family of homotopies $h_n(x) : (\iota_n \circ f_n)(x) = (f_{n+1} \circ \iota_n)(x)$ for $n < m$.*

Definition 4 (Cellular m-homotopies)**.** *Given two cellular m-maps $f_\bullet, g_\bullet : C_\bullet \to D_\bullet$, an m-homotopy between $f_\bullet$ and $g_\bullet$, denoted $f_\bullet \sim_m g_\bullet$, is a finite sequence of homotopies $(h_n : \iota_n \circ f_n = \iota_n \circ g_n)_{n\leq m}$ such that for $n < m$ and $x : C_n$ the following square commutes:*

$$\begin{array}{ccc} (\iota_{n+1} \circ f_{n+1} \circ \iota_n)\ x & \xlongequal{(h_{n+1}\circ\iota_n)(x)} & (\iota_{n+1} \circ g_{n+1} \circ \iota_n)\ x \\ \| & & \| \\ (\iota_{n+1} \circ \iota_n \circ f_n)\ x & \xlongequal[(\iota_{n+1}\circ h_n)(x)]{} & (\iota_{n+1} \circ \iota_n \circ g_n)\ x \end{array}$$

Theorem 1 (Cellular m-approximation theorem)**.** *Given two unordered CW skeleta $C_\bullet, D_\bullet$, a map $f : C_\infty \to D_\infty$ and $m : \mathbb{N}$, there merely exists an m-cellular map $f_\bullet : C_\bullet \to D_\bullet$ such that $\iota_\infty \circ f_m = f \circ \iota_\infty$.*

Theorem 2 (Cellular m-approximation theorem, part 2)**.** *Let $C_\bullet, D_\bullet$ be unordered CW skeleta and consider two cellular m-maps map $f_\bullet, g_\bullet : C_\bullet \to D_\bullet$ with a such that $f_m = g_m$. In this case, there merely exists a cellular m-homotopy $f_\bullet \sim_m g_\bullet$.*

Sketch of proofs. The proof of Theorem 1 is done by induction on m: if we have an n-approximation of f, we can use the principle of finite choice to obtain the mere existence of an $(n+1)$-approximation. Note that we only approximate f up to a fixed dimension m, so that the construction only needs finitely many calls to finite choice, which is constructively valid [13, exercise 3.22]. Theorem 2 is proved using the exact same techniques. □

Although our statements of the cellular approximation theorems are sufficient to develop cellular homology in HoTT [11], they are weaker than their classical counterparts on two points. Firstly, we only obtain the *mere existence* of an approximation. However, since every construction in HoTT has to be homotopy invariant, the untruncated version of Theorem 1 is actually inconsistent: when specialised to the unit type and the circle (which are both CW complexes), the untruncated approximation theorem amounts to the contractibility of the circle. Therefore, some amount of truncation is required to state the theorem in HoTT. Secondly, the classical cellular approximation theorems are stated for $m = \infty$, while ours only provide finite approximations. In fact, due to the fundamental reliance of the theorem on finite choice, we conjecture that the case $m = \infty$ is equivalent to the axiom of countable choice, and thus not provable in constructive HoTT.

Conjecture 1. *The case $m = \infty$ of the cellular approximation theorems is not provable in plain HoTT for unordered CW skeleta.*

However, in the case of *ordered* CW skeleta, we expect that it is possible to use the order on the sets of cells to pick a *minimal* approximation at each stage for some carefully defined order. This eschews the need for finite choice and thus we conjecture the following:

Conjecture 2. *The cellular approximation theorems hold for $m = \infty$ for ordered CW skeleta.*

References

[1] Guillaume Brunerie. *On the homotopy groups of spheres in homotopy type theory.* PhD thesis, Université Nice Sophia Antipolis, 2016.

[2] Guillaume Brunerie, Axel Ljungström, and Anders Mörtberg. Synthetic Integral Cohomology in Cubical Agda. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*, volume 216 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[3] Ulrik Buchholtz and Kuen-Bang Hou Favonia. Cellular Cohomology in Homotopy Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 521–529, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Ulrik Buchholtz and Egbert Rijke. The cayley-dickson construction in homotopy type theory. *ArXiv*, abs/1610.01134, 2016.

[5] Evan Cavallo. Synthetic Cohomology in Homotopy Type Theory. Master's thesis, Carnegie Mellon University, 2015.

[6] J. Daniel Christensen and Luis Scoccola. The Hurewicz theorem in Homotopy Type Theory. *Algebraic & Geometric Topology*, 23:2107–2140, 2023.

[7] Robert Graham. Synthetic Homology in Homotopy Type Theory, 2018. Preprint.

[8] Thomas Lamiaux, Axel Ljungström, and Anders Mörtberg. Computing cohomology rings in cubical agda. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, page 239–252, New York, NY, USA, 2023. Association for Computing Machinery.

[9] Daniel R. Licata and Eric Finster. Eilenberg-MacLane Spaces in Homotopy Type Theory. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, New York, NY, USA, 2014. Association for Computing Machinery.

[10] Axel Ljungström and Anders Mörtberg. Computational synthetic cohomology theory in homotopy type theory, 2024.

[11] Axel Ljungström, Anders Mörtberg, and Loïc Pujet. Cellular homology and the cellular approximation theorem, 2024. Extended abstract at HoTT/UF 2024.

[12] J.P. May. *A Concise Course in Algebraic Topology.* Chicago Lectures in Mathematics. University of Chicago Press, 1999.

[13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* Self-published, Institute for Advanced Study, 2013.

[14] Floris van Doorn. *On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory.* PhD thesis, Carnegie Mellon University, May 2018.

Revisiting the Steenrod Squares in HoTT

Axel Ljungström[1] and David Wärn[2]

[1] Stockholm University, Stockholm, Sweden
[2] University of Gothenburg, Gothenburg, Sweden

Given a type A and an abelian group G, we may define, in homotopy type theory (HoTT), the *nth cohomology group of A with coefficients in G* by $H^n(A,G) := \| A \to K(G,n) \|_0$. Here, $K(G,n)$ denotes the *nth Eilenberg MacLane space* as defined in [8] and $\|-\|_0$ denotes 0-truncation. The graded-commutative ring structure on $K(G,*)$ (and hence $H^*(A,G)$) induced by $\smile$, the cup product, is well-studied in HoTT [2, 1, 4, 7, 9]. Sometimes, however, the ring structure is not enough. There are many examples of non-equivalent types appearing in the wild whose cohomology rings are isomorphic. Sometimes, however, they can be distinguished using an even more fine grained invariant: *the Steenrod squares*. The Steenrod squares are a set of cohomology operations of type $H^m(X,\mathbb{Z}/2\mathbb{Z}) \to H^{n+m}(X,\mathbb{Z}/2\mathbb{Z})$. They were originally defined in HoTT by Brunerie [3] but little to nothing has been proved concerning their properties. The work we present here is a continuation dedicated precisely to verifying these properties. In particular, we present an ongoing project on the proof and computer formalisation of the following theorem. For ease of notation, we from now on let $K_n := K(\mathbb{Z}/2\mathbb{Z}, n)$.

Theorem 1. *There is a set of pointed maps* $\mathsf{Sq}^n : K_m \to_\star K_{m+n}$ *called the Steenrod squares s.t.*

1. $\mathsf{Sq}^0(x) = x$,
2. $\mathsf{Sq}^n(x) = 0$ *if* $n > m$,
3. $\mathsf{Sq}^n(x) = x \smile x$ *if* $n = m$,
4. $\mathsf{Sq}^n(x \smile y) = \sum_{i+j=n} \mathsf{Sq}^i(x) \smile \mathsf{Sq}^j(y)$ *(the Cartan formula)*
5. $\mathsf{Sq}^n \circ \mathsf{Sq}^k = \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{k-i-1}{n-2i} \mathsf{Sq}^{n+k-i} \circ \mathsf{Sq}^i$ *for* $0 < n < 2k$ *(the Adem relations)*
6. Sq^n *respects suspension, i.e. the following diagram commutes*

$$\begin{array}{ccc} \Omega(K_{m+1}) & \xrightarrow{\Omega(\mathsf{Sq}^n)} & \Omega(K_{(m+1)+n}) \\ \wr\downarrow & & \downarrow\wr \\ K_m & \xrightarrow{\mathsf{Sq}^n} & K_{m+n} \end{array}$$

While the formalisation of the exact statement of Theorem 1 is ongoing, we have already formalised one crucial theorem which implies the most involved axioms, namely 4 and 5.

Construction of the Steenrod Squares Our construction[1] of Sq^n follows that of Brunerie [3], but we make some alterations and generalisations[2]. The idea is to define all squares simultaneously by defining $\widehat{\mathsf{Sq}} : K_m \to_\star K_0 \times \cdots \times K_{2m}$ and letting $\mathsf{Sq}^n := \mathsf{proj}_{m+n} \circ \widehat{\mathsf{Sq}}$ if $n \leq m$ and

[1] We have, in fact, provided an alternative and very direct definition of Sq^n by analysis of its type, i.e. the function space $K_m \to_\star K_{m+n}$. One can show, using well-known results about connectivity and a result by Wärn [10], that the canonical map $\Omega : (K_m \to_\star K_{m+n}) \to (K_{m-1} \to_\star K_{(m-1)+n})$ has a left inverse when $n < m$. This allows for a recursive definition. While this definition trivially satisfies axioms 1,2,3 and 6, it is much less obvious why it should satisfy axiom 4 and, even less so, axiom 5.

[2] For instance, all of our constructions take place on the level of Eilenberg-MacLane spaces while Brunerie suggested working with $H^n(\mathbb{R}P^\infty, \mathbb{Z}/2\mathbb{Z})$

$\mathsf{Sq}^n(x) = 0$ otherwise. Let $\mathbb{R}P^\infty := \Sigma_{X:\mathsf{Type}} \| X \simeq \mathsf{Bool} \|_{-1}$ [5]. We simply write $X : \mathbb{R}P^\infty$ for types in $\mathbb{R}P^\infty$ and take the second field implicit. We now consider the composition of maps

$$\Pi_{X:\mathbb{R}P^\infty} \Pi_{n:X\to\mathbb{N}} \left(K_{n(-)} \to^X_\star K_{\Sigma\, n}\right) \xrightarrow{\phi_3} \Pi_{X:\mathbb{R}P^\infty} \Pi_{n:X\to\mathbb{N}} \left(\Pi_{x:X} K_{n(x)} \to K_{\Sigma\, n}\right)$$
$$K_m \to_\star (\mathbb{R}P^\infty \to K_{2m}) \xrightarrow{\phi_1} K_m \to_\star K_0 \times \cdots \times K_{2m} \quad (\phi_2 : \text{reverse})$$

where ϕ_1 is given by the Thom isomorphism [9], ϕ_2 is the diagonal maps defined by $\phi_2(F) := \lambda a\, \lambda X\, .\, F\, X\, (\lambda\, x\, .\, m)\, (\lambda\, x\, .\, a)$. The mysterious type appearing in the domain of ϕ_3 denotes the type of unordered (rel. $X : \mathbb{R}P^\infty$) bipointed maps. The idea is that, for any $X : \mathbb{R}P^\infty$, $A : X \to \mathsf{Type}$ and $B : \mathsf{Type}_\star$, there is a type $(A \to^X_\star B)$ equivalent to $A_0 \to_\star (A_1 \to_\star B)$ in case $X \simeq \mathsf{Bool}$. Brunerie [3] implicitly defined this type in terms of *unordered smash products*, but we use, for technical reasons, *unordered joins*.

Definition 1. *Given $X : \mathbb{R}P^\infty$ and $A : X \to \mathsf{Type}$, we define the unordered join of A (rel. X), denoted $\ast_{x:X} A(x)$, to be the pushout of the span $\Sigma_{x:X} A(x) \leftarrow A \times \Pi_{x:X} A(x) \xrightarrow{\mathsf{snd}} \Pi_{x:X} A(x)$.*

We may now define $(A \to^X_\star B) := \sum_{F:\Pi_{x:X} A(x)\to B} \mathsf{isBiHom}(\mathsf{F})$ where

$$\mathsf{isBiHom}(\mathsf{F}) := \prod_{f:\Pi_{x:X} A(x)} \left(\ast_{x:X} (f(x) = \star_{A(x)}) \to F(f) = \star_B \right)$$

In order to construct the Steenrod squares, we hence need to construct a map $S(X, -) : K_{n(-)} \to^X_\star K_{\Sigma\, n}$ for each $X : \mathbb{R}P^\infty$ and $n : X \to \mathbb{N}$. One can easily show, using results from [9] and [10], that the type of non-constant such functions is contractible with centre of contraction given by the cup product in the case $X := \mathsf{Bool}$.

Verifying the axioms With this definition, axioms 2 and 3 follow by construction. Both the Cartan formula and the Adem relations should follow from the fact that $S(X, S(Y, f)) = S(Y, S(X, \lambda\, y\, \lambda\, x\, .f(x, y)))$ for any $X, Y : \mathbb{R}P^\infty$ and $f : \Pi_{x:X} \Pi_{y:Y} K_{n(x,y)}$.[3] This, in turn, follows from the following theorem whose proof turns out to be surprisingly involved.

Theorem 2. *For any $X, Y : \mathbb{R}P^\infty$, there is a map $\ast_{x:X} \ast_{y:Y} A(x, y) \to \ast_{y:Y} \ast_{x:X} A(x, y)$.*

Yet another Brunerie number What about axioms 1 and 6? The following lemma is an easy consequence of the Cartan formula and an analysis of the cohomology of $\mathbb{R}P^\infty$.

Lemma 1. *If the composition $\mathbb{S}^1 \xrightarrow{e} K_1 \xrightarrow{\mathsf{Sq}^0} K_1$ is non-constant, then axioms 1 and 6 hold.*

Above, e is defined by sending the canonical 1-cell in $\mathbb{S}^1$ to the canonical 1-cell in K_1. In fact, this composition can be explicitly defined by

$$\mathbb{S}^1 \xrightarrow{e} K_1 \xrightarrow{x\mapsto(x^2, \mathsf{comm}_{1,1}(x,x))} \Sigma_{x:K_2} \Omega(K_2, x) \xrightarrow{(x,p)\mapsto \mathsf{ap}_{\lambda a\, .\, a-x}(p)} \Omega(K_2, \star_{K_2}) \xrightarrow{\sim} K_1$$

where $\mathsf{comm}_{1,1}(x, y) : x \smile y = y \smile x$. Let us call this composite map F. Lemma 1 should be provable by simply evaluating $\mathbb{1} \xrightarrow{\mapsto 1} \mathbb{Z} \xrightarrow{\sim} \Omega(S^1) \xrightarrow{\Omega(F)} \Omega(K_1) \xrightarrow{\sim} \mathbb{Z}/2\mathbb{Z}$ and observing that 1 is returned. Our attempts to compute this number in `Cubical Agda` have thus far failed and, while a normal 'pen-and-paper' proof Lemma of 1 should be achievable using similar methods to those used to compute the cohomology ring of the subcomplex $\mathbb{R}P^2$ in [9], we believe that this number serves as yet another interesting example of how HoTT allows us to reduce a non-trivial mathematical result to something as 'simple' as a computation.

[3] This approach is taken e.g. in [6]. For instance, the Cartan formula follows by setting $Y := \mathsf{Bool}$.

References

[1] Tim Baumann. The cup product on cohomology groups in homotopy type theory. Master's thesis, University of Augsburg, 2018.

[2] Guillaume Brunerie. *On the homotopy groups of spheres in homotopy type theory*. PhD thesis, Université Nice Sophia Antipolis, 2016.

[3] Guillaume Brunerie. The steenrod squares in homotopy type theory. Abstract at *23rd International Conference on Types for Proofs and Programs (TYPES 2017)*, 2016.

[4] Guillaume Brunerie, Axel Ljungström, and Anders Mörtberg. Synthetic Integral Cohomology in Cubical Agda. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*, volume 216 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[5] Ulrik Buchholtz and Egbert Rijke. The real projective spaces in homotopy type theory. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '17. IEEE Press, 2017.

[6] Jean-Claude Hausmann. *Steenrod Squares*, pages 325–354. Springer International Publishing, Cham, 2014.

[7] Thomas Lamiaux, Axel Ljungström, and Anders Mörtberg. Computing cohomology rings in cubical agda. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, page 239–252, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Daniel R. Licata and Eric Finster. Eilenberg-MacLane Spaces in Homotopy Type Theory. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, New York, NY, USA, 2014. Association for Computing Machinery.

[9] Axel Ljungström and Anders Mörtberg. Computational synthetic cohomology theory in homotopy type theory, 2024.

[10] David Wärn. Eilenberg–maclane spaces and stabilisation in homotopy type theory. *Journal of Homotopy and Related Structures*, 18(2):357–368, Sep 2023.

Session 14: Logic

Representing Temporal Operators with Dependent Event Types

Felix Bradley and Zhaohui Luo

Royal Holloway, University of London, Egham, U.K.
felix.bradley@rhul.ac.uk
zhaohui.luo@hotmail.co.uk

Event semantics were first proposed and studied by Davidson to express the occurrence of events in order to find suitable semantics for describing both actions and adverbial modifications [2], and further studied and developed by many in the fields of logic, philosophy and computer science [10, 8, 3, 9]. Later works showed that dependent types theories were suitable for formalising event semantics in the form of dependent event types, which allow for selection restriction through semantic roles such as the agent of an event [5].

In this work, we propose a further refinement of dependent event types by extending a type system with dependent event types by arbitrary semantic roles. We show that in the case of Church's simple type theory with dependent event types, this extension by arbitrary semantic roles is conservative. We focus on the semantic role of time via the introduction of a timeline type Time and subtypes of events parameterised by a timestamp, and use this to formulate traditional temporal operators and explore their properties in this system.

Dependent Event Types. In Davidsonian event semantics, there exists a type Evt of all events, and each event contains information associated with that event's semantic roles. When working in with Montague grammar, we can give meaning to sentences by interpreting them as types and using predicates to restrict their arguments. For example, we can interpret the sentence "Claire eats an apple" with agent 'Claire' and patient 'apple' as the type

$$\exists(e : \mathrm{Evt}).\mathrm{agent}(e, \mathrm{Claire}) \wedge \mathrm{patient}(e, \mathrm{apple}) \wedge \mathrm{eats}(e).$$

The development of dependent event types allows one to consider subtypes of Evt parameterised by an event's semantic roles, such as the agent of an event. By extending the type system with dependent event types and new types for each semantic role[1], we can instead consider subtypes of Evt paramaterised by semantic roles. For example, to express the sentence "Claire eats an apple" this way, we need the following rules

$$\frac{\Gamma \vdash a : \mathrm{Agent}}{\Gamma \vdash \mathrm{Evt}_A(a)\,\mathrm{type}} \qquad \frac{\Gamma \vdash a : \mathrm{Agent}}{\Gamma \vdash \mathrm{Evt}_P(p)\,\mathrm{type}} \qquad \frac{\Gamma \vdash a : \mathrm{Agent} \qquad \Gamma \vdash p : \mathrm{Patient}}{\Gamma \vdash \mathrm{Evt}_{AP}(a,p)\,\mathrm{type}}$$

and subsumptive subtyping relations

$$\frac{\Gamma \vdash a : \mathrm{Agent}}{\Gamma \vdash \mathrm{Evt}_A(a) \leq \mathrm{Evt}} \qquad \frac{\Gamma \vdash p : \mathrm{Patient}}{\Gamma \vdash \mathrm{Evt}_P(p) \leq \mathrm{Evt}}$$

$$\frac{\Gamma \vdash a : \mathrm{Agent} \qquad \Gamma \vdash p : \mathrm{Patient}}{\Gamma \vdash \mathrm{Evt}_{AP}(a,p) \leq \mathrm{Evt}_A(a)} \qquad \frac{\Gamma \vdash a : \mathrm{Agent} \qquad \Gamma \vdash p : \mathrm{Patient}}{\Gamma \vdash \mathrm{Evt}_{AP}(a,p) \leq \mathrm{Evt}_P(p)}.$$

In this framework, we are able to express this sentence as

$$\exists(e : \mathrm{Evt}_{AP}(\mathrm{Claire}, \mathrm{apple})).\mathrm{eats}(e)$$

[1] In the case of the given example, the semantic roles of agent and patient.

where eats : Evt $\to \mathbf{t}$, which is well-typed due to our use of subtyping.

Prior work on dependent event types has shown that extending Church's simple type theory with dependent event types is a conservative extension [6]. We extend these results to show that further extending this system with arbitrary semantic roles and subtypes of Evt parameterised by these semantic roles is also a conservative extension.

Theorem 1 (Conservativity)**.** *Let C be Church's simple type theory. Fix S a collection of semantic roles, and let $C_E[S]$ be C extended by dependent event types with subtypes parameterised by semantic roles in S. Then $C_E[S]$ is a conservative extension of C.*

Of particular interest, this means that extending dependent event types with time is conservative. We also extend the subtyping of Evt to allow for one to parameterise the time that an event occurs (Evt_T) and also for an interval in which an event occurs (Evt_{TT}).

Temporal operators. One of the original motivations for this work was to further study a modal type theory from a rules-first approach, where most prior work on modal type theories have been from a model-first approach. In particular, there was interest in extending dependent event types with the semantic role of time to enable the description and use of temporal operators.

There are typically two perspectives that are used in the study of temporal logic: those concerned with the analysis of computer software are likely to view temporal operators as a kind of function that takes as input a given moment in time and checks for truth of a given proposition at that moment of time; whereas those concerned with the description of natural language are likely to take the view that each statement has a 'speaking time', an inherent moment in time which it is in reference to, and can only be evaluated in comparison to that speaking time. Our work starts with the latter as a basis, and extends it to the former.

We can describe the traditional $\Box$ and $\Diamond$ operators

$$\Diamond A \stackrel{\text{def}}{=} \exists(t : \text{Time}).(\mathit{now} \preceq t \wedge A(t))$$

$$\Box A \stackrel{\text{def}}{=} \forall(t : \text{Time}).(\mathit{now} \preceq t \to A(t))$$

and use these to encode and represent simple sentences such as "John will talk" as $\Diamond A$, where $A(t) \stackrel{\text{def}}{=} \exists(e : \text{Evt}_{AT}(\text{John}, t).\text{talk}(e)$. However, these are rather restrictive in their use due to their fixed speaking time. While these can be used to express the sentence "there will be flying cars in the future", these fail to carry the importance of what point in time this sentence was instantiated or spoken. If we read this sentence is the year 2024, it carries different information and different meaning when it was spoken in the year 1985 versus the year 2023. We can adapt the above temporal operators to allow for variable speaking time.

$$\Diamond A(\mathit{ref}) \stackrel{\text{def}}{=} \exists(t : \text{Time}).(\mathit{ref} \preceq t \wedge A(t))$$

$$\Box A(\mathit{ref}) \stackrel{\text{def}}{=} \forall(t : \text{Time}).(\mathit{ref} \preceq t \to A(t))$$

Taking this approach also changes their type signature from (Time $\to \mathbf{t}$) $\to \mathbf{t}$ to (Time $\to \mathbf{t}$) $\to$ (Time $\to \mathbf{t}$), which allows us to nest multiple temporal operators together, e.g. $\Box\Diamond A$. This allows us to have the important distinction of different speaking times. For example, one may consider the sentence "John will eventually always have grey hair" encoded as $\Diamond(\Box A)\ \mathit{now}$, where A is the sentence "John has grey hair."

Conclusion. This work is currently ongoing, and we plan to extend our results from extending Church's simple type theory to extending modern type theories such as Martin-Löf's type theory. For Church's simple type theory, Montague grammar is a well-studied approach to natural language semantics which allows one to express the categorisation of objects by checking that they satisfy 'checking' propositions, but this can lead to syntactically correct but categorically incorrect sentences [11]. Different approaches prevent this, such as the use of subtyping to express the relationships between categories [1, 4], or by using a modern type theory to ensure that only categorically correct sentence are well-typed [12]. For example, "John is speaking" could be interpreted as

$$\exists(e : \mathrm{Evt}_A(\mathrm{John})).\mathrm{speaks}(e)$$

where speaks : Evt $\to$ **t** and the inhabitance of this type expresses the truth of this sentence. On the other hand, while one may be able to form an expression for a sentence such as "the table is talking," it could not be a well-typed sentence without further providing the sentence further context, such as Table $\leq$ Human.

However, working with Church's simple type theory allows for the use of subsumptive subtyping when describing subtypes of Evt, whereas working to extend modern type theories which allow for dependent types, polymorphism, and other more complex expressions requires a more nuanced approach through coercive subtyping [7].

Dependent event types extended with the semantic role of time bears some resemblance to functional reactive programming. However, this is outside the scope of this work.

References

[1] Stergios Chatzikyriakidis and Zhaohui Luo. On the interpretation of common nouns: Types versus predicates. In Stergios Chatzikyriakidis and Zhaohui Luo, editors, *Modern Perspectives in Type-Theoretical Semantics*, pages 43–70. Springer Cham, 2017. doi:10.1007/978-3-319-50422-3.

[2] Donald Davidson. The Logical Form of Action Sentences. In *Essays on Actions and Events*. Oxford University Press, 09 2001. doi:10.1093/0199246270.003.0006.

[3] Philippe de Groote and Yoad Winter. A type-logical account of quantification in event semantics. In *New Frontiers in Artificial Intelligence*. Springer, August 2015. doi:10.1007/978-3-662-48119-6_5.

[4] Zhaohui Luo. Common nouns as types. In D. Béchet and A. Dikovsky, editors, *Proceedings of the 7th International Conference on Logical Aspects of Computational Linguistics (LACL'12)*, pages 173–185, Berlin, Heidelberg, 2012. Springer-Verlag.

[5] Zhaohui Luo and Sergei Soloviev. Dependent event types. In *Proc of the 24th Workshop on Logic, Language, Information and Computation (WoLLIC'17)*, LNCS, pages 216–228, London, 2017. doi:10.1007/978-3-662-55386-215.

[6] Zhaohui Luo and Sergei Soloviev. Dependent event types. Manuscript, 2020.

[7] Zhaohui Luo, Sergei Soloviev, and Tao Xue. Coercive subtyping: Theory and implementation. *Information and Computation*, 223:18–42, 2013. URL: https://www.sciencedirect.com/science/article/pii/S0890540112001757, doi:10.1016/j.ic.2012.10.020.

[8] Claudia Maienborn. Event semantics. *Semantics - Theories*, 1:232–266, July 2019. doi:10.1515/9783110589245-008.

[9] Greg N. Carlson. Thematic roles and their role in semantic interpretation. In *Linguistics*, volume 22. De Gruyter Mouton, 1984. doi:10.1515/ling.1984.22.3.259.

[10] Terence Parsons. *Events in the Semantics of English*. MIT Press, January 1994.

[11] Richmond H. Thomason, editor. *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, 1974.

[12] Tao Xue, Zhaohui Luo, and Stergios Chatzikyriakidis. Propositional forms of judgemental interpretations. *Journal of Logic, Language and Information*, 32:733–758, May 2023. doi:10.1007/s10849-023-09397-y.

Normalization for classical natural deduction

Herman Geuvers[1] and Tonny Hurkens

[1] Radboud University Nijmegen & Technical University Eindhoven (NL)
herman@cs.ru.nl
[2] hurkens@science.ru.nl

Normalization for natural deduction has been studied by various authors, starting from the seminal work of Prawitz [8]. The intuitionistic case is studied in full by Prawitz, but the full classical case is restricted to the system without $\exists$ and $\vee$. Normalization states that if $\Gamma \vdash \varphi$ is derivable, then there is a derivation Σ of $\Gamma \vdash \varphi$ which is in normal form.

An important consequence of normalization is that for every derivable judgment $\Gamma \vdash \varphi$, there is a derivation Σ satisfying the *sub-formula property*: all formulas appearing in Σ are subformulas of φ or of a formula in Γ. This is a consequence of normalization because derivations in normal form satisfy the subformula property.

The sub-formula property also implies that the rules for the connectives are self-contained: a deduction of a formula involving (for example) only $\to$ does not require other connectives. For constructive logic, the idea of normalization is to eliminate so called *detours*, an introduction rule followed by an elimination rule, and to be able to *permute* elimination rules to make detours explicit. For classical predicate logic, the situation is more subtle and e.g. Prawitz[8] restricts the logic and Statman[10] restricts the ways in which the rules can be applied.

We give a proof of normalization of natural deduction for classical predicate logic using our earlier work on truth-table natural deduction (TT-ND, [1, 2, 3]). TT-ND gives a general format for propositional deduction rules and uses a term-interpretation of deductions to study normalization. We instantiate the general TT-ND format for the case of the well-known connectives ($\wedge$, $\vee$, $\to$ $\neg$) and we add the quantifiers $\forall$ and $\exists$. For the quantifiers we use classical rules, using so called *witnesses*, basically Hilbert's ε and τ choice operators. We show normalization by exhibiting a proof-term-reduction procedure (not detailed in this abstract) that terminates and we show that normal deductions have the sub-formula property.

Similar approaches to natural deduction, using so called "generalized" introduction and elimination rules, can be found, a.o., in the work of Von Plato and Siders [12] Milne [7], Kürbis [6, 5] and Shangin [9]. Our approach is specific as it uses classical rules for the quantifiers and can be flexibly adapted to include (or exclude) specific connectives. It gives the unrestricted sub-formula property for normal deductions. It also deals with the problematic (according to Milne [7]) classically valid $\forall x\, P\, x \vee \exists x\, \neg P\, x$ and a derivation of $(\forall x\, P\, x) \vee C$ from $\forall x\, P\, x \vee C$. In our system, these have derivations in normal form (thus satisfying the sub-formula property).

To summarize, the main results are the following.

1. Self-contained classical deduction rules for propositional connectives and $\forall, \exists$ that are equivalent to the the well-known ones.

2. Classical derivations, e.g. of $\vdash \exists x.((\exists y.P\, y) \to P\, x)$ and of $\forall x.(P\, x \vee C) \vdash (\forall x.P\, x) \vee C$ that satisfy the sub-formula property.

3. A term-interpretation of derivations, extending [4] to predicate logic, on which normalization can be defined as a reduction operation that creates derivations in normal form.

4. An approach that is modular: one can remove or add connectives with their rules, and the result remains.

We claim that also other classical quantifiers can be added with their own "stand alone" derivation rules and that the normalization approach extends to those.

The deduction rules are the classical ones derived from the standard approach in [1]. (Similar rules can be found in the work of Milne [7] and the general elimination rules of Von Plato [11].) We only give the rules for $\wedge, \to \neg$; we label the rules that are specifically classical with a c-subscript. The well-known rules are easily derivable: they usually occur as a simplified case.

$$\frac{\begin{array}{c}[A \wedge B]^k \\ \cdots \\ D\end{array} \qquad A \qquad B}{D}\ \wedge\text{-in}^k \qquad \frac{\begin{array}{c}[A]^k \\ \cdots \\ A \wedge B\end{array} \qquad D}{D}\ \wedge\text{-el}_1^k \qquad \frac{\begin{array}{c}[B]^k \\ \cdots \\ A \wedge B\end{array} \qquad D}{D}\ \wedge\text{-el}_2^k$$

$$\frac{\begin{array}{c}[A \to B]^k \\ \cdots \\ D\end{array} \qquad \begin{array}{c}[A]^k \\ \cdots \\ D\end{array}}{D}\ \to\text{-in}_c^k \qquad \frac{\begin{array}{c}[A \to B]^k \\ \cdots \\ D\end{array} \qquad B}{D}\ \to\text{-in}_2^k \qquad \frac{A \to B \qquad A \qquad \begin{array}{c}[B]^k \\ \cdots \\ D\end{array}}{D}\ \to\text{-el}^k$$

$$\frac{\begin{array}{c}[\neg A]^k \\ \cdots \\ D\end{array} \qquad \begin{array}{c}[A]^k \\ \cdots \\ D\end{array}}{D}\ \neg\text{-in}_c^k \qquad \frac{\neg A \qquad A}{D}\ \neg\text{-el}$$

For the classical rules of $\forall$ and $\exists$, we use *witness constants*: $a_{\forall x.\varphi}$ and $a_{\exists x.\varphi}$. Semantically they are interpreted in such a way that $\forall x.\varphi \Leftrightarrow \varphi(a_{\forall x.\varphi})$ and $\exists x.\varphi \Leftrightarrow \varphi(a_{\exists x.\varphi})$. The derivation rules also guarantee these equivalences. In the rules, t is an arbitrary term, while $a_{\forall x.\varphi}$ and $a_{\exists x.\varphi}$ are these special constants. Note that there are no side conditions in the rules, e.g. in $\exists$-el that $a_{\exists x.\varphi}$ should not occur in the other hypotheses.

$$\frac{\begin{array}{c}[\forall x.\varphi]^k \\ \cdots \\ D\end{array} \qquad \varphi(a_{\forall x.\varphi})}{D}\ \forall\text{-in}^k \qquad \frac{\forall x.\varphi \qquad \begin{array}{c}[\varphi(t)]^k \\ \cdots \\ D\end{array}}{D}\ \forall\text{-el}^k$$

$$\frac{\begin{array}{c}[\exists x.\varphi]^k \\ \cdots \\ D\end{array} \qquad \varphi(t)}{D}\ \exists\text{-in}^k \qquad \frac{\exists x.\varphi \qquad \begin{array}{c}[\varphi(a_{\exists x.\varphi})]^k \\ \cdots \\ D\end{array}}{D}\ \exists\text{-el}^k$$

Example. We now have (for x not in C): $\forall x.(P\,x \vee C) \vdash (\forall x.P\,x) \vee C$ with the following derivation satisfying the sub-formula property. (We abbreviate $a_\forall := a_{\forall x.P\,x}$ and we sometimes use simplified derivation rules.)

$$\frac{\dfrac{\dfrac{\forall x.(P\,x \vee C)}{P\,a_\forall \vee C}\ \forall\text{-el} \qquad \dfrac{\dfrac{[P\,a_\forall]^1}{\forall x.P\,x}\ \forall\text{-in}}{(\forall x.P\,x) \vee C}\ \vee\text{-in}_1 \qquad \dfrac{[C]^1}{(\forall x.P\,x) \vee C}\ \vee\text{-in}_2}{(\forall x.P\,x) \vee C}\ \vee\text{-el}^1$$

Our main result is an interpretation of derivations as proof-terms, extending [4], and a normalization procedure on these terms, showing that every classical derivation can be reduced to one satisfying the sub-formula property.

References

[1] H. Geuvers and T. Hurkens. Deriving natural deduction rules from truth tables. In *ICLA*, volume 10119 of *Lecture Notes in Computer Science*, pages 123–138. Springer, 2017.

[2] H. Geuvers and T. Hurkens. Proof Terms for Generalized Natural Deduction. In A. Abel, F. Nordvall Forsberg, and A. Kaposi, editors, *23rd International Conference on Types for Proofs and Programs (TYPES 2017)*, volume 104 of *LIPIcs*, pages 3:1–3:39, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[3] H. Geuvers, I. van der Giessen, and T. Hurkens. Strong normalization for truth table natural deduction. *Fundam. Inform.*, 170(1-3):139–176, 2019.

[4] Herman Geuvers and Tonny Hurkens. Classical natural deduction from truth tables. In Delia Kesner and Pierre-Marie Pédrot, editors, *28th International Conference on Types for Proofs and Programs, TYPES 2022, June 20-25, 2022, LS2N, University of Nantes, France*, volume 269 of *LIPIcs*, pages 2:1–2:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.TYPES.2022.2, doi:10.4230/LIPICS.TYPES.2022.2.

[5] Nils Kürbis. Normalisation and subformula property for a system of intuitionistic logic with general introduction and elimination rules. *Synthese*, 199(2):14223–14248, 2021. doi:10.1007/s11229-021-03418-8.

