Semi-formal verification as a routine tool

Neil Strickland

The Goal

The Goal

- Goal: Computer checkable proofs as a routine tool in ongoing mathematical research.

The Goal

- Goal: Computer checkable proofs as a routine tool in ongoing mathematical research.
- As far as I know, this is not really happening at the moment.

The Goal

- Goal: Computer checkable proofs as a routine tool in ongoing mathematical research.
- As far as I know, this is not really happening at the moment.
- I have tried to start doing this (with Agda, Coq, Isabelle) but have not succeeded.

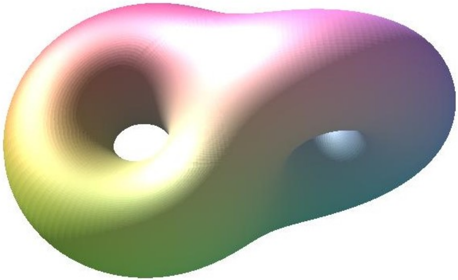
The Goal

- Goal: Computer checkable proofs as a routine tool in ongoing mathematical research.
- As far as I know, this is not really happening at the moment.
- I have tried to start doing this (with Agda, Coq, Isabelle) but have not succeeded.
- However, I have used a kind of semi-formal verification based on Maple for a wide range of projects, some of which are large.

The Goal

- Goal: Computer checkable proofs as a routine tool in ongoing mathematical research.
- As far as I know, this is not really happening at the moment.
- I have tried to start doing this (with Agda, Coq, Isabelle) but have not succeeded.
- However, I have used a kind of semi-formal verification based on Maple for a wide range of projects, some of which are large.
- In this talk I will discuss my experience of this, in the hope that it might provoke new ideas about the move to fully formal verification.

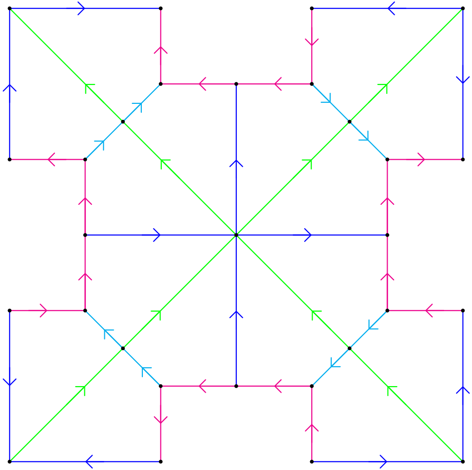
Geometry of embedded surfaces



$$EX^* = \{x \in S^3 \mid (3x_3^2 - 2)x_4 + \sqrt{2}(x_1^2 - x_2^2)x_3 = 0\},$$

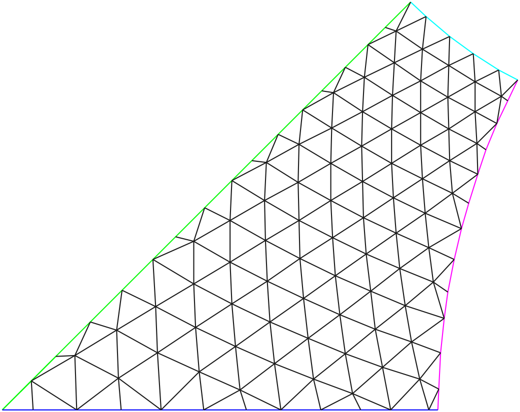
This is a large and complex project, with a 160 page monograph, 30000 lines of Maple code, and 2500 assertions checked by Maple. It involves Riemannian geometry of surfaces embedded in S^3

