Formalizing Discrete Exterior Calculus

Workshop on Discrete Exterior Calculus 2026

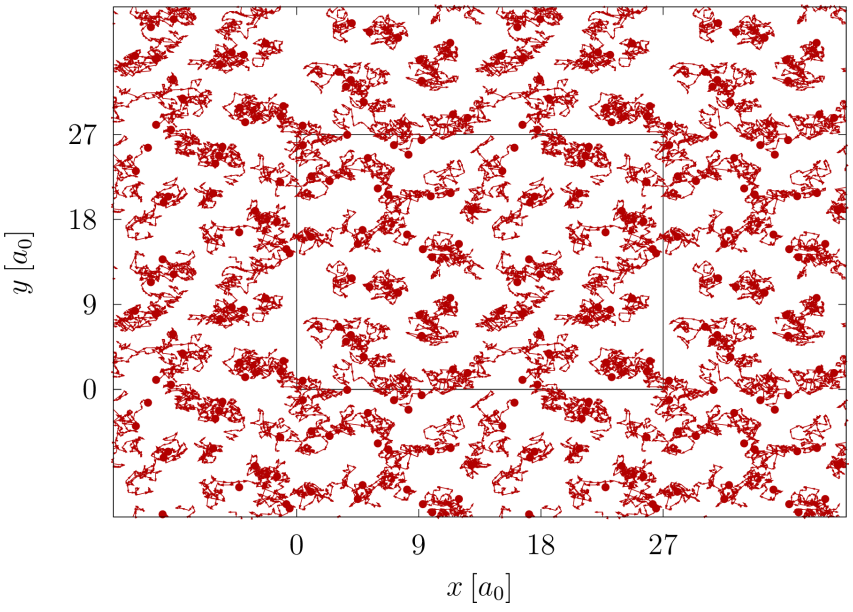
Sampsa Kiiskinen

University of Jyväskylä
Faculty of Information Technology

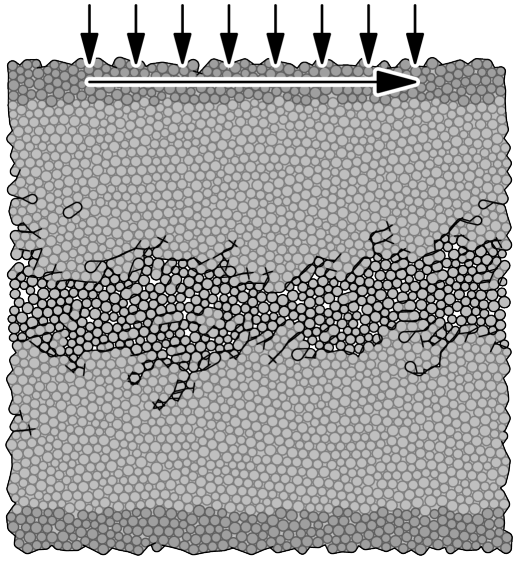
2026-04-29 11:30

S. Kiiskinen. *Formalizing Discrete Exterior Calculus*. 2026-04-29. URL: https://users.jyu.fi/~sapekiis/tmp/wdec-2026.pdf

Past Adventures



Path integral Monte Carlo simulation of helium



Discrete element method simulation of granite

S. Kiiskinen. “Polkuintegraaliperustilamenetelmä”. Cand. thesis. University of Jyväskylä, 2017-01-23. URL: https://urn.fi/URN:NBN:fi:jyu-201803271854

S. Kiiskinen. “Modeling Friction between Shearing Brittle Surfaces with a Discrete Element Method”. MA thesis. University of Jyväskylä, 2018-01-22. URL: https://urn.fi/URN:NBN:fi:jyu-201803271855

Past Adventures

“Physics”	“Bookkeeping”
Motivation	Data structures
Source of input parameters	Algorithms
Interpretation of results	Software engineering
⋮	⋮