[6] Nils Kürbis. Normalisation and subformula property for a system of classical logic with Tarski's rule. *Arch. Math. Log.*, 61(1-2):105–129, 2022. URL: https://doi.org/10.1007/s00153-021-00775-6, doi:10.1007/S00153-021-00775-6.

[7] P. Milne. Inversion principles and introduction rules. In *Dag Prawitz on Proofs and Meaning*, volume 7 of *Outstanding Contributions to Logic*, pages 189–224. Springer, 2015.

[8] D. Prawitz. *Natural deduction: a proof-theoretical study*. Almqvist & Wiksell, 1965.

[9] Vasily Shangin. A classical first-order normalization procedure with ∀ and ∃ based on the Milne-Kürbis approach. *Synthese*, 202(2):1–24, 2023. doi:10.1007/s11229-023-04276-2.

[10] Richard Statman. *Structural Complexity of Proofs*. PhD thesis, Stanford University, 1974.

[11] J. von Plato. Natural deduction with general elimination rules. *Arch. Math. Log.*, 40(7):541–567, 2001.

[12] Jan von Plato and Annika Siders. Normal derivability in classical natural deduction. *Rev. Symb. Log.*, 5(2):205–211, 2012. doi:10.1017/S1755020311000311.

Session 15: Equality and Evaluation

Useful Evaluation, Quantitatively

Pablo Barenbaum[1], Delia Kesner[2], and Mariana Milicich[2†]

[1] Universidad Nacional de Quilmes (CONICET), and Instituto de Ciencias de la Computación, UBA, Argentina
[2] Université Paris Cité, CNRS, IRIF, France

The *Invariance Thesis* [15] states that standard machine models, like Turing machines and random access machines, can simulate each other with polynomial overhead in time and constant overhead in space; they are called **invariant**. A key question is whether the λ-calculus is invariant, and in order to answer that, it is necessary to provide a *cost model*, *i.e.*, a way to determine the cost of a computation. For example, the length of a reduction sequence to normal form is one possible cost model to measure *time*. Proposing cost models for the λ-calculus involves two choices. Firstly, a *reduction strategy* needs to be fixed, such as *call-by-name* (CBN) or *call-by-value* (CBV). The second choice is a *term representation*. Tree-based term representation can lead to *size explosion* (see *e.g.* [7]), which has prompted the exploration of *succinct* representations based on *sharing*, such as those utilizing *explicit substitutions* (ES). By avoiding size explosion, invariant cost models can be obtained.

The question whether the λ-calculus is an invariant model remained open for many years, since avoiding size explosion is non-trivial. Indeed, in [7], leftmost-outermost reduction is shown to be an invariant cost model. This proof relies on a key notion called *useful evaluation*. Moreover, they restrict the invariance thesis to a *weak* variant, which ignores the space requirement.

Usefulness. The notion of useful evaluation is based on two ideas: using ESs to represent λ-terms, and restricting the copying of shared subterms to avoid size explosion. The study on usefulness has been extended to other strategies, such as open call-by-need [8] and different variants of CBV [1, 4, 6]. In these works, usefulness is specified by means of global restrictions on reduction steps at the meta-level. This diverges from the inductive way in which one usually reasons about the syntax and semantics of programming languages and proof assistants. Indeed, inductive methods offer a more structured and rigorous approach to understand, specify, and implement evaluation strategies in programming language theory. They provide clarity and precision, making it easier to achieve formal analysis and proofs.

Open Call-by-Value. In functional programming languages, evaluation is defined on *closed* terms, which do not contain free variables. Accordingly, evaluation is *weak*, not proceeding inside the bodies of abstractions. However, in proof assistants, evaluation is *strong*, allowing to compute inside abstractions, so it needs to operate on *open* terms, which may include occurrences of free variables.

This work is part of a broader, community-driven effort to understand the concept of useful *strong* CBV. Moreover, it has been noted that usefulness is not really required to obtain an invariant cost model in the *open* and *weak* case [6]. But in order to achieve a robust notion for useful strong CBV, it is essential to develop tools that enhance our understanding of usefulness within a less complex framework such as the one open CBV presents, which already presents numerous technical challenges [4].

†  *This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 945332.*

Quantitative Interpretations. Intersection Types (IT) [12] extend simple types with a new type constructor $\cap$ such that a program t becomes typable with $\alpha \cap \beta$ if t is typable with both types α and β independently. They were originally introduced as (*qualitative*) *models* capturing computational properties of functional programs. For example, termination of a particular evaluation strategy can be characterized by typability in an appropriate IT system, so that a program t is terminating for the evaluation strategy if and only if t is typable in the associated type system [11] (which means that typability becomes *undecidable* in these systems). Initially, the constructor $\cap$ was defined, in particular, as an idempotent type constructor (*i.e.*, $\sigma \cap \sigma = \sigma$), which means that an intersection $\alpha_1 \cap \ldots \cap \alpha_n$ can be written as a *set* $\{\alpha_1, \ldots, \alpha_n\}$. By instead adopting a *non-idempotent* notion of intersection [13, 10], types can be naturally understood as multisets. Just like their idempotent precursors, non-idempotent IT still allow for a characterization of operational properties of programs by means of typability [13, 10], but they also grant a substantial improvement: they provide *quantitative* measures about these operational properties. For example, it is still possible to prove that a program is terminating if and only if it is typable, but now an *upper bound* or even an *exact measure* for the number of steps to normal form can be obtained from the typing derivation. *Quantitative types* based on non-idempotent IT have been used to provide upper and exact measures for evaluation strategies in the λ-calculus (see [9, 2]).

One crucial insight is that *exact measures*, instead of upper bounds, can be obtained by considering minimal type derivations, called *tight* [2]. Using appropriate refined tight systems, it is also possible to obtain *independent* measures for different kinds of reduction steps. More precisely, quantitative typing systems are equipped with constants and counters, together with a condition called *tightness*, ensuring that a typing derivation is minimal. *Soundness* of the resulting IT system means that for any tight type derivation Φ of a program t with a counter m, the term t evaluates to a normal form in exactly m steps (generalized for steps of many possible kinds with counters $m_1, \ldots, m_n$).

On the other hand, *completeness* means that each reduction sequence of a given size has a corresponding (tight) typing derivation with appropriate counters. Exact measures based on tight systems have been extended to encompass different notions of evaluation, such as CBN [3] and CBV [5, 14].

Contributions. The focus of our work lies in enhancing the semantic understanding of useful open CBV, from a quantitative point of view. For this, we propose a *quantitative interpretation* for an inductive formulation of useful open CBV, based on non-idempotent IT. We equip this type system with a notion of tightness, and we show that the inductive definition of useful evaluation is *sound* and *complete*, meaning in particular that for any *tight* type derivation of a program t with independent counters m and e, the term t evaluates to a normal form in exactly m function application steps and e substitution steps. This is a novel result in the literature, as existing useful evaluation notions lack semantic interpretations, and existing quantitative interpretations do not consider usefulness.

References

[1] Beniamino Accattoli and Claudio Sacerdoti Coen. On the relative usefulness of fireballs. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 141–155. IEEE Computer Society, 2015.

[2] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. *Proc. ACM Program. Lang.*, 2(ICFP):94:1–94:30, 2018.

[3] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds, fully developed. *J. Funct. Program.*, 30:e14, 2020.

[4] Beniamino Accattoli and Giulio Guerrieri. Open call-by-value. In Atsushi Igarashi, editor, *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226, 2016.

[5] Beniamino Accattoli and Giulio Guerrieri. Types of fireballs. In Sukyoung Ryu, editor, *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, volume 11275 of *Lecture Notes in Computer Science*, pages 45–66. Springer, 2018.

[6] Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019.

[7] Beniamino Accattoli and Ugo Dal Lago. (leftmost-outermost) beta reduction is invariant, indeed. *Log. Methods Comput. Sci.*, 12(1), 2016.

[8] Beniamino Accattoli and Maico Leberle. Useful open call-by-need. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[9] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Log. J. IGPL*, 25(4):431–464, 2017.

[10] Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. PhD thesis, Ecole Doctorale Physique et Sciences de la Matière (Marseille), 2007.

[11] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Arch. Math. Log.*, 19(1):139–156, 1978.

[12] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Log.*, 21(4):685–693, 1980.

[13] Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software*, pages 555–574. Springer, 1994.

[14] Delia Kesner and Andrés Viso. Encoding tight typing in a unified framework. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[15] Cees F. Slot and Peter van Emde Boas. On tape versus core; an application of space efficient perfect hash functions to the invariance of space. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 391–400. ACM, 1984.

Splitting Booleans with Normalization-by-Evaluation

Kenji Maillard[1]

Gallinette Project-Team, Inria, Nantes, France

Abstract

Is the equational theory induced by extensional sum types $A + B$ decidable ? This problem has a positive answer with simple types but is open in presence of type dependency. We investigate a simple type theory $\Pi\mathbb{B}^{\text{ext}}$ featuring extensional booleans and report on a toy OCaml implementation of a typechecker based on normalization-by-evaluation (NbE).

Extensional Sums, Extensional Booleans The equational theory of the simply typed lambda-calculus is usually centered around the $\beta\eta$-equivalence induced by the type formers $\to$ and $\times$. The addition of extensional sum types $+$, a.k.a. coproducts, greatly enhances the complexity of the equational theory. In the non-empty case, its decidability was first established by Ghani [Gha95] using rewriting techniques, later simplified by Lindley [Lin07]. Normalization-by-evaluation techniques were proposed subsequently [Alt+01; AU04; BCF04]. Recently, Scherer [Sch17] also extended it to the empty case.

In the dependently typed case, however, the equational theory becomes undecidable in presence of an extensional empty type [McB09]. The decidability problem for extensional binary coproducts $A + B$ has yet to be settled. Binary coproducts $A + B$ can be encoded as $\Sigma(b : \mathbb{B}),\ \text{if } b \text{ then } A \text{ else } B$ using extensional Σ types and large elimination, and extensionality of $A + B$ reduces to the simpler case of extensional booleans. Hence, we consider here a type theory à la Martin-Löf (MLTT) with a universe $\square$, dependent products Π and an extensional boolean type $\mathbb{B}$ (with large elimination), that we note $\Pi\mathbb{B}^{\text{ext}}$. We focus on the decidability of the equational theory since the question of consistency is easily settled by reduction to Extensional Type Theory (ETT). [1]

What does extensionality mean for booleans in $\Pi\mathbb{B}^{\text{ext}}$? To answer that question, we first remark that in a dependently typed setting, the naive interpretation of boolean extensionality as rewriting coherently boolean subterms to `true` or `false` no longer suffices. Indeed, in the following example the term t is well-typed in MLTT+`Id`+$\mathbb{B}$+$\mathbb{N}$, and would be judgementally equal to 0 with extensional booleans.

$$\alpha : \mathbb{N} \to \mathbb{B} \vdash t \coloneqq \mathtt{ind}_{\mathbb{B}}\,(\lambda b, \forall (n : \mathbb{N}), \alpha\, n = b \to \mathbb{N})\,(\lambda n\, eq, 0)\,(\lambda n\, eq, 0)\,(\alpha\, 42)\ (\mathtt{refl}_{\mathbb{B}}(\alpha\, 42)) : \mathbb{N}$$

However, no rewriting of the occurrences of $\alpha\, 42$ in t to a literal boolean `true` or `false` preserves typing, because it would forget that the reflexivity proof is of the shape $\alpha\, n = b$. In order to implement typechecking with extensional booleans, these relationships between boolean expressions and literals have to be retained. Previous work [DS95; FS99; Alt+01] solve this issue swiftly by extending contexts with booleans constraints, assumptions reifying judgemental equalities at boolean type. Using such contexts, the extensionality rule for booleans takes the shape $\mathbb{B}\eta$: in order to show that t and u are judgementally equal, it is enough to operate a case splitting on an arbitrary well-typed boolean b and strengthen the assumptions with a

[1] Indeed, the extensionality principles of booleans are derivable propositionally in presence of an identity type using the dependent eliminator for $\mathbb{B}$, and the equality reflection rule then blurs the distinction between propositional and judgemental equality. Of course, this says nothing about the decidability of the judgemental equality in $\Pi\mathbb{B}^{\text{ext}}$, since the equational theory of ETT is undecidable [CCD17].

literal boolean value for b at the price of having now two subgoals.

$$\mathbb{B}\eta\ \frac{\Gamma \vdash b : \mathbb{B} \qquad \Gamma, b \equiv \mathtt{true} \vdash t \equiv u : A \qquad \Gamma, b \equiv \mathtt{false} \vdash t \equiv u : A}{\Gamma \vdash t \equiv u : A} \qquad \textsc{If}\ \frac{\Gamma \vdash b : \mathbb{B} \qquad \Gamma, b \equiv \mathtt{true} \vdash t : A \qquad \Gamma, b \equiv \mathtt{false} \vdash u : B}{\Gamma \vdash \ \mathrm{if}\ b\ \mathrm{then}\ t\ \mathrm{else}\ u : \ \mathrm{if}\ b\ \mathrm{then}\ A\ \mathrm{else}\ B}$$

Additionally, extensional booleans admit a very slick presentation of dependent elimination using case splitting (If): since boolean constraints are kept in the context, the elimination motive $\mathrm{if}\ b\ \mathrm{then}\ A\ \mathrm{else}\ B$ can be canonically inferred out of the types A, B of the branches without loss of generality.

An NbE algorithm for $\Pi\mathbb{B}^{\mathbf{ext}}$ The goal of this work is to implement a decision procedure for judgemental equality and typing in $\Pi\mathbb{B}^{\mathrm{ext}}$. However, in that context, the standard techniques for deciding equality with dependent types based on iterated weak-head reduction are inadequate. In a simply-typed context already, only the equation on the left should hold:

$$f : \mathbb{B}{\to}\mathbb{B}, x : \mathbb{B} \vdash f\,x \equiv f\,(f\,(f\,x)) : \mathbb{B} \qquad\qquad f : \mathbb{B}{\to}\mathbb{B}, x : \mathbb{B} \vdash x \not\equiv f\,(f\,x) : \mathbb{B}$$

In particular, terms should be compared by exploring their full β-short η-long normal form.

Normalization-by-evaluation (NbE) [BS91] provides a suitable tool for that purpose: terms are evaluated into a domain representing weak β-normal form and then read back to a canonical representation in the original syntax. Following Abel [Abe13]'s NbE for dependent types, the reading back procedure η-expand elements of the domain according to their type. During that process, a key idea is that no boolean expressions can be left as neutral: a boolean expression should be constrained by the evaluation environment either to `true` or `false`. Similarly to [AS19], we utilize a monadic interpreter to keep track of boolean constraints and generate fresh ones. The monad $\mathcal{C}$ provides an effectful primitive $\mathtt{split} : \mathtt{ne} \to \mathcal{C}\,\mathbb{B}$ assigning a value to neutral booleans and a partial run $\mathtt{filter} : (\mathtt{ne} \to \mathbb{B}) \to \mathcal{C}\,A \to \mathcal{C}\,(\mathtt{BinTree}_{\mathtt{ne}}\,A)$ that filter an effectful context with a predicate on nodes $p : \mathtt{ne} \to \mathbb{B}$, returning a binary tree representing a sequence of case-splits. Under the hood, $\mathcal{C}$ is implemented using binary trees labelled with neutral booleans. This monadic abstraction takes care in particular of inconsistent branches and duplicate splittings by arbitrarily well-ordering the nodes, ensuring that every equivalence class of booleans expressions are constrained to at most one literal value in each branch. Ultimately, the splitting is only allowed at neutral booleans in order to to compare and order boolean expressions syntactically.

In the end, the OCaml implementation provides a function $\mathtt{norm} : \mathtt{ctx} \to \mathtt{tm} \to \mathcal{C}\,\mathtt{tm}$ used in particular to decide judgemental equality and procedures $\mathtt{infer} : \mathtt{ctx} \to \mathtt{tm} \to \mathcal{C}\,\mathtt{ty}$ and $\mathtt{check} : \mathtt{ctx} \to \mathtt{tm} \to \mathtt{ty} \to \mathcal{C}\,\mathbb{B}$ implementing a simple bidirectional typechecking algorithm [Coq96; Len21; GSB19]. The monadic values returned by these procedures can be understood as *coverings* of the input context. Indeed, Altenkirch et al. [Alt+01] observe that the category of contexts with boolean constraints can be equipped with a Grothendieck topology[2] for which the judgements are sheaves ($\mathbb{B}\eta$). For a type A equipped with a decidable equality, the elements of $\mathcal{C}\,A$ can be enforced to be extensional, in the sense that any splitting is non-redundant. In particular, a term t typechecks at type T in context Γ when $\mathtt{check}\,\Gamma\,t\,T$ is the trivial computation $\mathtt{ret}_{\mathcal{C}}\ \mathtt{true}$. We provide some examples along the OCaml implementation [Mai24].

[2] Informally, a family of context $(\Delta_i)_i$ cover Γ if each Δ_i only add boolean constraints, and they handle in common all the possible cases, e.g. the pair of contexts $(b : \mathbb{B}, b \equiv \mathtt{true}; b : \mathbb{B}, b \equiv \mathtt{false})$ cover the context $b : \mathbb{B}$.

Towards decidability The implementation gives reasonable hope that an effective method can be employed to decide judgemental equality and typecheck $\Pi\mathbb{B}^{\text{ext}}$terms. However, correctness of the implementation does rely on the existence of a normal form for every term. Following existing work in formalizing NbE-style arguments [HJP23], we are working on a PER-model to establish the soundness and completeness of our NbE procedure for $\Pi\mathbb{B}^{\text{ext}}$.

References

[Abe13] Andreas Abel. "Normalization by Evaluation: Dependent Types and Impredicativity". Habilitation thesis. Institut für Informatik, Ludwig-Maximilians-Universität München, 2013.

[Alt+01] Thorsten Altenkirch et al. "Normalization by Evaluation for Typed Lambda Calculus with Coproducts". In: *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*. IEEE Computer Society, 2001, pp. 303–310. ISBN: 0-7695-1281-X. DOI: 10.1109/LICS.2001.932506. URL: https://doi.org/10.1109/LICS.2001.932506.

[AS19] Andreas Abel and Christian Sattler. "Normalization by Evaluation for Call-By-Push-Value and Polarized Lambda Calculus". In: *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*. Ed. by Ekaterina Komendantskaya. ACM, 2019, 3:1–3:12. ISBN: 978-1-4503-7249-7. DOI: 10.1145/3354166.3354168. URL: https://doi.org/10.1145/3354166.3354168.

[AU04] Thorsten Altenkirch and Tarmo Uustalu. "Normalization by Evaluation for $\lambda^{\to 2}$". en. In: *Functional and Logic Programming*. Ed. by Gerhard Goos et al. Vol. 2998. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 260–275. ISBN: 978-3-540-24754-8. DOI: 10.1007/978-3-540-24754-8_19. URL: http://link.springer.com/10.1007/978-3-540-24754-8_19 (visited on 10/04/2023).

[BCF04] Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore. "Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums". In: *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*. Ed. by Neil D. Jones and Xavier Leroy. ACM, 2004, pp. 64–76. ISBN: 1-58113-729-X. DOI: 10.1145/964001.964007. URL: https://doi.org/10.1145/964001.964007.

[BS91] Ulrich Berger and Helmut Schwichtenberg. "An Inverse of the Evaluation Functional for Typed lambda-calculus". In: *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*. IEEE Computer Society, 1991, pp. 203–211. ISBN: 0-8186-2230-X. DOI: 10.1109/LICS.1991.151645. URL: https://doi.org/10.1109/LICS.1991.151645.

[CCD17] Simon Castellan, Pierre Clairambault, and Peter Dybjer. "Undecidability of Equality in the Free Locally Cartesian Closed Category (Extended version)". In: *Log. Methods Comput. Sci.* 13.4 (2017). DOI: 10.23638/LMCS-13(4:22)2017. URL: https://doi.org/10.23638/LMCS-13(4:22)2017.

[Coq96] Thierry Coquand. "An Algorithm for Type-Checking Dependent Types". In: *Sci. Comput. Program.* 26.1-3 (1996), pp. 167–177. DOI: 10.1016/0167-6423(95)00021-6. URL: https://doi.org/10.1016/0167-6423(95)00021-6.

[DS95] D.J. Dougherty and R. Subrahmanyam. "Equality between functionals in the presence of coproducts". In: *Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*. 1995, pp. 282–291. DOI: 10.1109/LICS.1995.523263.

[FS99] Marcelo Fiore and Alex Simpson. "Lambda Definability with Sums via Grothendieck Logical Relations". en. In: *Typed Lambda Calculi and Applications*. Ed. by Gerhard Goos et al. Vol. 1581. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 147–161. ISBN: 978-3-540-65763-7 978-3-540-48959-7. DOI: 10.1007/3-540-48959-2_12. URL: http://link.springer.com/10.1007/3-540-48959-2_12 (visited on 03/06/2024).

[Gha95] Neil Ghani. "ßn-Equality for Coproducts". In: *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95, Edinburgh, UK, April 10-12, 1995, Proceedings*. Ed. by Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin. Vol. 902. Lecture Notes in Computer Science. Springer, 1995, pp. 171–185. ISBN: 3-540-59048-X. DOI: 10.1007/BFB0014052. URL: https://doi.org/10.1007/BFb0014052.

[GSB19] Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. "Implementing a modal dependent type theory". In: *Proc. ACM Program. Lang.* 3.ICFP (2019), 107:1–107:29. DOI: 10.1145/3341711. URL: https://doi.org/10.1145/3341711.

[HJP23] Jason Z. S. Hu, Junyoung Jang, and Brigitte Pientka. "Normalization by evaluation for modal dependent type theory". In: *Journal of Functional Programming* 33 (2023). DOI: 10.1017/S0956796823000060.

[Len21] Meven Lennon-Bertrand. "Complete Bidirectional Typing for the Calculus of Inductive Constructions". In: *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*. Ed. by Liron Cohen and Cezary Kaliszyk. Vol. 193. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 24:1–24:19. ISBN: 978-3-95977-188-7. DOI: 10.4230/LIPIcs.ITP.2021.24. URL: https://doi.org/10.4230/LIPIcs.ITP.2021.24.

[Lin07] Sam Lindley. "Extensional Rewriting with Sums". In: *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*. Ed. by Simona Ronchi Della Rocca. Vol. 4583. Lecture Notes in Computer Science. Springer, 2007, pp. 255–271. ISBN: 978-3-540-73227-3. DOI: 10.1007/978-3-540-73228-0_19. URL: https://doi.org/10.1007/978-3-540-73228-0%5C_19.

[Mai24] Kenji Maillard. *Splitting bools with NbE*. https://github.com/kyoDralliam/split-bool-nbe. 2024.

[McB09] Conor McBride. "Grins from my Ripley Cupboard". 2009. URL: http://strictlypositive.org/Ripley.pdf.

[Sch17] Gabriel Scherer. "Deciding equivalence with sums and the empty type". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. Ed. by Giuseppe Castagna and Andrew D. Gordon. ACM, 2017, pp. 374–386. ISBN: 978-1-4503-4660-3. DOI: 10.1145/3009837.3009901. URL: https://doi.org/10.1145/3009837.3009901.

Implementing Observational Equality with Normalisation by Evaluation

Matthew Sirman, Meven Lennon-Bertrand, and Neel Krishnaswami

University of Cambridge, UK

Abstract

We report on an experimental implementation of a type theory with an observational equality type, based on Pujet et al.'s CC^{obs}, extended with a form of inductive and quotient types. It features a normalisation by evaluation function, which is used to implement a bidirectional type checker. We also explore proof assistant features, notably the interaction of strict propositions with meta-variables, and a rudimentary "hole" system.

1 Observational Equality meets NbE

Observational equality In recent years, building on an early proposal by Altenkirch et al. [3, 4], Pujet et al. [10, 9, 11] have developed CC^{obs}, a dependent type theory featuring a new presentation of equality: *observational equality*. This equality is *definitionally proof-irrelevant*: any two proofs of equality are identified. Moreover, rather than being uniformly defined like the traditional inductive equality, observational equality has a specific behaviour at each type. Together, these aspects lead to an equality close to traditional mathematical one, with a seamless support of quotients, making it very attractive.

Normalisation by Evaluation Pujet et al.'s work come with an extensive meta-theoretic investigation, yet, they do not implement their type theory. We attack this unexplored aspect, by providing an experimental implementation, based on *normalisation by evaluation* (NbE) [1], a modern technique to decide *definitional equality*. To do so, the NbE approach efficiently computes normal forms by instrumenting the evaluation mechanism of the host language, and then compares these normal forms for a simple, structural notion of equality. In particular, abstractions/applications are handled by using functions of the host language.[1]

NbE for CC^{obs} In our implementation, we extend standard NbE techniques, as presented by *e.g.* Abel [1] and in Kovács' `elaboration-zoo` [7], to an extension of CC^{obs}. Our type theory features a sort of definitionally irrelevant propositions Ω [6], an observational equality valued in that sort, and quotient types. We also explore inductive types, as first-class construct equipped with a form of Mendler-style recursion [8]. We did not investigate the meta-theory of this presentation, but believe it would be an interesting avenue for future research.

Our Haskell code is available on GitHub [12]. Despite the standard NbE ideas required some care to adapt, they largely apply to CC^{obs}, witnessing their robustness.

2 Semantic propositions

Maybe the most important design decision in our implementation is the structure of the semantic domain D^{Ω} in which proofs are evaluated before being quoted back, in the standard NbE fashion. This should reflect the fact that we should never reduce such irrelevant terms.

[1] Technically, we depart from this by replacing functions with closures, but the philosophy still stands.

The simplest strategy is to erase irrelevant terms at evaluation, following Abel et al. [2], which amounts to set D^Ω to be a unit type. Unfortunately this means we cannot quote values, and have to present users with incomplete terms. This also has undesirable consequence when solving meta-variables.

A natural alternative is to keep a term during evaluation, *i.e.* to pick $\mathsf{D}^\Omega ::= \mathsf{Prop}\ t\ \rho$ – a term t and an environment ρ for its free variables. This is however still problematic, because terms, being represented with de Bruijn *indices*, do not support cheap lifting, a key operation on semantic values in NbE. Similarly, to be able to substitute a semantic value for a variable in such a proof term – an operation needed during evaluation –, we would have to quote the value, making evaluation and quoting mutually defined, which is unsatisfying.

We thus opt for a semantic domain, which uses closures to support substitution, and de Bruijn *levels* for free lifting. However, these values can represent term structure that would be normalised in relevant values, for instance $[\![(\lambda x.t)\ u]\!]^\Omega = \mathsf{PApp}(\mathsf{PLam}(\underline{\lambda} t)\rho)\ [\![u]\!]^\Omega\rho$, where $[\![\cdot]\!]^\Omega$ is evaluation of irrelevant terms, $\underline{\lambda}$ is a closure, and PApp and PLam are both constructors. Compare to relevant terms where the evaluation would give $[\![t]\!]^{\mathcal{U}}(\rho, [\![u]\!]^{\mathcal{U}}\rho)$.

To decide which evaluation function to use between $[\![\cdot]\!]^\Omega$ and $[\![\cdot]\!]^{\mathcal{U}}$, we need to sprinkle terms with relevance annotations: this is the case for binders, as done by Pujet et al. [10], but also for application nodes. This is not unsurprising, as with NbE we need to evaluate arguments of applications, while the reduction of Pujet et al. is call-by-name. Fortunately, all these annotations can be inferred during type-checking, so users do not need to put them in.

3 Cast-on-refl in term-directed NbE

An innovation of Pujet et al. [11] is the recovery of the definitional equality $\mathbf{cast}(A, A, e, t) \equiv t$ for any type A, which did not hold in earlier version of observational equality. The idea is to incorporate it not in reduction, but in conversion, i.e. when comparing two terms. We follow suit: our NbE implements β-normalisation only, and we implement cast-on-refl on a by-need basis when comparing semantic values, just like η-expansion. All in all, our algorithm is entirely term-directed.

4 Proof assistant features

We implement contextual meta-variables in the fashion of Matita [5] and Coq: a meta-variable has a context, and each of its use of comes with a substitution instantiating said context. This contrasts with the more standard use of lambda-lifting [7], proponents of which claim is simpler. Still, we found contextual meta-variables altogether not too complicated, and led to clearer code.

A very useful feature present in Agda is the ability to incrementally construct terms by filling holes, with the editor presenting information such as the expected type of a hole and the variables in scope. We implement a lightweight version of this mechanism: our syntax contains a primitive `?{t1, …, tn}`, which errors but reports to the user the expected type for this hole, and that of `t1` to `tn`, as exemplified by the following code.

```
let f : ℕ → ℕ =
  λx. S x
in
let x : ℕ =
  S (S (S 0))
in f ?{f, x}
```

```
[error]: Found proof goal.
    ┌─▶ <test-file>@6:6-6:13
  6 │     f ?{f, x}
    •       ┬─────
    •       └╴ Expected type [ℕ] at goal.
    •          List of relevant terms and their types:
    └──┘       f : ℕ → ℕ, x : ℕ
```

References

[1] Andreas Abel. "Normalization by Evaluation Dependent Types and Impredicativity". PhD thesis. München: Institut für Informatik Ludwig-Maximilians-Universität München, May 2013. URL: https://www.cse.chalmers.se/~abela/habil.pdf.

[2] Andreas Abel, Thierry Coquand, and Miguel Pagano. "A Modular Type-Checking Algorithm for Type Theory with Singleton Types and Proof Irrelevance". In: *Typed Lambda Calculi and Applications*. Ed. by Pierre-Louis Curien. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 5–19.

[3] Thorsten Altenkirch and Conor McBride. "Towards Observational Type Theory". 2006.

[4] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. "Observational Equality, Now!" In: *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification*. Freiburg Germany: ACM, Oct. 2007, pp. 57–68. ISBN: 978-1-59593-677-6. DOI: 10.1145/1292597.1292608. URL: https://dl.acm.org/doi/10.1145/1292597.1292608 (visited on 05/08/2023).

[5] Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. "User Interaction with the Matita Proof Assistant". In: *Journal of Automated Reasoning* 39 (2007), pp. 109–139.

[6] Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. "Definitional Proof-Irrelevance without K". In: *Proceedings of the ACM on Programming Languages*. POPL'19 3.POPL (Jan. 2019), pp. 1–28. DOI: 10.1145/329031610.1145/3290316. URL: https://hal.inria.fr/hal-01859964.

[7] András Kovács. *Elaboration-Zoo*. May 2023. URL: https://github.com/AndrasKovacs/elaboration-zoo (visited on 05/14/2023).

[8] Nax Paul Mendler. "Inductive Types and Type Constraints in the Second-Order Lambda Calculus". In: *Annals of Pure and Applied Logic* 51.1-2 (Mar. 1991), pp. 159–172. ISSN: 01680072. DOI: 10.1016/0168-0072(91)90069-X. URL: https://linkinghub.elsevier.com/retrieve/pii/016800729190069X (visited on 05/04/2023).

[9] Loïc Pujet. "Computing with Extensionality Principles in Dependent Type Theory". PhD thesis. Nantes Université, Dec. 2022.

[10] Loïc Pujet and Nicolas Tabareau. "Observational Equality: Now for Good". In: *Proceedings of the ACM on Programming Languages* 6.POPL (Jan. 2022), pp. 1–27. ISSN: 2475-1421. DOI: 10.1145/3498693. URL: https://dl.acm.org/doi/10.1145/3498693 (visited on 04/26/2023).

[11] Loïc Pujet and Nicolas Tabareau. "Impredicative Observational Equality". In: *Proc. ACM Program. Lang.* 7.POPL (Jan. 2023). DOI: 10.1145/3571739.

[12] Matthew Sirman. *observational-type-theory*. May 2023. URL: https://github.com/matthew-sirman/observational-type-theory (visited on 11/29/2023).

Towards a logical framework modulo rewriting and equational theories

Bruno Barras, Thiago Felicissimo, and Théo Winterhalter

Université Paris-Saclay, INRIA, Deducteam, Laboratoire de Méthodes Formelles, ENS Paris-Saclay

Dedukti [ABC+23, BDG+23] is a logical framework based on a dependent type theory (the $\lambda\Pi$-calculus) extended with user-defined rewrite rules. It can be used to encode a wide variety of systems (set theory, pure type systems, first and higher-order logic...). Yet, some features of proof systems remain out of reach. Among them we find universes that seem hard to encode with a confluent rewriting system, typically due to non-linear and non-terminating rules such as those pertaining to the maximum of universe levels: $\ell \vee \ell \equiv \ell$ or $\ell \vee \ell' \equiv \ell' \vee \ell$. In practice this causes problems when encoding the type theories of Agda [Nor09], Coq [Coq22] or Lean [dMU21]. Another important theory that is hard to encode using only rewrite rules is the relatively recent cubical type theory [CCHM16] in which type-checking requires deciding the inclusion of hypercube faces expressed in an algebraic structure related to de Morgan algebra.

To better support such theories, we argue for an extension of the $\lambda\Pi$-calculus in which one can augment the definitional equality not only with rewrite rules $l \longmapsto r \in \mathcal{R}$, but also with *equational theories* $\mathcal{E}$, which are sets of non-oriented equations $t \approx u$. This is thus closely related to the work of Strub *et al.* [Str10, BJSW11] on Coq modulo theory, and of Bauer *et al.* [BHP20] on Andromeda 2 with user-defined definitional equalities. In this talk we will identify potential steps to take towards that goal as well as pitfalls that should be avoided. This work is still in a very early stage and the goal would be to eventually reach a logical framework with user-provided decidable theories.

Towards $\lambda\Pi$ modulo equational theories. We parametrise $\lambda\Pi$ modulo with a signature of typed constants Σ, an equational theory $\mathcal{E}$, a first-order signature $\mathfrak{S}$, and set of rewrite rules $\mathcal{R}$. Syntax is given by the following grammar where $(c : A) \in \Sigma$ and $(f(\Delta) : B) \in \mathfrak{S}$ with $|\Delta| = n$.

$$A, B, t, u, v \quad ::= \quad x \mid \mathsf{TYPE} \mid \mathsf{KIND} \mid \Pi(x : A).B \mid \lambda(x : A).t \mid t\ u \mid c \mid f(t_1, \dots, t_n)$$

We write e for *first-order terms*, *i.e.*, terms made only of variables and first-order function symbols f, and suppose all equations in $\mathcal{E}$ to be of the form $e \approx e'$. Conversion $\equiv$ is defined as usual as the congruence closure of β, $\mathcal{E}$ and $\mathcal{R}$. We write $\simeq_{\mathcal{E}}$ for the congruence closure of $\mathcal{E}$.

An example, a problem. Consider the following example mixing booleans with an equational theory of lists, where true, false, negb are declared in Σ and $[\_]$, $\_{+}{+}\_$, $\mathsf{hd}(\_)$ in $\mathfrak{S}$.

$$\begin{array}{lcl} \mathcal{R} & := & \{\ \mathsf{negb\ true} \longrightarrow \mathsf{false};\quad \mathsf{negb\ false} \longrightarrow \mathsf{true}\ \} \\ \mathcal{E} & := & \{\ \mathsf{hd}([x]{+}{+}l) \approx x,\quad l_1{+}{+}(l_2{+}{+}l_3) \approx (l_1{+}{+}l_2){+}{+}l_3,\quad \dots\ \} \end{array}$$

Now consider the term $\mathsf{negb\ hd}(l)$ with $l = ([\mathsf{true}]{+}{+}[\mathsf{false}]){+}{+}[\mathsf{true}]$. To be able to trigger the rewrite rule for negb, one first needs to observe that $\mathsf{hd}(l) \simeq \mathsf{true}$ so that $\mathsf{negb\ hd}(l) \simeq \mathsf{negb\ true} \longrightarrow \mathsf{false}$. This illustrates the fact that rewriting modulo an equational theory $\mathcal{E}$ requires not only to decide $\mathcal{E}$ but also to match rewrite-rules left-hand sides *modulo* $\mathcal{E}$ (see [Con04] dealing with matching modulo associativity and commutativity). Note that this happens even though $\mathcal{R}$ and $\mathcal{E}$ have completely disjoint signatures. Moreover, it is interesting to note that even by removing the booleans, this example can still be reproduced by considering interactions of β with a theory of lists of functions. In order to address this problem, we propose solutions with varying restrictions, starting with the most restrictive.