Geometry of embedded surfaces



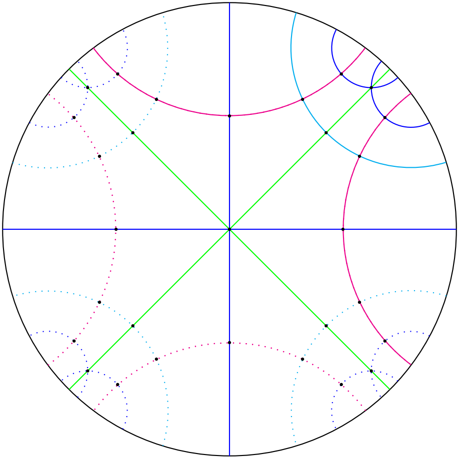
. . . equivariant combinatorics of simplicial complexes with finite group action

Geometry of embedded surfaces



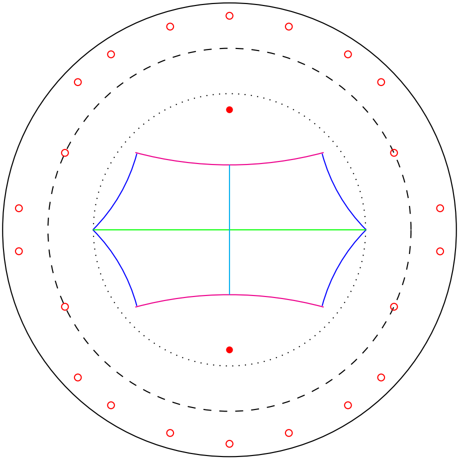
. . . equivariant combinatorics of simplicial complexes with finite group action

Geometry of embedded surfaces



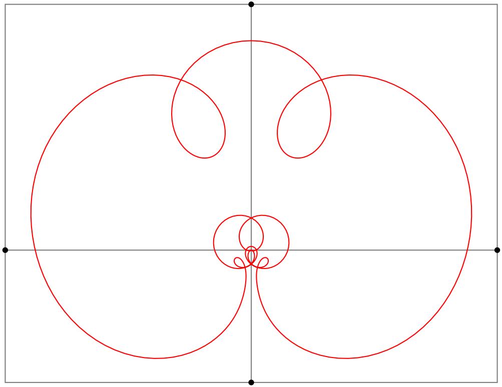
. . . hyperbolic geometry and combinatorial group theory of Fuchsian groups

Geometry of embedded surfaces



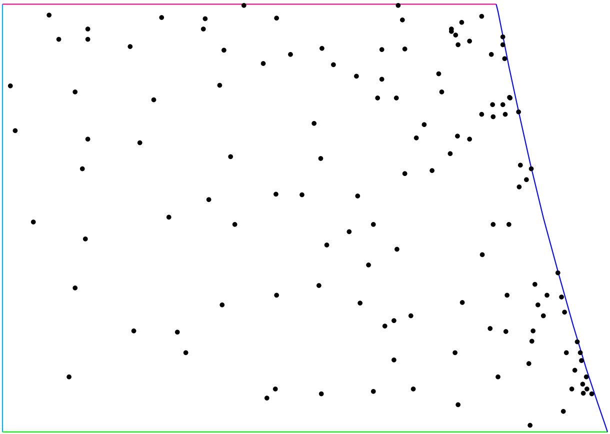
. . . analytic and geometric theory of differential equations, Schwarzian derivatives and conformal mappings

Geometry of embedded surfaces



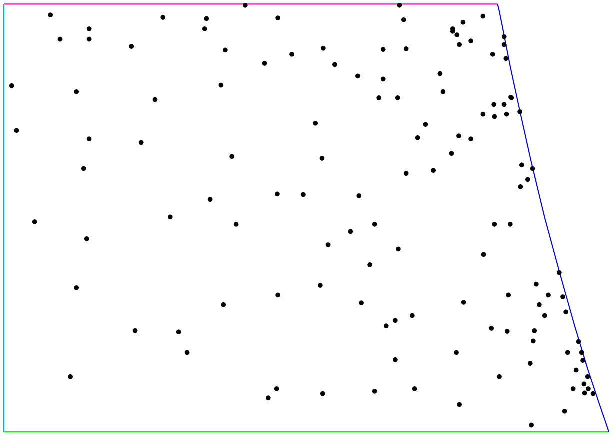
. . . theory of automorphic forms

Geometry of embedded surfaces



. . . brute force solution of multivariable diophantine equations

Geometry of embedded surfaces



. . . brute force solution of multivariable diophantine equations

. . . and various other ingredients.

