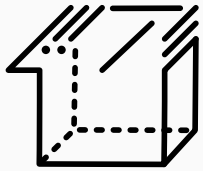


Towards Computational UIP in Cubical Agda

Yee-Jian Tan, Andreas Nuyts, Dominique Devriese

TYPES 2026 (Gothenburg, Sweden)
06 May 2026

DistriNet, KU Leuven, Belgium

Cubical-UIP

Cubical Agda [VMA21]

Based on [Coh+17] with De Morgan cubes.

- transp and hcomp
- I, *Glue* types, higher inductive types

Glue types (source of univalence) incompatible with Uniqueness of Identity Proofs (UIP) and K.

Cubical-UIP

Cubical Agda [VMA21]

Based on [Coh+17] with De Morgan cubes.

- transp and hcomp
- I, *Glue* types, higher inductive types

Glue types (source of univalence) incompatible with Uniqueness of Identity Proofs (UIP) and K. (But we can disable it!)

FunExt, Quotients, possibly **UIP**... good for working with sets! [Coc19, Shu17]

Motivation

Choices for working with **sets in Agda** (with Uniqueness of Identity Proofs):

	FunExt	**UIP holds**	**UIP computes**	**Quotient**	**Projects**
Agda `--with-K`	❌	✅	✅	❌	(Many)
Agda + Setoids	❌	Setoid ⚠️	✅	Setoid ⚠️	[CND25, HC21]

Motivation

Choices for working with **sets in Agda** (with Uniqueness of Identity Proofs):

	FunExt	**UIP holds**	**UIP computes**	**Quotient**	**Projects**
Agda `--with-K`	❌	✅	✅	❌	(Many)
Agda + Setoids	❌	Setoid ⚠️	✅	Setoid ⚠️	[CND25, HC21]
`--cubical` + *h*-set proofs	✅	by hand ⚠️	✅	✅	[Agd25, Nuy24, Sch+25]

Motivation

Choices for working with **sets in Agda** (with Uniqueness of Identity Proofs):

	FunExt	UIP holds	UIP computes	Quotient	Projects
Agda `--with-K`	❌	✅	✅	❌	(Many)
Agda + Setoids	❌	Setoid ⚠️	✅	Setoid ⚠️	[CND25, HC21]
`--cubical` + *h*-set proofs	✅	by hand ⚠️	✅	✅	[Agd25, Nuy24, Sch+25]
`--cubical=no-glue` + postulated UIP	✅	✅	❌	✅	[CNT26]

Motivation

Choices for working with **sets in Agda** (with Uniqueness of Identity Proofs):

	FunExt	UIP holds	UIP computes	Quotient	Projects
Agda `--with-K`	❌	✅	✅	❌	(Many)
Agda + Setoids	❌	Setoid ⚠️	✅	Setoid ⚠️	[CND25, HC21]
`--cubical` + *h*-set proofs	✅	by hand ⚠️	✅	✅	[Agd25, Nuy24, Sch+25]
`--cubical=no-glue` + postulated UIP	✅	✅	❌	✅	[CNT26]
`--cubical` + *h*-level instances	✅	Auto (external)	✅	✅	[1La24]

Motivation

Choices for working with **sets in Agda** (with Uniqueness of Identity Proofs):

	FunExt	UIP holds	UIP computes	Quotient	Projects
Agda `--with-K`	❌	✅	✅	❌	(Many)
Agda + Setoids	❌	Setoid ⚠️	✅	Setoid ⚠️	[CND25, HC21]
`--cubical` + *h*-set proofs	✅	by hand ⚠️	✅	✅	[Agd25, Nuy24, Sch+25]
`--cubical=no-glue` + postulated UIP	✅	✅	❌	✅	[CNT26]
`--cubical` + *h*-level instances	✅	Auto (external)	✅	✅	[1La24]
`--cubical=uip`	✅	**Auto (internal)**	✅	✅	

Cubical-UIP

Cubical Agda [VMA21]

Based on [Coh+17] with De Morgan cubes.