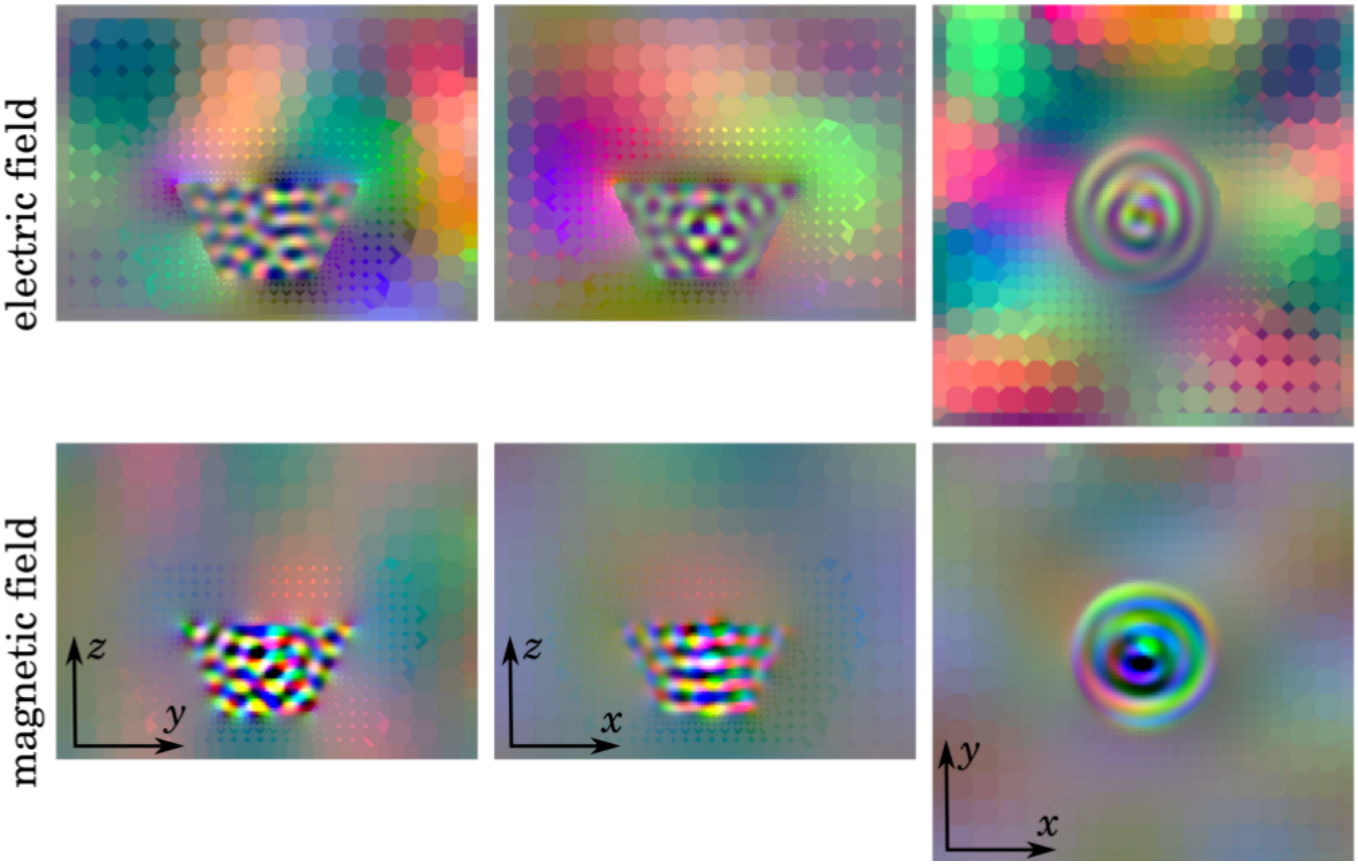
Dichotomy of my previous supervisor

Current Adventures

Alum	Title	Name
	Professor	Tuomo Rossi
	Professor	Lauri Kettunen
	University Lecturer	Sanna Mönkölä
	University Teacher	Sampsa Kiiskinen
	University Teacher	Tytti Saksa
†	Post-Doctoral Researcher	Jukka Räbinä
	Doctoral Student	Markus Kivioja
	Doctoral Student	Mikael Myyrä
†	Doctoral Student	Jonni Lohi