Examples of data and formulae

	χ_0	χ_1	χ_2	χ_3	χ_4	χ_5	χ_6	χ_7	χ_8	χ_9
1	1	1	1	1	1	1	1	1	2	2
λ^2	1	1	1	1	1	1	1	1	-2	-2
$\mu\nu$	1	1	-1	-1	1	1	-1	-1	2	-2
$\lambda^2\mu\nu$	1	1	-1	-1	1	1	-1	-1	-2	2
$\lambda^{\pm 1}$	1	-1	1	-1	-1	1	-1	1	0	0
$\mu, \lambda^2\mu$	1	1	-1	-1	-1	-1	1	1	0	0
$\lambda^{\pm 1}\mu$	1	-1	-1	1	1	-1	-1	1	0	0
$\nu, \lambda^2\nu$	1	1	1	1	-1	-1	-1	-1	0	0
$\lambda^{\pm 1}\nu$	1	-1	1	-1	1	-1	1	-1	0	0
$\lambda^{\pm 1}\mu\nu$	1	-1	-1	1	-1	1	1	-1	0	0

Examples of data and formulae

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0			0	$\frac{\pi}{2}$	π	$-\frac{\pi}{2}$	$\frac{\pi}{4}$	$\frac{3\pi}{4}$	$-\frac{3\pi}{4}$	$-\frac{\pi}{4}$				
1	0	π					$\frac{\pi}{2}$		$-\frac{\pi}{2}$					
2	0	π						$\frac{\pi}{2}$		$-\frac{\pi}{2}$				
3				$\frac{\pi}{2}$		$-\frac{\pi}{2}$						0		π
4			$-\frac{\pi}{2}$		$\frac{\pi}{2}$						0		π	
5	0											π		
6	0										π			
7		0												π
8		0											π	

$$
\begin{array}{lll}
\lambda(c_0(t)) = c_0(t+\pi/2) & \mu(c_0(t)) = c_0(-t) & \nu(c_0(t)) = c_0(-t) \\
\lambda(c_1(t)) = c_2(t) & \mu(c_1(t)) = c_2(t+\pi) & \nu(c_1(t)) = c_2(-t) \\
\lambda(c_2(t)) = c_1(-t) & \mu(c_2(t)) = c_1(t+\pi) & \nu(c_2(t)) = c_1(-t) \\
\lambda(c_3(t)) = c_4(t) & \mu(c_3(t)) = c_3(t+\pi) & \nu(c_3(t)) = c_3(-t) \\
\lambda(c_4(t)) = c_3(-t) & \mu(c_4(t)) = c_4(-t-\pi) & \nu(c_4(t)) = c_4(t) \\
\lambda(c_5(t)) = c_6(t) & \mu(c_5(t)) = c_7(t) & \nu(c_5(t)) = c_5(t) \\
\lambda(c_6(t)) = c_5(-t) & \mu(c_6(t)) = c_8(-t) & \nu(c_6(t)) = c_6(-t) \\
\lambda(c_7(t)) = c_8(t) & \mu(c_7(t)) = c_5(t) & \nu(c_7(t)) = c_7(t) \\
\lambda(c_8(t)) = c_7(-t) & \mu(c_8(t)) = c_6(-t) & \nu(c_8(t)) = c_8(-t)
\end{array}
$$