- transp and hcomp
- I, *Glue* types, higher inductive types

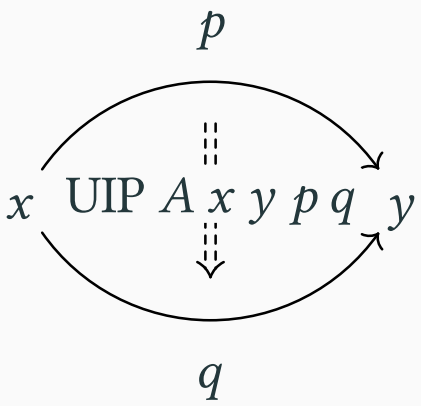
Cubical-UIP

✅ Agda `--cubical=no-glue` variant (merged in 2.9.0)

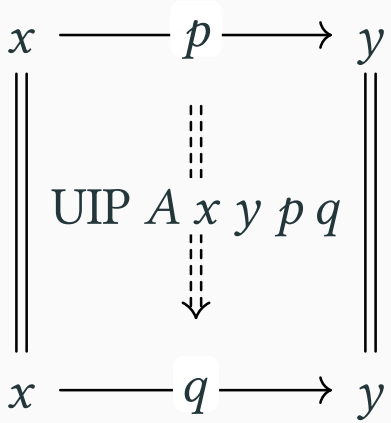
⌚ Agda `--cubical=uip` variant with a “UIP primitive” that recurses on types

UIP: working with h-Sets

UIP: squares with reflexive sides have a filler.



UIP

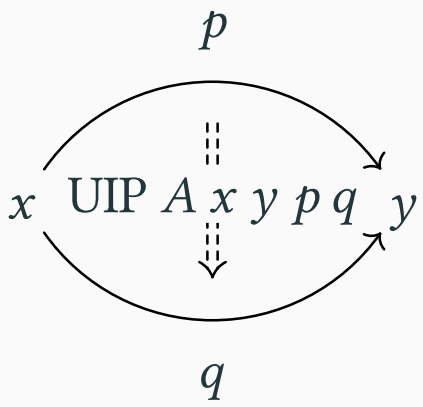


Reflexive sides

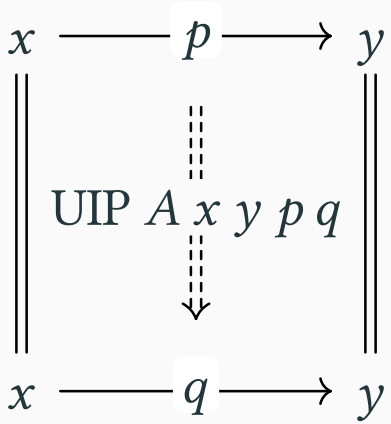
[1] [Agd25]

UIP: working with h-Sets

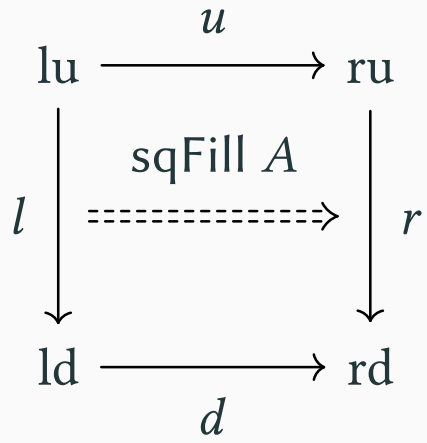
UIP: squares with reflexive sides have a filler.



UIP



Reflexive sides



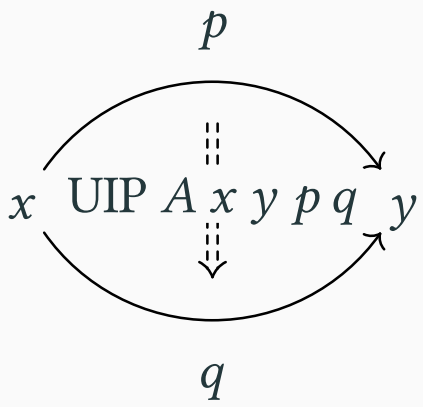
Square-filling

SqFill: any given square has a filler.

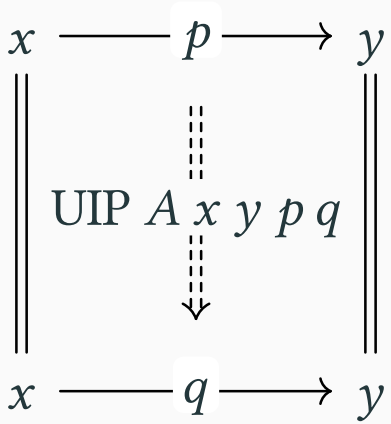
[1] [Agd25]

UIP: working with h-Sets

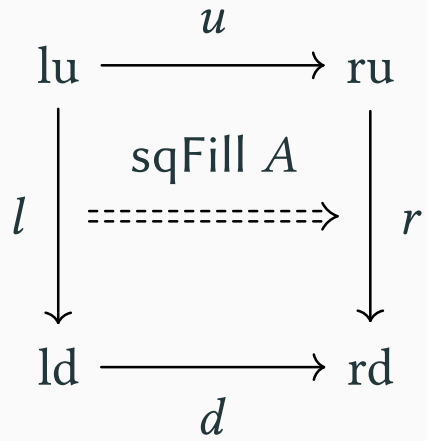
UIP: squares with reflexive sides have a filler.