My research groupoid

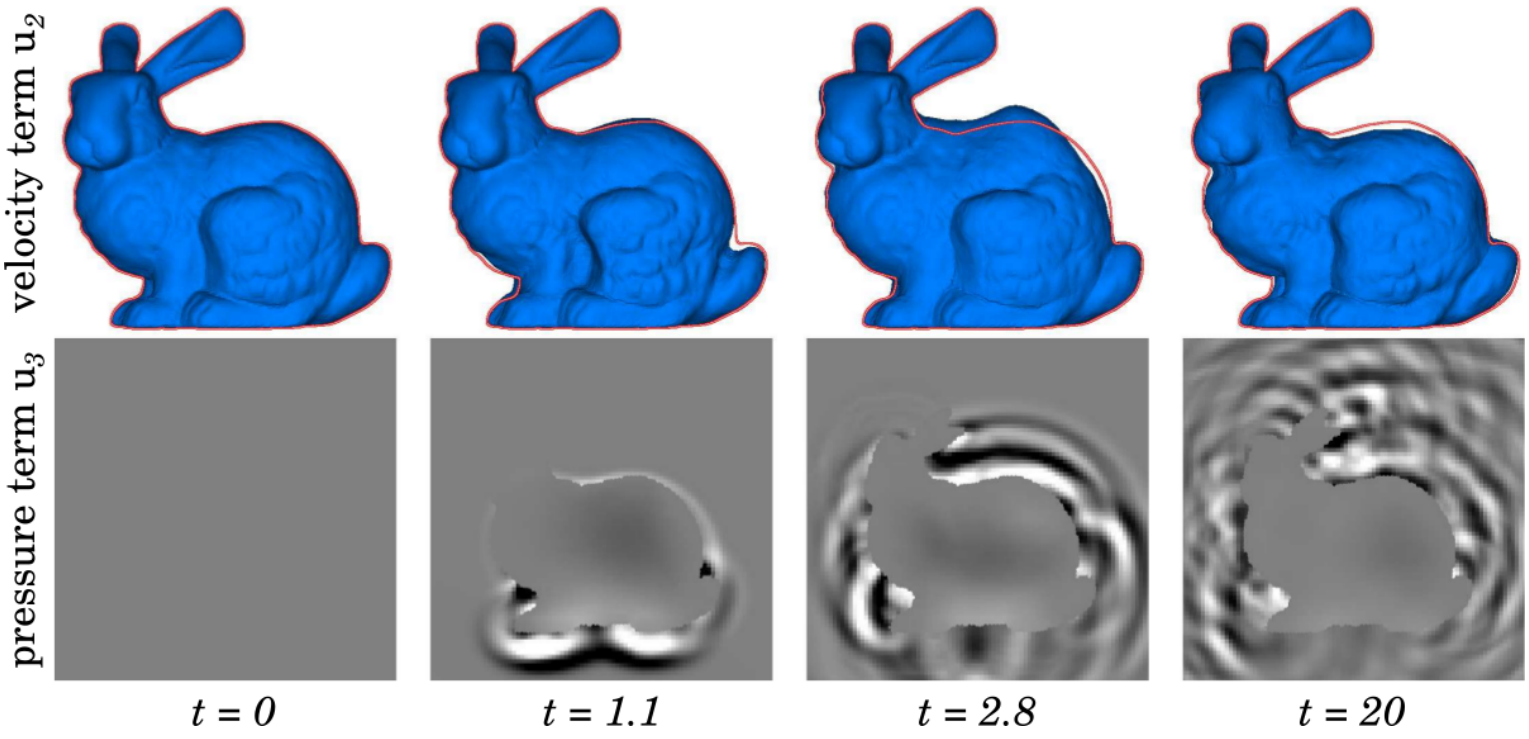
Current Adventures



Dielectric heating in a microwave oven

J. Räbinä et al. “High-Quality Discretizations for Microwave Simulations”. In: *2016 URSI International Symposium on Electromagnetic Theory*. IEEE, 2016-08, pp. 129–132. URL: https://urn.fi/URN:NBN:fi:jyu-201609264208