Examples of data and formulae

$$
\begin{aligned}
c_0(t) &= j'(-\sqrt{(a^{-1}-a)^2+4\sin^2(2t)},\ e^{it},\ -e^{-it}) \\
c_1(t) &= j'\left(\frac{1+i}{8\sqrt{2}}\sin(t)\sqrt{16\cos(t)^2+(a+a^{-1})^2\sin(t)^4},\ \frac{1+\cos(t)}{2},\ \frac{1-\cos(t)}{2}i\right) \\
c_2(t) &= \lambda(c_1(t)) \\
c_3(t) &= j'\left(-i\frac{a^{-1}-a}{8}\sin(t)\sqrt{(1+a)^4-(1-a)^4\cos(t)^2}\sqrt{(1+a)^2-(1-a)^2\cos(t)^2},\right. \\
&\qquad \left.\frac{(1+a)+(1-a)\cos(t)}{2},\ \frac{(1+a)-(1-a)\cos(t)}{2}\right) \\
c_4(t) &= \lambda(c_3(t)) \\
c_5(t) &= j\left(\frac{\sin(t)}{8}\sqrt{2a(3-\cos(t))(4-a^4(1-\cos(t))^2)},\ a\frac{1-\cos(t)}{2}\right) \\
c_6(t) &= \lambda(c_5(t)) \\
c_7(t) &= \mu(c_5(t)) \\
c_8(t) &= \lambda\mu(c_5(t)).
\end{aligned}
$$

Here $a \in (0,1)$ so all square roots are of positive quantities. Maple is able to take this into account when simplifying but there is no documentation of the algorithm used.

Examples of data and formulae

There are morphisms of elliptic curves

$$E^+(a) \xrightarrow{\pi^+} E^-(a) \xrightarrow{\pi^-} E^+(a)$$

given generically by

$$\pi^+(j(y,x)) = j\left(\frac{\sqrt{2}y((1-x)^2+b_-^2x^2)}{((1-x)^2-b_-^2x^2)^2}, \frac{2x(x-1)}{((1-x)^2-b_-^2x^2)}\right)$$
$$\pi^-(j(y,x)) = j\left(\frac{\sqrt{2}y((1-x)^2-b_+^2x^2)}{((1-x)^2+b_+^2x^2)^2}, \frac{2x(x-1)}{((1-x)^2+b_+^2x^2)}\right).$$

We need to verify that the denominators do not cause trouble. For this we use homogeneous coordinates, check some polynomial identities that were found by Gröbner methods, and apply some logic.

Examples of data and formulae

Fuchsian group with generators $\beta_0, \ldots, \beta_7, \lambda, \mu, \nu$ and relations

$$\beta_{k+4} = \beta_k^{-1} \qquad \beta_0\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7 = 1 \qquad \lambda^4 = \mu^2 = \nu^2 = (\lambda\nu)^2 = 1$$

$$(\lambda\mu)^2 = \beta_7\beta_6 \qquad (\nu\mu)^2 = \beta_6\beta_0\beta_7\beta_6 \qquad \lambda\beta_k\lambda^{-1} = \lambda_*(\beta_k) \qquad \mu\beta_k\mu = \mu_*(\beta_k) \qquad \nu\beta_k\nu = \nu_*(\beta_k).$$

	β_0	β_1	β_2	β_3	β_4	β_5	β_6	β_7
λ_*	β_2	β_3	β_4	β_5	β_6	β_7	β_0	β_1
μ_*	$\beta_2\beta_0\beta_1$	$\beta_5\beta_4\beta_3$	$\beta_0\beta_7\beta_6$	$\beta_2\beta_3\beta_1$	$\beta_5\beta_4\beta_6$	$\beta_7\beta_0\beta_1$	$\beta_2\beta_3\beta_4$	$\beta_5\beta_7\beta_6$
ν_*	β_0	$\beta_2\beta_1\beta_2$	β_6	$\beta_0\beta_7\beta_0$	β_4	$\beta_6\beta_5\beta_6$	β_2	$\beta_4\beta_3\beta_4$

Fix $b \in (0,1)$ with $b_\pm = \sqrt{1 \pm b^2}$ and consider action on unit disc by

$$\lambda(z) = iz \qquad \beta_0(z) = \frac{b_+ z + 1}{z + b_+}$$

$$\mu(z) = \frac{b_+ z - b^2 - i}{(b^2 - i)z - b_+} \qquad \beta_1(z) = \frac{b_+^3 z - (2+i)b^2 - i}{((i-2)b^2 + i)z + b_+^3}$$

$$\nu(z) = \overline{z} \qquad \beta_{2n}(z) = i^n \beta_0(z/i^n)$$

$$\beta_{2n+1}(z) = i^n \beta_1(z/i^n).$$

Methods for semi-formal verification

Methods for semi-formal verification

- Somewhat similar philosophy to automated unit testing in software engineering.