UIP



Reflexive sides



Square-filling

SqFill: any given square has a filler.

$$\text{UIP} \leftrightarrow^{1} \text{SqFill}$$

[1] [Agd25]

How sqFill computes

The sqFill primitive performs **induction on the type**, repeatedly applying the SqFill-preservation proof [TND25] for the outermost type former:

How sqFill computes

The sqFill primitive performs **induction on the type**, repeatedly applying the SqFill-preservation proof [TND25] for the outermost type former:

$$\mathsf{sqFill}\ (A \times B) \quad \to_\beta \quad \mathsf{sqFill}_\times \underbrace{(\mathsf{sqFill}\ A)}_{\text{recurse}} \underbrace{(\mathsf{sqFill}\ B)}_{\text{recurse}}$$

$$\mathsf{sqFill}\ (\Pi\ (a : A)\ .\ B\ a) \quad \to_\beta \quad \mathsf{sqFill}_\Pi \underbrace{(\mathsf{sqFill}\ B)}_{\text{recurse}}$$

How sqFill computes

The sqFill primitive performs **induction on the type**, repeatedly applying the SqFill-preservation proof [TND25] for the outermost type former:

$$\mathsf{sqFill}\ (A \times B) \quad \to_\beta \quad \mathsf{sqFill}_\times\ \underbrace{(\mathsf{sqFill}\ A)}_{\text{recurse}}\ \underbrace{(\mathsf{sqFill}\ B)}_{\text{recurse}}$$

$$\mathsf{sqFill}\ (\Pi\ (a : A)\ .\ B\ a) \quad \to_\beta \quad \mathsf{sqFill}_\Pi \underbrace{(\mathsf{sqFill}\ B)}_{\text{recurse}}$$

Base types: a proof term is directly given.

$$\mathsf{sqFill}\ \mathsf{Bool} \quad \to_\beta \quad \mathsf{sqFill}_{\mathsf{Bool}}$$

$$\mathsf{sqFill}\ \mathbb{N} \quad \to_\beta \quad \mathsf{sqFill}_{\mathbb{N}}$$

How sqFill computes

The sqFill primitive performs **induction on the type**, repeatedly applying the SqFill-preservation proof [TND25] for the outermost type former:

$$\mathsf{sqFill}\ (A \times B) \quad \to_\beta \quad \mathsf{sqFill}_\times\ \underbrace{(\mathsf{sqFill}\ A)}_{\text{recurse}}\ \underbrace{(\mathsf{sqFill}\ B)}_{\text{recurse}}$$

$$\mathsf{sqFill}\ (\Pi\ (a : A)\ .\ B\ a) \quad \to_\beta \quad \mathsf{sqFill}_\Pi \underbrace{(\mathsf{sqFill}\ B)}_{\text{recurse}}$$

Base types: a proof term is directly given.

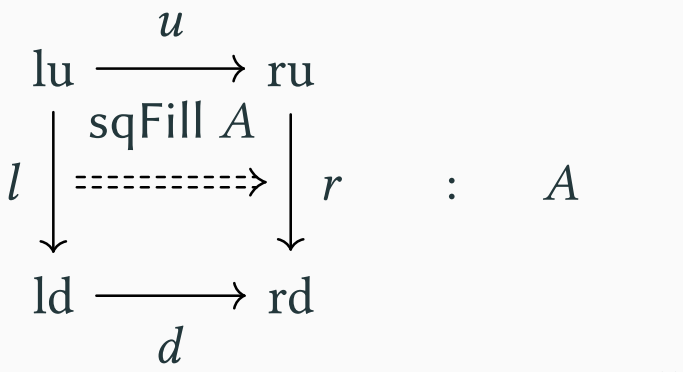
$$\mathsf{sqFill}\ \mathsf{Bool} \quad \to_\beta \quad \mathsf{sqFill}_{\mathsf{Bool}}$$

$$\mathsf{sqFill}\ \mathbb{N} \quad \to_\beta \quad \mathsf{sqFill}_{\mathbb{N}}$$