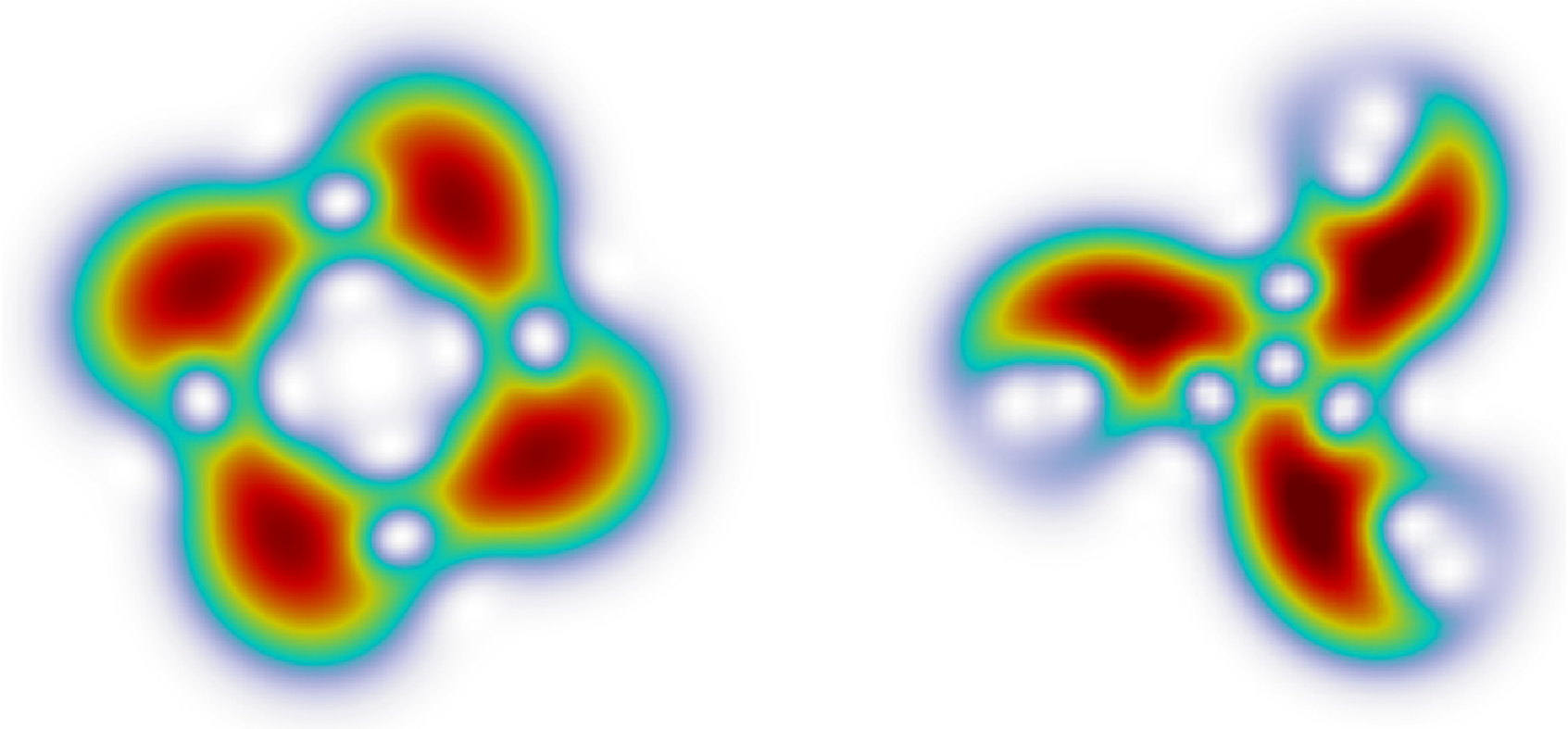
Current Adventures



Coupling of acoustic and elastic waves

J. Räbinä et al. “Generalized Wave Propagation Problems and Discrete Exterior Calculus”. In: *ESAIM: M2AN* 52.3 (2018-05), pp. 1195–1218. URL: https://urn.fi/URN:NBN:fi:jyu-201809214210