Solution 1: Confinement. A radical approach is to completely forbid the interactions between $\mathcal{R}$ and $\mathcal{E}$ with the use of *confinement* [ADJL17, FB24]. This approach consists in syntactically isolating the first-order fragment over which $\mathcal{E}$ operates from the rest of the syntax. The global syntax is then allowed to refer to the confined first-order syntax but, crucially, not the other way around, rendering terms like $[\mathsf{true}]$ syntactically ill formed. Even if these restrictions rule out interesting examples like the one given above, many useful theories can still be encoded using confinement. This is the case of (predicative) universe polymorphism, for which the use of confinement has also allowed for its first confluent encoding in Dedukti [FB24]. Yet, one can wonder if a more permissive account of rewriting modulo is possible.

Solution 2: Collapsing rules. Our example shows that (in the non-confined case) a first-order context $e[x_1, x_2, x_3] = \mathsf{hd}(([x_1] +\!\!+ [x_2]) +\!\!+ [x_3])$ can be inserted between two constants negb and true, blocking the reduction as long as $\mathcal{E}$ is not applied to collapse $e[x_1, x_2, x_3]$ to x_1. If $\mathcal{R}$ is left-linear and does not mention any first-order symbols f, then only *collapsible* first-order terms can block a reduction, leading us to consider the rules $\mathcal{C} = \{e \longmapsto x \mid e \neq x \wedge e \simeq_{\mathcal{E}} x\}$. With $\mathcal{C}$, we can avoid the use of matching modulo in our example by first applying $\mathcal{C}$ to reduce $\mathsf{hd}(([\mathsf{true}] +\!\!+ [\mathsf{false}]) +\!\!+ [\mathsf{true}])$ to true, and then $\mathcal{R}$ to reduce $\mathsf{negb}\ \mathsf{true}$ to false. If $\mathcal{C}$ allows us to avoid matching modulo, its integration with $\mathcal{R}\beta$ can sometimes break desirable rewriting properties, and further theoretical study is needed in this direction.

Solution 3: Pattern symbols. The least restrictive solution we consider would let us use equational theory symbols also in *patterns* of rewrite rules. In the example above, we might for instance want to consider $[\_]$ to be a pattern, but intuitively, this should not be the case for hd. Formally, the problem with hd is that it's not injective: $\mathsf{hd}([x] +\!\!+ l) \equiv x \equiv \mathsf{hd}([x] +\!\!+ l')$.

Our goal is to generalise the collapse rule above to be able to uniquely determine the head pattern symbol of an expression. For this, we assume that we have a normalisation function $\Downarrow$ on terms built over $\mathfrak{S}$ such that variables are considered normal forms ($\Downarrow x = x$) and which respects $\mathcal{E}$: $\Downarrow u \simeq_{\mathcal{E}} u$. Pattern symbols will be injective symbols p such that they are normal forms: $\Downarrow p(e_1, \ldots, e_n) = p(\Downarrow e_1, \ldots, \Downarrow e_n)$. Note that we can probably weaken this property to have $\Downarrow$ only produce one pattern symbol without fully normalising the term. Crucially, when $u \simeq_{\mathcal{E}} p(u_1, \ldots, u_n)$ then we must have $\Downarrow u = p(v_1, \ldots, v_n)$ with $u_i \simeq_{\mathcal{E}} v_i$. This approach solves the problem at hand but it also lets us have rewrite rules that match on *e.g.*, $[\_]$.

Equality checking. We claim that in the above cases, equality checking can be performed in the usual style by weak-head reduction, comparing the heads, and proceeding recursively on the subterms. In our setting the weak-head reduction needs to include $\mathcal{C}$ or $\Downarrow$ in addition to the usual $\beta\mathcal{R}$. Also, the comparison of head normal forms needs to consider the extra cases related to first-order symbols. First, a term $f(t_1, \ldots, t_n)$ is never convertible to a non-first-order term, unless it is an instance of a collapsible term, which would contradict that we have normalized w.r.t. $\mathcal{C}$ or $\Downarrow$. Second, when both sides have a first-order symbol at the head, we need to split the terms to compare (t_1, t_2) as a combination of two first-order terms with a substitution (that is such that $t_1 = e_1\sigma$ and $t_2 = e_2\sigma$ for some first-order terms e_1, e_2 and some substitution σ). For the sake of completeness, and also when either side of the equations are not linear, the above decomposition needs to be generalized. But let us first show an example to illustrate this.

Consider the associative commutative theory of $\oplus$, merging sorted lists, the conversion problem $(u, v) = ([(\lambda x.x)\,\mathsf{true}] \oplus [c\,t] \oplus [\mathsf{true}],\ [\mathsf{true}] \oplus [\mathsf{true}] \oplus [(\lambda x.c\,x)\,t])$ results in the comparison of $u' = [Y] \oplus [Z] \oplus [Y]$ and $v' = [Y] \oplus [Y] \oplus [Z]$ (with substitution $\sigma = (Y \mapsto \mathsf{true},\ Z \mapsto c\,t)$) in the equational theory:

$$u \equiv_{\beta\mathcal{R}\mathcal{C}} [\mathsf{true}] \oplus [c\,t] \oplus [\mathsf{true}] = u'\sigma \simeq_{\mathcal{E}} v'\sigma = [\mathsf{true}] \oplus [\mathsf{true}] \oplus [c\,t] \equiv_{\beta\mathcal{R}\mathcal{C}} v$$

Actually we need to linearize the first-order terms to allow the different instances of the same variable. This leads to considering the substitution

$$\sigma = (Y_1 \mapsto (\lambda x.x)\,\mathsf{true},\ Y_2 \mapsto \mathsf{true},\ Z_1 \mapsto c\,t,\ Z_2 \mapsto (\lambda x.c\,x)\,t)$$

such that $u = ([Y_1] \oplus [Z_1] \oplus [Y_2])\sigma$ and $v = ([Y_2] \oplus [Y_2] \oplus [Z_2])\sigma$. Then we consider a renaming

$$\rho = (Y_1 \mapsto Y, Y_2 \mapsto Y, Z_1 \mapsto Z, Z_2 \mapsto Z)$$

that maps convertible subterms to the same variable ($Y_1\sigma \equiv Y_2\sigma$ and $Z_1\sigma \equiv Z_2\sigma$). Finally we check that $(Y_1 \oplus Z_1 \oplus Y_2)\rho = u' \simeq_{\mathcal{E}} v' = (Y_2 \oplus Y_2 \oplus Z_2)\rho$.

This optimization of the substitution may require N^2 conversions (with N the number of distinct subterms), but we expect this can be reduced to $N.\log N$ (using a total order on terms in normal form) or even linear in some interesting cases (*e.g.*, non commutative theories).

References

[ABC+23] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a logical framework based on the $\lambda\pi$-calculus modulo theory, 2023.

[ADJL17] Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, and Jiaxiang Liu. Untyped confluence in dependent type theories. 2017.

[BDG+23] Frédéric Blanqui, Gilles Dowek, Emilie Grienenberger, Gabriel Hondet, and François Thiré. A modular construction of type theories. *Logical Methods in Computer Science*, Volume 19, Issue 1, February 2023.

[BHP20] Andrej Bauer, Philipp G Haselwarter, and Anja Petković. Equality checking for general type theories in andromeda 2. In *International Congress on Mathematical Software*, pages 253–259. Springer, 2020.

[BJSW11] Bruno Barras, Jean-Pierre Jouannaud, Pierre-Yves Strub, and Qian Wang. CoqMTU: A higher-order type theory with a predicative hierarchy of universes parametrized by a decidable first-order theory. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science*, pages 143–151, 2011.

[CCHM16] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. *arXiv preprint arXiv:1611.02108*, 2016.

[Con04] Évelyne Contejean. A certified ac matching algorithm. In *International Conference on Rewriting Techniques and Applications*, 2004.

[Coq22] The Coq Development Team. The Coq Proof Assistant, September 2022.

[dMU21] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, pages 625–635, 2021.

[FB24] Thiago Felicissimo and Frédéric Blanqui. Sharing proofs with predicative theories through universe polymorphic elaboration, 2024. https://arxiv.org/abs/2308.15465.

[Nor09] Ulf Norell. Dependently typed programming in agda. In *Proceedings of TLDI'09: 2009 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, Savannah, GA, USA, January 24, 2009*, pages 1–2, 2009.

[Str10] Pierre-Yves Strub. Coq modulo theory. In *International Workshop on Computer Science Logic*, pages 529–543. Springer, 2010.

Session 16: Computability

Comodule Representations of Second-Order Functionals*

Danel Ahman[1] and Andrej Bauer[2,3]

[1] University of Tartu (Estonia)
[2] University of Ljubljana (Slovenia)
[3] Institute of Mathematics, Physics and Mechanics (Slovenia)

In information-theoretic terms, a map is *continuous* when a finite amount of information about the input suffices for computing a finite amount of information about the output. Already Brouwer [7] observed that this idea can be used to represent a continuous functional $f : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ with a certain *well-founded question-answer tree* t_F. The value $f\,\alpha$ can be computed by traversing a path in t_F according to α, until a leaf containing the result is encountered. Similarly, a continuous functional $F : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ may be represented by a sequence of such trees. Other variations of tree-based representations of functionals have subsequently been studied [11, 12, 10, 8, 9], of which a recent account [4] considers functionals[1] $F : \left(\prod_{a:A} P\,a\right) \to \left(\prod_{b:B} Q\,b\right)$.

A tree representation of such an F can be organised into the following diagram:

$$\begin{array}{ccc} \prod_{a:A} P\,a & \xrightarrow{\quad F \quad} & \prod_{b:B} Q\,b \\ & {\scriptstyle \mathsf{c}_{A,P}} \searrow \quad \nearrow {\scriptstyle \lambda\alpha b.\, \mathsf{e}_F\, b\, (\alpha(\mathsf{t}_F\, b))} & \\ & \prod_{t:\mathsf{Tree}(A,P)} \mathsf{Path}_{A,P}(t) & \end{array} \tag{1}$$

Here, $\mathsf{Tree}(A, P)$ is the type of *well-founded A-labelled P-branching trees* (with unlabelled leaves) and $\mathsf{Path}_{A,P}(t)$ the type of *paths* in t from the root to leaves. The map $\mathsf{c}_{A,P}$ takes $h : \prod_{a:A} Pa$ and a tree t, and computes a path in t by choosing branches according to h. The map $\mathsf{t}_F : B \to \mathsf{Tree}(A, P)$ returns a tree representation of F (with unlabelled leaves) at each $b : B$, while $\mathsf{e}_F : \prod_{b:B} \mathsf{Path}_{A,P}(\mathsf{t}_F\, b) \to Q\, b$ labels the paths (equivalently, leaves) in $\mathsf{t}_F\, b$ with values of F.

The data in (1) may be recast in the language of containers [1]. A *container* $A \lhd P$ comprises a type of *shapes* A and a family of *positions* $P : A \to \mathbf{Type}$. A *container morphism* $f \lhd g : A \lhd P \to B \lhd Q$ is given by a shape map $f : A \to B$ and a position map $g : \prod_{a:A} Q\,(f\,b) \to P\,a$. The assignment $T : A \lhd P \mapsto \mathsf{Tree}(A, P) \lhd (\lambda t.\, \mathsf{Path}_{A,P}(t))$ is the functor part of the *tree monad* on the *category of containers* **Cont**. In fact, the container $T(A \lhd P)$ is the initial algebra of the functor $B \lhd Q \mapsto (\mathbb{1} + \sum_{a:A} (P\,a \to B)) \lhd [\mathsf{inl}(\star) \mapsto \mathbb{1}, \mathsf{inr}(a, v) \mapsto \sum_{p:P\,a} Q\,(v\,p)]$, or more abstractly, of $B \lhd Q \mapsto \mathsf{Id}^{\mathsf{c}} +^{\mathsf{c}} (A \lhd P) \circ^{\mathsf{c}} (B \lhd Q)$. In (1), we may also discern a morphism $\mathsf{t}_F \lhd \mathsf{e}_F : B \lhd Q \to T(A \lhd P)$ in the *Kleisli category* $\mathbf{Cont}_T$ for the tree monad T, additionally overlaid with the (contravariant) functor[2] $\langle\!\langle - \rangle\!\rangle : \mathbf{Cont}^{\mathrm{op}} \to \mathbf{Type}$, given by $\langle\!\langle A \lhd P \rangle\!\rangle = \prod_{a:A} P\,a$.[3]

To explain the presence of the map $\mathsf{c}_{A,P}$ in (1), and thereby complete the category-theoretic picture, we need one more ingredient. Given a monad T on $\mathcal{C}$, define a *right T-comodule* (F, c) in $\mathcal{D}$ to be given by a (contravariant) functor $F : \mathcal{C}^{\mathrm{op}} \to \mathcal{D}$ together with a natural transformation $\mathsf{c} : F \to F \circ T$, called the *comodule structure map*, satisfying $F\eta \circ \mathsf{c} = \mathsf{id}$ and $\mathsf{c}_T \circ \mathsf{c} = F\mu \circ \mathsf{c}$. The map $\mathsf{c}_{A,P}$ in (1) is precisely such a comodule structure map, for $\langle\!\langle - \rangle\!\rangle : \mathbf{Cont}^{\mathrm{op}} \to \mathbf{Type}$.

We have thus managed to recast tree-based representations of functionals in purely category-theoretic terms that generalises to any monad T and comodule structure on the functor $\langle\!\langle - \rangle\!\rangle$.

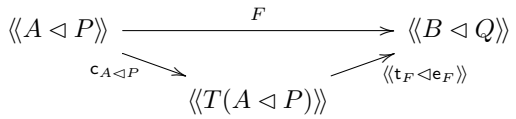
*This work has received support from the COST Action EuroProofNet (CA20111). This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-21-1-0024.

[1] Most sources study continuity of functionals $F : \left(\prod_{a:A} Pa\right) \to Q$, but generalising Q to a product of a type family is easy and results in composable functionals.

[2] We construe **Type** as a category with types as objects and functions as morphisms, assuming sufficient extensionality principles. Alternatively, the reader may interpret our work in a suitable model of type theory.

[3] Note that $\langle\!\langle A \lhd P \rangle\!\rangle$ is isomorphic to the cointerpretation [3] of $A \lhd P$ at the terminal object $\mathbb{1}$.

Definition 1. Let T be a monad on **Cont** and $\mathsf{c} : \langle\!\langle - \rangle\!\rangle \to \langle\!\langle T(-) \rangle\!\rangle$ a right T-comodule structure map on $\langle\!\langle - \rangle\!\rangle$. A *comodule representation* of a functional $F : \left(\prod_{a:A} P\,a\right) \to \left(\prod_{b:B} Q\,b\right)$, with respect to c, is given by a container morphism $\mathsf{t}_F \lhd \mathsf{e}_F : B \lhd Q \to T(A \lhd P)$, such that



A functional which has such a representation is said to be *comodule representable*.

The monad and comodule laws together guarantee that representations compose as morphisms in $\mathbf{Cont}_T$, and the passage from the representations to functionals is functorial. Further investigation of the category-theoretic structure is needed. Here we satisfy ourselves with showing that comodule representations arise in many interesting situations in addition to continuity.

Finite support. Say that $F : \left(\prod_{a:A} P\,a\right) \to \left(\prod_{b:B} Q\,b\right)$ has *finite support* when for every $b : B$ there is a finite subtype $A_{F,b} \subseteq A$ such that $F\,h\,b$ can be computed already from the restriction $h{\restriction}_{A_{F,b}} : \prod_{a':A_{F,b}} Pa'$. We can capture this with the monad $T(A \lhd P) = \mathcal{P}_f A \lhd \lambda A'.\, \prod_{a':A'} Pa'$, where $\mathcal{P}_f -$ is the finite powerset monad. The comodule $\mathsf{c}_{A \lhd P}$ is the restriction ${\restriction}$, $\mathsf{t}_F\,b$ computes an appropriate finite subtype, and e_F the values of F from a finitely supported map.

Instance reductions. The previous example hints at a general phenomenon: a monad M on **Type** for which **Type** itself is an M-algebra[4] induces a monad T_M on **Cont**, defined as $T_M(A \lhd P) = M(A) \lhd P^\dagger$. Sometimes P needs to be restricted so that an M-algebra can be obtained. One such example arises when we take *propositional containers* $A \lhd\!\!| P$, where $P : A \to \mathbf{Prop}$, and let M be the inhabited powerset monad, with $P^\dagger = \lambda A'.\, \exists a' {:} A'.\, Pa'$. The comodule representable functionals $F : \left(\forall a {:} A.\, P\,a\right) \to \left(\forall b {:} B.\, Q\,b\right)$ are precisely the *instance reductions*, as studied in reverse constructive mathematics [5] and related to Weihrauch reducibility [6].

Functional functionals. Even the identity monad with the trivial comodule structure map is mildly interesting, as it yields *functional functionals* F that (functionally) reduce each codomain instance $b : B$ to a single domain instance $a : A$, so that $F\,h\,b$ queries h on only one such a. When restricted to propositional containers, these are the *functional instance reductions*.

Exceptional functionals. Alternatively, we can choose M to be the exception monad $MA = \mathbb{1} + A$, with $P^\dagger = [\mathsf{inl}(\star) \mapsto \mathbb{1}, \mathsf{inr}(a) \mapsto P\,a]$ and $\mathsf{c}_{A \lhd P}\,h = [\mathsf{inl}(\star) \mapsto \star, \mathsf{inr}(a) \mapsto h\,a]$. Now $F\,h\,b$ either queries h on one $a : A$ (like in the previous example) or gives a default answer.

Interactive functionals. Further variations on computational effects are possible, with some also indicating at natural generalisations of the notion of comodule representation. Take as M the input-output monad $\mathsf{IO}(A)$. Here we endow **Type** with an IO-algebra based on IO-*traces*. We can then talk about comodule representation of functionals relative to an IO-*runner* [13, 14, 2] modelling the environment of the functionals. This way we capture functionals that compute values by interacting with the environment through input and output operations. However, using the functor $\langle\!\langle - \rangle\!\rangle$, as before, would only capture representations that reset their environments when composed, and thus would not satisfy the comodule (co)associativity law. In order to properly propagate environment changes across representations (and thus the functionals) in composition, we need to rethink Definition 1 and recast it in terms of the functor $\langle\!\langle A \lhd P \rangle\!\rangle_R = \prod_{a:A} (R \to P\,a \times R)$, where R carries an IO-runner structure. Further investigation of this and analogous generalisations is warranted.

[4] A suitable way to formulate such structure is to endow **Type** with a Mendler-style M-algebra structure [15], given as a family of maps $(-)^\dagger_A : (A \to \mathbf{Type}) \to (MA \to \mathbf{Type})$ compatible with the monad structure.

References

[1] Michael Gordon Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing strictly positive types. *Theoretical Computer Science*, 342(1):3–27, 2005.

[2] Danel Ahman and Andrej Bauer. Runners in action. In Peter Müller, editor, *Programming Languages and Systems – 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings*, volume 12075 of *Lecture Notes in Computer Science*, pages 29–55. Springer, 2020.

[3] Danel Ahman and Tarmo Uustalu. Update monads: Cointerpreting directed containers. volume 26 of *LIPIcs*, pages 1–23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

[4] Martin Baillon. *Continuity in Type Theory.* PhD thesis, Nantes Université, December 2023.

[5] Andrej Bauer. Instance reducibility and Weihrauch degrees. *Logical Methods in Computer Science*, 18(3), 2022.

[6] Vasco Brattka and Guido Gherardi. Weihrauch degrees, omniscience principles and weak computability. *The Journal of Symbolic Logic*, 76(1):143–176, 2011.

[7] L. E. J. Brouwer. Über Definitionsbereiche von Funktionen. *Mathematische Annalen*, 97(1):60–75, 1927.

[8] Martín Escardó. Continuity of Gödel's System T Definable Functionals via Effectful Forcing. *Electronic Notes in Theoretical Computer Science*, 298:119–141, 2013. Proceedings of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics, MFPS XXIX.

[9] Neil Ghani, Peter G. Hancock, and Dirk Pattinson. Continuous Functions on Final Coalgebras. volume 249 of *Electronic Notes in Theoretical Computer Science*, pages 3–18. Elsevier, 2009.

[10] Martin Hofmann, Aleksandr Karbyshev, and Helmut Seidl. What Is a Pure Functional? In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 199–210. Springer, 2010.

[11] Stephen C. Kleene. Recursive Functionals and Quantifiers of Finite Types I. *Transactions of the American Mathematical Society*, 91(1):1–52, 1959.

[12] John Longley. The sequentially realizable functionals. *Annals of Pure and Applied Logic*, 117(1):1–93, 2002.

[13] Gordon Plotkin and John Power. Tensors of comodels and models for operational semantics. In *Proceedings of 24th Conference on Mathematical Foundations of Programming Semantics, MFPS XXIV*, volume 218 of *ENTCS*, pages 295–311. Elsevier, 2008.

[14] Tarmo Uustalu. Stateful runners of effectful computations. *Electronic Notes in Theoretical Computer Science*, 319:403–421, 2015.

[15] Tarmo Uustalu and Varmo Vene. Mendler-style inductive types, categorically. *Nordic Journal of Computing*, 6(3):343, 1999.

Oracle modalities

Andrew W Swan

University of Ljubljana
wakelin.swan@gmail.com

Already in [Hyl82, Section 17] Hyland showed an interesting connection between topos theory and computability theory: the Turing degrees embed into the lattice of subtoposes of the effective topos. We can update this idea to work in HoTT through lex modalities [RSS20] and cubical assemblies [Uem19, SU21]. For this work, we can think of lex modalities as generalisations of subtoposes to higher dimensions, so as models of type theory that we can define and work with from within another, larger model, characterised by a reflection operation that finds a "closest approximation" in the submodel of an object in the larger one. In particular we can consider ∇, the modality of double negation sheaves [RSS20, Example 3.41] as a subuniverse of assemblies where we can carry out classical logic and access all external functions in sets including non computable ones. Functions $\mathbb{N} \to \mathbb{N}$ in the reflective subuniverse for ∇ can be viewed as functions $\mathbb{N} \to \nabla\mathbb{N}$ in the original universe. We define the *oracle modality* $\bigcirc_\chi$ to be the smallest modality that forces $\chi : \mathbb{N} \to \nabla\mathbb{N}$ to be a total function $\mathbb{N} \to \mathbb{N}$. We can think of functions $\mathbb{N} \to \bigcirc_\chi\mathbb{N}$ as functions that can be computed using an oracle Turing machine with oracle χ. This can be made precise using cubical assemblies, where every function $\mathbb{N} \to \mathbb{N}$ in sets appears as a function $\mathbb{N} \to \nabla\mathbb{N}$ in cubical assemblies, and two functions χ, χ' have the same Turing degree if and only if the modalities $\bigcirc_\chi$ and $\bigcirc_{\chi'}$ are equal in cubical assemblies.

The aim here is to use HoTT as a "unifying" approach that enables us to combine ideas from a number of different areas.

Generalisations of Turing reducibility Kihara [Kih23] has emphasised the potential of Lawvere-Tierney topologies in the effective topos as generalisations of Turing degrees. We can push this idea further by incorporating the higher dimensional aspects of cubical assemblies, and also consider non-topological modalities (i.e. modalities that are not sheafification for a Lawvere-Tierney topology). I'll talk about one of the simplest example of such modalities, to illustrate the potential of this approach. Given any modality $\bigcirc$, we can construct the suspension modality $\bigcirc^=$ such that a type is $\bigcirc^=$-modal precisely when it is $\bigcirc$-separated [CORS20]. Although the construction was originally motivated by homotopy theory, it is also natural from a computational point of view, since it gives us fine grained control over the hLevel where new information is introduced. E.g. In $\bigcirc_\chi X$ we can use an oracle to construct new points of X, and also to construct new paths, new homotopies between paths, etc. On the other hand, in $\bigcirc^=_\chi X$, we cannot use the oracle to construct new points, but only new paths, homotopies, etc. In particular $\bigcirc^=_\chi\mathbb{Z} = \mathbb{Z}$, since $\mathbb{Z}$ is a discrete type that only has points,[1] but the first homotopy group of $\bigcirc^=_\chi\mathbb{S}^1$ is $\bigcirc_\chi\mathbb{Z}$, since we can use the oracle to construct paths in $\mathbb{S}^1$, which then appear as elements of the homotopy group. By iterating the suspension, we can get variants of n-dimensional spheres where some homotopy groups are computable (and non trivial) and some are non computable.

Synthetic computability Synthetic computability [Ric83, Bau06, Bau17, For21] is an approach to computability where instead of working directly with explicit descriptions of objects, one works with simpler, more natural constructions in constructive mathematics, which can

[1] More precisely we use the fact that $\mathbb{Z}$ is a $\neg\neg$-separated set.

then be interpreted in realizability models, to recover the original versions. Recently, several synthetic definitions of Turing reducibility have been developed [Bau20, FKM23]. The approach here is equivalent to one of the earliest synthetic definitions of Turing reducibility: already in the conclusion to [Bau06], Bauer points out that using Hyland's result, one can talk about Turing reductions via Lawvere-Tierney topologies in the effective topos.

HoTT style arguments in synthetic computability One of the most elegant developments in HoTT is a new approach to group theory first studied by Buchholtz, Van Doorn and Rijke [BvDR18] where groups are formulated as pointed connected 1-truncated types. We can recover the traditional definition of group from a pointed type (A, a) using the *loop space* $\Omega(A, a) := \mathrm{Id}_A(a, a)$, where the group multiplication is defined using the transitivity of identity. As Buchholtz et al point out, this approach to group theory has the advantage of generalising easily to higher dimension. It can also help with formalisation, since group multiplication is already implicitly part of a type, rather than extra algebraic structure that we need to carry around, and some constructions in group theory are more natural from this point of view, compared to more traditional approaches to group theory.

I'll give an example application of this approach to a simple result in computability theory: if two Turing degrees induce isomorphic permutation groups on $\mathbb{N}$ then they are equal. To do this, the first important point is that we can encode any function $\chi : \mathbb{N} \to 2$ as a permutation of $\mathbb{N}$. We first view χ as an element of $\prod_{n:\mathbb{N}} \mathbb{Z}/2\mathbb{Z}$. This group is precisely the loop space of $\lambda x.2$ in $\mathcal{U}^{\mathbb{N}}$. We have a map $\mathcal{U}^{\mathbb{N}} \to \mathcal{U}$ that sends $X : \mathbb{N} \to \mathcal{U}$ to $\sum_{n:\mathbb{N}} Xn$. Applying action on paths then induces an inclusion from $\Omega(\mathcal{U}^{\mathbb{N}}, \lambda x.2)$ to $\Omega(\mathcal{U}, \mathbb{N})$. The second key point is that in the proof of the theorem we make use of the fact that the inclusion factors through the wreath product of permutations on $\mathbb{N}$ and 2. We can again formulate this elegantly in HoTT. The map $\mathcal{U}^{\mathbb{N}} \to \mathcal{U}$ factors through $\sum_{A:\mathcal{U}} \mathcal{U}^A$ by sending $X : \mathbb{N} \to \mathcal{U}$ to $(\mathbb{N}, X)$, and sending (A, X) to $\sum_{a:A} Xa$. We note that the loop space on $(\mathbb{N}, \lambda x.2)$ is the wreath product $S_2 \wr S_{\mathbb{N}}$. These observations are used together with some technical lemmas to prove the theorem.

A synthetic approach to the study of Lawvere-Tierney topologies in realizability Aside from working in HoTT via cubical assemblies, our approach to topological modalities differs from earlier work by Hyland [Hyl82], Lee and Van Oosten [LvO13] and Kihara [Kih23] in carrying out more of the construction in the internal logic, as opposed to working externally, using explicit descriptions of the objects of the topos. In particular, this required a strengthening of Markov's principle that we call *Markov induction*, which is equivalent to the generalised Markov's principle formulated by Hofmann, Van Oosten and Streicher in [HvOS06]. Markov induction implies not only that Markov's principle holds, but also that it holds in the reflective subuniverses for the modalities that we consider. To see why Markov induction seems necessary here, note that in order to apply Markov's principle we require a computable function $\mathbb{N} \to 2$ as antecedent. However, a binary sequence in the reflective subuniverse is a map $\mathbb{N} \to \bigcirc_\chi 2$. There is no apparent way to convert this into a computable function $\mathbb{N} \to 2$ in general, since e.g. we have no upper bound on the total oracle queries used before the sequence reaches 1.

The fact that Markov's principle holds in reflective subuniverses was used in turn for various other results, including the theorem above that Turing degrees are equal when they induce isomorphic permutation groups. Since more of the arguments here take place internally they are easier to formalise electronically compared to earlier work on Lawvere-Tierney topologies in realizability such as [LvO13] and [Kih23]. Most of the definitions and results mentioned in this abstract have been formalised[2] using the cubical mode [VMA19] of Agda.

[2]The formalisation is available at `https://github.com/awswan/oraclemodality`.

References

[Bau06] Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006. Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI).

[Bau17] Andrej Bauer. On fixed-point theorems in synthetic computability. *Tbilisi Mathematical Journal*, 10(3):167 – 181, 2017.

[Bau20] Andrej Bauer. Synthetic mathematics with an excursion into computability theory (slide set), 2020. University of Wisconsin Logic seminar.

[BvDR18] Ulrik Buchholtz, Floris van Doorn, and Egbert Rijke. Higher groups in homotopy type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 205–214, New York, NY, USA, 2018. Association for Computing Machinery.

[CORS20] J. Daniel Christensen, Morgan Opie, Egbert Rijke, and Luis Scoccola. Localization in homotopy type theory. *Higher Structures*, 4:1–32, 2020.

[FKM23] Yannick Forster, Dominik Kirst, and Niklas Mück. Oracle computability and turing reducibility in the calculus of inductive constructions. In Chung-Kil Hur, editor, *Programming Languages and Systems*, pages 155–181, Singapore, 2023. Springer Nature Singapore.

[For21] Yannick Forster. *Computability in Constructive Type Theory*. PhD thesis, Saarland University, 2021.

[HvOS06] M. Hofmann, J. van Oosten, and T. Streicher. Well-foundedness in realizability. *Archive for Mathematical Logic*, 45(7):795–805, Oct 2006.

[Hyl82] J. M. E. Hyland. The effective topos. In A. S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium Proceedings of the Conference held in Noordwijkerhout*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 165 – 216. Elsevier, 1982.

[Kih23] Takayuki Kihara. Lawvere-Tierney topologies for computability theorists. *Transactions of the American Mathematical Society. Series B*, 10:48–85, 2023.

[LvO13] Sori Lee and Jaap van Oosten. Basic subtoposes of the effective topos. *Annals of Pure and Applied Logic*, 164(9):866 – 883, 2013.

[Ric83] Fred Richman. Church's thesis without tears. *Journal of Symbolic Logic*, 48(3):797–803, 1983.

[RSS20] Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, Volume 16, Issue 1, January 2020.

[SU21] Andrew W. Swan and Taichi Uemura. On Church's thesis in cubical assemblies. *Mathematical Structures in Computer Science*, 31(10):1185–1204, 2021.

[Uem19] Taichi Uemura. Cubical Assemblies, a Univalent and Impredicative Universe and a Failure of Propositional Resizing. In Peter Dybjer, José Espírito Santo, and Luís Pinto, editors, *24th International Conference on Types for Proofs and Programs (TYPES 2018)*, volume 130 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[VMA19] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: A dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019.

"Proofs *are* programs" in MLTT

Pierre-Marie Pédrot

INRIA

Martin-Löf type theory truly is the paradise of constructive mathematics, as it is indeed both a logical foundation *and* a programming language. Yet the previous sentence does not do justice to the profound essence of MLTT: it is not just some fancy logical reasoning strapped onto a run-of-the-mill programming language. In MLTT, computation and logic are literally identified into a single monistic view, and the choice to consider an object as logical or computational can be framed as an opinion. It is thus a truism that "proofs *are* programs" in MLTT, a self-evident truth which we will dub the *credo* of Church's church for reasons that will become clear soon. This identification holds by construction and does not require e.g. any realizability model.

With this in mind, it should come as a surprise that MLTT enjoys non-computational models, such as the **Set** model where MLTT functions are interpreted as ZFC functions. The reason for this discrepancy is that, in MLTT, the proof-as-program identification is an external fact not reflected in the theory itself. Thankfully, there is a well-known solution to bridge this gap: the internal Church Thesis [8]. In higher-order arithmetic, this principle can be stated as

$$\mathsf{CT} : \forall(f : \mathbb{N} \to \mathbb{N}).\, \exists(p : \mathbb{N}).\, \forall(n : \mathbb{N}).\, \exists(k : \mathbb{N}).\, \mathsf{T}\ p\ n\ k\ (f\ n)$$

where T is the decidable Kleene predicate. Namely, $\mathsf{T}\ p\ n\ k\ v$ holds whever p is the code of some Turing machine, and running p on the input n terminates in less than k steps and returns the value v. Said otherwise, CT guarantees that any internal function $f : \mathbb{N} \to \mathbb{N}$ is reflected by an actual algorithm $p : \mathbb{N}$ from within the logic, i.e. is extensionally a program.

The CT principle was heavily used by the Russian constructivist school [6], and is known to be consistent with higher-order arithmetic. The typical way to prove this is via Kleene realizability, where proofs are interpreted as concrete codes. One has to be wary that CT is quite an oddball, though. Assuming enough choice, it contradicts both weak forms of excluded middle like LLPO *and* function extensionality. This is not a problem for MLTT, which does not validate either.

After having accumulated that much evidence, the reader could rightfully assume that the compatibility of MLTT with CT is a classic, if not folklore result. As a matter of fact, MLTT+CT is the foundation for synthetic computability [3], another offshoot of the synthetic trend that trivializes the annoying parts of computability proofs by working directly and implicitly with computable functions. Surely one does not add axioms lightly when it comes to developping a sizable formalized library. So, MLTT + CT ought to be known to be consistent. Right?

Interestingly, the answer to this question so far was: *it depends*[1]. The problem boils down to the precise definition of CT in our theory. Many type theories feature several kind of existential types, typically contrasting actual existence $\Sigma x : A.\, B$ with propositional existence $\exists x : A.\, B$. Since the arithmetic statement of CT features an existential quantification, there are as many ways to interpret CT as there are existential types[2], i.e. we have two principles

$$\begin{aligned} \mathsf{CT}_{\Sigma} &: \Pi(f : \mathbb{N} \to \mathbb{N}).\, \Sigma(p : \mathbb{N}).\, \Pi(n : \mathbb{N}).\, \Sigma(k : \mathbb{N}).\, \mathsf{T}\ p\ n\ k\ (f\ n) \\ \mathsf{CT}_{\exists} &: \Pi(f : \mathbb{N} \to \mathbb{N}).\, \exists(p : \mathbb{N}).\, \Pi(n : \mathbb{N}).\, \Sigma(k : \mathbb{N}).\, \mathsf{T}\ p\ n\ k\ (f\ n) \end{aligned}$$

[1] One could not have expected less from a dependent type theory.