Methods for semi-formal verification

- Somewhat similar philosophy to automated unit testing in software engineering.
- For small, finite combinatorial structures: exhaustive check of cases.

Methods for semi-formal verification

- Somewhat similar philosophy to automated unit testing in software engineering.
- For small, finite combinatorial structures: exhaustive check of cases.
- For polynomial identities: trust Maple to verify.

Methods for semi-formal verification

- Somewhat similar philosophy to automated unit testing in software engineering.
- For small, finite combinatorial structures: exhaustive check of cases.
- For polynomial identities: trust Maple to verify.
- Rational function identities: clear denominators, with care.

Methods for semi-formal verification

- Somewhat similar philosophy to automated unit testing in software engineering.
- For small, finite combinatorial structures: exhaustive check of cases.
- For polynomial identities: trust Maple to verify.
- Rational function identities: clear denominators, with care.
- Polynomial identites mod relations: trust Maple's Gröbner package (but could ask for witnesses).

Methods for semi-formal verification

- Somewhat similar philosophy to automated unit testing in software engineering.
- For small, finite combinatorial structures: exhaustive check of cases.
- For polynomial identities: trust Maple to verify.
- Rational function identities: clear denominators, with care.
- Polynomial identites mod relations: trust Maple's Gröbner package (but could ask for witnesses).
- Expressions with trig functions, radicals, positivity conditions: trust Maple, more cautiously. Also check random examples.

Methods for semi-formal verification

- Somewhat similar philosophy to automated unit testing in software engineering.
- For small, finite combinatorial structures: exhaustive check of cases.
- For polynomial identities: trust Maple to verify.
- Rational function identities: clear denominators, with care.
- Polynomial identites mod relations: trust Maple's Gröbner package (but could ask for witnesses).
- Expressions with trig functions, radicals, positivity conditions: trust Maple, more cautiously. Also check random examples.
- More complex proofs: isolate independent (in)equalities as far as possible, check them as above or with random examples, with high precision numerics where appropriate.

Comments on the framework

Comments on the framework

- We performed very extensive interactive explorations in Maple to find out what was true, supported by excellent visualisation tools etc. We then used the same framework to set up automated verification tests. This is convenient.

Comments on the framework

- We performed very extensive interactive explorations in Maple to find out what was true, supported by excellent visualisation tools etc. We then used the same framework to set up automated verification tests. This is convenient.
- Maple can do a very wide range of things. There are some things that it does not do especially well, but it makes life much easier to have everything in the same package.

Comments on the framework

- We performed very extensive interactive explorations in Maple to find out what was true, supported by excellent visualisation tools etc. We then used the same framework to set up automated verification tests. This is convenient.
- Maple can do a very wide range of things. There are some things that it does not do especially well, but it makes life much easier to have everything in the same package.
- Maple allows, but does not enforce, various kinds of type checking. This limits the level of rigour that can be achieved, but in practice it seems a good compromise.

Comments on the framework

- We performed very extensive interactive explorations in Maple to find out what was true, supported by excellent visualisation tools etc. We then used the same framework to set up automated verification tests. This is convenient.
- Maple can do a very wide range of things. There are some things that it does not do especially well, but it makes life much easier to have everything in the same package.
- Maple allows, but does not enforce, various kinds of type checking. This limits the level of rigour that can be achieved, but in practice it seems a good compromise.
- Automated testing definitely catches many errors. A large fraction are just transcription problems, or arise from adjustments in one part of the project not properly carried over to other parts. Some others are more subtle and interesting.