Current Adventures



Formation of quantum vortices in condensed matter

M. Kivioja et al. “GPU-Accelerated Time Integration of Gross–Pitaevskii Equation with Discrete Exterior Calculus”. In: *Computer Physics Communications* 278 (2022-09), p. 108427. URL: https://urn.fi/URN:NBN:fi:jyu-202208174180

Current Adventures

Open Questions

1. What is the most general set of problems that can be solved with dec?
   - We know several classes of problems, where the method excels.
   - We have found a few special cases, where the method seems to work.
   - We have yet to discover the exact boundary between the possible and the impossible.

Current Adventures

Open Questions

1. What is the most general set of problems that can be solved with dec?
   - We know several classes of problems, where the method excels.
   - We have found a few special cases, where the method seems to work.
   - We have yet to discover the exact boundary between the possible and the impossible.
2. How should dec be implemented to obtain maximal flexibility and correctness guarantees?
   - We have several numerical solvers that have been used in research papers.
   - We have one incomplete draft of a formal specification.
   - We do not have much in the way of quality software engineering.

Current Adventures

Attacking the Open Questions

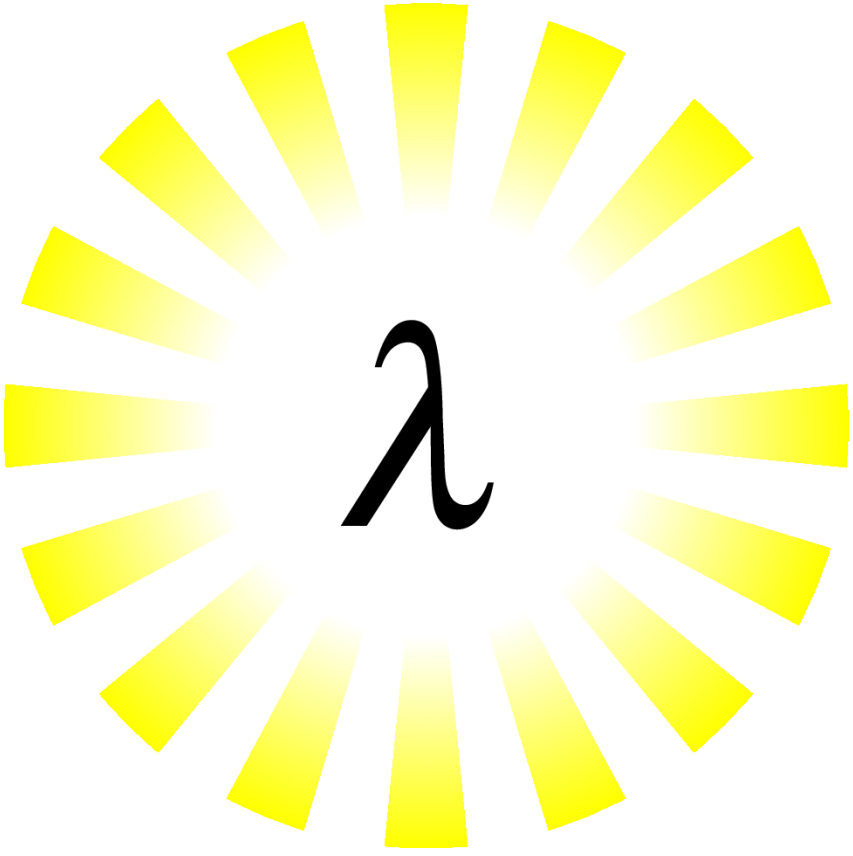
1. What is the most general set of problems that can be solved with dec?
   - Use dependent type theory to formalize the foundations of dec.
   - Implement the formalization in a proof assistant and prove its most essential properties.