[2] The translation choice for the second existential quantification does not matter in most settings.

which are in general not equivalent. As $\exists$ is usually intended to be uncomputational and thus does not satisfy choice, $\mathsf{CT}_\exists$ is the nicest of the two, i.e. it does not contradict classical logic nor function extensionality. As a result, it was showed to be consistent not only with MLTT, but also with univalence [7]. By contrast, as Σ-types validate non-choice automatically, CT_Σ is the portal to an algorithmic hell featuring a quote function $\mathfrak{q} : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$.

At the risk of repeating ourselves, "proofs *are* programs" in MLTT, even if only externally so. It should thus be easy to extend MLTT with CT_Σ. At this point the waters become extremely murky. The few published results on the topic [5, 4] are only able to prove the consistency of CT_Σ with a crippled subset of MLTT deprived of the ξ rule, i.e. congruence for λ-abstractions, which prevents conversion to proceed under binders. Furthermore, they hint that handling full-blown MLTT is a hard problem.

This was an unbearable situation for us. First, we believe that the ξ rule is a critical feature of MLTT. Second, MLTT + CT_Σ is obviously consistent, because remember that "proofs *are* programs". So, it was a categorical imperative to actually prove it, and not merely on paper, as nobody trusts paper proofs about type theory. Therefore, as the only reasonable path forward, we formalized in Coq the consistency of "MLTT", an extension of MLTT that proves CT_Σ. This settles the question for good.

In a nutshell, "MLTT" is a variant of MLTT with one universe, negative Π and Σ types with definitional η-rules, additionally featuring empty, identity and natural number types. It features three additional quotation primitives

$$\frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N}}{\Gamma \vdash \mathfrak{q}\ M : \mathbb{N}} \qquad \frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N} \qquad \Gamma \vdash N : \mathbb{N}}{\Gamma \vdash \mathfrak{r}\ M\ N : \mathbb{N}} \qquad \frac{\Gamma \vdash M : \mathbb{N} \to \mathbb{N} \qquad \Gamma \vdash N : \mathbb{N}}{\Gamma \vdash \varrho\ M\ N : M\ \mathfrak{q}\ N}$$

where $M\ \mathfrak{q}\ N := \mathsf{T}\ (\mathfrak{q}\ M)\ N\ (\mathfrak{r}\ M\ N)\ (M\ N)$. In other words, these three operations implement the skolemization of CT_Σ. In particular, "MLTT" proves CT_Σ trivially, hence

""proofs *are* programs" in "MLTT"".

For brevity, we will not present in detail the computational rules of these operations, but give the general idea. Basically, these operations will only compute on closed deep normal forms. For instance, the conversion rule for $\mathfrak{q}$ is given as

$$\mathfrak{q}\ M \equiv \lceil M \rceil \quad \text{when } M \text{ closed deep normal form}$$

where $\lceil \cdot \rceil : \mathtt{Term} \to \mathbb{N}$ is a quotation function in the metatheory, which together with T defines a *computational model* for "MLTT".

Under mild hypotheses on this model, it is possible to show that not only "MLTT" is consistent, but is also strongly normalizing and enjoys canonicity. We prove these facts through essentially the same logical relation used by Abel et al. to prove decidability of conversion [1]. The main difference is that we annotate our semantic proofs of conversions with the fact that not only they are normalizing for head reduction, but also for iterated head (i.e. deep) reduction. This addition is virtually transparent and did not require any non-trivial change to the relation. The only additional material needed for the proof is a fair amount of rewriting theory for the untyped fragment of "MLTT", including confluence and standardization. Moreover, some care has to be taken to properly handle the definitional η-rules of negative types, which adds some unwanted technicity. The Coq formalization is based on the `logrel-coq` project [2] and can be found at https://github.com/ppedrot/quote-mltt. Although we did not prove decidability of type-checking for "MLTT", this should be an easy byproduct of this development.

It appears that the solution to the internalization of CT in MLTT is conceptually trivial: simply restrict computation to closed normal forms. While this seems to go against the type-theoretical ethos, it turns out that this plays well with the usual expectations on MLTT such as canonicity and strong normalization. As a result, we believe that this cheap trick can go a long way to internalize externally derivable rules in MLTT. We leave the study of the class of axioms that can be implemented this way to future work.

References

[1] A. Abel, J. Öhman, and A. Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL), Dec. 2017.

[2] A. Adjedj, M. Lennon-Bertrand, K. Maillard, P.-M. Pédrot, and L. Pujet. Martin-Löf à la Coq. In S. Blazy, B. Pientka, A. Timany, and D. Traytel, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*. ACM, 2024.

[3] Y. Forster. *Computability in constructive type theory.* PhD thesis, Saarland University, Germany, 2021.

[4] H. Ishihara, M. E. Maietti, S. Maschio, and T. Streicher. Consistency of the intensional level of the minimalist foundation with Church's thesis and axiom of choice. *Arch. Math. Log.*, 57(7-8):873–888, 2018.

[5] M. E. Maietti. A minimalist two-level foundation for constructive mathematics. *Ann. Pure Appl. Log.*, 160(3):319–354, 2009.

[6] M. Margenstern. L'école constructive de Markov. *Revue d'histoire des mathématiques*, 1995.

[7] A. W. Swan and T. Uemura. On Church's thesis in cubical assemblies. *Math. Struct. Comput. Sci.*, 31(10):1185–1204, 2021.

[8] A. Troelstra and D. Dalen. *Constructivism in Mathematics: An Introduction.* Number vol. 1 in Constructivism in Mathematics. North-Holland, 1988.

Session 18: Proofs

OnlineProver: A proof assistant for online teaching of formal logic and semantics*

Joachim Tilsted Kristensen[1], Ján Perháč[2], Lars Tveito[1], Lars-Bo Husted Vadgaard[1], Michael Kirkedal Thomsen[1,3], Oleks Shturmov[1,3], Samuel Novotný[2], Sergej Chodarev[2], and William Steingartner[2]

[1] Department of Informatics, University of Oslo, Oslo, Norway
{joachkr,larstvei,larsvad,michakt,olekss}@ifi.uio.no
[2] Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Košice, Slovakia
{jan.perhac,samuel.novotny,sergej.chodarev,william.steingartner}@tuke.sk
[3] Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

1 Introduction

CS students often struggle when presented with exercises in formal CS courses that require mathematical rigour. Research [10], as well as personal experience [11], suggests that the teaching approaches that instills confidence in CS students differ from the approaches that work well with math students (analogous results have for physics students [13]). CS students have usually been exposed to several programming courses before encountering a course dedicated to theoretical computer science. In programming classes students may have developed a confidentiality with the trial-and-error approach [15]. We hypothesise, that supported by the right tool, a flavour of trial-and-error will increase student learning among CS students when exercising mathematical rigorousness. Following tradition in CS education [3], we set out to design such a tool, that encourages reflection but also provides immediate feedback on trial-and-error.

Although "traditional" proof assistants such as Coq, or Agda use programming trial-and-error style, students tend to be confused when the formalism in the exercise looks different than the book, and it takes a significant amount of time to teach the underlying language(s), sufficiently well that the students can start working on formal proofs. On the other hand, we know from introduction to programming, that students like the ease of block-based frameworks (like Scratch), but that they find them less authentic [15].

The main goal of our project[1] is to develop a proof assistant as a teaching tool OnlineProver[2]. It simulates the teaching environment by a visualization of an exercise class with paper proofs, while providing automated (but latent) feedback to the students, without the need of learning additional language. We do not tie into a specific framework; instead OnlineProver can be used in a wide range of introductory theoretical computer science teaching settings. We will discuss the details of this in Section 2.

Several other tools have been developed with a focus on a specific framework and implements trial-and-error approach, that gives students less room for reflection. Such tools are dedicated

* Iceland Liechtenstein Norway grants Norway grants
The authors thanks EEA and Norway Grants for support of this work under initiative no. FBR-PDI-025. "*Working together for a green, competitive and inclusive Europe.*" https://www.eeagrants.sk/

[1] OnlineProver webpage: https://onlineprover.github.io/
[2] OnlineProver tool: http://onlineprover.com/

to various frameworks e.g. proof trees for Hoare logic [7], natural deduction [1,9,12], or sequent calculus [4] in the Gentzen style [6], Tableau [2, 14], and λ-calculus [5]. references. However, these are designed to be limiting for the students and can exhibit the same problems as block-based programming frameworks. Closer to our approach is the work by [8], that in a more general and flexible framework implements support for Fitch style proofs.

2 The OnlineProver Tool Design

The OnlineProver system consists of three parts: A pair of domain-specific programming languages for specifying languages and exercises, a web-editor for interacting with the exercises as programs, and a web-service that serves and checks programs according to the language definition provided by the teacher.

An initial version of the tool has been developed for simple Gentzen style derivations. In this version, students are provided with a list of derivations, and must provide a derivation as an answer to the exercise.

As an example, the teacher provides a document, formally describing the syntax and semantics in sections as exemplified below.

```
# syntax **Syntax for terms and values**

t : Term
  | `(` t0 `,` t1 `)`
  | ...

v : Value
  | ...

# judgement **Evaluation** [ gamma `|-` t `->` v ]

gamma |- t1 ~> v1
gamma |- t2 ~> v2
---------------------------------------(pair-rule)
gamma |- (t1, t2) ~> (v1, v2)
```

Together with an exercise description program[3], this compiles into a text/tree format that can be served and manipulated by the web-service and user-interface, and solutions (trees) are completed when they are closed and unify with the handout.

Developing a theorem prover specifically for teaching, means that the error messages can be guided by suggested solutions from the teacher's program. We plan to conduct empirical studies on, for instance, the effect of different variations of feedback on the number of steps it takes a student to complete the proof. Another benefit, is that the students do not have to spend weeks on learning a functional programming language, before doing proofs as programs.

The user interface is language/semantics agnostic, and does not tie into a specific logical framework; instead OnlineProver can be configured by changing the user-interface's representation of the trees.

The formal-language description language acts as a lightweight Lex/Yacc for specifying small toy programming languages in which the teacher, and the students will be able to focus on those toy languages without dealing with the overhead of figuring out how to use a fully fledged theorem prover like Coq or Agda.

[3] Exercise example program left out in the interest of saving space.

References

[1] Krysia Broda, Jiefei Ma, Gabrielle Sinnadurai, and Alexander Summers. Pandora: A reasoning toolbox using natural deduction style. *Logic Journal of the IGPL*, 15(4):293–304, 2007.

[2] Rafael del Vado Vírseda, Eva Pilar Orna, Eduardo Berbis, and Saúl de León Guerrero. A logic teaching tool based on tableaux for verification and debugging of algorithms. In Patrick Blackburn, Hans van Ditmarsch, María Manzano, and Fernando Soler-Toscano, editors, *Tools for Teaching Logic*, pages 239–248. Springer Berlin Heidelberg, 2011.

[3] Stephen H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '04, page 26–30. Association for Computing Machinery, 2004.

[4] Arno Ehle, Norbert Hundeshagen, and Martin Lange. The sequent calculus trainer with automated reasoning - helping students to find proofs. In Pedro Quaresma and Walther Neuper, editors, Proceedings 6th International Workshop on *Theorem proving components for Educational software,* Gothenburg, Sweden, 6 Aug 2017, volume 267 of *Electronic Proceedings in Theoretical Computer Science*, pages 19–37. Open Publishing Association, 2018.

[5] Mario Frank and Christoph Kreitz. A theorem prover for scientific and educational purposes. In Pedro Quaresma and Walther Neuper, editors, Proceedings 6th International Workshop on *Theorem proving components for Educational software,* Gothenburg, Sweden, 6 Aug 2017, volume 267 of *Electronic Proceedings in Theoretical Computer Science*, pages 59–69. Open Publishing Association, 2018.

[6] Olivier Gasquet, François Schwarzentruber, and Martin Strecker. Panda: A proof assistant in natural deduction for all. a gentzen style proof assistant for undergraduate students. In Patrick Blackburn, Hans van Ditmarsch, María Manzano, and Fernando Soler-Toscano, editors, *Tools for Teaching Logic*, pages 85–92. Springer Berlin Heidelberg, 2011.

[7] Joomy Korkut. A proof tree builder for sequent calculus and hoare logic. In Pedro Quaresma, João Marcos, and Walther Neuper, editors, Proceedings 11th International Workshop on *Theorem Proving Components for Educational Software,* Haifa, Israel, 11 August 2022, volume 375 of *Electronic Proceedings in Theoretical Computer Science*, pages 54–62. Open Publishing Association, 2023.

[8] Graham Leach-Krouse. Carnap: An open framework for formal reasoning in the browser. In Pedro Quaresma and Walther Neuper, editors, Proceedings 6th International Workshop on *Theorem proving components for Educational software,* Gothenburg, Sweden, 6 Aug 2017, volume 267 of *Electronic Proceedings in Theoretical Computer Science*, pages 70–88. Open Publishing Association, 2018.

[9] Benjamín Machín and Luis Sierra. Yoda: a simple tool for natural deduction. In *Proceedings of the Third International Congress on Tools for Teaching Logic (TICTTL'11)*, volume 6680, 2011.

[10] Carroll Morgan. (in-)formal methods: The lost art. In Zhiming Liu and Zili Zhang, editors, *Engineering Trustworthy Software Systems*, pages 1–79, Cham, 2016. Springer International Publishing.

[11] John Newsome Crossley. What is mathematical logic? an australian odyssey. *Logic Journal of the IGPL*, 31(6):1010–1022, 2023.

[12] Jeremy Seligman and Declan Thompson. Teaching natural deduction in the right order with natural deduction planner, 2015.

[13] Bruce L. Sherin. A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning*, 6(1):1–61, 2001.

[14] Davi Romero Vasconcelos. Anita: Analytic tableau proof assistant. In Pedro Quaresma, João Marcos, and Walther Neuper, editors, Proceedings 11th International Workshop on *Theorem Proving Components for Educational Software,* Haifa, Israel, 11 August 2022, volume 375 of *Electronic*

Proceedings in Theoretical Computer Science, pages 38–53. Open Publishing Association, 2023.

[15] David Weintrop and Uri Wilensky. To block or not to block, that is the question: students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, page 199–208, New York, NY, USA, 2015. Association for Computing Machinery.

Proof and Consequences: Separating Construction and Checking of eBPF Loops

Mikkel Kragh Mathiesen and Ken Friis Larsen

Department of Computer Science, University of Copenhagen, Denmark
{mkm,kflarsen}@di.ku.dk

1 Introduction

The Linux kernel contains a subsystem called eBPF that allows safe execution of untrusted user-defined functions given as an assembly-level byte-code inside the kernel. Before the eBPF functions run they are analysed using static analysis by the in-kernel *verifier*. The verifier guards the integrity of the kernel by disallowing erroneous or malicious behaviour such as out-of-bounds memory access and non-termination. Thus, the verifier is not complete (but is hopefully sound). Termination is ensured by disallowing all back jumps (including recursion), thus eliminating all loops and recursion. Instead the kernel provides access to a so-called *helper function* named `bpf_loop` (amongst others), that can best be described as a higher-order function for performing loops with an upper bound.

The verifier is only informally described, and in some cases details are only given in a implementation-near manner or not at all. Thus it is hard to know which rules you have to obey as a programmer; the situation is even worse if you are writing a code-generator for eBPF. Gershuni et al. [1] presents an alternative formulation and implementation of an eBPF verifier. They describe that their analysis can handle loops, but cannot handle helper functions like `bpf_loop`.

We show how a type-system based on weakest pre-conditions can be used to faithfully model the in-kernel verifier. Post-conditions/abstract interpretation together with SMT solving enables automatic inference, while still permitting manual proofs when it falls short. In our presentation we will describe a number of examples demonstrating typical use of loops in eBPF that you have to use `bpf_loop` for, how we model the static semantics of eBPF as a type system and detail our implementation, demonstrating that realistic programs can be checked with our system.

Our work is a confluence of ideas, borrowing some ideas from Gershuni et al. [1], the idea of a typed assembly language from TAL [3, 4], and of weakest pre-conditions in Floyd–Hoare logic [2].

2 The Type System

Our approach separates eBPF verification into two parts: a type system and strategies for inferring types. The type system requires that each instruction has been assigned a *pre-condition* describing what must be true about the program state before executing that instruction. Type rules stipulate a logical relationship between an instruction's pre-conditions and the pre-conditions of its continuations, including safety constraints such as bounds checking for memory access. Given a program annotated with pre-conditions together with certificates for every logical implication, a simple type checker can verify the safety of the program.

Some amount of automatic inference is required for this system to be practical. If eBPF code is generated from a safe high-level language, the compiler would presumably be responsible for

$$
\begin{array}{rll}
r, s & ::= \texttt{r0} \mid \ldots \mid \texttt{r9} \mid \texttt{fp} & \text{Registers} \\
L & ::= r_d \mid e_0[e_1 : e_2] & \text{Locations} \\
e & ::= x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \cdot e_2 \mid \ldots & \text{Expressions} \\
P, Q & ::= \top \mid \bot \mid \neg P \mid P \wedge Q \mid P \vee Q & \text{Propositions} \\
& \quad \mid \exists x.P \mid \forall x.P \mid L \mapsto e \mid e_1 = e_2 \mid e_1 \leq e_2 &
\end{array}
$$

Figure 1: Syntax for conditions

this (perhaps based on annotations in the high-level language). For the time being, we focus on inferring types of eBPF programs directly as a litmus test for the viability of our approach.

The in-kernel verifier as well as proposed alternatives are generally based on symbolic execution or abstract interpretation. This approach is fully automatic and works well for programs that happen to conform to the verifier's expectations, but is quite sensitive to even small code alterations (e.g., as the result of compiler optimisations). Weakest pre-conditions, on the other hand, can verify any program (subject to limitations of the underlying logic) but are liable to produce huge verification conditions.

Our system combines the two approaches. Abstract interpretation is used to establish easily inferable information at each instruction. Weakest pre-conditions are then propagated, while being simplified at each step according to the inferred information. In practice the final conditions are small and strong, and often close to what a programmer would write. This means that the logical implications can generally be established directy by an SMT solver.

When the abstract interpretation fails to reason usefully about the program the consequence is that the verification conditions become more complicated, requiring more effort by the prover. In this way the system degrades gracefully and gradually.

3 Example

Appendix A shows an eBPF function that computes the sum of a byte array. Demonstrating how to use `bpf_loop` to traverse an array and compute a value. Similar to Gershuni et al. [1] we have illustrative examples that uses `bpf_loop` to copy, compare, initialize the content of memory regions, compute checksums and so on.

4 Formal System

The language of conditions is based on first-order arithmetic. Its syntax can be seen in Figure 1. The state of the program can be observed through the containment relation, $L \mapsto e$, which stipulates that a location (a register or a piece of memory) contains a specific value. Memory is modelled via an abstract type of regions.

A point in the program execution is identified by an instruction label ℓ and a stack σ to keep track of (nested) loops being executed. Annotating a program means constructing a function ϕ which assigns a condition to each point (ℓ, σ). The main typing judgement is of the form $\phi \vdash \ell; \sigma : I : \ell'; \sigma'$ which is read as: given an annotation function ϕ and an instruction I with label ℓ before and label ℓ' after, running I with stack σ is safe and results in stack σ'. See appendix B for example rules from our formalisation of the system.

References

[1] Elazar Gershuni, Nadav Amit, Arie Gurfinkel, Nina Narodytska, Jorge A. Navas, Noam Rinetzky, Leonid Ryzhyk, and Mooly Sagiv. Simple and precise static analysis of untrusted linux kernel extensions. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 1069–1084, New York, NY, USA, 2019. Association for Computing Machinery.

[2] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, Oct. 1969.

[3] Greg Morrisett, Karl Crary, Neal Glew, and David Walker. Stack-based typed assembly language. *Journal of Functional Programming*, 12(1):43–88, January 2002.

[4] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From System F to typed assembly language. In *25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 85–97, San Diego, CA, USA, January 1998. ACM Press.

A Example: Sum of byte array

```
// r1 points to a byte array, r2 is the number of bytes
r6 := fp      // fp points to the top of the stack frame
r6 += -16     // make room for two 64-bit words
r0 := 0
r6[0] := r0   // first word is the sum
r6[8] := r1   // second word is the array pointer
r1 := r2      // set number of iterations to the number of bytes
r3 := r6      // set the loop context to the stack frame
call bpf_loop @body
r0 := r6[0]  // set the program return value to the computed sum
exit

@body
// r1 is the current iteration, r2 contains the loop context
r3 := r2[0]  // load partial sum
r4 := r2[8]  // load byte pointer
r4 += r1
r5b := r4[0] // load current byte
r3 += r5
r2[0] := r3  // store partial sum
r0 := 0      // 0 means continue (rather than break)
exit
```

B Type system rules

$$\frac{\phi(\ell,\sigma) \vdash (\exists n.s_d \mapsto n \wedge \phi(\ell',\sigma)[n/r_d]) \vee (\exists m,i,\rho:[m].s_d \mapsto \rho@i \wedge \phi(\ell',\sigma)[\rho@i/r_d])}{\phi \vdash \ell;\sigma : r_d \texttt{:=} s_d : \ell';\sigma} \quad \text{(move-reg64)}$$

$$\frac{\phi(\ell,\cdot) \vdash \texttt{r0}_d \mapsto 0 \vee \texttt{r0}_d \mapsto 1}{\phi \vdash \ell;\cdot : \texttt{exit} : \ell';\sigma'} \quad \text{(exit-final)} \qquad \frac{\phi(\ell,(\sigma,\ell_0)) \vdash \phi(\ell_0,\sigma) \wedge (\texttt{r0}_d \mapsto 0 \vee \texttt{r1}_d \mapsto 1)}{\phi \vdash \ell;(\sigma,\ell_0) : \texttt{exit} : \ell';\sigma'} \quad \text{(exit-loop)}$$

Session 19: HoTT and Sets

Constructive Ordinal Exponentiation in Homotopy Type Theory

Tom de Jong[1], Nicolai Kraus[1], Fredrik Nordvall Forsberg[2], and Chuangjie Xu[3]

[1] University of Nottingham, Nottingham, UK
{tom.dejong, nicolai.kraus}@nottingham.ac.uk
[2] University of Strathclyde, Glasgow, UK
fredrik.nordvall-forsberg@strath.ac.uk
[3] SonarSource GmbH, Bochum, Germany
chuangjie.xu@sonarsource.com

Constructive ordinals Ordinals are a powerful tool for establishing consistency of logical theories, proving termination of processes and justifying induction and recursion. Constructively, there are many different approaches to ordinals, such as ordinal notation systems [6], or Brouwer trees [4], or as wellfounded trees with finite or countable branchings [5, 1]. The homotopy type theory book follows the classical idea of considering ordinals as order types of well ordered sets, and defines ordinals as types equipped with an order relation that is transitive, extensional (elements with the same predecessors are equal) and wellfounded (the order admits the principle of transfinite induction) [9, §10.3]. Notably, the univalence axiom is used to exhibit the type of (small) ordinals as a (large) ordinal; specifically, it is used to show that the relation on ordinals given by bounded simulations is extensional. This gives rise to a fascinating theory of ordinals, often skirting the edges of what is constructively achievable. With this in mind, is it possible to develop a constructive theory of arithmetic for such ordinals, with operations of addition, multiplication and exponentiation which extend the usual arithmetic for the natural numbers?

Ordinal exponentiation via case distinction Addition and multiplication can be realised by disjoint union and Cartesian product of the underlying types of the ordinals, respectively. Their basic properties were investigated by Escardó [3].

The case of exponentiation is constructively more challenging. From the classical theory of ordinals, we know what the specification should be: for zero and successors, exponentiation should be repeated multiplication, and it should be continuous as soon as the base is non-zero:

$$\begin{aligned} \alpha^0 &= 1 \\ \alpha^{\beta+1} &= \alpha^\beta \times \alpha \\ \alpha^{\sup_{i:I} f(i)} &= \sup_{i:I} \alpha^{f(i)} \quad (\text{for } \alpha \neq 0,\ I \text{ inhabited}) \\ 0^\beta &= 0 \qquad\qquad\qquad\qquad (\text{for } \beta \neq 0) \end{aligned} \tag{†}$$

Using classical logic, this is already a definition of exponentiation, but not so in a constructive setting, where the ability to make definitions by case distinctions on arbitrary ordinals is not available. In fact, we can show:

Theorem 1. *There is an operation α^β satisfying the specification (†) for all ordinals α and β if and only if excluded middle holds.*

In fact, excluded middle follows as soon as there is an exponentiation operator which is monotone in the exponent, and satisfies the first two equations of the specification (†). There is thus no hope of defining ordinal exponentiation constructively for arbitrary ordinals.

Ordinal exponentiation as functions with finite support However, we could still hope to define exponentiation for restricted classes of ordinals. For $\alpha > 0$ (which is equivalent to α having a least element $\bot$), Sierpiński [8, §XIV.15] gives an explicit construction of the exponential α^β as the collection of functions $\beta \to \alpha$ with *finite support*, i.e., functions $f : \beta \to \alpha$ such that $f(x) > \bot$ for only finitely many x. While this definition works well classically, the order relation it induces does not seem to be well-behaved constructively. The usual classical argument that the exponential is an ordinal requires decidability of the order on α, and decidable equality on α seems to be required to verify the expected properties (such as the specification (†)) of this ordinal. In general, neither of these assumptions are constructively justified.

Constructive exponentiation for ordinals with a detachable least element Let α be an ordinal of the form $\alpha = 1 + \gamma$ for some ordinal γ. That is, let α be an ordinal with a least element which is detachable — we can decide if a given element is the least one or not. For such α, we are able to define the exponential α^β constructively, by considering a "combinatorial" variant of Sierpiński's construction.

Since β is an ordinal, we can think of a finitely supported function from β to γ as a finite list of output-input[1] pairs $[(c_0, b_0), (c_1, b_1), \dots, (c_n, b_n)] : \mathsf{List}\,(\gamma \times \beta)$ which is strictly decreasing in the second argument (to enforce that each input has a unique output), with all inputs not occurring in the list being sent to the least element. Write

$$\mathsf{D_2List}(\gamma, \beta) :\equiv \big(\Sigma\, \ell : \mathsf{List}\,(\gamma \times \beta)\big)\ \mathsf{is\text{-}decreasing}\ (\mathsf{map}\,\pi_2\,\ell)$$

for the type of such lists of pairs decreasing in the second component. The idea of the combinatorial presentation using lists is similar to—but more general than—Setzer's sketch [7, App. A] of the construction of exponentials with base ω.

Theorem 2. *The type* $\mathsf{D_2List}(\gamma, \beta)$ *is an ordinal, when ordered lexicographically. Moreover, it satisfies the specification* (†) *for* $\alpha = 1 + \gamma$.

Proof sketch. Because the list is decreasing, the lexicographic order is wellfounded.

For verifying the specification (†), note that the only list with elements from $\gamma \times \mathbf{0}$ is the empty list, so $\mathsf{D_2List}(\gamma, \mathbf{0}) = \mathbf{1}$. For checking that $\mathsf{D_2List}(\gamma, \beta + \mathbf{1}) = \mathsf{D_2List}(\gamma, \beta) \times (1 + \gamma)$, note that a list ℓ in $\mathsf{D_2List}(\gamma, \beta + \mathbf{1})$ contains at most one head of the form $(c_0, \mathsf{inr}\,\star)$, followed by a list ℓ_1 in $\mathsf{D_2List}(\gamma, \beta)$. If the head is of the form $(c_0, \mathsf{inr}\,\star)$, the list ℓ corresponds to the pair $(\ell_1, \mathsf{inr}\,c_0)$, and otherwise it corresponds to the pair $(\ell, \mathsf{inl}\,\star)$. For the supremum case, it is crucial that the lists are decreasing in the second component; see the Agda code for details. □

Alternative constructions of exponentials In work in progress, we are investigating alternative definitions of exponentials. In particular, building on a suggestion by David Wärn, we consider the following definition by transfinite recursion:

$$\alpha^\beta :\equiv \sup_{\mathbf{1}+\beta} \begin{cases} \mathsf{inl}\,\star \mapsto \mathbf{1} \\ \mathsf{inr}\,b \mapsto \alpha^{\beta\downarrow b} \times \alpha, \end{cases}$$

where $\beta \downarrow b$ is the initial segment of β consisting of elements strictly smaller than b. This is motivated by the specification (†) and the observation that every ordinal γ is the supremum of the successors of its initial segments, i.e., $\gamma = \sup_{c:\gamma}((\gamma \downarrow c) + \mathbf{1})$. This operation indeed satisfies the specification (†) for $\alpha \geq 1$. Relating this construction to the one above is ongoing work.

[1] We prefer to use output-input pairs rather than input-output pairs so that their order corresponds to the usual order on the product $\gamma \times \beta$, which is reverse lexicographic.

Formalisation We have formalised our results in Agda, building on Escardó's TypeTopology development [2]. We have found Agda extremely valuable in developing our proofs as the intensional nature of our construction makes for rather combinatorial arguments. The source code can be found at https://github.com/fredrikNordvallForsberg/TypeTopology/blob/exponentiation/source/Ordinals/Exponentiation/.

References

[1] Thierry Coquand, Henri Lombardi, and Stefan Neuwirth. Constructive theory of ordinals. In Marco Benini, Olaf Beyersdorff, Michael Rathjen, and Peter Schuster, editors, *Mathematics for Computation*, pages 287–318. World Scientific, 2023.

[2] Martín H. Escardó and contributors. TypeTopology. https://github.com/martinescardo/TypeTopology. Agda development.

[3] Martín Hötzel Escardó et al. Ordinals in univalent type theory in Agda notation. Agda development, HTML rendering available at: https://www.cs.bham.ac.uk/~mhe/TypeTopology/Ordinals.index.html, 2018.

[4] Nicolai Kraus, Fredrik Nordvall Forsberg, and Chuangjie Xu. Type-theoretic approaches to ordinals. *Theoretical Computer Science*, 957, 2023.

[5] Per Martin-Löf. *Notes on constructive mathematics*. Almqvist & Wiksell, 1970.

[6] Fredrik Nordvall Forsberg, Chuangjie Xu, and Neil Ghani. Three equivalent ordinal notation systems in cubical Agda. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020*, pages 172–185. ACM, 2020.

[7] Anton Setzer. Proof theory of Martin-Löf type theory. An overview. *Mathématiques et sciences humaines*, 165:59–99, 2004.

[8] Wacław Sierpiński. *Cardinal and Ordinal Numbers*, volume 34 of *Monografie Matematyczne*. Państwowe Wydawnictwo Naukowe, 1958.

[9] Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

Extensional Finite Sets and Multisets in Type Theory *

Clemens Kupke, Fredrik Nordvall Forsberg, and Sean Watters

University of Strathclyde, UK
{clemens.kupke, fredrik.nordvall-forsberg, sean.watters}@strath.ac.uk

The type of lists is one of the most elementary inductive data types. It has been studied and used extensively by computer scientists and programmers for decades. Two conceptually similar structures are those of finite sets and multisets, which can be thought of as unordered analogues to lists. However, capturing unordered structures in a data type while maintaining desirable properties such as decidable equality and the correct equational theory is challenging.

The usual approach to formalise unordered structures in mathematics is to represent them as functions (with finite support): finite sets as $X \to 2$, and finite multisets as $X \to \mathbb{N}$, respectively. However, these representations do not enjoy decidable equality, even if the underlying type X does.

Meanwhile the approach taken in most programming languages is to pretend — one uses a list (or another ordered structure for efficiency) internally, but hides it and any invariants behind a layer of abstraction provided by an API. However, each set or multiset can then be represented by many different lists, meaning that the equational theory might not be correct. This is a problem in a dependently typed setting, where having equality as a first-class type allows us to distinguish between different representations of the same set.

In the setting of homotopy type theory (HoTT) [14], we can use higher inductive types (HITs) to define the identities on an inductive type simultaneously with its elements. This allows us to define a data type which enjoys both decidable equality and the right equational theory, as demonstrated by Choudhury and Fiore [3]. However, many proof assistants today do not support HITs; thus, the main question we set out to answer in this work is whether it is possible in ordinary dependent type theory to define data types of finite sets and multisets, which:

(i) have decidable equality iff the underlying set has decidable equality; and

(ii) satisfy the equational theories of finite sets and multisets.

For property (ii), we take as our success criteria the facts that the type of finite sets is the free idempotent commutative monoid [7], and that finite multisets are the free commutative monoid. Thus, we are really aiming to find data types for the free idempotent commutative monoid and free commutative monoid, which satisfy the above property (i). We accomplish this by restricting our attention to only those sets with decidable equality that can be totally ordered. We can then form a type of sorted lists over such a set. Provided we treat the existence of the ordering data carefully, this type turns out to give us exactly finite sets when the order is strict, and finite multisets when it is non-strict.

We show that our constructions satisfy universal properties, in the sense that they are left adjoints to forgetful functors — this is the standard way to state freeness in the language of category theory. However, note that the notion of freeness is with respect to e.g. totally ordered monoids, rather than all monoids. For proving the universal properties and for defining the categories involved, we need function extensionality. However we stress that the constructions themselves work in ordinary dependent type theory, without function extensionality.

*This is an extended abstract of a paper that was published at APLAS 2023 [10]. All results are formalised in Agda and are available at: https://www.seanwatters.uk/agda/fresh-lists/.

Fresh Lists Fresh lists, the key inductive data type of this work, were first introduced by C. Coquand to represent contexts in the simply typed lambda calculus [4]. The type of fresh lists is a parameterised data type similar to the type of ordinary lists, with the additional requirement that in order to adjoin a new element x to a list xs, that element x must be "fresh" with respect to all other elements already present in the list xs. We follow the Agda standard library [1] in considering a generalised notion of freshness, given by an arbitrary binary relation on the carrier set. We can recover Coquand's original notion of freshness by choosing inequality as our freshness relation.

Finite Sets as Sorted Lists Our candidate representation for finite sets satisfying the above properties (i) and (ii) is the type of sorted lists without duplicates. We obtain this by the appropriate instantiation of the type of fresh lists; namely, $\mathsf{FList}(A, <)$ for some type $A : \mathsf{Set}$ and a strict total order $< : A \to A \to \mathsf{Prop}$. We then prove an extensionality principle analogous to set extensionality which allows us to show that $\mathsf{FList}(A, <)$ is an idempotent commutative monoid with the empty list as the unit, and the operation which merges two sorted lists as the multiplication.

To establish (ii), we would like to show that this type is the *free* idempotent commutative monoid. However, there is a wrinkle — the domain of the sorted list functor cannot be simply the category of sets Set, since we require that the underlying set is equipped with a strict total order in order to form the type of sorted lists. Assuming that any set can be equipped with such an order is a strongly classical axiom called the Ordering Principle which is strictly weaker than the well-ordering principle [8, Ch. 5 §5], but still implies LEM [13]. Therefore to remain constructive, we must restrict the domain of the functor to strictly totally ordered sets. Thus, we define the categories STO of strictly totally ordered sets, and OICMon of *ordered* idempotent commutative monoids (ordering data is also required for the monoids so that it can be preserved by the forgetful functor; this is satisfied for $\mathsf{FList}(A, <)$ via the lexicographic ordering). With the categories in place, we can prove that the type of sorted lists is functorial, and left adjoint to the forgetful functor $\mathcal{U} : \mathsf{OICMon} \to \mathsf{STO}$, giving us the desired universal property.

Other Free Algebraic Structures The choice to implement sorted lists as an instantiation of the type of fresh lists reveals further paths to explore; what happens for other instantiations of the freshness relation? It turns out that different choices each yield a different free structure.

In particular, it should come as no surprise that finite multisets are represented by sorted lists with duplicates (i.e., fresh lists over a total order $\leqslant$). The proof of the adjunction is very similar to the previous case, however we obtain a different extensionality principle: since the membership relation for multisets is valued in Set rather than Prop, we must prove an isomorphism rather than merely a bi-implication. Other such results are summarised in Table 1.