Implemented: **Pi**, **Sigma**, **PathP**; **List**, **Maybe**, **(Co-)products** ; Base types: **Unit**, **Bool**, **Nat**

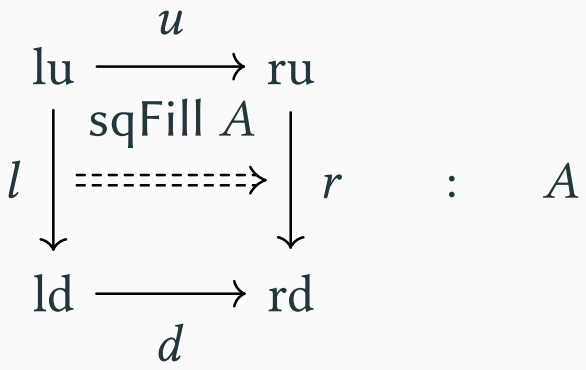
Welcome to experiment: https://github.com/yeejian-tan/agda/tree/cubical-uip

Non-dependent vs Dependent

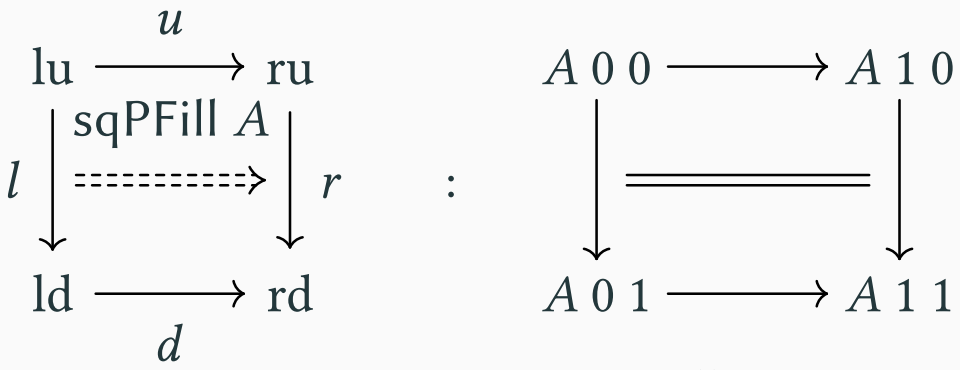


Homogenous SqFill

Non-dependent vs Dependent

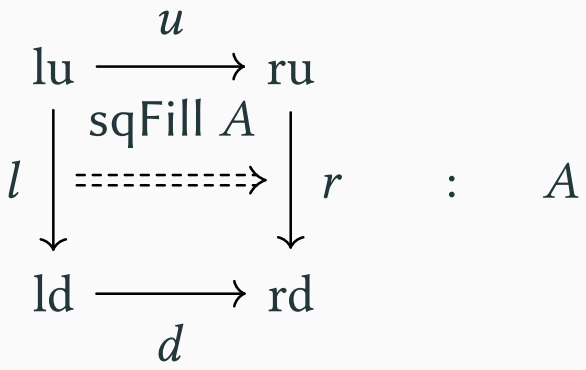


Homogenous SqFill

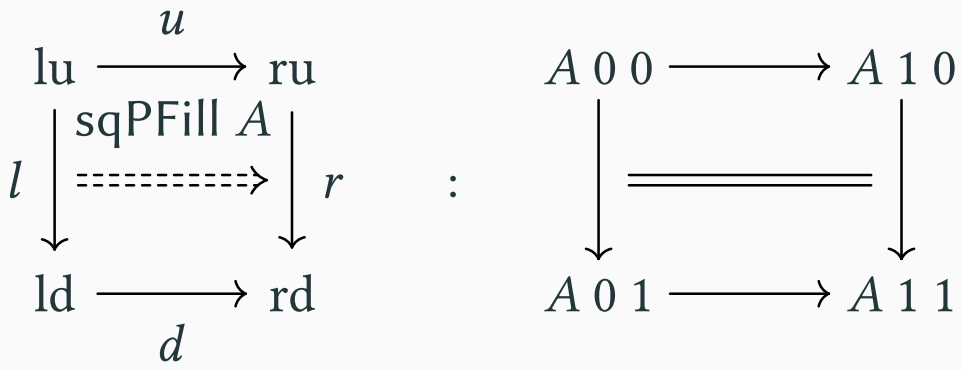


Heterogeneous SqPFill

Non-dependent vs Dependent



Homogenous SqFill



Heterogeneous SqPFill

SqPFill (heterogenous) $\leftrightarrow$ pointwise SqFill

Observation: Complexity of Preservation Proofs

Our proofs attempt to fill squares directly instead of instantiating more general results [TND25].