Current Adventures

Attacking the Open Questions

1. What is the most general set of problems that can be solved with dec?
   - Use dependent type theory to formalize the foundations of dec.
   - Implement the formalization in a proof assistant and prove its most essential properties.
2. How should dec be implemented to obtain maximal flexibility and correctness guarantees?
   - Extract a reference implementation from the formalization.
   - Link existing implementations with the reference implementation and test them for conformance.

Current Adventures



This looks like a job for functional programming!

Current Adventures

Outline of the Attack

1. Define the type classes.
   - Type classes are essentially a mechanism for imposing constraints on parametrically polymorphic types.
   - They can be used to represent abstract mathematical structures without choosing a concrete representation.
   - Ultimately, we want to find type classes for dec and its constituents.
2. Prove the most interesting properties of the type classes.
3. Construct useful instances of the type classes.
4. Extract code from the proofs and instances.
5. Link the extracted code to existing implementations.

Current Adventures

Outline of the Attack

1. Define the type classes.
2. Prove the most interesting properties of the type classes.
   - Due to the Curry–Howard correspondence, formal proofs correspond to programs in a dependently-typed programming language.
   - The compiler verifies the proofs by type checking them.
   - We would like to prove various coherence and convergence conditions of dec.
   - We might end up having to prove a lot more.
3. Construct useful instances of the type classes.
4. Extract code from the proofs and instances.
5. Link the extracted code to existing implementations.

Current Adventures

Outline of the Attack

1. Define the type classes.
2. Prove the most interesting properties of the type classes.
3. Construct useful instances of the type classes.
   - Instances are concrete types that satisfy the constraints of the corresponding type classes.
   - They are weakly canonical, so the compiler can resolve them automatically.
4. Extract code from the proofs and instances.
5. Link the extracted code to existing implementations.

Current Adventures

Outline of the Attack

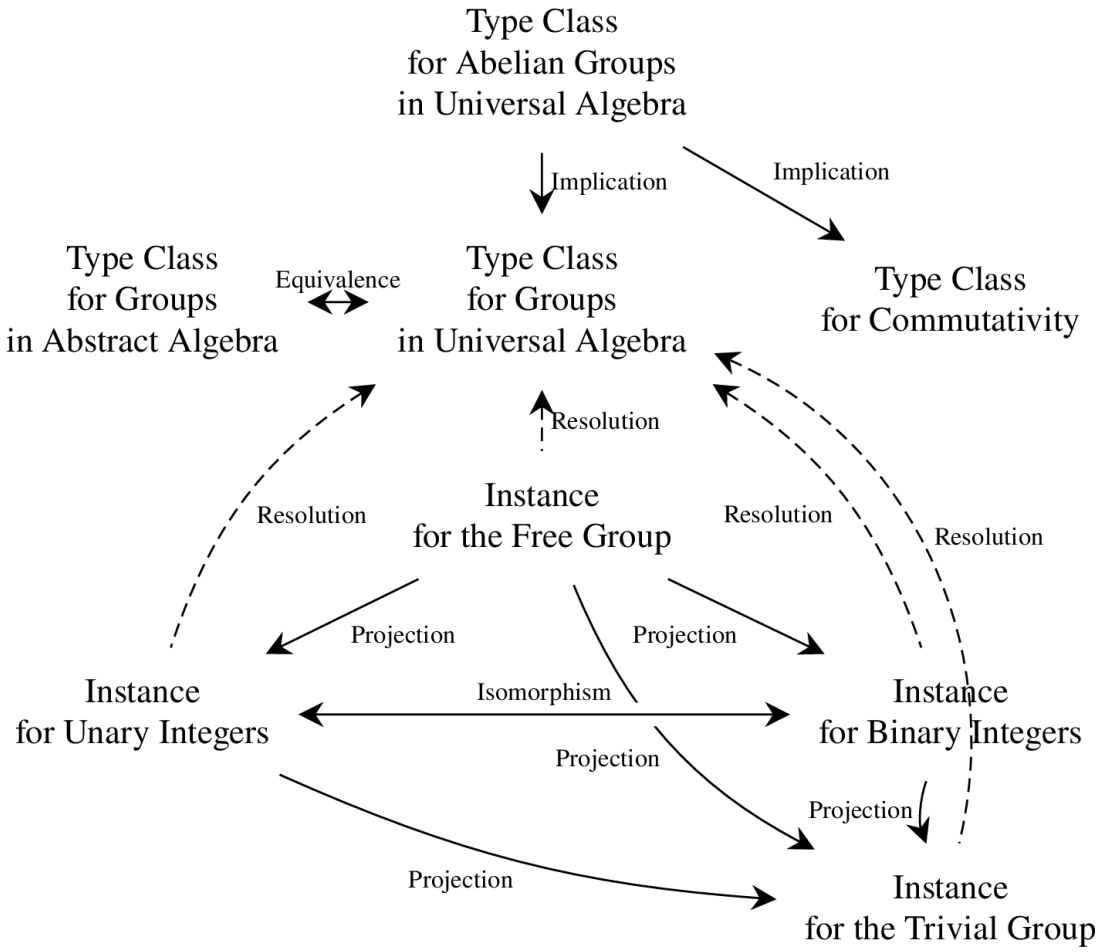
1. Define the type classes.
2. Prove the most interesting properties of the type classes.
3. Construct useful instances of the type classes.
4. Extract code from the proofs and instances.
   - Since formal proofs correspond to programs, we should be able extract executable code from our proofs.
   - We need to be careful, however, since most proofs do not contain any useful algorithms.
   - This issue can be avoided by assuming additional axioms like proof irrelevance, even though doing so is metatheoretically displeasing.
5. Link the extracted code to existing implementations.