Freshness Relation	Free Algebraic Structure	Data Structure
$\leqslant$, a total order	Ordered Commutative Monoid	Sorted lists
$<$, a strict total order	Ordered Idempotent Comm. Monoid	Sorted lists w/o duplicates
$\lambda x.\lambda y.\bot$	Pointed Set	Maybe
$\lambda x.\lambda y.\top$	Monoid	List
$\neq$	Left-Regular Band Monoid	Lists without duplicates
$=$	Reflexive Partial Monoid	$1 + (A \times \mathbb{N}^{>0})$

Table 1: Free algebraic structures as instantiations of freshlists (carrier set A)

Related Work Appel and Leroy [2] recently introduced canonical binary tries as an extensional representation of finite *maps*. These can be used to construct finite sets with elements from the index type. Krebbers [9] extended this technique to form extensional finite maps over arbitrary countable sets of keys.

The technique of using underlying ordering data to construct extensional data structures is not new, and has been employed in a number of Coq libraries for many years [5][6][12][11].

References

[1] The Agda Community. Agda standard library, 2023. https://github.com/agda/agda-stdlib.

[2] Andrew W. Appel and Xavier Leroy. Efficient extensional binary tries. *Journal of Automated Reasoning*, 67(1):8, 2023.

[3] Vikraman Choudhury and Marcelo Fiore. Free commutative monoids in Homotopy Type Theory. In Justin Hsu and Christine Tasson, editors, *Mathematical Foundations of Programming Semantics (MFPS '22)*, volume 1 of *Electronic Notes in Theoretical Informatics and Computer Science*, 2023.

[4] Catarina Coquand. A formalised proof of the soundness and completeness of a simply typed lambda-calculus with explicit substitutions. *Higher Order Symbolic Computation*, 15(1):57–90, 2002.

[5] Arthur Azevedo de Amorim et al. `extructures`. Coq library available at https://github.com/arthuraa/extructures.

[6] Cyril Cohen et al. `finmap`. Coq library available at https://github.com/math-comp/finmap.

[7] Dan Frumin, Herman Geuvers, Léon Gondelman, and Niels van der Weide. Finite sets in homotopy type theory. In *International Conference on Certified Programs and Proofs CPP '18*, pages 201–214. Association for Computing Machinery, 2018.

[8] Thomas Jech. *The Axiom of Choice*. North-Holland, 1973.

[9] Robbert Krebbers. Efficient, extensional, and generic finite maps in coq-std++. In *The Coq Workshop 2023*, 2023.

[10] Clemens Kupke, Fredrik Nordvall Forsberg, and Sean Watters. A fresh look at commutativity: Free algebraic structures via fresh lists. In *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 135–154. Springer, 2023.

[11] Christian Strub. `ssrmisc`. Coq library available at https://github.com/strub/ssrmisc/blob/master/fset.v.

[12] Pierre-Yves Strub. `fset`. Coq library available at https://www.ps.uni-saarland.de/formalizations/fset/html/libs.fset.html.

[13] Andrew Swan. `irreflexive-extensional-order-on-every-set-gives-excluded-middle`. Agda formalisation by Tom De Jong available at https://www.cs.bham.ac.uk/~mhe/TypeTopology/Ordinals.WellOrderingTaboo.html.

[14] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.

Comparing Quotient- and Symmetric Containers

Philipp Joram and Niccolò Veltri

Department of Software Sciences, Tallinn University of Technology, Tallinn, Estonia

Abstract

Quotient containers [2] are set-valued containers with a selection of permissible automorphisms of positions. They are interpreted as endofunctors on sets where labellings are identified according to those isomorphisms. More recently [7], symmetric containers were introduced to model types with symmetries. These are interpreted as endofunctors on groupoids. We investigate the relationship between the two notions. To any quotient container, we associate a symmetric container by delooping automorphism groups of positions. In the other direction, obtaining a quotient container is difficult constructively: it suffices to divide each groupoid of shapes into its pointed connected components, which should be implied by a variant of the axiom of choice. In any case we can deduce that a quotient container and its associated symmetric container have equivalent interpretations as set-endofunctors when truncated appropriately.

Container datatypes [1] are represented by a *container* $(S \triangleright P)$ consisting of a set of shapes S, and for each shape s a set of positions $P(s)$ where data can be stored. A container is interpreted as an endofunctor $[\![S \triangleright P]\!] : \mathsf{Set} \to \mathsf{Set}$ given by $[\![S \triangleright P]\!](X) := \sum_{s \in S} P(s) \to X$, called its *extension*. Container datatypes cover a wide range of polymorphic datatypes. Polymorphism is expressed by proving that $[\![-]\!]$ is fully faithful, i.e. showing that morphisms of containers are exactly natural transformations of their extensions. To also model quotient datatypes, [2] introduced *quotient containers* $(S \triangleright P/G)$, which additionally specify groups $G_s \subseteq \mathrm{Aut}(P(s))$ for each shape $s : S$. The extension of a quotient container identifies labelled positions by G_s-actions: $[\![S \triangleright P/G]\!]_/(X) := \sum_{s \in S} P(s) \to X/\!\sim_s$, where $f \sim_s g$ whenever $f = g \circ \sigma$ for some $\sigma \in G_s$. Specifying the groups G_s gives control over which pieces of X-labelled data are identified in the extension.

More recently, containers have been studied in a *univalent* setting [9, 11, 6]. Here, interpretation of containers as Set-endofunctors is no longer appropriate, as shapes and positions can be higher (homotopy) groupoids. In the following, we are working in univalent foundations with universes of homotopy sets Set and homotopy groupoids Gpd and higher inductive types, such as set truncation $\|-\|$ and deloopings $\mathbf{B}G$ of groups. To help keep track of the higher structure involved, we have prototyped the majority the following results in Cubical Agda.

In [7], Gylterud defines *symmetric containers* as those for which shapes are groupoids and positions are valued in sets. For such containers, extension is a functor $[\![-]\!] : \mathsf{Gpd} \to \mathsf{Gpd}$. Both quotient- and symmetric containers are used to model datatypes with symmetries: Multisets, for example, are "lists up to reordering", and arise both from a quotient container $(\mathbb{N} \triangleright \mathrm{Fin}/S_n)$ [2, Ex. 3.7] and a symmetric container whose shapes are deloopings $\mathbf{B}S_n$ [7, Ex. 3.1.5]. We show that classically, both notions of "containers with symmetries" have essentially the same semantics. We reiterate, in the language of HoTT/UF, a point raised in e.g. [9, 4]: dataypes with symmetries ought to have an interpretation in groupoids, and quotient containers are their proof-irrelevant shadows.

Working on multisets in [8], we observed that quotient containers naturally give rise to symmetric containers:

Definition 1. Each quotient container $Q = (S \triangleright P/G)$ defines an associated symmetric container $Q^\uparrow = (S^\uparrow \triangleright P^\uparrow)$ with shapes given by $S^\uparrow := \sum_{s:S} \mathbf{B}G_s$ and positions $P^\uparrow$ defined from P by recursion on the delooping $\mathbf{B}G_s$.

As expected, $Q^{\uparrow}$ represents the same functor as Q when truncated appropriately:

Theorem 2. *Denote the type of quotient- and symmetric containers by $\mathcal{Q}$ and $\mathcal{S}$, respectively. The following diagram commutes:*

$$\begin{array}{ccc} \mathcal{S} & \xrightarrow{[\![-]\!]} & \mathsf{Gpd} \to \mathsf{Gpd} \\ {\scriptstyle (-)^{\uparrow}}\uparrow & & \downarrow{\scriptstyle \|-\|\circ -} \\ \mathcal{Q} & \xrightarrow[[\![-]\!]_{/}]{} & \mathsf{Set} \to \mathsf{Set} \end{array}$$

We would like to strengthen the above to a square of functors $\mathcal{Q} \to (\mathsf{Set} \to \mathsf{Set})$. So far, we have defined the category of quotient containers following [1] and given a constructive proof that $[\![-]\!]_{/}$ is a left Kan extension when defined in terms of a HIT of set quotients. The extension $[\![-]\!]$ is a functor between the *wild categories* [5], and we believe that a proof of [3] can be adapted to show that $[\![-]\!]$ is fully faithful. Fully-faithfulness of both extension functors implies that $\mathcal{S}$ is a univalent category, and that $\mathcal{Q}$ is not. We are interested to know in what sense the former is a completion of the latter.

Following our intuition for multisets, we would like to construct containers in the other direction. However, we encounter coherence issues: Given a symmetric container $(S \triangleright P)$, the associated quotient container is supposed to have as shapes the truncated $\|S\|$, with the higher path spaces of S put into a subgroup of automorphisms of positions. This leaves us to define positions as a function from a set $\|S\|$ to the groupoid Set, which in general requires P to be constant on paths [10].

The shapes defined by $(-)^{\uparrow}$ are special, in the following sense: We say that a groupoid G has a *skeleton* if its connected components are pointed, i.e. we can exhibit a map of type $\mathrm{Skeleton}(G) := \prod_{x:\|G\|} \mathrm{fiber}_{|-|}(x)$.

Corollary 3. *For any $(S \triangleright P/G) : \mathcal{Q}$, $S^{\uparrow}$ has pointed connected components, namely $\mathbf{B}G_s$.*

Strengthening our assumptions on symmetric containers, we obtain the following:

Theorem 4. *For any $C = (S \triangleright P)$ with a skeleton on S, there is an associated quotient container $C^{\downarrow} = (S^{\downarrow} \triangleright P^{\downarrow}/G^{\downarrow})$ given by $S^{\downarrow} := \|S\|$, $P^{\downarrow} := P \circ \mathrm{sk}_S$, and $G^{\downarrow}_s := \{\sigma \mid \exists (p : s = s).\ \mathrm{cong}_P(p) = \sigma\}$. Here, $\mathrm{sk}_S : \|S\| \to S$ denotes the map selecting a point for each connected component.*

Like in category theory, arbitrary skeletons are obtained from classical assumptions:

Proposition 5. *Denoting by $\mathcal{S}^{\circ}$ the type of symmetric containers with a skeleton on their shapes, the evident map $\mathcal{S}^{\circ} \to \mathcal{S}$ is an equivalence if the axiom of choice holds.*

We plan to investigate in which way the use of the axiom of choice is necessary for $\mathcal{S}^{\circ}$ and $\mathcal{S}$ to coincide.

References

[1] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. "Categories of Containers". In: *Foundations of Software Science and Computation Structures*. Springer Berlin Heidelberg, 2003, pp. 23–38. ISBN: 9783540365761. DOI: 10.1007/3-540-36576-1_2.

[2] Michael Abbott et al. "Constructing Polymorphic Programs with Quotient Types". In: *Mathematics of Program Construction*. Springer Berlin Heidelberg, 2004, pp. 2–15. ISBN: 9783540277644. DOI: 10.1007/978-3-540-27764-4_2.

[3] Thorsten Altenkirch and Stefania Damato. "Revisiting Containers in Cubical Agda". In: *29th International Conference on Types for Proofs and Programs*. Ed. by Eduardo Hermo Reyes and Alicia Villanueva. Valencia (Spain), 2023. URL: https://types2023.webs.upv.es/TYPES2023.pdf.

[4] John C. Baez and James Dolan. "From Finite Sets to Feynman Diagrams". In: *Mathematics Unlimited — 2001 and Beyond*. Springer Berlin Heidelberg, 2001, pp. 29–50. ISBN: 9783642564789. DOI: 10.1007/978-3-642-56478-9_3.

[5] Paolo Capriotti and Nicolai Kraus. "Univalent higher categories via complete Semi-Segal types". In: *Proceedings of the ACM on Programming Languages* 2.POPL (Dec. 2017), pp. 1–29. ISSN: 2475-1421. DOI: 10.1145/3158132.

[6] Eric Finster et al. "A Cartesian Bicategory of Polynomial Functors in Homotopy Type Theory". In: *Proc. of 37th Conf. on Mathematical Foundations of Programming Semantics, MFPS'21*. Ed. by Ana Sokolova. Vol. 351. EPTCS. 2021, pp. 67–83. DOI: 10.4204/EPTCS.351.5.

[7] Håkon Robbestad Gylterud. "Symmetric Containers". MA thesis. Department of Mathematics, Faculty of Mathematics and Natural Sciences, University of Oslo, 2011. URL: https://hdl.handle.net/10852/10740.

[8] Philipp Joram and Niccolò Veltri. "Constructive Final Semantics of Finite Bags". en. In: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. DOI: 10.4230/LIPICS.ITP.2023.20.

[9] Joachim Kock. "Data Types with Symmetries and Polynomial Functors over Groupoids". In: *Proc. of 28th Conf. on Mathematical Foundations of Programming Semantics, MFPS'12*. Ed. by Ulrich Berger and Michael Mislove. Vol. 286. ENTCS. Elsevier, 2012, pp. 351–365. DOI: 10.1016/j.entcs.2013.01.001.

[10] Nicolai Kraus. "The General Universal Property of the Propositional Truncation". en. In: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. DOI: 10.4230/LIPICS.TYPES.2014.111.

[11] Brent Yorgey. "Combinatorial Species and Labelled Structures". PhD thesis. University of Pennsylvania, 2014. URL: http://ozark.hendrix.edu/~yorgey/pub/thesis.pdf.

Univalent material set theory: Hierarchies of n-types

Håkon Robbestad Gylterud Elisabeth Stenholm

May 10, 2024

Abstract

The relationship between structural and material set theory is well studied (See for instance Shulman's article on the topic[5]). Homotopy type theory/Univalent Foundations (HoTT/UF) can be seen as an extension of structural set theory to higher homotopy levels. This talk is part of an effort to investigate what the corresponding extension of material set theory to higher homotopy levels could look like. This work has been formalised in Agda[6] using Agda-Unimath[4] and a preprint covering the basis for this talk is available[3]. The formalisation can be found at: `https://git.app.uib.no/hott/hott-set-theory`

Central to this talk will be the notion of an $\in$-structure, which extends the notion of model of set theory to include higher structures.

Definition 1 ( ). *An* $\in$***-structure*** *is a pair* $(V, \in)$ *where* $V : \mathrm{Type}$ *and* $\in : V \to V \to \mathrm{Type}$*, which is* ***extensional****: for each* $x, y : V$*, the canonical map* $x = y \to \prod_{z:V} z \in x \simeq z \in y$ *is an equivalence of types.*

Given this core notion, we formulate analogies of the familiar set theoretic axioms (with focus on the constructive ones) at each type level; the 0-th level being the usual set theoretic ones*. While the axiom of exponentiation (formation of function sets) has a uniform formulation most other axioms branch out into various forms depending on type levels. For instance, the 1-form of pairing, denoted $\{a, b\}_1$, is *multiset pairing*, while the 0-form, $\{a, b\}_0$ is the usual idempotent set pairing.

We then proceed to show consistency by constructing models, V^n, of these generalised axioms at each level – including V^∞ which is the underlying type of Aczel's model of set theory in type theory[1]. These types are independently interesting, as they are the initial algebras of functors P^n, which generalise the powerset functor to n-truncated maps. Of these only P^∞ is a strictly positive type former, so the existence of these initial algebras is not immediate. The corresponding terminal coalgebras would also be models of univalent material set theory, but have resisted construction thus far.

Connecting the $\in$-structures back to their ambient type theory, there is a notion that an element in a $\in$-structure internalises a type, namely its type of elements $\mathrm{El}\ a := \sum_{x:V} x \in a$. It turns out that the generalised forms of the axiom of replacement is tightly connected to the questions of how different internalisations of a given type relate. The type level of El is bounded by the type level of V, since $\in$ is one level lower, which makes V^n with El into a family of n-types indexed by an n-type. This is notably different from the usual situation in HoTT/UF where the type of all n-types gives a family of n types indexed by an $n+1$-type. In joint work with Gratzer and Mörtberg, the authors have given a detailed account of the categorical properties of V^0[2].

In this talk we will take a closer look also at V^1, which internalises all (U-small) classifying spaces of groups. These internalisations can be described using $\in$: For instance $B\mathbb{Z}_2$ is internalised by the set $\{\{\emptyset, \emptyset\}_1\}_0$, which is uniquely characterised by the equation $(x \in \{\{\emptyset, \emptyset\}_1\}_0) = \| \prod_{y:V} y \in x = (2 \times (y = \emptyset))\|_{-1}$, for all x. In fact this particular type can be formed in all $\in$-structures satisfying two of the aforementioned generalised axioms, called axioms 1-pairing and 0-singletons. In general, one also needs U-restricted 1-separation to get the classifying type of every (U-small) group.

While it is trivial that all small n-types are internalised in V^{n+1}, the status of small n-types in V^n is open. For V^0 this is connected to Shulman's axiom of well-founded materialisation[5], while in V^1 the same question is related to the ability to express groupoids as sums of groups.

In conclusion, we hope to demonstrate that univalent material set theory is an interesting object of study, forming a bridge between traditional set theoretical notions and the higher structures of HoTT/UF.

*We use "type level" here to denote the homotopy level of a type.

[1] Peter Aczel. “The Type Theoretic Interpretation of Constructive Set Theory”. In: *Logic Colloquium '77*. Ed. by A. MacIntyre, L. Pacholski, and J. Paris. North–Holland, Amsterdam-New York, 1978, pp. 55–66.

[2] Daniel Gratzer et al. *The Category of Iterative Sets in Homotopy Type Theory and Univalent Foundations*. 2024. arXiv: 2402.04893 [cs.LO].

[3] Håkon Robbestad Gylterud and Elisabeth Stenholm. *Univalent Material Set Theory*. 2023. arXiv: 2312.13024 [math.LO].

[4] Egbert Rijke et al. *The agda-unimath library*. URL: https://github.com/UniMath/agda-unimath/.

[5] Michael Shulman. “Stack semantics and the comparison of material and structural set theories”. In: (Apr. 2010). DOI: 10.48550/arXiv.1004.3802. URL: https://arxiv.org/abs/1004.3802.

[6] The Agda development team. *Agda*. URL: https://github.com/agda/agda.

Session 20: Category Theory

The Univalence Maxim and Univalent Double Categories

Nima Rasekh, Niels van der Weide, Benedikt Ahrens, Paige Randall North

Formalization of mathematics is often seen as a way to formally verify mathematical definitions and theorems and gain more confidence in their veracity. However, in certain mathematical contexts and appropriately chosen type theories, pursuing formalization can result in additional advantages that can broaden our mathematical perspective. In this talk we will focus on the particular case of the **Univalence Maxim** and its application to the study and formalization of **double category theory**.

1 The Univalence Maxim for Categorical Structures

The *univalence maxim* broadens our understanding of categorical structures via formalization in univalent foundations, by relating categorical structures and various notions of equivalences. Before providing a more precise characterization, we will first examine two relevant examples.

Categories were first formalized in univalent foundations by Ahrens–Kapulkin–Shulman [AKS15]. As part of their formalization they also defined *univalent* categories and established a *univalence principle*, proving that identities of the type of categories coincide with equivalences. Moreover, as an immediate implication of their argument it follows that the type of categories with a set of objects has identities given by isomorphisms of categories. We hence witness that in univalent foundations we have *two* notions of categories: categories with a set of objects, who are invariant under isomorphisms, and univalent categories, who are invariant under equivalences. We are hence witnessing a correspondence between possible formalizations of categories in UniMath and possible notions of equivalences of categories. This should be understood as a first manifestation of the above-mentioned maxim.

The formalization of categories has been generalized to a formalization of 2-categories and bicategories in univalent foundations [AFM+21]. The authors in particular formalize univalent bicategories and establish their univalence principle, by proving that identities in the type of univalent bicategories coincide with biequivalences of bicategories. One implication of this result is that we need to relax the categorical structure from a 2-category, where the composition of 1-morphisms is strictly associative and unital, to a bicategory, meaning weaken those unitality and associativity conditions, in order to prove the univalence principle. Using similar methods, we can prove a univalence principle for 2-categories (i.e., a univalent category enriched in setcategories), by observing that their identities coincide with functors that are isomorphisms of objects and local equivalences of hom categories.

Here for the first time we are witnessing that obtaining a univalence principle for a given choice of equivalence of a categorical structure, can necessitate adjusting, and particularly weakening, the categorical structure. The examples we have presented allow us to now articulate the **Univalence Maxim for Categorical Structures** in a more precise manner: For every categorical structure and for every possible notion of equivalence of that structure, there exists a corresponding notion of univalent categorical structure, such that its identities precisely correspond to the chosen equivalences. We can in particular understand this maxim as a feedback loop between category theory and univalent mathematics. Indeed, advancing our knowledge of categories benefits from formalizing it (an application of formalization to category theory), and formalizing categories along with a chosen equivalence in univalent foundations requires understanding possible weakening of the chosen categorical structure (an application of categorical literature to formalization).

2 The Maxim in Action: Double Categories

In a variety of situations objects witness more than one relevant notion of morphism. Important examples include sets, which come with functions and relations, categories, with functors and profunctors, or rings, with ring homomorphisms and modules. This motivates defining a categorical structure which generalizes categories and can capture the data of two types of morphisms, which is known as a *double category*.

Double categories are a categorical structure consisting of objects and two types of morphisms (called horizontal and vertical morphisms) that interact well with each other via appropriately chosen squares [Ehr63]. They were introduced by Ehresmann as a tool to better understand categories, and played an important role in formal category theory and the theory of equipments [Woo82, Woo85]. Beyond those original applications, double categories have also found a variety of applications in applied mathematics and computer science; see, for instance, its applications in systems theory [Cou20, Mye21, BCV22] and programming languages theory [DM13, NL23]. As part of this project we discuss the formalization of double categories, their results, and their examples, as well as provide further evidence for the *univalence maxim* in the context of the formalization of double categorical notions in univalent foundations.

As the definition of a double category involves far more data than a category, double categories exhibit many different notions of equivalences. This includes standard notions such as an isomorphism of double categories. However, we can also define a *horizontal equivalence* defined as inducing equivalences on the following two underlying categories: the one given by objects and horizontal morphisms, and the one given by vertical morphisms and squares. Similarly, we can define *vertical equivalences*. Moreover, we can generalize both notions to *horizontal (vertical) biequivalences* [MSV20]. Finally we also have symmetric notions of equivalences, such as *gregarious equivalences* [Cam20, ANST21].

As part of our work, we apply the univalence maxim to double categorical structures. Using this method we obtain a correspondence between invariances of double categories and corresponding double categorical structures with appropriately chosen strictness of unitality and associativity, which we also formalize in univalent foundations. We can summarize it as follows:

Invariance	**Structure**	**Source**	**Formalization**
Iso. of Double Cat.	Set Double Cat.	[Ehr63]	[Weic]
Iso. of Pseudo Double Cat.	Set Pseudo Cat.	[Gra20]	[Weia]
Horizontal Equiv. of Double Cat.	Univalent Pseudo Double Cat.	[Gra20]	[vdWRAN24, Weib]
Gregarious Equiv. of Double Cat.	Univalent Double Bicat.	[Ver11, ANST21]	[RWAN24, Weid]

Here, a pseudo-double category has strict horizontal composition but only weakly associative and unital vertical composition. Moreover, double bicategories have weakly associative and unital compositions in both directions, fitting the symmetric nature of gregarious equivalences. Its categorical properties and formalization are part of [RWAN24], building on the univalence principle developed in [ANST21].

3 Conclusion

Our work has two accomplishments. As an application to category theory, we applied the univalence maxim to obtain a structured understanding of double categorical equivalences. In addition, we formalized a wide range of double categorical notions, their properties and their univalence principles.

References

[AFM+21] Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. Bicategories in univalent foundations. *Math. Struct. Comput. Sci.*, 31(10):1232–1269, 2021.

[AKS15] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the Rezk completion. *Math. Structures Comput. Sci.*, 25(5):1010–1039, 2015.

[ANST21] Benedikt Ahrens, Paige Randall North, Michael Shulman, and Dimitris Tsementzis. The univalence principle. *arXiv preprint*, 2021. arXiv:2102.06275, To appear in Memoirs of the AMS.

[BCV22] John C. Baez, Kenny Courser, and Christina Vasilakopoulou. Structured versus Decorated Cospans. *Compositionality*, 4, September 2022.

[Cam20] Alexander Campbell. The gregarious model structure for double categories. *Talk at Masaryk University Algebra Seminar*, 2020. Slides available online.

[Cou20] Kenny Courser. *Open Systems: A Double Categorical Perspective*. PhD thesis, University of California, Riverside, 2020.

[DM13] Pierre-Évariste Dagand and Conor McBride. A categorical treatment of ornaments. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 530–539. IEEE Computer Society, 2013.

[Ehr63] Charles Ehresmann. Catégories structurées. In *Annales scientifiques de l'École Normale Supérieure*, volume 80, pages 349–426, 1963.

[Gra20] Marco Grandis. *Higher dimensional categories*. World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2020. From double to multiple categories.

[MSV20] Lyne Moser, Maru Sarazola, and Paula Verdugo. A model structure for weakly horizontally invariant double categories. *arXiv preprint*, 2020. arXiv:2007.00588.

[Mye21] David Jaz Myers. Double categories of open dynamical systems (extended abstract). *Electronic Proceedings in Theoretical Computer Science*, 333:154–167, feb 2021.

[NL23] Max S. New and Daniel R. Licata. A formal logic for formal category theory. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 113–134. Springer, 2023.

[RWAN24] Nima Rasekh, Niels van der Weide, Benedikt Ahrens, and Paige North. Insights from univalent foundations: A case study using double categories. *arXiv preprint*, 2024. arXiv:2402.05265.

[vdWRAN24] Niels van der Weide, Nima Rasekh, Benedikt Ahrens, and Paige Randall North. Univalent double categories. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2024, page 246–259, New York, NY, USA, 2024. Association for Computing Machinery.

[Ver11] Dominic Verity. Enriched categories, internal categories and change of base. *Repr. Theory Appl. Categ.*, (20):1–266, 2011.

[Weia] Niels van der Weide. The category of pseudo double setcategories. available at `https://github.com/UniMath/UniMath/blob/master/UniMath/Bicategories/DoubleCategories/Core/CatOfPseudoDoubleCats.v#L766`.

[Weib] Niels van der Weide. The category of pseudo double setcategories. available at `https://github.com/UniMath/UniMath/blob/master/UniMath/Bicategories/DoubleCategories/Core/BicatOfDoubleCats.v#L1909`.

[Weic] Niels van der Weide. The univalent category of strict double categories. available at `https://github.com/UniMath/UniMath/blob/master/UniMath/Bicategories/`

DoubleCategories/Core/CatOfStrictDoubleCats.v#L429.

[Weid] Niels van der Weide. Verity double bicategories. available at https://github.com/UniMath/UniMath/blob/master/UniMath/Bicategories/DoubleCategories/DoubleBicat/VerityDoubleBicat.v.

[Woo82] R. J. Wood. Abstract proarrows. I. *Cahiers Topologie Géom. Différentielle*, 23(3):279–290, 1982.

[Woo85] R. J. Wood. Proarrows. II. *Cahiers Topologie Géom. Différentielle Catég.*, 26(2):135–168, 1985.

Synthetic Stone Duality

Felix Cherubini, Thierry Coquand, Freek Geerligs*, and Hugo Moeneclaey

University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden

We propose a variation of the axiomatization of Zariski (higher) topos in synthetic algebraic geometry [CCH23] to an axiomatization of (separable) Stone spaces, with Stone duality, within a univalent type theory. The roles of affine schemes and schemes are taken over by Stone spaces and compact Hausdorff spaces respectively. In the theory, we can show that any map between compact Hausdorff spaces is continuous, hence the negation of WLPO holds. However, we can show that LLPO and Markov's principle do hold. We conjecture that internal results on light condensed sets [Ásg21; CS24; Sch19] can be shown using univalent type theory extended by these axioms. Furthermore, this work can be seen as a variation of the work in [XE13]. We also expect to build a constructive sheaf model of these axioms, similar to the constructive model of synthetic algebraic geometry presented in [CCH23].

We denote the type of countably presented Boolean algebras by Boole. Given a Boolean algebra B, we define $Sp(B)$, the spectrum of B as the set of Boolean morphisms from B to 2. A type of the form $Sp(B)$ for $B : \mathsf{Boole}$ is called Stone. Two motivating examples of elements of Boole are as follows:

- C is the free Boolean algebra on countably many generators $(p_n)_{n\in\mathbb{N}}$. The corresponding set $Sp(C)$ is Cantor space $2^{\mathbb{N}}$.
- The Boolean algebra B_∞ is the quotient of C by the relations $p_n \wedge p_m = 0$ for $n \neq m$. A term of $Sp(B_\infty)$ sends p_n to 1 for at most one n. For this reason, $Sp(B_\infty)$ is denoted $\mathbb{N}_\infty$.

Axiom 1 (Stone duality)**.** *For any $B : \mathsf{Boole}$, the evaluation map $B \to 2^{Sp(B)}$ is an isomorphism.*

It follows from Stone duality that being Stone is a proposition and Sp defines an embedding from Boole to any universe $\mathcal{U}$. We denote its image Stone. Any $X : \mathit{Stone}$ has a topology where basic clopens are given by decidable subsets. Using Stone duality we can show that any map from a Stone space to $\mathbb{N}$ is uniformly continuous. Both Stone and Boole have a natural structure of a category, and Stone duality gives that Sp induces a dual equivalence between them.

Axiom 2 (Surjections are Formal Surjections)**.** *A map $Sp(B') \to Sp(B)$ is surjective iff the corresponding Boolean map $B \to B'$ is injective.*

Using this axiom, we can show that if B is nontrivial, $Sp(B)$ is merely inhabited. Note that the sum of the maps $\mathbb{N}_\infty \to \mathbb{N}_\infty$ sending n to $2n, 2n+1$ respectively has no section. However, we can use the above axiom to show that it is surjective. This implies that $\mathbb{N}_\infty$ is not projective and that LLPO holds. However, we can also show the negation of WLPO from Axiom 1.

Analogously to synthetic algebraic geometry, we need an axiom of local choice.

Axiom 3 (Local choice)**.** *Given X Stone, E, F arbitrary types, a map $X \to F$ and $E \twoheadrightarrow F$ surjective, there is some Y Stone, a surjection $Y \twoheadrightarrow X$ and a map $Y \to E$ such that the following diagram commutes:*

$$\begin{array}{ccc} Y & \dashrightarrow & E \\ \vdots & & \downarrow \\ X & \longrightarrow & F \end{array}$$

*Speaker.

We define a type to be compact Haussdorf if it is the quotient of a Stone type by a closed equivalence relation. We denote CHaus for the the type of compact Hausdorff types. A motivating example for compact Haussdorf types is the unit interval, which can be given as a quotient of Cantor space. To show that such an interval is isomorphic to the standard Cauchy interval, we need LLPO and an axiom of dependent choice.

Axiom 4 (Dependent Choice)**.** *Given a sequence of arbitrary types* $(X_n)_{n:\mathbb{N}}$ *and surjections* $X_n \twoheadrightarrow X_{n-1}$*, all the limit projection maps* $X \to X_n$ *are surjective.*

It is important that these two last axioms are stated for *arbitrary* types and not types that are only homotopy sets. One application should be a proof of $H^n(S, \mathbb{Z}) = 0$ for $n > 0$, for S Stone, that we have checked for $n = 1$ (similar to the proof of $H^1(X, R) = 0$ for X affine in the setting of [CCH23]). For this, we use the general definition of cohomology group in Homotopy Type Theory [Pro13], which refers to types that are not necessarily homotopy sets. We also expect to have for $X : \mathsf{CHaus}$ that $H^n(X, \mathbb{Z})$ coincides with the singular cohomology of X.

Finally, we checked that all axioms suggested recently by R. Barton and J. Commelin [BC] follow from our axioms.

References

[Ásg21] Dagur Ásgeirsson. "The Foundations of Condensed Mathematics". MA thesis. Université de Paris, 2021. URL: https://dagur.sites.ku.dk/condensed-foundations/ (cit. on p. 1).

[BC] Reid Barton and Johan Commelin. *lean-ctt-snapshot*. URL: https://github.com/jcommelin/lean-ctt-snapshot (cit. on p. 2).

[CCH23] Felix Cherubini, Thierry Coquand, and Matthias Hutzler. *A Foundation for Synthetic Algebraic Geometry*. 2023. arXiv: 2307.00073 [math.AG]. URL: https://www.felix-cherubini.de/iag.pdf (cit. on pp. 1, 2).

[CS24] Dustin Clausen and Peter Scholze. *Analytic Stacks*. Lecture series. 2023-2024. URL: https://www.youtube.com/playlist?list=PLx5f8IelFRgGmu6gmL-Kf_Rl_6Mm7juZO (cit. on p. 1).

[Pro13] The Univalent Foundations Program. *Homotopy type theory: Univalent foundations of mathematics*. 2013 (cit. on p. 2).

[Sch19] Peter Scholze. *Lectures on Condensed Mathematics*. 2019. URL: https://people.mpim-bonn.mpg.de/scholze/Condensed.pdf (cit. on p. 1).

[XE13] Chuangjie Xu and Martín Hötzel Escardó. "A Constructive Model of Uniform Continuity". In: *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*. Ed. by Masahito Hasegawa. Vol. 7941. Lecture Notes in Computer Science. Springer, 2013, pp. 236–249. DOI: 10.1007/978-3-642-38946-7_18. URL: https://doi.org/10.1007/978-3-642-38946-7%5C_18 (cit. on p. 1).

A formal framework for univalent completions

Kobe Wullaert

Delft University of Technology, The Netherlands

1 Introduction

Abstract summary In this abstract, we provide a formal case study on the theory of *weak equivalences* (or, equivalence up to univalence) and *Rezk/univalent completions*. We generalize the results in [1] and [5]; concerning univalent (enriched) categories, weak equivalences, and Rezk completions. Furthermore, we rely on the theory of (univalent) bicategories [2], and Yoneda structures [3] respectively. (The latter has been developed in set-theoretic foundations.) In particular, we provide a universal characterization of *essentially surjective on objects* (eso) functors. Furthermore, we develop some of the theory of "*(1-dimensionally) univalent 2-categories*".

C.T. in HoTT/UF Internal to HoTT/UF [4], category theory often comes in two flavors: set-category theory, where one deals with categories whose type of objects is a (h)set; or, univalent category theory [1], where one deals with categories with a structure-identity principle for isomorphic objects. Univalent category theory is often preferred for multiple reasons. First, there are not many set-categories, while univalent categories are plenty. Furthermore, univalent categories have the desired type of equivalences: a fully faithful and (mere) essentially surjective functor, between univalent categories, is an equivalence (even an isomorphism/identity) of categories. However, traditional constructions of universal objects (e. g., Kleisli objects) leave the world of univalent categories. A process to turn a non-univalent category into a univalent one is provided in [1]; it provides a construction of the "free univalent" category, referred to as the Rezk completion.

Special cases The theory of univalent categories, weak equivalences, and the Rezk completion, has been generalized to monoidal categories [6], and enriched categories [5]. A key ingredient in the aforementioned papers is the Yoneda embedding, whose (replete) image factorization provides a concrete construction of the Rezk completion.

Axiomatic framework In this project, we provide an axiomatic framework generalizing the concrete construction of [1, 5] to a more abstract (already existing) setting: 2-categories equipped with a Yoneda structure [3]. We observe that a Yoneda structure on a 2-category provides sufficient structure to suitably interpret weak equivalences as morphisms which are essentially surjective on objects and fully faithful. This interpretation is based on (a slight generalization of) Proposition 23 in [3].

Limitations The framework presented here provides a general blueprint for the construction of the Rezk completion. However, the framework does not cover the case of monoidal categories. Indeed, the monoidal Yoneda embedding does not equip a 2-category of monoidal categories with a Yoneda structure. There are two avenues that I consider, fixing the case. A first approach is by considering monoidal categories as "pseudomonoids", relative to a 2-category equipped with a Yoneda structure. Another approach is by weakening the structure provided by the Yoneda structure (i. e., a generalization hereof).