Preservation proofs differ in complexity for SqFill and SqPFill:

	SqFill (non-dependent)	SqPFill (dependent)
Pi	**Trivial** (no Kan operations)	**Complicated**: transport-fill-align
Sigma	**Complicated**: transport-fill-align	**Trivial** (no Kan operations)
Coproducts	**Encode-decode** (J, hcomp) [Uni13]	**Encode-decode** (J, comp) [Uni13]
Path Types	**Simple** (a single hcomp)	**Complicated**: transport-fill-align

Complexity of SqFill and SqPFill preservation proofs

Evaluation

Candidates:

- Chen, Nordvall Forsberg, and Tsai's TT-as-QIIRT [CNT26]
  Replacing UIP postulate by our sqFill primitive
- Bitvector Library in Katamaran [Huy+23]
  computation over proofs of "proof-irrelevance"

Evaluation

Candidates:

- Chen, Nordvall Forsberg, and Tsai's TT-as-QIIRT [CNT26]
  Replacing UIP postulate by our sqFill primitive
- Bitvector Library in Katamaran [Huy+23]
  computation over proofs of "proof-irrelevance"

Do you have evaluation targets that **compute** over **UIP proofs**?

- Possibly need: **FunExt**, **Quotient types**?
- Would benefit from **computational UIP/proof-irrelevance**?
- No need: **univalence**?

Related Work

System	UIP	Equality … compute
Cubical-UIP	Propositional (+ automation)	proofs

Related Work

System	UIP	Equality … compute
Cubical-UIP	Propositional (+ automation)	proofs
XTT	**Definitional**	proofs

XTT [SAG22]

- *Definitional* UIP (c.f. our propostional UIP)
- type-checking algorithm requires inj. of type formers (contradicts at least LEM)

Related Work

System	UIP	Equality … compute
Cubical-UIP	Propositional (+ automation)	proofs
XTT	**Definitional**	proofs
Observational Type Theory (OTT)	**Definitional** (needs SProp)	**types**

XTT [SAG22]

- *Definitional* UIP (c.f. our propostional UIP)
- type-checking algorithm requires inj. of type formers (contradicts at least LEM)

Observational Type Theory [Alt99, AMS07, PLT25]

- UIP needs proof-irrelevant propositions
- Equality **types** compute instead of **proofs**

$$\sim_{A \times B} = \sim_A \times \sim_B$$

(c.f. *isomorphic* in Cubical)

Open question: Universe 🪐

What happens to sqFill U?

Open question: Universe 🪐

What happens to sqFill U?

Option 1: U is an inductive-recursive universe

```
data U : Type₁ where
  Pi : (A : U) →  (B : El A → Type) → U
  ...
```

Advantage: sqFill U (and hcomp U) compute by recursing on typecodes.

Open question: Universe 🪐

What happens to sqFill U?

Option 1: U is an inductive-recursive universe

```
data U : Type₁ where
  Pi : (A : U) →  (B : El A → Type) → U
  ...
```

Advantage: sqFill U (and hcomp U) compute by recursing on typecodes.

But...

- separate universe hierachy from the rest of Cubical Agda

Open question: Universe 🪐

What happens to sqFill U?

Option 1: U is an inductive-recursive universe

```
data U : Type₁ where
  Pi : (A : U) →  (B : El A → Type) → U
  ...
```

Advantage: sqFill U (and hcomp U) compute by recursing on typecodes.

But...

- separate universe hierachy from the rest of Cubical Agda
- hcomp U does *not* compute in other Cubical Agda variants
- anti-classical: contradicts with at least excluded middle.

Open question: Universe 🪐

What happens to sqFill U?

Option 1: U is an inductive-recursive universe

```
data U : Type₁ where
  Pi : (A : U) →  (B : El A → Type) → U
  ...
```

Advantage: sqFill U (and hcomp U) compute by recursing on typecodes.

But...

- separate universe hierachy from the rest of Cubical Agda
- hcomp U does *not* compute in other Cubical Agda variants
- anti-classical: contradicts with at least excluded middle.

Open question: Universe 🪐

Option 2: Universe is *not* closed

- sqFill U as a new type… ?
- sqFill only for small types?

Future work

Universes.

Records: nested Σ-types?