Current Adventures

Outline of the Attack

1. Define the type classes.
2. Prove the most interesting properties of the type classes.
3. Construct useful instances of the type classes.
4. Extract code from the proofs and instances.
5. Link the extracted code to existing implementations.
   - Functional programming languages by and large rely on runtime system features like garbage collection, which makes them unsuitable for heavy numerical computations.
   - We still need to use existing implementations written in procedural programming languages.
   - Our plan is to hide them behind a common mathematically principled interface, which follows from code extraction.

Current Adventures



Everything in sight commutes

Current Adventures

Example Definitions

```
1  | Class IsAssoc (A : Type) (X : A -> A -> Prop)
2  |   (k : A -> A -> A) : Prop :=
3  |   assoc (x y z : A) : X (k x (k y z)) (k (k x y) z).
4  |
5  | Class IsGrp (A : Type) (X : A -> A -> Prop)
6  |   (x : A) (f : A -> A) (k : A -> A -> A) : Prop := {
7  |   grp_is_mon :> IsMon X x k;
8  |   grp_is_inv :> IsInv X x f k;
9  |   grp_is_proper :> IsProper (X ==> X) f;
10 | }.
```

Current Adventures

Example Proofs

```
1  | Context (A : Type) (X : A -> A -> Prop)
2  |   (x : A) (f : A -> A) (k : A -> A -> A) `{IsGrp X x f k}.
3  |
4  | #[export] Instance grp_is_invol : IsInvol X f.
5  | Proof with note.
6  |   intros y...
7  |   rewrite <- (unl_elem_r (- (- y))).
8  |   rewrite <- (inv_l y).
9  |   rewrite (assoc (- (- y)) (- y) y).
10 |   rewrite (inv_l (- y)).
11 |   rewrite (unl_elem_l y).
12 |   reflexivity.
13 | Qed.
14 |
15 | #[export] Instance grp_is_cancel : IsCancel X k.
16 | Proof. esplit; typeclasses eauto. Qed.
```