2 Framework

In this section, we fix a 2-category $\mathcal{K}$ equipped with a Yoneda structure, see [3]. (Informally, the objects of $\mathcal{K}$ are to be interpreted as (V-enriched) 1-categories.) The Yoneda structure on $\mathcal{K}$ assigns to every object X an object $\mathcal{P}\,X$ (its object of presheaves) and a morphism $\yoneda_X : X \to \mathcal{P}\,X$ (its Yoneda morphism), see [3] for the universal property of $(X, \yoneda_X)$.

The main idea behind a Yoneda structure is that every morphism is uniquely determined by its action on "generalized objects" and "generalized morphisms" respectively. The idea is made formal by the following construction, due to Street and Walters.

Construction 1. Every precomposition functor $\mathcal{K}(f, Z)$ factors through a displayed category over the source (hom-)category, denoted $ExNat(f, Z)$:

$$\begin{array}{ccc} & & ExNat(f,Z) \\ & {\scriptstyle (f \cdot^{e}_{Z} -)} \nearrow & \downarrow{\scriptstyle \pi_1} \\ \mathcal{K}(Y,Z) & \xrightarrow[(f\cdot -)]{} & \mathcal{K}(X,Z) \end{array}$$

Definition 2. An object Z is **univalent** if for any $f : X \to Y$, the category $ExNat(f, Z)$ is univalent.

Theorem 3. *Let $f : X \to Y$ be a morphism. The following are equivalent:*

1. *f is a **weak equivalence**: for every univalent object Z, the precomposition functor $\mathcal{K}(f, Z)$ is an isomorphism of (hom-)categories;*
2. *f is **fully faithful** (see [3]) and **essentially surjective**, that is: $(f \cdot^{e}_{Z})$ is a weak equivalence of (univalent) categories, for every univalent Z.*

Applying the construction to $\mathcal{K} := \mathsf{Cat}$, we have:

Example 4. Let $f : X \to Y$ be a functor between categories, the following are equivalent:

1. f is a "fully faithful functor", i. e. , for any $x_1, x_2 : X$, the action on morphisms, i. e. ,

$$X(x_1, x_2) \xrightarrow{f_{x_1,x_2}} Y(f\,x_1, f\,x_2)$$

   is an equivalence of types, if and only if f is fully faithful (as a morphism);

2. f is "essentially surjective (on objects)": for every object $y : Y$, there merely exists an object $x : X$ such that $f\,x$ is "isomorphic" to y if and only if f is essentially surjective.

In particular, we recover that weak equivalence is necessarily equivalent to a functor which is essentially surjective and fully faithful.

References

[1] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. "Univalent categories and the Rezk completion". In: *Mathematical Structures in Computer Science* 25.5 (2015), pp. 1010–1039. DOI: 10.1017/S0960129514000486.

[2] Benedikt Ahrens et al. "Bicategories in univalent foundations". In: *Mathematical Structures in Computer Science* 31.10 (2021), pp. 1232–1269. DOI: 10.1017/S0960129522000032.

[3] Ross Street and Robert Walters. "Yoneda structures on 2-categories". In: *Journal of Algebra* 50.2 (1978), pp. 350–379.

[4] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

[5] Niels van der Weide. *Univalent Enriched Categories and the Enriched Rezk Completion*. 2024. arXiv: 2401.11752 [cs.LO].

[6] Kobe Wullaert, Ralph Matthes, and Benedikt Ahrens. "Univalent Monoidal Categories". In: *28th International Conference on Types for Proofs and Programs (TYPES 2022)*. Ed. by Delia Kesner and Pierre-Marie Pédrot. Vol. 269. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 15:1–15:21. ISBN: 978-3-95977-285-3. DOI: 10.4230/LIPIcs.TYPES.2022.15. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2022.15.

The Internal Language of Univalent Categories

Niels van der Weide[1]

Radboud University Nijmegen, The Netherlands
nweide@cs.ru.nl

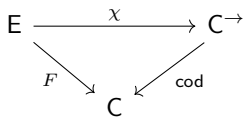
Type theory and category theory are deeply connected. This connection is exhibited by various theorems classifying the internal language of some kind of structured category as a suitable type theory. For example, the simply typed lambda calculus is the language of Cartesian closed categories [17, Theorem 11.3], and Martin-Löf type theory is the language of locally Cartesian closed categories [8, 11, 19]. Such theorems allow us to use type theory to reason about the objects and morphisms of a category.

However, in univalent foundations [21] several subtleties arise when we try to find a suitable internal language for classes of univalent categories. Various models of type theory, such as categories with families (CwFs) [12], assume strictness of the types in the model. This strictness requirement actually eliminates the CwFs whose types are given by sets or presheaves in univalent foundations, and, more generally, it is not so that every univalent category with finite limits gives rise to a CwF with suitable type formers. A related problem comes up when one defines the syntax of type theory as a quotient inductive-inductive type [5]: without assuming UIP, one cannot interpret the types in the syntax as sets. Note that one could instead use iterative sets [13], if one is interested in strict rather than univalent categories.

In this abstract, we study the internal language of univalent categories. More specifically, we construct an equivalence between univalent categories with finite limits and type theories with suitable type formers (unit types, binary product types, extensional identity types, and sigma types). This gives an analogue of [8, Theorem 6.1] for univalent categories. Our results are formalized in Coq [20] using the UniMath library [22].

1 Univalent Comprehension Categories

Throughout the years, many different kinds of categorical models for dependent type theory have been developed [4, 6, 10, 12, 14, 15]. For our purposes, we need one that does not enforce strictness upon the types. Since we are interested in univalent categories, we would not have a set of type in most actual examples (e.g., sets and presheaves). In addition, we would like substitution to be expressed via a universal property. If we were to express substitution as an operation, then we would also need to find the correct coherences. For universal properties, such coherences are automatically satisfied and there is no need to formulate them explicitly. One particular kind of model of dependent type theory that satisfies these requirements, is given by **comprehension categories**. Recall that a comprehension category [15, 16] is a strictly commuting diagram of functors.



Here we require F to be a fibration and χ to preserve cartesian morphisms, and we require that C has a terminal object, which we denote by $[]$. In addition, we assume that our comprehension categories are full, meaning that χ is fully faithful as well. We denote the fiber along F of objects

$\Gamma : \mathsf{C}$ by $\mathsf{Ty}(\Gamma)$, and given a morphism $s : \Delta \to \Gamma$, we have a functor by $s^* : \mathsf{Ty}(\Gamma) \to \mathsf{Ty}(\Delta)$. Objects of C are called **contexts** and objects $A : \mathsf{Ty}(\Gamma)$ are **types** in context Γ. Context extension is given by the functor χ: it sends every type $A : \mathsf{Ty}(\Gamma)$ to a morphism $\pi_A : \Gamma.A \to \Gamma$ where $\Gamma.A$ is the context Γ extended with A. The functor s^* gives substitution of types.

In univalent foundations, one formulates comprehension categories using displayed categories [3, Definition 6.1]. We also require the involved categories to be univalent. More specifically, a comprehension category is **univalent** if both C and E are univalent [2].

In the remainder, we look at univalent comprehension categories with additional structure.

Definition 1. A **democratic finite limit (DFL) comprehension category** is a univalent full comprehension category with the additional structure described below.

- For every $\Gamma : \mathsf{C}$, $\mathsf{Ty}(\Gamma)$ has a terminal object, binary products, and equalizers;
- for every morphism s, the substitution functor s^* preserves finite limits;
- the substitution functors s^* have left adjoints satisfying the Beck-Chevally condition;
- for every $\Gamma : \mathsf{C}$ there is a type $\overline{\Gamma} : \mathsf{Ty}([])$ and an isomorphism $\Gamma \cong [].\overline{\Gamma}$.

We also require that the canonical maps from $\Gamma.\mathbb{1}$ to Γ and from $\Gamma.A.B$ to $\Gamma.(\Sigma A.B)$ are isomorphisms, where we write $\mathbb{1}$ for the fiberwise terminal object and $\Sigma A.B$ for action of the left adjoint of π_A^* on B.

Note that one can construct extensional identity types using fiberwise equalizers [16, Theorem 10.5.10], so all DFL comprehension categories have unit types, binary product types, extensional identity types, and sigma types. The final requirement in Definition 1 expresses that every DFL comprehension category is **democratic** [8, Definition 2.6].

Definition 2. We define the bicategory $\mathsf{DFLCompCat}$ of DFL comprehension categories as the bicategories whose objects are DFL comprehension categories. For the 1-cells, we pick functors that preserve all type formers, the empty context, and context extension up to isomorphism.

The bicategory $\mathsf{DFLCompCat}$ is univalent [1]. Note that one might be interested in other bicategories of comprehension categories where the 1-cells only are only required to be lax morphisms [9, Definition 3.3.1.5]. In addition, we do not add the requirement that the 1-cells preserve democracy in contrast to [7, 8]. This is because every functor preserves democracy. The reason for that, is that all morphisms in the diagram of Definition 3.6 in [8] are isomorphisms, and thus there is a unique isomorphism d_Γ witnessing the preservation of democracy.

2 The Internal Language Theorem

We write FinLim for the bicategory whose objects are univalent categories with finite limits, 1-cells are functors that preserve finite limits, and whose 2-cells are natural transformations. Now we give a univalent analogue of [8, Theorem 6.1].

Theorem 3. *We have a biequivalence between* $\mathsf{DFLCompCat}$ *and* FinLim.

Intuitively, categories with finite limits correspond to a class of comprehension categories. As such, extensional type theory with unit types, binary product types, and sigma types is the internal language of categories with finite limits. Since the involved bicategories are univalent, the types of DFL comprehension categories and of categories with finite limits are equivalent. In the formalization, this biequivalence is also extended to locally Cartesian closed categories [8, Theorem 6.1] and to various classes of toposes using ideas from [18].

References

[1] Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. Bicategories in univalent foundations. *Math. Struct. Comput. Sci.*, 31(10):1232–1269, 2021.

[2] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the rezk completion. *Math. Struct. Comput. Sci.*, 25(5):1010–1039, 2015.

[3] Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed categories. *Log. Methods Comput. Sci.*, 15(1), 2019.

[4] Benedikt Ahrens, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. Categorical structures for type theory in univalent foundations. *Log. Methods Comput. Sci.*, 14(3), 2018.

[5] 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.

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

[7] Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: unityped, simply typed, and dependently typed. In *Joachim Lambek: the interplay of mathematics, logic, and linguistics*, volume 20 of *Outst. Contrib. Log.*, pages 135–180. Springer, Cham, [2021] ©2021.

[8] Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and martin-löf type theories. *Math. Struct. Comput. Sci.*, 24(6), 2014.

[9] Greta Coraglia. *Categorical structures for deduction*. PhD thesis, PhD thesis, Universita degli Studi di Genova, 2023.

[10] Greta Coraglia and Ivan Di Liberti. Context, judgement, deduction. *CoRR*, abs/2111.09438, 2021.

[11] Pierre-Louis Curien, Richard Garner, and Martin Hofmann. Revisiting the categorical interpretation of dependent type theory. *Theor. Comput. Sci.*, 546:99–119, 2014.

[12] Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 1995.

[13] Daniel Gratzer, Håkon Gylterud, Anders Mörtberg, and Elisabeth Stenholm. The category of iterative sets in homotopy type theory and univalent foundations. *CoRR*, abs/2402.04893, 2024.

[14] Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and logics of computation (Cambridge, 1995)*, volume 14 of *Publ. Newton Inst.*, pages 79–130. Cambridge Univ. Press, Cambridge, 1997.

[15] Bart Jacobs. Comprehension categories and the semantics of type dependency. *Theor. Comput. Sci.*, 107(2):169–207, 1993.

[16] Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in logic and the foundations of mathematics*. North-Holland, 2001.

[17] J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1986.

[18] Maria Emilia Maietti. Modular correspondence between dependent type theories and categories including pretopoi and topoi. *Math. Struct. Comput. Sci.*, 15(6):1089–1149, 2005.

[19] Robert Seely. Locally Cartesian closed categories and type theory. *C. R. Math. Rep. Acad. Sci. Canada*, 4(5):271–275, 1982.

[20] The Coq Development Team. The coq proof assistant, July 2023.

[21] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[22] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. Unimath — a computer-checked library of univalent mathematics. available at http://unimath.org.

Session 21: Universes

Universes in simplicial type theory

Ulrik Buchholtz[1], Daniel Gratzer[2], and Jonathan Weinberger[3]

[1] The University of Nottingham Nottingham, England
[2] Aarhus University Aarhus, Denmark
[3] Johns Hopkins University Baltimore, Maryland, USA

Simplicial type theory Simplicial type theory was introduced by Riehl and Shulman [RS17] in order to provide a synthetic account of ∞-categories.[1] To begin with, the ∞-category of ∞-categories may be realized as a reflective subcategory of the category of simplicial spaces [Rez01]. Riehl and Shulman then observe that simplicial spaces—as an ∞-topos—support a model of homotopy type theory (HoTT) [Uni13]. They further observe that by extending HoTT with a few key operations and axioms from this model, homotopy type theory may be transformed into a tool for reasoning directly about ∞-categories, in fact *internal* ∞-categories [MW23], implemented as (complete) Segal objects, due to the semantical results of [Ras21; Rie24; RS17; RV22; Shu19; Wei22b].

Roughly, simplicial type theory (STT) extends HoTT with a new type $\mathbb{I}$ such that (1) $\mathbb{I}$ is an h-set (2) $\mathbb{I}$ is equipped with the structure of a bounded distributive lattice $(\wedge, \vee, 0, 1)$ (3) $\mathbb{I}$ is totally ordered.[2] By viewing $\mathbb{I}$ as a *directed interval* or, equivalently, the walking arrow $\{0 \longrightarrow 1\}$, we build up objects from simplicial homotopy theory using ordinary HoTT such as $\Delta^n \subseteq \mathbb{I}^n$ or $\Lambda^n_k \subseteq \Delta^n$. Each type in STT comes with an intrinsic notion of "hom-space" in the form of the function space $\mathbb{I} \to X$. However, these putative hom-spaces need not enjoy a composition operator or otherwise behave like hom-spaces in any meaningful way. We isolate those types for which hom-spaces behave appropriately and take them as our definition of *synthetic* ∞-category [Rie23; RS17]:

Definition 1 ([RS17])**.** A type X is *Segal* whenever the canonical map $X^{\Delta^2} \to X^{\Lambda^2_1}$ is an equivalence. A Segal type X is *Rezk* whenever $\mathrm{ev}_0 : X^{\mathbb{E}} \to X$ is an equivalence. Here $\mathbb{E}$ is the "walking isomorphism" formed by gluing two degenerate 2-simplices onto $\mathbb{I}$ to add left and right inverses.

Definition 2 ([RS17])**.** A synthetic ∞-category is a Rezk type. A synthetic ∞-category for which every morphism is invertible (i.e., $X^{\mathbb{E}} \to X^{\mathbb{I}}$ is an equivalence) is a synthetic ∞-groupoid.

STT is already enough to prove a number of results: Riehl and Shulman [RS17] prove the Yoneda lemma, develop the theory of adjunctions and discrete fibrations, Buchholtz and Weinberger [BW23] and Weinberger [Wei22a] systematically study the theory of cocartesian fibrations and fibered category theory, and Bardomiano Martínez [Bar22] introduces limits and colimits to the theory as well as exponentiable fibrations. A number of results from these works have been formalized [KRW04; sHo24] in Kudasov's proof assistant Rzk [Kud23]. Despite these results, however, aspects of STT remain underdeveloped.

Most urgently, STT lacks well-adapted *universes of* ∞*-categories* [Cis19; Ras24]. As a variant of HoTT, STT enjoys a hierarchy of univalent universes $\mathcal{U}$ and we may isolate the subuniverse $\mathcal{U}_{\mathrm{grp}}$ which carve out e.g. ∞-groupoids. Unfortunately, maps $\mathbb{I} \to \mathcal{U}_{\mathrm{grp}}$ do not meaningfully correspond to morphisms between ∞-groupoids and, consequently, this type is neither Segal nor

[1] Here, by "∞-categories" we mean $(\infty, 1)$-categories.
[2] We note that Riehl and Shulman [RS17] and subsequent works on simplicial type theory [BW23; Wei22a]

Rezk; the "points" of $\mathcal{U}_{\mathsf{grp}}$ correspond to ∞-groupoids but none of the higher structure encodes maps between ∞-groupoids. Phrased more technically, $\mathcal{U}_{\mathsf{grp}}$ does not classify covariant (left) fibrations. The difference matters: the universe $\mathcal{S}$ classifying left fibrations is the ∞-category of ∞-groupoids and plays the role of **Set** among ∞-categories. Without $\mathcal{S}$, many results such as the Yoneda lemma must be phrased indirectly to avoid mentioning the type of presheaves on a given type. We now discuss our work-in-progress construction $\mathcal{S}$ and types like it through a modal enhancement of STT.

Triangulated type theory Constructing a type classifying left fibrations is akin to constructing a universe in cubical type theory classifying Kan fibrations [Coh+17]. Taking inspiration from Licata et al. [Lic+18] and Weaver and Licata [WL20], we construct $\mathcal{S}$ *internally* to type theory by extending our base theory with several modalities and, in particular, a modality representing the "amazing right adjoint" to $\mathbb{I} \to -$. A small wrinkle immediately complicates this story: $\mathbb{I} \to -$ does not have a right adjoint in the intended model of simplicial spaces. We circumvent this problem by embedding simplicial spaces into cubical spaces [SW21] (see also [KV20; Sat19]) and obtain an ∞-topos containing simplicial spaces (and therefore ∞-categories) in which $\mathbb{I} \to -$ has the required right adjoint. The upshot of this detour is that we no longer need $\mathbb{I}$ to be totally ordered; those types local for $i \leq j \vee j \leq i$ represent simplicial spaces while general types are cubical. All told, we combine simplicial type theory with a variant of MTT [Gra+20] to obtain a type theory for cubical spaces *triangulated type theory* $\mathrm{TT}_{\boxslash}$.[3] Put concisely, we are working with simplicial type theory in an ambient *cubical* type theory.

$\mathrm{TT}_{\boxslash}$ extends STT with several modalities and axioms. Among the modalities, the most important are the global sections modality $\square$ and the "amazing right adjoint" $(-)_{\mathbb{I}}$; neither of these modalities are fibered and so the apparatus of MTT is necessary to include them into type theory. We further ensure that $(\mathbb{I} \to -) \dashv (-)_{\mathbb{I}}$ [Gra+21, Chapter 10]. Among the additional axioms with which we have extended $\mathrm{TT}_{\boxslash}$, the most important is the *synthetic quasicoherence* axiom inspired by Blechschmidt [Ble23]. This axiom roughly captures the property that cubical spaces classify (flat) distributive lattices [Spi16] and practically ensures that, in certain situations, we can control maps into $\mathbb{I}$:

Axiom 3 (Theorem 4.11 [Ble23])**.** *Fix R to be a finitely-presented distributive lattice over $\mathbb{I}$. The canonical map $R \to (\hom_{\mathbb{I}}(R, \mathbb{I}) \to \mathbb{I})$ is an equivalence.*

Using this apparatus, we may take proposition $\mathrm{Cov}\, A$ stating that a family $A : \mathbb{I} \to \mathcal{U}$ is covariant and *transpose* along $(\mathbb{I} \to -) \dashv (-)_{\mathbb{I}}$ to obtain the "amazingly covariant proposition" $\mathrm{ACov} : \mathcal{U} \to \mathcal{U}_{\mathbb{I}}$. Post-composing this with the action of $(-)_{\mathbb{I}}$ on the universe $(-)_{\mathbb{I}} : \mathcal{U}_{\mathbb{I}} \to \mathcal{U}$, we isolate a putative ∞-category of spaces: $\mathcal{S} = \sum_{A:\mathcal{U}_{\mathrm{simp}}} \mathrm{ACov}(A)_{\mathbb{I}}$, where $\mathcal{U}_{\mathrm{simp}}$ is the subuniverse of simplicial types. It remains to show that $\mathcal{S}$ is closed under the expected operations and to characterize maps $\mathbb{I} \to \mathcal{S}$. We have shown the following:

Theorem 4. *$\mathcal{S}$ is closed under identity types, dependent sums, and other standard connectives (notably,* not *dependent products).*

Note that failure of closure under dependent products is to be expected and in line with the semantics, as not every functor between ∞-categories is exponentiable.

Theorem 5. *An arrow $P : \mathbb{I} \to \mathcal{S}$ is equivalent to a function $P\,0 \to P\,1$.*

This last result implies that $\mathcal{S}$ is Segal and Rezk. Using some external reasoning in part, we have shown that $\mathcal{S}$ is a genuine simplicial space as opposed to a cubical space.

[3] We have avoided naming our type theory the more apt "cubical type theory" for obvious reasons.

Acknowledgments JW is grateful for financial support by the US Army Research Office under the MURI Grant W911NF-20-1-0082.

References

[Bar22] César Bardomiano Martínez. *Limits and exponentiable functors in simplicial homotopy type theory*. 2022. arXiv: 2202.12386. URL: https://arxiv.org/abs/2202.12386 (cit. on p. 1).

[Ble23] Ingo Blechschmidt. *A general Nullstellensatz for generalized spaces*. Rough draft. 2023 (cit. on p. 2).

[BW23] Ulrik Buchholtz and Jonathan Weinberger. "Synthetic fibered $(\infty, 1)$-category theory". In: *Higher Structures* 7 (1 2023), pp. 74–165. DOI: 10.21136/HS.2023.04 (cit. on p. 1).

[Cis19] Denis-Charles Cisinski. *Higher Categories and Homotopical Algebra*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2019. DOI: 10.1017/9781108588737. URL: http://www.mathematik.uni-regensburg.de/cisinski/CatLR.pdf (cit. on p. 1).

[Coh+17] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical Type Theory: a constructive interpretation of the univalence axiom". In: *IfCoLog Journal of Logics and their Applications* 4.10 (2017), pp. 3127–3169. arXiv: 1611.02108 [cs.LO] (cit. on p. 2).

[Gra+21] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. "Multimodal Dependent Type Theory". In: *Logical Methods in Computer Science* Volume 17, Issue 3 (July 2021). DOI: 10.46298/lmcs-17(3:11)2021. URL: https://lmcs.episciences.org/7713 (cit. on p. 2).

[Gra+20] Daniel Gratzer, G.A. Kavvos, Andreas Nuyts, and Lars Birkedal. "Multimodal Dependent Type Theory". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. ACM, 2020. DOI: 10.1145/3373718.3394736 (cit. on p. 2).

[KV20] Krzysztof Kapulkin and Vladimir Voevodsky. "A cubical approach to straightening". In: *Journal of Topology* 13.4 (2020), pp. 1682–1700. DOI: 10.1112/topo.12173 (cit. on p. 2).

[Kud23] Nikolai Kudasov. *Rzk*. An experimental proof assistant based on a type theory for synthetic ∞-categories. 2023. URL: https://github.com/rzk-lang/rzk (cit. on p. 1).

[KRW04] Nikolai Kudasov, Emily Riehl, and Jonathan Weinberger. "Formalizing the ∞-Categorical Yoneda Lemma". In: *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2004, pp. 274–290. DOI: 10.1145/3636501.3636945 (cit. on p. 1).

[Lic+18] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. "Internal Universes in Models of Homotopy Type Theory". In: *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*. Ed. by H. Kirchner. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018, 22:1–22:17. DOI: 10.4230/LIPIcs.FSCD.2018.22. eprint: 1801.07664 (cit. on p. 2).

[MW23] Louis Martini and Sebastian Wolf. "Internal higher topos theory". In: (2023). URL: https://arxiv.org/abs/2303.06437 (cit. on p. 1).

[Ras21] Nima Rasekh. "Quasi-categories vs. Segal spaces: Cartesian edition". English. In: *J. Homotopy Relat. Struct.* 16.4 (2021), pp. 563–604. ISSN: 2193-8407. DOI: 10.1007/s40062-021-00288-2 (cit. on p. 1).

[Ras24] Nima Rasekh. "A Model for the Higher Category of Higher Categories". In: *Theory and Applications of Categories* 41.2 (2024), pp. 21–70. URL: http://www.tac.mta.ca/tac/volumes/41/2/41-02abs.html (cit. on p. 1).

[Rez01] Charles Rezk. "A model for the homotopy theory of homotopy theory". In: *Transactions of the AMS* 353.3 (2001), pp. 973–1007 (cit. on p. 1).

[Rie23] Emily Riehl. "Could ∞-Category Theory Be Taught to Undergraduates?" In: *Notices of the American Mathematical Society* 70.5 (2023). DOI: 10.1090/noti2692 (cit. on p. 1).

[Rie24] Emily Riehl. "On the ∞-topos semantics of homotopy type theory". In: *Bulletin of the London Mathematical Society* 56.2 (2024), pp. 461–517. DOI: 10.1112/blms.12997 (cit. on p. 1).

[RS17] Emily Riehl and Michael Shulman. "A type theory for synthetic ∞-categories". In: *Higher Structures* 1 (1 2017), pp. 147–224. DOI: 10.21136/HS.2017.06 (cit. on p. 1).

[RV22] Emily Riehl and Dominic Verity. *Elements of ∞-Category Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2022. DOI: 10.1017/9781108936880 (cit. on p. 1).

[Sat19] Christian Sattler. *Idempotent completion of cubes in posets*. Online. 2019. URL: https://arxiv.org/abs/1805.04126v2 (cit. on p. 2).

[sHo24] The sHoTT Community. *sHoTT Library in Rzk*. 2024. URL: https://rzk-lang.github.io/sHoTT/ (cit. on p. 1).

[Shu19] Michael Shulman. *All $(\infty, 1)$-toposes have strict univalent universes*. 2019. arXiv: 1904.07004 [math.AT] (cit. on p. 1).

[Spi16] Bas Spitters. *Cubical sets and the topological topos*. 2016. arXiv: 1610.05270 [cs.LO] (cit. on p. 2).

[SW21] Thomas Streicher and Jonathan Weinberger. "Simplicial sets inside cubical sets". In: *Theory and Applications of Categories* 37.10 (2021), pp. 276–286 (cit. on p. 2).

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL: https://homotopytypetheory.org/book (cit. on p. 1).

[WL20] Matthew Z. Weaver and Daniel R. Licata. "A Constructive Model of Directed Univalence in Bicubical Sets". In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '20. ACM, July 2020. DOI: 10.1145/3373718.3394794 (cit. on p. 2).

[Wei22a] Jonathan Weinberger. "A Synthetic Perspective on $(\infty, 1)$-Category Theory: Fibrational and Semantic Aspects". PhD thesis. Technische Universität Darmstadt, 2022. DOI: 10.26083/tuprints-00020716 (cit. on p. 1).

[Wei22b] Jonathan Weinberger. *Strict stability of extension types*. 2022. URL: https://arxiv.org/abs/2203.07194 (cit. on p. 1).

Large Universe Construction by Indexed Induction-Recursion in Agda

Yuta Takahashi

Aomori University, Aomori, Japan
y.takahashi@aomori-u.ac.jp

Background and Aim. Martin-Löf [13] motivated the introduction of universe types to his type theory in terms of a reflection principle: "whatever we are used to doing with types can be done" inside a universe. Since its introduction, universe types in Martin-Löf type theory (MLTT) have been investigated and utilised in various ways (for an overview, see [6]). For instance, several methods of generic programming using universe types have been proposed [3, 2, 4, 19, 10]. Moreover, universe types in a system of MLTT with W-types are often treated as type-theoretic counterparts of large cardinals or sets. Aczel's constructive set theory **CZF** with a large set axiom is interpreted in a system of MLTT with the corresponding universe type (see, e.g., [1, 16, 15]). In the area of proof theory called ordinal analysis, ordinal notation systems using recursive analogues of large cardinals have been formulated, and well-ordering proofs for these notation systems were given in the corresponding systems of MLTT [17, 18].

In particular, Rathjen, Griffor and Palmgren [16] introduced the system $\mathbf{CZF}_\pi$, which is an extension of **CZF** with Mahlo's inaccessible sets of all transfinite orders [12]. They also introduced an extension of MLTT called **MLQ** and showed that $\mathbf{CZF}_\pi$ is interpretable in **MLQ**. The system **MLQ** has two large universe types M and Q: the latter is a "universe of universe operators" [14], and the former is a universe closed under the operators in Q. Roughly speaking, Q is an inductive type of codes for operators which provides universes closed under universe operators constructed previously.

The universe construction in **MLQ** was generalised by Palmgren [14]. He introduced a family $\mathbf{ML}^n$ of systems of higher-order universe operators. It was shown that **MLQ** is an instance of these systems (i.e., an instance of the system $\mathbf{ML}^3$) and higher-order construction was proposed further. The type Q in **MLQ** is considered as a universe of first-order universe operators whose elements are introduced by a second-order universe operator, and this construction is extended to the $(n+1)$-th order universe operators for an arbitrary natural number $n : \mathbb{N}$.

Yet another universe type corresponding to large sets is the Mahlo universe type introduced by Setzer [18]. A Mahlo universe $\mathbb{M}$ has a reflection property similar to that of weakly Mahlo cardinals: for any function f on families $\Sigma_{(x:\mathbb{M})}(\mathrm{T}_{\mathbb{M}}x \to \mathbb{M})$ of small types in $\mathbb{M}$, $\mathbb{M}$ has a subuniverse U_f which is closed under f. The resulting system is called **MLM**, and Setzer's purpose of introducing **MLM** is to obtain an extension of MLTT which is able to prove the well-ordering property of an ordinal notation system built on one recursively Mahlo ordinal.

A framework to unify the large universes above has been already provided by Dybjer-Setzer's theory of induction-recursion, which is a finite axiomatisation of Dybjer's general schema of simultaneous inductive-recursive definitions [5]. An *external* Mahlo universe, which is a proof-theoretically weak variant of a Mahlo universe, can be formulated by induction-recursion [7]. Moreover, in [8], induction-recursion was generalised to *indexed* induction-recursion, and it was shown that the systems $\mathbf{ML}^n$ of higher-order universe operators for an arbitrary $n : \mathbb{N}$ can be defined by indexed induction-recursion. Since the system **MLQ** is an instance of $\mathbf{ML}^n$ as seen above, this implies that one obtains the large universe types above in the theory **IIR** of indexed induction-recursion, except that this theory has defined a weak variant of Mahlo universes only.

Recently, Dybjer and Setzer [9] constructed Kahle-Setzer's predicative Mahlo universe [11] in MLTT with indexed induction-recursion which goes beyond that of [8].

Investigating large universe types in MLTT further, we first extend the Mahlo universe type by the higher-order construction in the style of $\mathbf{ML}^n$. We call this extended universe $\mathbb{MH}$ the *Mahlo universe type with higher-order subuniverses*, since each "subuniverse" U_f^n is closed under a given $(n+1)$-th order operator f on $\mathbb{M}$ and so contains the n-th order operators constructed by applying f. Subuniverses of the usual Mahlo universe are constructed in the case of the 0-th order operators. We then provide an Agda implementation of all large universe types above including the external variant of $\mathbb{MH}$, noting that the external variant of $\mathbb{MH}$ in fact coincides with the union $\bigcup_{n:\mathbb{N}} \mathbf{ML}^n$.[1] This implementation, which uses Agda's indexed induction-recursion, gives us a computer-checked formulation of $\mathbb{MH}$.