Datatypes:

- HITs: add sqFill constructor
- Non-HITs: add sqFill constructor; or "pattern-match" on corners (like hcomp)

Contribution

✅ `--cubical=no-glue`: (Cubical Agda - Glue)

⌚ `--cubical=uip`: (Cubical Agda - Glue + sqFill primitive recursing on types) [TND25]

Supports (Bool, Nat, Unit), (Pi, Sigma, PathP, List, Maybe, (Co-)products).

Welcome to experiment: https://github.com/yeejian-tan/agda/tree/cubical-uip

Questions?

References

[1La24] The 1Lab Development Team. 2024. The 1Lab. Retrieved from https://1lab.dev/

[Agd25] The Agda Community. 2025. Cubical Agda Library. Retrieved from https://github.com/agda/cubical

[AMS07] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. 2007. Observational equality, now!. In *Proceedings of the ACM Workshop Programming Languages meets Program Verification, PLPV 2007, Freiburg, Germany, October 5, 2007*, 2007. ACM, 57–68. https://doi.org/10.1145/1292597.1292608

[Alt99] Thorsten Altenkirch. 1999. Extensional Equality in Intensional Type Theory. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, 1999. IEEE Computer Society, 412–420. https://doi.org/10.1109/LICS.1999.782636

[CND25] Joris Ceulemans, Andreas Nuyts, and Dominique Devriese. 2025. BiSikkel: A Multimode Logical Framework in Agda. *Proc. ACM Program. Lang.* 9, POPL (2025), 210–240. https://doi.org/10.1145/3704844

[CNT26] Liang-Ting Chen, Fredrik Nordvall Forsberg, and Tzu-Chun Tsai. 2026. Can We Formalise Type Theory Intrinsically without Any Compromise? A Case Study in Cubical Agda. In *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '26)*, 2026. Association for Computing Machinery, Rennes, France, 201–215. https://doi.org/10.1145/3779031.3779090

[Coc19] Jesper Cockx. 2019. Comment: A variant of Cubical Agda that is consistent with UIP. Retrieved from https://github.com/agda/agda/issues/3750#issuecomment-490070548

References

[Coh+17] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2017. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. *FLAP* 4, 10 (2017), 3127–3170. Retrieved from http://collegepublications.co.uk/ifcolog/?00019

[HC21] Jason Z. S. Hu and Jacques Carette. 2021. Formalizing category theory in Agda. In *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, 2021. ACM, 327–342. https://doi.org/10.1145/3437992.3439922

[Huy+23] Sander Huyghebaert, Steven Keuchel, Coen De Roover, and Dominique Devriese. 2023. Formalizing, Verifying and Applying ISA Security Guarantees as Universal Contracts. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, 2023. ACM, 2083–2097. https://doi.org/10.1145/3576915.3616602

[Nuy24] Andreas Nuyts. 2024. Algebraic theories with contexts, in Cubical Agda. Retrieved from https://github.com/anuyts/ctx-alg

[PLT25] Loïc Pujet, Yann Leray, and Nicolas Tabareau. 2025. Observational Equality Meets CIC. *ACM Trans. Program. Lang. Syst.* 47, 2 (2025), 6:1–6:35. https://doi.org/10.1145/3719342

[Sch+25] Steven Schaefer, Nathan Varner, Pedro Henrique Azevedo de Amorim, and Max S. New. 2025. Intrinsic Verification of Parsers and Formal Grammar Theory in Dependent Lambek Calculus. *Proc. ACM Program. Lang.* 9, PLDI (June 2025). https://doi.org/10.1145/3729281

[Shu17] Michael Shulman. 2017. Cubical type theory with UIP. Retrieved from https://groups.google.com/g/homotopytypetheory/c/UegPjMrnx6I/m/UNBLHfZNBAAJ

References

[SAG22] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. 2022. A Cubical Language for Bishop Sets. *Log. Methods Comput. Sci.* 18, 1 (2022). https://doi.org/10.46298/LMCS-18(1:43)2022

[TND25] Yee-Jian Tan, Andreas Nuyts, and Dominique Devriese. 2025. Towards Computational UIP in Cubical Agda. Retrieved from https://arxiv.org/abs/2511.21209

[Uni13] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study. Retrieved from https://homotopytypetheory.org/book

[VMA21] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2021. Cubical Agda: A dependently typed programming language with univalence and higher inductive types. *J. Funct. Program.* 31, (2021), e8. https://doi.org/10.1017/S0956796821000034

Bonus: Demo

See a square compute! 👀