Current Adventures

Example Instances

```
Module Additive.
#[export] Instance Z_has_null_op : HasNullOp Z := 0.
#[export] Instance Z_has_un_op : HasUnOp Z := Z.opp.
#[export] Instance Z_has_bin_op : HasBinOp Z := Z.add.
End Additive.

Module Multiplicative.
#[export] Instance Z_has_null_op : HasNullOp Z := 1.
#[export] Instance Z_has_bin_op : HasBinOp Z := Z.mul.
End Multiplicative.

Ltac ecrush :=
  repeat (try typeclasses eauto; esplit);
  hnf in *; eauto with zarith.

#[export] Instance Z_add_is_grp : IsGrp _=_ 0 Z.opp Z.add.
Proof. ecrush. Qed.
```

Current Adventures

$$\mathbb{R} \simeq \mathbb{F}$$

real numbers

IEEE-754 floating-point numbers

If only theory agreed with practice...

Future Adventures

Some Issues

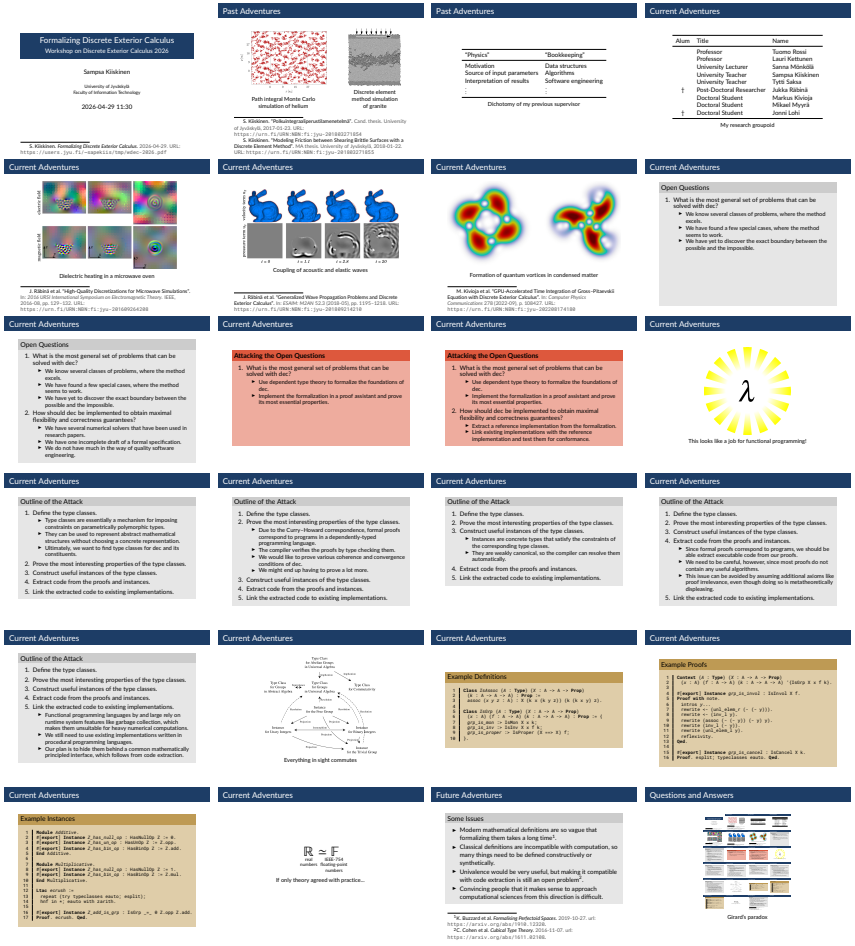
- Modern mathematical definitions are so vague that formalizing them takes a long time[1].
- Classical definitions are incompatible with computation, so many things need to be defined constructively or synthetically.
- Univalence would be very useful, but making it compatible with code extraction is still an open problem[2].
- Convincing people that it makes sense to approach computational sciences from this direction is difficult.

---

[1] K. Buzzard et al. *Formalising Perfectoid Spaces*. 2019-10-27. url: https://arxiv.org/abs/1910.12320.

[2] C. Cohen et al. *Cubical Type Theory*. 2016-11-07. url: https://arxiv.org/abs/1611.02108.

Questions and Answers



Girard's paradox