Formulation. Below we use the logical framework adopted by Agda. As seen above, the Mahlo universe type $\mathbb{M} : \mathrm{Set}$ has the following property: for any function $f : \Sigma_{(x:\mathbb{M})}(\mathrm{T}_{\mathbb{M}} x \to \mathbb{M}) \to \Sigma_{(x:\mathbb{M})}(\mathrm{T}_{\mathbb{M}} x \to \mathbb{M})$, there is a subuniverse $\widehat{\mathrm{U}}_f : \mathbb{M}$ with the decoding function (or the injection) $\widehat{\mathrm{T}}_f : \mathrm{U}_f \to \mathbb{M}$ such that U_f is closed under f. We define T_f as the composition $\mathrm{T}_f := \mathrm{T}_{\mathbb{M}} \circ \widehat{\mathrm{T}}_f$, then the closedness of U_f under f is represented by the two constructors $\mathsf{res}_f^1 : \Sigma_{(x:\mathrm{U}_f)}(\mathrm{T}_f x \to \mathrm{U}_f) \to \mathrm{U}_f$ and $\mathsf{res}_f^2 : (c : \Sigma_{(x:\mathrm{U}_f)}(\mathrm{T}_f x \to \mathrm{U}_f)) \to \mathrm{T}_f(\mathsf{res}_f^1 c) \to \mathrm{U}_f$ with the computation rules

$$\widehat{\mathrm{T}}_f(\mathsf{res}_f^1 c) = \mathsf{p}_1(f\ (\widehat{\mathrm{T}}_f(\mathsf{p}_1 c), \lambda y.\widehat{\mathrm{T}}_f(\mathsf{p}_2 c\, y))), \quad \widehat{\mathrm{T}}_f(\mathsf{res}_f^2 c\, a) = \mathsf{p}_2(f\ (\widehat{\mathrm{T}}_f(\mathsf{p}_1 c), \lambda y.\widehat{\mathrm{T}}_f(\mathsf{p}_2 c\, y)))\, a.$$

Informally, the constructors $\mathsf{res}_f^1, \mathsf{res}_f^2$ are the restriction of f to U_f:

$$\begin{array}{ccc} \Sigma_{(x:\mathrm{U}_f)}(\mathrm{T}_f x \to \mathrm{U}_f) & \xrightarrow{\mathsf{res}_f^1, \mathsf{res}_f^2} & \Sigma_{(x:\mathrm{U}_f)}(\mathrm{T}_f x \to \mathrm{U}_f) \\ \downarrow{\scriptstyle \widehat{\mathrm{T}}_f} & & \downarrow{\scriptstyle \widehat{\mathrm{T}}_f} \\ \Sigma_{(x:\mathbb{M})}(\mathrm{T}_{\mathbb{M}} x \to \mathbb{M}) & \xrightarrow[f]{} & \Sigma_{(x:\mathbb{M})}(\mathrm{T}_{\mathbb{M}} x \to \mathbb{M}) \end{array}$$

On the other hand, the higher-order universe operators in $\mathbf{ML}^n$ are defined via encoding to elements of universe types. We first define the type $\mathrm{O} : \mathbb{N} \to \mathrm{Set}_1$ of operators of finite order and the type $\mathrm{F} : \mathbb{N} \to \mathrm{Set}_1$ of families of operators simultaneously as $\mathrm{O}\ 0 = \mathrm{Set}$, $\mathrm{O}\ (n+1) = \mathrm{F}\ n \to \mathrm{F}\ n$ and $\mathrm{F}\ n = \Sigma_{(A:\mathrm{Set})}(A \to \mathrm{O}\ n)$. For an arbitrary natural number n and any families $\mathcal{P} = c_n, c_{n-1}, \ldots, c_0$ with $c_i : \mathrm{F}\ i$, codes for higher-order universe operators of $\mathbf{ML}^{n+1}$ are constructed from $\mathcal{P}$ as elements of the universes $\mathrm{U}^n(\mathcal{P}), \mathrm{U}^{n-1}(\mathcal{P}), \ldots, \mathrm{U}^0(\mathcal{P})$ of finite order operators, where $\mathrm{U}^0(\mathcal{P})$ is a universe type in the usual sense. For instance, a new element of type $\Sigma_{(x:\mathrm{U}^0(\mathcal{P}))}(\mathrm{T}^0(\mathcal{P})\ x \to \mathrm{U}^{k-1}(\mathcal{P}))$, i.e., a new family of $(k-1)$-th order operators is obtained by applying a code $o : \mathrm{U}^k(\mathcal{P})$ of a k-th order operator to a given family $c : \Sigma_{(x:\mathrm{U}^0(\mathcal{P}))}(\mathrm{T}^0(\mathcal{P})\ x \to \mathrm{U}^{k-1}(\mathcal{P}))$ of $(k-1)$-th order operators:

$$\begin{array}{ccc} \Sigma_{(x:\mathrm{U}^0(\mathcal{P}))}(\mathrm{T}^0(\mathcal{P})\ x \to \mathrm{U}^{k-1}(\mathcal{P})) & \xrightarrow{o} & \Sigma_{(x:\mathrm{U}^0(\mathcal{P}))}(\mathrm{T}^0(\mathcal{P})\ x \to \mathrm{U}^{k-1}(\mathcal{P})) \\ \downarrow{\scriptstyle \text{decoding}} & & \downarrow{\scriptstyle \text{decoding}} \\ \Sigma_{(A:\mathrm{Set})}(A \to \mathrm{O}\ (k-1)) & \xrightarrow[\mathrm{T}^n_{k-1} o]{} & \Sigma_{(A:\mathrm{Set})}(A \to \mathrm{O}\ (k-1)) \end{array}$$

We define the Mahlo universe type $\mathbb{MH}$ with higher-order subuniverses by mimicking the higher-order construction of $\mathbf{ML}^n$ *inside* the Mahlo universe. We first define the type $\mathrm{O}_{\mathbb{MH}}$ of

[1] https://github.com/takahashi-yt/large-universes

small operators of finite order and the type $\mathrm{F}_{\mathbb{MIH}}$ of families of such operators as $\mathrm{O}_{\mathbb{MIH}}\,0 = \mathbb{MIH}$, $\mathrm{O}_{\mathbb{MIH}}\,(i+1) = \mathrm{F}_{\mathbb{MIH}}\,i \to \mathrm{F}_{\mathbb{MIH}}\,i$ and $\mathrm{F}_{\mathbb{MIH}}\,i = \Sigma_{(x:\mathbb{MIH})}(\mathrm{T}_{\mathbb{MIH}}x \to \mathrm{O}_{\mathbb{MIH}}\,i)$. Let $\mathcal{P} : (n : \mathbb{N}) \to \mathrm{F}_{\mathbb{MIH}}\,n$ be families of finite order small operators. As a generalisation of the reflection property of the Mahlo universe type, the codes representing the restriction of the k-th order small operators in $\mathcal{P}\,k$ to the subuniverse $\mathrm{U}^k_{\mathbb{MIH}}(\mathcal{P})$ are introduced. Moreover, as in $\mathbf{ML}^n$, we define higher-order universe operators in $\mathbb{MIH}$ via encoding to the elements of $\mathbb{MIH}$'s subuniverses. Roughly, applying a code $o : \mathrm{U}^k_{\mathbb{MIH}}(\mathcal{P})$ of a k-th order small operator to $c : \Sigma_{(x:\mathrm{U}^0_{\mathbb{MIH}}(\mathcal{P}))}(\mathrm{T}^0_{\mathbb{MIH}}(\mathcal{P})\;x \to \mathrm{U}^{k-1}_{\mathbb{MIH}}(\mathcal{P}))$ provides a new family of $(k-1)$-th order small operators as below:

$$\begin{array}{ccc}
\Sigma_{(x:\mathrm{U}^0_{\mathbb{MIH}}(\mathcal{P}))}(\mathrm{T}^0_{\mathbb{MIH}}(\mathcal{P})\;x \to \mathrm{U}^{k-1}_{\mathbb{MIH}}(\mathcal{P})) & \xrightarrow{\;o\;} & \Sigma_{(x:\mathrm{U}^0_{\mathbb{MIH}}(\mathcal{P}))}(\mathrm{T}^0_{\mathbb{MIH}}(\mathcal{P})\;x \to \mathrm{U}^{k-1}_{\mathbb{MIH}}(\mathcal{P})) \\
\Big\downarrow{\scriptstyle \text{decoding}} & & \Big\downarrow{\scriptstyle \text{decoding}} \\
\Sigma_{(x:\mathbb{MIH})}(\mathrm{T}^0_{\mathbb{MIH}}\;x \to \mathrm{O}_{\mathbb{MIH}}\;(k-1)) & \xrightarrow[\mathrm{T}^k_{\mathbb{MIH}}(\mathcal{P})\;o]{} & \Sigma_{(x:\mathbb{MIH})}(\mathrm{T}^0_{\mathbb{MIH}}\;x \to \mathrm{O}_{\mathbb{MIH}}\;(k-1))
\end{array}$$

References

[1] Peter Aczel. The type theoretic interpretation of constructive set theory: Inductive definitions. In R. B. Marcus, G. J. Dorn, and G. J. W. Dorn, editors, *Logic, Methodology, and Philosophy of Science VII*, pages 17–49. North-Holland, 1986.

[2] Thorsten Altenkirch, Conor McBride, and Peter Morris. Generic programming with dependent types. In Roland Backhouse, Jeremy Gibbons, Ralf Hinze, and Johan Jeuring, editors, *Datatype-Generic Programming*, pages 209–257, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[3] Marcin Benke, Peter Dybjer, and Patrik Jansson. Universes for generic programs and proofs in dependent type theory. *Nord. J. Comput.*, 10(4):265–289, 2003.

[4] James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. The gentle art of levitation. In Paul Hudak and Stephanie Weirich, editors, *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*, pages 3–14. ACM, 2010.

[5] Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *J. Symb. Log.*, 65(2):525–549, 2000.

[6] Peter Dybjer and Erik Palmgren. Intuitionistic Type Theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2020 edition, 2020.

[7] Peter Dybjer and Anton Setzer. Induction-recursion and initial algebras. *Ann. Pure Appl. Log.*, 124(1-3):1–47, 2003.

[8] Peter Dybjer and Anton Setzer. Indexed induction-recursion. *J. Log. Algebraic Methods Program.*, 66(1):1–49, 2006.

[9] Peter Dybjer and Anton Setzer. The extended predicative Mahlo universe in Martin-Löf type theory. *Journal of Logic and Computation*, 05 2023.

[10] Lucas Escot and Jesper Cockx. Practical generic programming over a universe of native datatypes. *Proc. ACM Program. Lang.*, 6(ICFP), aug 2022.

[11] Reinhard Kahle and Anton Setzer. An Extended Predicative Definition of the Mahlo Universe. In Ralf Schindler, editor, *Ways of Proof Theory*, pages 315–340. De Gruyter, Berlin, Boston, 2010.

[12] Paul Mahlo. Über lineare transfinite Mengen. *Berichte über die Verhandlungen der Königlich Sächsischen Gesellschaft der Wissenschaften zu Leipzig. Mathematisch-Physische Klasse*, 63:187–225, 1911.

[13] Per Martin-Löf. An intuitionistic theory of types. In G. Sambin and Jan M. Smith, editors, *Twenty-five years of constructive type theory*, volume 36 of *Oxford Logic Guides*, pages 127–172. Clarendon Press, 1998.

[14] Erik Palmgren. On universes in type theory. In Giovanni Sambin and Jan M. Smith, editors, *Twenty Five Years of Constructive Type Theory*, Oxford Logic Guides, pages 191–204. Oxford University Press, 1998.

[15] Michael Rathjen. Realizing Mahlo set theory in type theory. *Arch. Math. Log.*, 42(1):89–101, 2003.

[16] Michael Rathjen, Edward R. Griffor, and Erik Palmgren. Inaccessibility in constructive set theory and type theory. *Ann. Pure Appl. Log.*, 94(1-3):181–200, 1998.

[17] Anton Setzer. Well-ordering proofs for Martin-Löf type theory. *Ann. Pure Appl. Logic*, 92(2):113–159, 1998.

[18] Anton Setzer. Extending Martin-Löf type theory by one Mahlo-universe. *Arch. Math. Log.*, 39(3):155–181, 2000.

[19] Stephanie Weirich and Chris Casinghino. Generic programming with dependent types. In Jeremy Gibbons, editor, *Generic and Indexed Programming: International Spring School, SSGIP 2010, Oxford, UK, March 22-26, 2010, Revised Lectures*, pages 217–258. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

A Canonical Form for Universe Levels in Impredicative Type Theory

Yoan Géran[1,2]

[1] Centre de Recherche en Informatique, Mines Paris - PSL University France
[2] Université Paris-Saclay, INRIA project Deducteam, Laboratoire de Méthodes Formelles, ENS Paris-Saclay, 91190 France
yoan.geran@minesparis.psl.eu

To allow quantification over Type, the Calculus of Constructions [5] can be extended with a countable sequence of universes $\mathtt{U}_0 \colon \mathtt{U}_1 \colon \cdots$ [4], the indices being referred to as universe levels (here, we consider $\mathtt{U}_0$ to be the universe of the propositions). In order to provide an impredicative universe of propositions, we need to use a slightly different maximum for the typing rule of the dependent product, to ensure that quantifying over any universe to produce a proposition yields a product in $\mathtt{U}_0$.

Impredicative maximum is the function $\mathrm{imax} \colon \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ defined by $\mathrm{imax}(n, 0) = 0$ and $\mathrm{imax}(n, S(m)) = \max(n, S(m))$. It is used to type dependent products in the Calculus of Constructions with universes (CC^∞), which typing rule is

$$(\textsc{Prod})\ \frac{\Gamma \vdash A \colon \mathtt{U}_i \qquad \Gamma, x \colon A \vdash B \colon \mathtt{U}_j}{\Gamma \vdash \Pi x \colon A \cdot B \colon \mathtt{U}_{\mathrm{imax}(i,j)}}$$

The imax function is easily evaluated and universes such as $\mathtt{U}_1$ and $\mathtt{U}_{\mathrm{imax}(1,1)}$ are the same since 1 and $\mathrm{imax}(1, 1)$ are evaluated to the same expression.

Universe polymorphism allows quantification over universes [12, 10, 6]. We obtain $\mathrm{CC}^\infty_\forall$ by adding level variables and prenex quantification on them to CC^∞. The universe polymorphic identity is then the term $\lambda i \colon \mathbb{L} \cdot \lambda A \colon \mathtt{U}_i \cdot \lambda x \colon A \cdot x$, where $\mathbb{L}$ is the set of the levels, defined with the grammar

$$\ell \coloneqq 0 \mid S(\ell) \mid \max(\ell, \ell) \mid \mathrm{imax}(\ell, \ell) \mid x$$

where x is an element of a countable set of variables $\mathcal{X}$.

Level equivalence is more difficult to check with level variables. Indeed, we cannot reduce level expressions anymore, and equivalence is no more the syntactic equality since $\max(x, S(x))$ and $S(x)$ for instance, are equivalent. Besides, $\mathtt{U}_{\max(x,S(x))}$ and $\mathtt{U}_{S(x)}$ should be identified, which motivates equivalence checking procedures in the imax-successor algebra.

Definition 1. *A function $\sigma \colon \mathcal{X} \to \mathbb{N}$ is called a valuation. For all valuations σ, we define inductively the value of a level ℓ over σ, denoted as $[\![\ell]\!]_\sigma$, with*

$$[\![0]\!]_\sigma = 0 \qquad [\![S(\ell)]\!]_\sigma = 1 + [\![\ell]\!]_\sigma \qquad [\![x]\!]_\sigma = \sigma(x)$$
$$[\![\max(\ell_1, \ell_2)]\!]_\sigma = \max\big([\![\ell_1]\!]_\sigma, [\![\ell_2]\!]_\sigma\big) \qquad [\![\mathrm{imax}(\ell_1, \ell_2)]\!]_\sigma = \mathrm{imax}\big([\![\ell_1]\!]_\sigma, [\![\ell_2]\!]_\sigma\big)$$

Definition 2. *Let $\ell_1, \ell_2 \in \mathbb{L}$. We say that $\ell_1 \leqslant \ell_2$ if for all valuations σ, $[\![\ell_1]\!]_\sigma \leqslant [\![\ell_2]\!]_\sigma$. In the same way, we say that $\ell_1 \equiv \ell_2$ if for all valuations σ, $[\![\ell_1]\!]_\sigma = [\![\ell_2]\!]_\sigma$.*

Motivation. Our main motivation lies in the definition of $\mathrm{CC}^{\infty}_{\forall}$ in the logical framework $\lambda\Pi/\equiv$ [2]. We aim the translation of the theories of Coq and Lean into implementations of $\lambda\Pi/\equiv$ such as Dedukti[1], Lambdapi[11] or Kontroli[7]. This requires a level translation where equivalent levels are convertible. Since convertibility in $\lambda\Pi/\equiv$ is modulo a set of rewrite rules, a computable canonical from could help to make equivalent levels convertible (by rewriting them to their canonical form), whereas an algorithm to check inequality, as presented in [3], seems unsuitable. The predicative case of Agda has been handled by Genestier in [9] and Férey also worked on the encoding of universe polymorphism [8].

Contribution. We study the imax-successor algebra, and we provide a canonical form for its terms. This gives us a way to decide level equivalence by syntactic comparison of the canonical form.

In [13], Voevodsky represented each level of the predicative case (so with max instead of imax) as a maximum of terms that do not contain maximum. Here, we follow the same idea. We establish equivalences that permit to pull max and imax out of imax and S, and we obtain a restricted grammar for the levels.

Theorem 1. *For all $t \in \mathbb{L}$, there exists $u_1, \dots, u_n$ in the grammar*

$$\ell \coloneqq S^k(x) \mid S^{k+1}(0) \mid \mathrm{imax}(\ell, x)$$

such that $t \equiv \max(u_1, \dots, u_n)$.

However, the goal is not yet reached. For instance $\max(\mathrm{imax}(x, y), x) \equiv \max(x, y)$. The imax function is too complex: its second argument should always be taken into account while its first one is taken into account under certain conditions. Then, we want to replace imax with a simpler primitive. That is why we extend the levels with two symbols $\mathcal{V}$ and $\mathcal{C}$.

Definition 3 (Extended levels)**.** *An extended level is a term of the grammar*

$$\ell \coloneqq 0 \mid S(\ell) \mid \max(\ell, \ell) \mid \mathrm{imax}(\ell, \ell) \mid x \mid \mathcal{V}(\{\ell, \dots, \ell\}, \ell, k) \mid \mathcal{C}(\{\ell, \dots, \ell\}, k)$$

where $k \in \mathbb{N}$. We extend $[\![\cdot]\!]_\sigma$ and the level comparison to the extended levels with

$$[\![\mathcal{V}(E, u, k)]\!]_\sigma = \begin{cases} 0 & \text{if } \exists v \in E, [\![v]\!]_\sigma = 0 \\ [\![u]\!]_\sigma + k & \text{else} \end{cases} \quad \text{and} \quad [\![\mathcal{C}(E, k)]\!]_\sigma = \begin{cases} 0 & \text{if } \exists u \in E, [\![u]\!]_\sigma = 0 \\ k & \text{else} \end{cases}$$

These symbols permit getting rid of imax since $\mathrm{imax}(u, v) \equiv \max(\mathcal{V}(\{v\}, u, 0), v)$. Then, we obtain a notion of canonical sublevels sufficient to express any level.

Definition 4 (Canonical sublevels)**.** *A canonical sublevel is an element of the set*

$$\mathbf{S} = \{\mathcal{V}(E, x, k) \mid E \subset \mathcal{X}, x \in E\} \cup \{\mathcal{C}(E, k) \mid E \subset \mathcal{X}, k > 0\}.$$

Theorem 2. *Let $t \in \mathbb{L}$. Then there exists a finite $U \subset \mathbf{S}$ such that $t \equiv \max U$.*

Elements of $\mathbf{S}$ are easily comparable, and their equivalence is the syntactic equality. This is the key of our main result. We introduce the notion of minimal representation and show that a level has one and only one minimal representation.

Definition 5. *Let $U \subset \mathbf{S}$. We say that $\max U$ is minimal if and only if for all $u, v \in U$ such that $u \leqslant v$, we have $u = v$. We denote by $\mathbf{R}$ the set of such terms $\max U$.*

Theorem 3 (Representation)**.** *For all $t \in \mathbb{L}$, there exists a unique $U \in \mathbf{R}$ such that $t \equiv U$. We say that U is the minimal representation of t.*

Finally, we propose an algorithm to compute the minimal representation of any extended level (having an algorithm for all the extended levels permits to implement variable substitution in minimal representation). It offers a new decision procedure for equality in the imax-successor algebra and this representation is used in a WIP translator from Lean to Dedukti[1].

Minimal representations also give us a simple procedure to decide level inequality since for all $U, V \in \mathbf{R}$, $U \leqslant V$ if and only if for all $u \in U$, there exists $v \in V$ such that $u \leqslant v$.

References

[1] Ali Assaf et al. "Dedukti : a Logical Framework based on the λΠ-Calculus Modulo Theory". In: 2016.

[2] Mathieu Boespflug, Quentin Carbonneaux, and Olivier Hermant. "The λΠ-calculus Modulo as a Universal Proof Language". In: *CEUR Workshop Proceedings* 878 (June 2012).

[3] Mario Carneiro. "The Type Theory of Lean". MA thesis. Carnegie Mellon University, 2019, p. 44. URL: https://github.com/digama0/lean-type-theory/releases.

[4] Thierry Coquand. "An Analysis of Girard's Paradox". In: *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS 1986)*. Cambridge, MA, USA: IEEE Computer Society Press, June 1986, pp. 227–236.

[5] Thierry Coquand and Gérard Huet. "The calculus of constructions". In: *Information and Computation* 76.2 (1988), pp. 95–120. ISSN: 0890-5401. DOI: https://doi.org/10.1016/0890-5401(88)90005-3. URL: https://www.sciencedirect.com/science/article/pii/0890540188900053.

[6] Judicaël Courant. "Explicit Universes for the Calculus of Constructions". In: *Theorem Proving in Higher Order Logics*. Ed. by Victor A. Carreño, César A. Muñoz, and Sofiène Tahar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 115–130. ISBN: 978-3-540-45685-8.

[7] Michael Färber. "Safe, Fast, Concurrent Proof Checking for the Lambda-Pi Calculus modulo Rewriting". In: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2022. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 225–238. ISBN: 9781450391825. DOI: 10.1145/3497775.3503683. URL: https://doi.org/10.1145/3497775.3503683.

[8] Gaspard Férey. "Higher-Order Confluence and Universe Embedding in the Logical Framework . (Confluence d'ordre supérieur et encodage d'univers dans le Logical Framework". PhD thesis. École normale supérieure Paris-Saclay, France, 2021. URL: https://lmf.cnrs.fr/downloads/Perso/Ferey-thesis.pdf.

[9] Guillaume Genestier. "Encoding Agda Programs Using Rewriting". In: *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*. Ed. by Zena M. Ariola. Vol. 167. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 31:1–31:17. ISBN: 978-3-95977-155-9. DOI: 10.4230/LIPIcs.FSCD.2020.31. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12353.

[1] https://github.com/Deducteam/lean2dk

[10] Robert Harper and Robert Pollack. “Type Checking with Universes”. In: *2nd International Joint Conference on Theory and Practice of Software Development*. TAPSOFT ’89. Barcelona, Spain: Elsevier Science Publishers B. V., 1991, pp. 107–136.

[11] Gabriel Hondet and Frédéric Blanqui. “The New Rewriting Engine of Dedukti (System Description)”. In: *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*. Ed. by Zena M. Ariola. Vol. 167. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 35:1–35:16. ISBN: 978-3-95977-155-9. DOI: 10.4230/LIPIcs.FSCD.2020.35. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12357.

[12] Matthieu Sozeau and Nicolas Tabareau. “Universe Polymorphism in Coq”. In: *Interactive Theorem Proving*. Ed. by Gerwin Klein and Ruben Gamboa. Cham: Springer International Publishing, 2014, pp. 499–514. ISBN: 978-3-319-08970-6.

[13] Vladimir Voevodsky. *A universe polymorphic type system*. An unfinished unreleased manuscript. Oct. 2014. URL: https://www.math.ias.edu/Voevodsky/files/files-annotated/Dropbox/Unfinished_papers/Type_systems/UPTS_current/Universe_polymorphic_type_sytem.pdf.

Predicativity of the Mahlo Universe in Type Theory

Peter Dybjer[1] and Anton Setzer[2]

[1] Dept. of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden http://www.cse.chalmers.se/%7Epeterd/
[2]Swansea University, Dept. of Computer Science, Swansea, UK
https://www.swansea.ac.uk/staff/a.g.setzer/ https://csetzer.github.io/

Abstract

We provide a constructive, predicative justification of Setzer's Mahlo universe in type theory. Our approach is closely related to Kahle and Setzer's construction of an extended predicative Mahlo universe in Feferman's Explicit Mathematics. However, we work directly in Martin-Löf type theory extended with a Mahlo universe and obtain informal meaning explanations which extend (and slightly modify) those in Martin-Löf's article *Constructive Mathematics and Computer Programming*. We also present mathematical models in set-theoretic metalanguage and explain their relevance to the informal meaning explanations.

Martin-Löf's first published paper on type theory was entitled "An intuitionistic type theory: predicative part" [12]. This theory had an infinite hierarchy of universes. Its proof-theoretic strength was determined to be Γ_0 [5, 6], the limit of predicativity in Feferman and Schütte's sense [20, 9, 8, 4, 18, 17]. In his article *Constructive Mathematics and Computer Programming* [13] Martin-Löf added W-types, and the theory became impredicative in Feferman's and Schütte's sense. Nevertheless, Martin-Löf still considered the theory predicative in an extended sense, since *meaning explanations* were given suggesting how the types and terms of the theory are built up from below. They explain how the objects of the theory are trees that are built by a well-founded process of repeated lazy evaluation of expressions to canonical form.

Martin-Löf type theory was later extended with several higher universe constructions, such as Palmgren's universe operators, the super universe [15], quantifier universes [16], and Setzer's Mahlo universe [19]. All these extensions were intended to be constructive and predicative in the sense of Martin-Löf's meaning explanations. However, the predicativity of the Mahlo universe was not so clear, especially after Palmgren [15] discovered that adding a natural elimination rule for it led to an inconsistency. Maybe Mahlo is a natural limit of Martin-Löf's extended predicativity as we conjectured in our paper on a finite axiomatisation of inductive-recursive definitions [2]?

A universe in type theory is a type closed under all the standard type formers of Martin-Löf type theory, such as $\Pi, \Sigma, 0, 1, 2, \mathrm{N}, \mathrm{W}$, and the identity type I. Universes can either be formulated à la Russell, where an element $A : \mathrm{U}$ is also a type A, or à la Tarski, where an element $a : \mathrm{U}$ is a "name" or "code" of a type A and there is a decoding map T such that $\mathrm{T}\, a = A$.

A super universe is a universe closed under an operator on families of sets that maps a universe $(\mathrm{U}_n, \mathrm{T}_n)$ to the next universe $(\mathrm{U}_{n+1}, \mathrm{T}_{n+1})$ in the hierarchy. One can then form an operator mapping a super universe to the next and form a super^2 universe closed under this operator. This process can be iterated and thus one obtains super^n universes. More generally, one can define universes closed under arbitrary operators on families of sets. A Mahlo universe is a universe that contains all universes generated by family operators. Moreover, the latter are *subuniverses* of the Mahlo universe. One can show that these subuniverses arise as special cases of the inductive-recursive definitions in our theory **IR** [2]. This theory is formulated

as an extension of Martin-Löf's logical framework [14], where there is a type Set of "sets" in Martin-Löf's sense: "to know a *set* is to know how the elements of the set are formed and how equal elements are formed", a phrase indicating that sets should be inductively (or inductive-recursively) generated. Therefore we will refer to $\Pi, \Sigma, 0, 1, 2, \mathrm{N}, \mathrm{W}, \mathrm{I}$, etc, as *set formers* rather than type formers, when we work in this version of type theory. (Note that Martin-Löf's notion of "set" is different from the "h-sets" in homotopy type theory.)

Let f be an operator on families of sets split into two components (f_0, f_1) where f_0 returns the index set and f_1 returns the family. Then we can define a subuniverse $\mathrm{U}\, f_0\, f_1 : \mathrm{Set}$ with decoding $\mathrm{T}\, f_0\, f_1 : \mathrm{U}\, f_0\, f_1 \to \mathrm{Set}$ as an instance of an inductive-recursive definition in **IR**. In this way Set encodes Setzer's Mahlo universe [19]. We call it an *external* Mahlo universe to contrast it with the *internal* Mahlo universe that arises if we introduce a set $\mathrm{M} : \mathrm{Set}$ with the Mahlo property. This M goes beyond inductive-recursive definitions in **IR**.

When Martin-Löf extended his meaning explanation to the 1986 version based on a logical framework [14], he did *not* stipulate that "to know a *type* is to know how the objects of the type are formed and how equal objects are formed". (We refer to Martin-Löf's Leiden lectures [10, 11] for a comprehensive account of the philosophical foundations of intuitionistic type theory with the distinction between types and sets.) The type Set is to be understood as "open" to extension with new inductive(-recursively) defined sets when we need them. Hence it is not natural to add an elimination rule for it. In contrast to this, $\mathsf{M} : \mathrm{Set}$ is to be understood as "closed". Nevertheless, as Palmgren showed, adding a natural elimination rule for it leads to an inconsistency. This paradox makes us doubt whether the internal Mahlo universe is a good predicative set according to Martin-Löf's conception.

In spite of this, we shall argue that the Mahlo universe is predicative and constructive by giving Martin-Löf style meaning explanations for it. Our argument can be applied both to the external and internal Mahlo universe, although we only discuss the simpler external version.

We first construct a crude set-theoretic model of logical framework-based type theory with Set as a Mahlo universe and $\mathrm{U}\, f_0\, f_1 : \mathrm{Set}$ with decoding $\mathrm{T}\, f_0\, f_1 : \mathrm{U}\, f_0\, f_1 \to \mathrm{Set}$ as subuniverses. This model is an adaptation of our model of **IR** [2], where type-theoretic function spaces are interpreted as sets of all set-theoretic functions. We work in the classical set theory **ZFC** with a Mahlo cardinal M and an inaccessible cardinal I above it. (Note that we use the term "set" both for sets in Martin-Löf type theory and for sets in the set-theoretic metalanguage. We hope this will not lead to confusion.) We interpret the collection of all types as V_{I} and Set as V_{M}. Let $\mathcal{F}am(V) = \{(X, Y) \mid X \in V, Y : X \to V\}$ be the set of families of sets in V. An operator on families of sets in the type theory is interpreted as a function $f : \mathcal{F}am(\mathrm{V}_{\mathsf{M}}) \to \mathcal{F}am(\mathrm{V}_{\mathsf{M}})$. We then use the Mahlo property of M to show that there is an inaccessible cardinal $\kappa_f < \mathsf{M}$ such that $f : \mathcal{F}am(\mathrm{V}_{\kappa_f}) \to \mathcal{F}am(\mathrm{V}_{\kappa_f})$ and interpret the subuniverses $\mathrm{U}\, f_0\, f_1$ as $\mathcal{U}\, f_0\, f_1 = \mathrm{V}_{\kappa_f}$ à la Russell, that is, the decoding $\mathrm{T}\, f_0\, f_1$ is interpreted as the injection $\mathcal{T}\, f_0\, f_1 : \mathrm{V}_{\kappa_f} \hookrightarrow \mathrm{V}_{\mathsf{M}}$.

We then construct a second "predicative" set-theoretic model where we interpret the inductive-recursively defined type-theoretic subuniverses $(\mathrm{U}\, f_0\, f_1, \mathrm{T}\, f_0\, f_1)$ in set theory in terms of inductively generated graphs $\mathcal{T}\, f_0\, f_1$ with domain $\mathcal{U}\, f_0\, f_1$ in the standard set-theoretic way following Allen [1]. Moreover, in order to interpret Set as an inductively defined set $\mathcal{S}et$ we make use of Kahle and Setzer's extended predicative Mahlo universe in Feferman's theory of Explicit Mathematics [7, 3]. The key idea is that it suffices to require that the family operator f on families of sets is total on families over the subuniverse $\mathcal{U}\, f_0\, f_1$ when we add $\mathcal{U}\, f_0\, f_1$ to $\mathcal{S}et$. Although this may seem impredicative, we show that it results in an inductive definition of $\mathcal{S}et \subseteq \mathrm{V}_{\mathsf{M}}$. Moreover, we show that $\mathcal{T}\, f_0\, f_1 : \mathcal{U}\, f_0\, f_1 \to \mathcal{S}et$.

The final step is to provide Martin-Löf style meaning explanations inspired by the second set-theoretic model. The usual situation in type theory is that the meaning explanation for a

type is determined by the formation rule and the introduction rules, and the computation rules for the elimination constant is given by the equality rules. However, in the case of the Mahlo universe this pattern is broken. If we take the formation rule for the subuniverses $\mathrm{U}\, f_0\, f_1$ as a type-checking condition (a *matching condition*), then we get a non-wellfounded type-checking process, because of Palmgren's paradox. (We remark that "type checking" here refers to the matching of canonical terms with canonical types in Martin-Löf's meaning explanations, and not to the type-checking of judgements in intensional type theory, as implemented in proof-assistants.)

Instead we let the second set-theoretic model suggest the type-checking conditions. As an example we give one of the type-checking conditions for the judgement $A : \mathrm{Set}$. If A has canonical form $\mathrm{U}\, f_0\, f_1$, then we check whether

$$f_0\,(\mathrm{T}\, f_0\, f_1 u)\,(\lambda x.\mathrm{T}\, f_0\, f_1(tx)) : \mathrm{Set}$$

in the context $u : \mathrm{U}\, f_0\, f_1, t : \mathrm{T}\, f_0\, f_1\, u \to \mathrm{U}\, f_0\, f_1$, and

$$f_1\,(\mathrm{T}\, f_0\, f_1 u)\,(\lambda x.\mathrm{T}\, f_0\, f_1(tx))\, y : \mathrm{Set}$$

in the context $u : \mathrm{U}\, f_0\, f_1, t : \mathrm{T}\, f_0\, f_1\, u \to \mathrm{U}\, f_0\, f_1, y : f_0\,(\mathrm{T}\, f_0\, f_1 u)\,(\lambda x.\mathrm{T}\, f_0\, f_1(tx))$. We avoid the circularity by only type-checking for arguments in the image of $\mathrm{T}\, f_0\, f_1 : \mathrm{U}\, f_0\, f_1 \to \mathrm{Set}$ and this avoids checking for $\mathrm{U}\, f_0\, f_1 : \mathrm{Set}$. Nevertheless, the formation rule for $\mathrm{U}\, f_0\, f_1$ can be justified on this basis.

References

[1] Stuart Allen. A non-type-theoretic definition of Martin-Löf's types. In *Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987*, pages 215–221, 1987. Available from https://ecommons.cornell.edu/server/api/core/bitstreams/c714d33b-6c35-4958-9e54-20845ba11ca4/content.

[2] Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, April 1999. doi:10.2307/2586554.

[3] Peter Dybjer and Anton Setzer. The extended predicative Mahlo universe in Martin-Löf type theory. *Journal of Logic and Computation*, May 2023. doi:10.1093/logcom/exad022.

[4] Solomon Feferman. Systems of predicative analysis. *The Journal of Symbolic Logic*, 29(1):pp. 1–30, 1964. URL: http://www.jstor.org/stable/2269764.

[5] Solomon Feferman. Iterated inductive fixed-point theories: Application to Hancock's conjecture. In George Metakides, editor, *Patras Logic Symposion*, volume 109 of *Studies in Logic and the Foundations of Mathematics*, pages 171 – 196. Elsevier, 1982. doi:10.1016/S0049-237X(08)71364-8.

[6] Peter Hancock. *Ordinals and interactive programs*. PhD thesis, LFCS, University of Edinburgh, 2000. URL: https://era.ed.ac.uk/bitstream/handle/1842/376/ECS-LFCS-00-421.pdf?sequence=2&isAllowed=y.

[7] Reinhard Kahle and Anton Setzer. An extended predicative definition of the Mahlo universe. In Ralf Schindler, editor, *Ways of Proof Theory*, Ontos Series in Mathematical Logic, pages 309 – 334, Berlin, Boston, 2010. De Gruyter. doi:10.1515/9783110324907.315.

[8] Georg Kreisel. La prédicativité. *Bulletin de la Société Mathématique de France*, 88:371–391, 1960. Available from http://archive.numdam.org/article/BSMF_1960__88__371_0.pdf. URL: http://eudml.org/doc/86990.

[9] Georg Kreisel et al. Ordinal logics and the characterization of informal concepts of proof. In *Proceedings of the International Congress of Mathematicians*, volume 14, pages 289–299. Cambridge University Press, 1958. URL: https://www.mathunion.org/fileadmin/ICM/Proceedings/ICM1958/ICM1958.ocr.pdf.

[10] Per Martin-Löf. Philosophical aspects of intuitionistic type theory. Lectures given at the Faculteit der Wijsbegeerte, Rijksuniversiteit Leiden, 23 September – 16 December, 1993, edited and transferred to LaTeXby Ansten Klev. URL: https://pml.flu.cas.cz/uploads/PML-LeidenLectures93.pdf.

[11] Per Martin-Löf. Sets, types, and categories. Lecture given at the Symposium of Constructive Type Theory, Rijksuniversiteit Leiden, 6 February 2004, edited and transferred to LaTeXby Ansten Klev. URL: https://pml.flu.cas.cz/uploads/PML-Leiden06Feb04.pdf.

[12] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975. doi:10.1016/S0049-237X(08)71945-1.

[13] Per Martin-Löf. Constructive mathematics and computer programming. In L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski, editors, *Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. North-Holland, 1982. doi:10.1016/S0049-237X(09)70189-2.

[14] Bengt Nordström, Kent Petersson, and Jan Smith. *Programming in Martin-Löf's Type Theory: an Introduction*. Oxford University Press, 1990. Book out of print. Online version available via http://www.cs.chalmers.se/Cs/Research/Logic/book/.

[15] Erik Palmgren. On universes in type theory. In G. Sambin and J.M. Smith, editors, *Twenty-five years of constructive type theory: Proceedings of a Congress held in Venice, October 1995*, volume 36, pages 191 – 204. Oxford University Press, 1998. https://global.oup.com/academic/product/twenty-five-years-of-constructive-type-theory-9780198501275?cc=gb&lang=en&.

[16] Michael Rathjen, Edward Griffor, and Erik Palmgren. Inaccessibility in constructive set theory and type theory. *Annals of Pure and Applied Logic*, 94(1-3):181–200, 1998. doi:10.1016/S0168-0072(97)00072-9.

[17] Kurt Schütte. Eine Grenze Für die Beweisbarkeit der Transfiniten Induktion in der Verzweigten Typenlogik. *Archive for Mathematical Logic*, 7:45–60, 1964. doi:10.1007/BF01972460.

[18] Kurt Schütte. Predicative well-orderings. In J.N. Crossley and M.A.E. Dummett, editors, *Formal Systems and Recursive Functions*, volume 40 of *Studies in Logic and the Foundations of Mathematics*, pages 280–303. Elsevier, 1965. URL: https://www.sciencedirect.com/science/article/pii/S0049237X0871694X, doi:https://doi.org/10.1016/S0049-237X(08)71694-X.

[19] Anton Setzer. Extending Martin-Löf type theory by one Mahlo-universe. *Arch. Math. Log.*, 39:155 – 181, 2000. doi:10.1007/s001530050140.

[20] Nik Weaver. Predicativity beyond Γ_0, 2009. arXiv:math/0509244.

Session 24: Models of Type Theory

Semantics of Axiomatic Type Theory

Daniël Otten, University of Amsterdam
Matteo Spadetto, University of Leeds

Overview. We compare two semantics for type theory: *comprehension categories* [Jac93], which closely follow the syntax and intricacies of type theory, and *path categories* [vdBM18], which are relatively simple structures that take inspiration from homotopy theory. Both are only semantics in a weak way because they only specify substitutions up to isomorphism. However, it is known that the class of comprehension categories enjoys *coherence*: we can turn comprehension categories into actual models by 'splitting' them [LW15, Boc22]. We show that this can also be done for path categories by proving an equivalence between path categories and certain comprehension categories. Specifically, we show that the 2-category of path categories is equivalent to the 2-category of comprehension categories that have:

- contextuality (every context must be build out of a finite number of types),
- $=$-types with only a propositional β-rule,
- Σ-types with a definitional β-rule and a definitional η-rule,

in a weakly stable way. Here, weak stability is a technical condition and precisely what we need to obtain a (genuine) model by applying the left-adjoint splitting functor. Thanks to this 2-categorical equivalence, our coherence result for path categories follows by observing that the left-adjoint splitting functor preserves these characteristics. This result makes it precise how path categories provide semantics for a minimal dependent type theory: one with only $=$-types and, moreover, where the β-rule is only an *axiom* (a propositional equality) and not a *reduction* (a definitional equality). This is the notion that appears in Cubical Type Theory [CCHM18] as well as in Axiomatic Type Theory (type theory without reductions).

Path categories make no distinction between contexts and types: $\Gamma, x : A, y : B[x]$ doubles as $\Gamma, z : \Sigma(x : A)\, B[x]$. Hence, comprehension categories associated to path categories are endowed with Σ-types. Because this is not always desirable, we also introduce a more fine-grained notion: that of a *display path category* where we do make this distinction. We show that display path categories are equivalent to contextual comprehension categories only endowed with propositional $=$-types in a weakly stable way, without any assumption regarding Σ-types. We obtain the following diagram:

$$\begin{array}{ccc} \mathrm{PathCat} & \xrightarrow{\sim} & \mathrm{ComprehensionCat}_{\mathrm{Contextual},=,\Sigma_{\beta\eta}} \\ {\scriptstyle U}\uparrow \dashv \downarrow{\scriptstyle C} & & {\scriptstyle F}\uparrow \dashv \downarrow{\scriptstyle U} \\ \mathrm{DisplayPathCat} & \xrightarrow[\sim]{} & \mathrm{ComprehensionCat}_{\mathrm{Contextual},=} \end{array}$$

where the arrows denoted with U are forgetful 2-functors, C is a cofree 2-functor that interprets every context extension as a type, and F is a free 2-functor that adds Σ-types.

Axiomatic Type Theory. An *Axiomatic Type Theory (ATT)* is a dependent type theory without any reduction rules. (Display) path categories provide semantics for a minimal ATT, and we can model more extensive versions of ATT by adding more structure. Removing the reductions from a type theory makes it easier to find models and reduces the complexity of

type checking (from non-elementary to quadratic [vdBdB21]). Despite these simplifications, ATT does not lose much deductive strength: by adding only two principles to ATT—binder extensionality and uniqueness of identity proofs—we can prove the same as Extensional Type Theory [Win20], which has the maximal number of reductions. For more, see [Boc23].

Path Categories. A path category extends Brown's notion of a category of fibrant objects [Bro73] and Joyal's notion of a tribe [Joy17] as seen in Van den Berg [vdB18]. In this way, we can view it both as a framework for homotopy theory as well as a semantics for type theory. From the first perspective: it is a setting in which we can define homotopy, build factorisation systems, and prove lifting theorems. From the second perspective: it simplifies semantics by removing explicit elimination and computation rules.

To model the basic structure of dependent type theory, we have a category $\mathcal{C}$ and a collection of morphisms called *fibrations*. We view an object as a type or a context and a fibration $A \twoheadrightarrow \Gamma$ as a type or telescope A in context Γ. To model propositional $=$-types, we require that every object A has an object PA of paths in A. The formation rule is modelled by the source and target maps $(s,t) : PA \twoheadrightarrow A \times A$ and the introduction rule is modelled by the constant path map $r : A \to PA$. Now, instead of adding elimination and computation rules, we require that we are given a collection of morphisms called *(weak) equivalences*, and that r is in this collection. These morphisms will model the equivalences of the type theory. Lastly, we require that fibrations, equivalences, and path objects satisfy some simple axioms.

Display Path Categories. In a display path category we use *display maps* as a primitive notion instead of fibrations. Intuitively, these are the fibrations $\Delta \twoheadrightarrow \Gamma$ that only extend Γ with a single type. We define the fibrations as the maps that can be built as a composition of isomorphisms and display maps. In addition, we replace the path objects for objects Γ with path objects for display maps $A \twoheadrightarrow \Gamma$. This appears weaker but is sufficient, as they can be used to inductively construct path objects for general objects using a lifting theorem and a notion of transport. Hence, a display path category is in particular a path category.

Equivalences. To interpret a path category as a comprehension category we interpret the objects as contexts and the fibrations as types. We can recover the elimination and computation rules for propositional $=$-types using a lifting theorem for path categories; the details can be found in Van den Berg [vdB18]. We get contextuality because every map $A \to 1$ is a fibration, and Σ-types because fibrations are closed under composition. For the other direction, we interpret compositions of display maps and isomorphisms as our fibrations, and the homotopy equivalences according to the propositional $=$-type structure as our weak equivalences.

On the level of display path categories we can be a bit more precise because we have additional structure. When interpreting a display path category as a comprehension category we only interpret display maps as types instead of arbitrary fibration. This also means that we retain more structure when interpreting a suitable comprehension category as a display path category, namely which fibrations are display maps.

Additional Structure. To extend this work, we are hoping to interpret additional propositional type constructors in (display) path categories. The idea is that Σ-types and Π-types should be weakenings of existing presentations: as left and right adjoints of the pullback functor. In this way, we aim to achieve similar simplifications: requiring that certain maps are equivalences and thereby being able to omit computation rules.

References

[Boc22] Rafaël Bocquet. Strictification of Weakly Stable Type-Theoretic Structures Using Generic Contexts. In Henning Basold, Jesper Cockx, and Silvia Ghilezan, editors, *27th International Conference on Types for Proofs and Programs (TYPES 2021)*, volume 239 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Boc23] Rafaël Bocquet. Towards coherence theorems for equational extensions of type theories, 2023.

[Bro73] Kenneth S. Brown. Abstract homotopy theory and generalized sheaf cohomology. *Transactions of the American Mathematical Society*, 186:419–458, 1973.

[CCHM18] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:34, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Jac93] Bart Jacobs. Comprehension categories and the semantics of type dependency. *Theoretical Computer Science*, 107(2):169–207, 1993.

[Joy17] Andre Joyal. Notes on clans and tribes, 2017.

[LW15] Peter Lefanu Lumsdaine and Michael A. Warren. The local universes model: An overlooked coherence construction for dependent type theories. *ACM Trans. Comput. Logic*, 16(3), jul 2015.

[vdB18] Benno van den Berg. Path categories and propositional identity types. *ACM Trans. Comput. Logic*, 19(2), jun 2018.

[vdBdB21] Benno van den Berg and Martijn den Besten. Quadratic type checking for objective type theory, 2021.

[vdBM18] Benno van den Berg and Ieke Moerdijk. Exact completion of path categories and algebraic set theory. Part I: Exact completion of path categories. *Journal of Pure and Applied Algebra*, 222(10):3137–3181, 2018.

[Win20] Théo Winterhalter. *Formalisation and meta-theory of type theory.* PhD thesis, Université de Nantes, 2020.

Identity types in predicate logic

Sori Lee

https://orcid.org/0000-0002-3911-8909

I will report on work in progress that draws on Martin-Löf's identity types [6, 7] to describe universal properties of model constructions in categorical predicate logic based on equivalence relations and partial equivalence relations. The driving example is the tripos-to-topos construction [1, 3], which can be viewed as composed of the following steps:

$$\text{Triposes} \xrightarrow[\substack{\text{take PERs as objects and descent data as predicates}}]{(1)} \xrightarrow[\text{'virtualise'}]{(2)} \xrightarrow[\substack{\text{take functional relations as arrows}}]{(3)} \xrightarrow[\substack{\text{identify arrows by extensionality}}]{(4)} \xrightarrow[\substack{\text{take underlying category}}]{(5)} \text{Toposes}$$

Our attention is on the first two steps. (1) is the aforementioned 'model construction based on PERs', which shall henceforth be referred to as the *PER construction*. *Virtualisation* (2) is a procedure that turns chosen predicates into truths, and in this case has the effect that it turns the PERs emerged in the previous step into equivalence relations. The objective of this talk is to outline these two constructions in general forms and describe their universal properties.

A related work with regard to the tripos-to-topos construction is [9], which discusses a different decomposition of the construction and contains details about steps (3), (4) and (5).

Identity objects We work with indexed preorders, which are an interpretation of many-sorted predicate logic:

Definition 1. An *indexed preorder* P consists of a category P^0 and a functor $P^1 \colon (P^0)^{\mathrm{op}} \to \mathrm{Pre}$.

Taking the view that many-sorted predicate logic is a highly truncated version of dependent type theory, we obtain the following adaptation of the inductive axioms of identity types to indexed preorders. Let P be an indexed $(\wedge, \top)$-preorder over a binary-product category.

Definition 2. An *identity object* on an object $X \in P^0$ is an element $\mathrm{Id}_X \in P^1(X \times X)$ such that

1. (*introduction*) $\top \leq (X \xrightarrow{\delta} X \times X)^*(\mathrm{Id}_X)$, and
2. (*elimination*) for any object Y in P^0 and $p, q \in P^1(X \times X \times Y)$, if
$$(X \times Y \xrightarrow{\delta \times Y} X \times X \times Y)^*(p) \leq (X \times Y \xrightarrow{\delta \times Y} X \times X \times Y)^*(q),$$
then $(X \times X \times Y \xrightarrow{\pi_1, \pi_2} X \times X)^*(\mathrm{Id}_X) \wedge p \leq q$.

We say P has *identity objects* if each X has an identity object.

This Martin-Löf notion of equality turns out, perhaps as expected, to be equivalent to Lawvere's one as extracted by Maietti and Rosolini [2, 5]:

Theorem 3. *An indexed $(\wedge, \top)$-poset over a finite-product category has identity objects if and only if it is an elementary doctrine.*

This means Pasquali's 'elementary completion' result [8] is telling that the Maietti-Rosolini 'effective-quotient completion' construction [4], or the *ER construction* as would fit our nomenclature, is a right-biadjoint completion with respect to identity objects. Briefly, this construction assigns to P an indexed preorder $\mathrm{ER}(P)$ such that an object in $\mathrm{ER}(P)^0$ is a pair $(X, \sim)$ with $\sim \in P^1(X \times X)$ an equivalence relation, an arrow $(X, \sim_X) \to (Y, \sim_Y)$ is an arrow $f \colon X \to Y$ satisfying $\sim_X \leq P^1(f \times f)(\sim_Y)$, and an element in $\mathrm{ER}(P)^1(X, \sim)$ is an element $p \in P^1(X)$

satisfying $P^1(\pi_1)(p) \wedge {\sim} \leq P^1(\pi_2)(p)$. Pasquali's result adapted to our settings reads:

Theorem 4. *The assignment* $P \mapsto \mathrm{ER}(P)$ *extends to a 2-functor* $\mathrm{IdxPre}_{\mathrm{pn}}^{\times,\wedge,\top} \to \mathrm{IdxPre}_{\mathrm{pn}}^{\times,\wedge,\top,\mathrm{Id}}$ *that is right biadjoint to the inclusion 2-functor.*

Here, the notation e.g. $\mathrm{IdxPre}_{\mathrm{pn}}^{\times,\wedge,\top,\mathrm{Id}}$ denotes the 2-category of indexed $(\wedge,\top)$-preorders with identity objects over binary-product categories, **p**seudo**n**atural morphisms that preserves $\times$, $\wedge$, $\top$ and Id, and 2-morphisms; these morphisms and 2-morphisms are defined the same way as in [5, 4, 8], except that our morphisms have a *pseudo*natural-transformation component.

This statement may be viewed as a (extremely) truncated instance of the conceivable notion that the (higher) groupoid interpretation serves as a completion with respect to identity types.

Partial identity objects Let P be an indexed $\wedge$-preorder over a binary-product category. Define an indexed preorder $\mathrm{PER}(P)$ the same way as $\mathrm{ER}(P)$ but with as objects in $\mathrm{PER}(P)^0$ *partial* equivalence relations in P instead. This is the PER construction. Now the following weakened form of identity objects is going to give us a result analogous to Theorem 4 for the PER construction.

Definition 5. We say P *has partial identity objects* if each object $X \in P^0$ is *equipped* with an element $\mathrm{PId}_X \in P^1(X \times X)$, such that

1. (*partial reflexivity*) $\mathrm{PId}_X \leq (X \times X \overset{\pi_1}{\to} X \times X)^*(\mathrm{PId}_X), (X \times X \overset{\pi_2}{\to} X \times X)^*(\mathrm{PId}_X)$,
2. (*paravirtual elimination*) for each object $Y \in P^0$ and elements $p, q \in P^1(X \times X \times Y)$, if
$$(X \times Y \overset{\pi_1,\pi_1}{\to} X \times X)^*(\mathrm{PId}_X) \wedge (X \times Y \overset{\pi_2,\pi_2}{\to} Y \times Y)^*(\mathrm{PId}_Y) \wedge$$
$$(X \times Y \overset{\delta\times Y}{\to} X \times X \times Y)^*(p) \leq (X \times Y \overset{\delta\times Y}{\to} X \times X \times Y)^*(q)$$
then $(X \times X \times Y \overset{\pi_3,\pi_3}{\to} Y \times Y)^*(\mathrm{PId}_Y) \wedge (X \times X \times Y \overset{\pi_1,\pi_2}{\to} X \times X)^*(\mathrm{PId}_X) \wedge p \leq q$.
3. each arrow $f\colon X \to Y$ in P^0 satisfies $\mathrm{PId}_X \leq (f \times f)^*(\mathrm{PId}_Y)$, and
4. $\mathrm{PId}_{X\times Y} \simeq (X\times Y\times X\times Y \overset{\cong}{\to} X\times X\times Y\times Y)^*(\mathrm{PId}_X \times \mathrm{PId}_Y)$ for each objects $X, Y \in P^0$.

Theorem 6. *The assignment* $P \mapsto \mathrm{PER}(P)$ *extends to a 2-functor* $\mathrm{IdxPre}_{\mathrm{pn}}^{\times,\wedge} \to \mathrm{IdxPre}_{\mathrm{pn}}^{\times,\wedge,\mathrm{PId}}$ *that is right biadjoint to the forgetful 2-functor.*

Virtualisation An indexed preorder P is *oplaxly sectioned* if each object $X \in P^0$ is equipped with an element $\mathrm{os}_X \in P^1(X)$, and every arrow $f\colon X \to Y$ in P^0 satisfies $\mathrm{os}_X \leq f^*(\mathrm{os}_Y)$. We regard an indexed preorder with partial identity objects as oplaxly sectioned by $\mathrm{os}_X \coloneqq (X \overset{\delta}{\to} X \times X)^*(\mathrm{PId}_X)$. Let P be an oplaxly sectioned indexed $\wedge$-preorder.

Definition 7. The *virtualisation* of P is the indexed preorder $\mathrm{Virt}(P)$ given by $\mathrm{Virt}(P)^0 \coloneqq P^0$ and $\mathrm{Virt}(P)^1(X) \coloneqq (\mathrm{U}_{\mathrm{Set}}P^1(X), \overset{\mathrm{v}}{\leq})$ where $p \overset{\mathrm{v}}{\leq} q$ if and only if $\mathrm{os}_X \wedge p \leq q$.

$\mathrm{Virt}(P)^1$ is in fact a Kleisli as well as Eilenberg-Moore object for a (necessarily idempotent) comonad in the Pre-category $[(P^0)^{\mathrm{op}}, \mathrm{Pre}^{\wedge}]_{\mathrm{o}}$ of functors and *oplax* natural transformations.

Note that the os_X become top elements in $\mathrm{Virt}(P)$. Moreover, if P has partial identity objects, then $\mathrm{Virt}(P)$ has identity objects.

Now a universal property of virtualisation is as follows; beware that the morphisms involved here are **o**plax-**n**atural morphisms, rather than pseudonatural morphisms used previously.

Theorem 8. *The assignment* $P \mapsto \mathrm{Virt}(P)$ *extends to a 2-functor* $\mathrm{IdxPre}_{\mathrm{on}}^{\wedge,\mathrm{os}} \to \mathrm{IdxPre}_{\mathrm{on}}^{\wedge,\top}$ *that is right biadjoint to the 'inclusion' 2-functor.*

References

[1] J.M.E. Hyland, P.T. Johnstone, and A.M. Pitts. Tripos theory. *Mathematical Proceedings of the Cambridge Philosophical Society*, 88(2):205–232, 1980.

[2] F. W. Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. In A. Heller, editor, *Proceedings of the AMS Symposium on Pure Mathematics*, volume XVII, pages 1–14. American Mathematical Society, 1970.

[3] S. Lee. Subtoposes of the effective topos. Master's thesis, Utrecht University, 2011.

[4] M.E. Maietti and G. Rosolini. Elementary quotient completion. *Theory and Applications of Categories*, 27(17):445–463, 2013.

[5] M.E. Maietti and G. Rosolini. Quotient completion for the foundation of constructive mathematics. *Logica Universalis*, 7(3):371–402, 2013.

[6] P. Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 179–216. Elsevier, 1971.

[7] P. Martin-Löf. An intuitionistic theory of types: predicative part. In *Logic Colloquium '73*, pages 73–118. Elsevier, 1975.

[8] F. Pasquali. A co-free construction for elementary doctrines. *Applied Categorical Structures*, 23(1):29–41, 2015.

[9] F. Pasquali. Remarks on the tripos to topos construction: Comprehension, extensionality, quotients and functional-completeness. *Applied Categorical Structures*, 24(2):105–119, 2016.

Partial Combinatory Algebras for Intensional Type Theory

Sam Speight

University of Birmingham, UK

Realizability over partial combinatory algebras

An important class of models for the meta-theoretic study of type theory comes from realizability. Not only can these models be used to show consistency of constructive principles (eg. Church's thesis, which is valid in Hyland's *effective topos* [9]), but they are also able to interpret polymorphism or impredicative universes in dependent type theory [10].

Traditionally, the starting point for a realizability interpretation is a *partial combinatory algeba* (PCA). A PCA embodies a notion of untyped (or unityped) computation (the untypedness is actually necessary for impredicativity [3, 14, 12]). Formally, a PCA consists of a set $\mathbb{A}$ and a partial "application" operation $(-)\cdot(?) : \mathbb{A} \times \mathbb{A} \rightharpoonup \mathbb{A}$. Additionally, there must exist particular elements ("combinators") obeying certain laws. Most often one sees the combinators k and s satisfying:

$$\mathsf{k}ab = a \qquad\qquad \mathsf{s}ab\downarrow \quad \text{and} \quad \mathsf{s}abc = ac(bc)$$

(we surpress the application symbol and associate to the left). The existence of these combinators is enough to guarantee *combinatorial completeness*, which means that every "polynomial" over $\mathbb{A}$ (built up from variables and elements of $\mathbb{A}$ using application) is represented by some "code" (element of $\mathbb{A}$) [6]. In this way, a PCA can mimic λ-abstraction, which, together with application, satisfies the β-law.

Among the first PCAs one encounters are the λ-calculus Λ, "Kleene's first algebra" $\mathcal{K}_1$ and categorical models of the λ-calculus (reflexive objects in cartesian closed categories). Λ is the set of λ-terms modulo β together with the application of the λ-calculus. The underlying set of $\mathcal{K}_1$ is $\mathbb{N}$ and application is: $n\cdot m \coloneqq \{n\}(m)$, ie. the result of applying the n^{th} partial computable function to m.

Realizability for intensional type theory

With the advent of *homotopy type theory*, in the context of *intensional type theory* (ITT), evidence (a proof term) for an identification may be thought of as a path between points in some space [18]. Insofar as realizability interpretations formalize the BHK interpretation (in that realizers play the role of evidence for propositions), one might think that—in the context of ITT—realizers should carry higher-dimensional (categorical, homotopical) structure.

In this spirit, Angiuli and Harper have formulated a cubical generalization of *Martin-Löf's meaning explanations* [1]. Related to this is *higher-dimensional (cubical) computational type theory*, which can be seen as a realizability model of cubical type theory [2]. The starting point here is a cubical programming language that has sorts for dimensions and terms. Terms, which may contain free dimension names, can be seen as abstract cubes.

On the categorical side, [15] studies a groupoidal generalization of *partitioned assemblies*. Realizers derive from a *realizer category* $\mathbf{R}$ containing an interval (co-groupoid) $\mathbb{I} \in \mathbf{R}$. The interval furnishes a fundamental groupoid construction $\Pi : \mathbf{R} \to \mathbf{Gpd}$. A partitioned assembly has an underlying groupoid, whose objects are realized by points in the fundamental groupoid ΠA of some "realizer type" $A \in \mathbf{R}$ and whose morphisms are realized by paths ΠA. If the

realizer type is always some fixed *universal* object U, the notion of realizability in untyped. An alternative approach is to consider higher-dimensional structures in traditional realizability models of *extensional* type theory, eg. *cubical assemblies* (cubical objects internal to the model of extensional type theory in assemblies over $\mathcal{K}_1$) [17, 16].

Partial combinatory algebras in groupoids

The notion of PCA makes sense in any *cartesian restriction category* (CRC; restriction categories formalize the idea of categories containing partial maps) [4]. **The goal of this work is to give examples of PCAs in CRCs of groupoids that may be used for constructing realizability models of ITT.**

The first example we give is a *higher-dimensional λ-calculus*, very much inspired by *cubical type theory* [5]. In fact, different calculi could be formulated depending on the notion of "shape" (eg. globular, cubical, etc.). For simplicity, we discuss a relatively simple 1-dimensional globular λ-calculus. Judgements in this calculus are of the form

$$\Psi \mid \Gamma \vdash t$$

where Ψ is a context of dimension variables and Γ is a context of regular variables. We have constants:

$$\cdot \mid \cdot \vdash 0 \qquad\qquad \cdot \mid \cdot \vdash 1$$

As well as the usual rules for λ-abstraction, application and β (uniform in dimension context), we have rules for composition, identities and inverses. For example:

$$\frac{i \mid \Gamma \vdash \alpha \qquad i \mid \Gamma \vdash \beta \qquad \cdot \mid \Gamma \vdash \beta[0/i] = \alpha[1/i]}{i \mid \Gamma \vdash \beta \circ \alpha}\ \text{comp}$$

Identities are obtained by weakening the dimension context. These term constructors satisfy the usual groupoid equations, ensuring that we obtain a groupoid $\Pi\Lambda$ with:

- objects: terms (in context, up to α-equivalence) of the form $\cdot \mid \Gamma \vdash t$;
- morphisms $(\cdot \mid \Gamma \vdash t) \to (\cdot \mid \Gamma \vdash u)$: terms $i \mid \Gamma \vdash \alpha$ satisfying $\cdot \mid \Gamma \vdash \alpha[0/i] = t$ and $\cdot \mid \Gamma \vdash \alpha[1/i] = u$;
- composition, identities and inverses given by the corresponding term constructors.

The groupoid $\Pi\Lambda$ is a PCA in the category of groupoids and functors (with the trivial restriction structure). The application functor is given by application of terms (given how substitution behaves and the various term constructors interact) and the combinators k and s are determined respectively by:

$$\cdot \mid \cdot \vdash \lambda xy.\ x \qquad\qquad \cdot \mid \cdot \vdash \lambda fgx.\ fx(gx)$$

Moving on, there is a class of examples coming from *2-dimensional models of the λ-calculus*, ie. cartesian closed bicategories $\mathcal{C}$ with a pseudoreflexive object U. Instances of these include *generalised species of structures* [7], *profunctorial Scott semantics* [8] and *categorified relational* ("distributors-induced") [13] and *graph* models [11]); realizer categories $(\mathbf{R}, \mathbb{I}, U)$ as discussed above also gives rise to such structures. The carrier of the (total) PCA is the groupoid $\mathcal{C}(1, U)$. This results in a "pseudo PCA", where the combinator laws hold up to isomorphism.

Further work-in-progress is to establish a groupoidal analogue of $\mathcal{K}_1$ based on a notion of *partial recursive functor over the groupoid of finite sets and bijections*. This will live in the CRC of groupoids and *partial* functors (with non-trivial restriction structure).

References

[1] Carlo Angiuli and Robert Harper. Meaning explanations at higher dimension. *Indagationes Mathematicae*, 29(1):135–149, 2018.

[2] Carlo Angiuli, Robert Harper, and Todd Wilson. Computational higher-dimensional type theory. *SIGPLAN Not.*, 52(1):680–693, jan 2017.

[3] Lars Birkedal. A general notion of realizability. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 7–17. IEEE Computer Society, 2000.

[4] J.R.B. Cockett and P.J.W. Hofstra. Introduction to turing categories. *Annals of Pure and Applied Logic*, 156(2):183–209, 2008.

[5] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:34, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[6] Solomon Feferman. A language and axioms for explicit mathematics. In John Newsome Crossley, editor, *Algebra and Logic*, pages 87–139, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[7] M. Fiore, N. Gambino, M. Hyland, and G. Winskel. The cartesian closed bicategory of generalised species of structures. *Journal of the London Mathematical Society*, 77(1):203–220, 2008.

[8] Zeinab Galal. A Profunctorial Scott Semantics. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[9] J.M.E. Hyland. The effective topos. In A.S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 165–216. Elsevier, 1982.

[10] J.M.E. Hyland. A small complete category. *Annals of Pure and Applied Logic*, 40(2):135–165, 1988.

[11] Axel Kerinec, Giulio Manzonetto, and Federico Olimpieri. Why are proofs relevant in proof-relevant models? *Proc. ACM Program. Lang.*, 7(POPL), jan 2023.

[12] Peter Lietz and Thomas Streicher. Impredicativity entails untypedness. *Mathematical. Structures in Comp. Sci.*, 12(3):335–347, June 2002.

[13] Federico Olimpieri. Intersection type distributors. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–15, 2021.

[14] Edmund Robinson and Giuseppe Rosolini. An abstract look at realizability. In *International Workshop on Computer Science Logic*, pages 173–187. Springer, 2001.

[15] Sam Speight. *Groupoidal Realizability for Intensional Type Theory*. PhD thesis, University of Oxford, 2023.

[16] Andrew W. Swan and Taichi Uemura. On church's thesis in cubical assemblies. *Mathematical Structures in Computer Science*, 31(10):1185–1204, 2021.

[17] Taichi Uemura. Cubical Assemblies, a Univalent and Impredicative Universe and a Failure of Propositional Resizing. In Peter Dybjer, José Espírito Santo, and Luís Pinto, editors, *24th International Conference on Types for Proofs and Programs (TYPES 2018)*, volume 130 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[18] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

A directed type theory for 1-categories

Fernando Chu[1], Éléonore Mangel[2], and Paige Randall North[1]

[1] Utrecht University, Netherlands
[2] École normale supérieure Paris-Saclay, France

Background

In Martin-Löf Type Theory (MLTT), the identity types are defined as the family of types $\mathsf{Id}_A(x, y)$ for A a type, and x and y two elements of A, generated inductively by $\mathsf{refl}_a : \mathsf{Id}_A(a, a)$. We can deduce the usual properties of equality: reflexivity, symmetry and transitivity. In fact, Hofmann and Streicher [1] discovered that these properties correspond respectively to the groupoidal operations of identity, inverse and composition, when identities between two terms of a type A are interpreted as morphisms between two objects of A. This discovery showed that MLTT can be used to prove and verify theorems about groupoids and even ∞-groupoids.

Currently, there are many attempts towards finding a similar type theory that instead codifies the structures of (higher) category theory and directed homotopy theory, but no clear consensus has been reached yet. One natural idea to solve this problem is to replace the symmetric identity types with directed homomorphism types. Indeed, whereas groupoid theory and homotopy theory are used to study structures with symmetric paths, (higher) category theory and directed homotopy theory are used to study structures with directed paths.

Related Works

Our work builds on the previous work of the third-listed author [2]. We keep the same goal of having a homomorphism type former with simple rules analogous to the identity type former of the MLTT. We solve one of the main problems of the previous article: our type theory includes both directed homomorphism types and Martin-Löf's original identity types without the collapse of the former into the latter.

We are also inspired by Nuyts [3], especially how variances of assumptions and terms are marked in judgments, and we improve on it by building an interpretation of our syntax.

Our work diverges from other attempts in the literature. Indeed, the works of Licata and Harper [4] and Ahrens, North and Van Der Weide [5] don't have a homomorphism type former and the work of Riehl and Shulman [6] builds on Cubical Type Theory rather than MLTT. Finally, we differ from Kavvos [7] by giving a syntax for our semantics.

Our new attempt

As mentioned, our theory builds upon previous theories by adding orientations $+$, $-$ and $\circ$, while also introducing a hom-type former and an Id-type former. We present a sketch of this construction on the following paragraphs.

We start with MLTT and add orientations that mark the variance of assumptions and terms.

$$\Gamma \vdash_{\ell \vdash \omega} t : A,$$

where ℓ is a list of orientations (one for each type in Γ) and ω is the orientation of the term t.

The first two orientations are $+$ and $-$, corresponding respectively to covariance and contravariance. For example, given a contravariant term $t : B$ depending covariantly on a variable $x : A$ and a morphism $\varphi : a \to a'$ in A, then we obtain a morphism from $t(a)$ to $t(a')$ in B.

However, many mathematical notions that we want to express in our system are neither covariant nor contravariant. For example, the identity type of x and y can depend neither covariantly nor contravariantly on x, nor on y. If it did, it would allow us to transport identity along morphisms and thus the two ends of every morphism would be identified. We want identities to transport along isomorphisms, which motivates the introduction of a third orientation for this case, denoted $\circ$, which will correspond to isovariance.

With $t : B$ a covariant term depending isovariantly on a variable $x : A$, a morphism $\phi : a \to a'$ in A will give us no information between $t(a)$ and $t(a')$. But if ϕ is an isomorphism, then we will have an isomorphism between $t(a)$ and $t(a')$ in B. Conversely, if $t : B$ is an isovariant term depending on $x : A$ covariantly, any morphism from a to a' in A will induce an isomorphism between $t(a)$ and $t(a')$ in B.

We then introduce a $\mathsf{hom}_A(x, y)$ type (following [2]) that respects these orientations in a coherent way. Its first three rules are the following (to which must be added hom-LEFT-ELIM and corresponding computation rules).

hom-FORM
$$\frac{\Gamma \vdash_{\ell \vdash +} A : \mathcal{U}_k}{\Gamma, x : A, y : A \vdash_{\ell, x^-, y \vdash +} \mathsf{hom}_A(x, y) : \mathcal{U}_k}$$

hom-INTRO
$$\frac{\Gamma \vdash_{\ell \vdash +} A : \mathcal{U}_k}{\Gamma, x : A \vdash_{\ell, x^\circ \vdash \circ} 1_x : \mathsf{hom}_A(x, x)}$$

hom-LEFT-ELIM
$$\frac{\Gamma \vdash_{\ell \vdash +} A : \mathcal{U}_k \qquad \Gamma, x : A, y : A, z : \mathsf{hom}_A(x, y) \vdash_{\ell, x^\circ, y, z \vdash +} D(x, y, z) : \mathcal{U}_k \qquad \Gamma, x : A \vdash_{\ell, x^\circ \vdash \omega} d(x) : D(x, x, 1_x)}{\Gamma, x : A, y : A, z : \mathsf{hom}_A(x, y) \vdash_{\ell, x^\circ, y, z \vdash \omega} j_d^L(x, y, z) : D(x, y, z)}$$

One of our goals was to not divert too much from the spirit of MLTT: erasing the orientation will give us exactly the rules for the identity types of MLTT.

Similarly, we also have identity types, with the difference being that they are limited to isovariant terms.

Results

Internal 1-category theory. Working inside this theory, we are able to develop the theory of 1-categories in a synthetic manner, similar to how Homotopy Type Theory (HoTT) develops the theory of ∞-groupoids. For example, we derive:

Theorem (Yoneda)**.** *For A a type, a an element of A and P a presheaf, we have an equivalence*

$$\Pi_{x:A^\circ}(\mathsf{hom}_A(x, a) \to P(x)) \simeq P(a),$$

where $x : A^\circ$ indicates x appears isovariantly in $\mathsf{hom}_A(x, a) \to P(x)$.

1-Categorical semantics. We develop a semantic model of this type theory in the category of 1-categories. In this interpretation, the orientations described earlier are interpreted as endofunctors of Cat: $+$ is the identity, $-$ maps a category to its opposite category, and $\circ$ is the functor taking a category to its core (i.e. its maximal sub-groupoid).

Contexts $\Gamma \vdash_\ell$ are modeled as categories Γ^ℓ, while we model types A in a context $\Gamma \vdash_\ell$ as a functor from Γ^ℓ to Cat.

Future works

Following the developments in HoTT, we would also like to introduce an adequate notion of higher inductive types. Additionally, we also expect some version of directed univalence to

be compatible with this theory. Finally, an important question is whether this theory can be extended to higher dimensions. We would like to investigate under what conditions the n-th iterated hom type gives precisely the n-cells of a higher category.

References

[1] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. *Twenty-five years of constructive type theory (Venice, 1995)*, 36:83–111, 1998.

[2] Paige Randall North. Towards a directed homotopy type theory. *Electronic Notes in Theoretical Computer Science*, 347:223–239, 2019.

[3] Andreas Nuyts. Towards a directed homotopy type theory based on 4 kinds of variance. *Mém. de mast. Katholieke Universiteit Leuven*, 2015.

[4] Daniel R Licata and Robert Harper. 2-dimensional directed type theory. *Electronic Notes in Theoretical Computer Science*, 276:263–289, 2011.

[5] Benedikt Ahrens, Paige Randall North, and Niels Van Der Weide. Bicategorical type theory: semantics and syntax. *Mathematical Structures in Computer Science*, 33(10):868–912, 2023.

[6] Emily Riehl and Michael Shulman. A type theory for synthetic ∞-categories. *arXiv preprint arXiv:1705.07442*, 2017.

[7] G. A. Kavvos. A quantum of direction. *Preprint*, 2019. https://seis.bristol.ac.uk/~tz20861/papers/meio.pdf.

Reviewers

Program Committee

Patrick Bahr	IT University of Copenhagen
Henning Basold	LIACS, Leiden University
Andrej Bauer	University of Ljubljana
Marco Carbone	IT University of Copenhagen
Jesper Cockx	Delft University of Technology
Greta Coraglia	LUCI Lab, Department of Philosophy, University of Milan
Peter Dybjer	Chalmers University of Technology
Yannick Forster	Inria
Hugo Herbelin	INRIA
Patricia Johann	Appalachian State University
Marie Kerjean	CNRS, LIPN, Université Sorbonne Paris Nord
Ekaterina Komendantskaya	Department of Computer Science, Heriot-Watt University
Meven Lennon-Bertrand	University of Cambridge
Assia Mahboubi	INRIA
Sonia Marin	University of Birmingham
Anders Mörtberg	Stockholm University
Rasmus Ejlers Møgelberg	IT University of Copenhagen
Benjamin Pierce	University of Pennsylvania
Jakob Rehof	University of Dortmund
Simona Ronchi Della Rocca	Universita' di Torino - dipartimento di Informatica
Kristina Sojakova	Vrije Universiteit Amsterdam
Ana Sokolova	University of Salzburg
Bas Spitters	Aarhus University
Wouter Swierstra	Utrecht University
Philip Wadler	The University of Edinburgh

Additional Reviewers

Cavallo, Evan
Dudenhefner, Andrej
Kirst, Dominik
Laarmann, Felix
Lamiaux, Thomas
Stahl, Christoph