Other projects and libraries

Other projects and libraries

- I have a library of Maple code designed for semi-formal verification. Some projects use this library, some will be migrated, some are unsuitable for migration.

Other projects and libraries

- I have a library of Maple code designed for semi-formal verification. Some projects use this library, some will be migrated, some are unsuitable for migration.
- There is a lot of code for finite combinatorial structures: different kinds of (chains of) orders, partitions, graphs, trees, maps, relations.

Other projects and libraries

- I have a library of Maple code designed for semi-formal verification. Some projects use this library, some will be migrated, some are unsuitable for migration.
- There is a lot of code for finite combinatorial structures: different kinds of (chains of) orders, partitions, graphs, trees, maps, relations.
- There is code for dealing with various topological spaces. Most of these can be thought of as compact subspaces $X \subset \mathrm{Map}(A, \mathbb{R})$ (for some finite A) defined by polynomial (in)equalities, spectral conditions etc.

Other projects and libraries

- I have a library of Maple code designed for semi-formal verification. Some projects use this library, some will be migrated, some are unsuitable for migration.
- There is a lot of code for finite combinatorial structures: different kinds of (chains of) orders, partitions, graphs, trees, maps, relations.
- There is code for dealing with various topological spaces. Most of these can be thought of as compact subspaces $X \subset \text{Map}(A, \mathbb{R})$ (for some finite A) defined by polynomial (in)equalities, spectral conditions etc.
- Some spaces are quotients of spaces as above.

Other projects and libraries

- I have a library of Maple code designed for semi-formal verification. Some projects use this library, some will be migrated, some are unsuitable for migration.
- There is a lot of code for finite combinatorial structures: different kinds of (chains of) orders, partitions, graphs, trees, maps, relations.
- There is code for dealing with various topological spaces. Most of these can be thought of as compact subspaces $X \subset \mathrm{Map}(A, \mathbb{R})$ (for some finite A) defined by polynomial (in)equalities, spectral conditions etc.
- Some spaces are quotients of spaces as above.
- There is code for various rings, often arising as (generalised) cohomology rings of specific topological spaces. This typically involves Gröbner bases.

Other projects and libraries

- I have a library of Maple code designed for semi-formal verification. Some projects use this library, some will be migrated, some are unsuitable for migration.
- There is a lot of code for finite combinatorial structures: different kinds of (chains of) orders, partitions, graphs, trees, maps, relations.
- There is code for dealing with various topological spaces. Most of these can be thought of as compact subspaces $X \subset \mathrm{Map}(A, \mathbb{R})$ (for some finite A) defined by polynomial (in)equalities, spectral conditions etc.
- Some spaces are quotients of spaces as above.
- There is code for various rings, often arising as (generalised) cohomology rings of specific topological spaces. This typically involves Gröbner bases.
- There is code for many morphisms and (co)operad structures.

Chained preorders

Chained preorders

- Example: given a finite set A and $N > 0$ let $\mathrm{ICP}_N(A)$ be the set of chains $(Q_1, \dots, Q_N)$ of preorders on A such that Q_1 is total, Q_N is separated, any two elements are Q_i-comparable iff Q_{i-1}-equivalent. This set is itself finite, and partially ordered.

Chained preorders

- Example: given a finite set A and $N > 0$ let $\mathrm{ICP}_N(A)$ be the set of chains $(Q_1, \ldots, Q_N)$ of preorders on A such that Q_1 is total, Q_N is separated, any two elements are Q_i-comparable iff Q_{i-1}-equivalent. This set is itself finite, and partially ordered.
- There is code to recognise elements of $\mathrm{ICP}_N(A)$. This could easily be converted to a definition of $\mathrm{ICP}_N(A)$ as a dependent type in Agda/Coq/Isabelle. Architecture is designed with this in mind.

Chained preorders

- Example: given a finite set A and $N > 0$ let $\mathrm{ICP}_N(A)$ be the set of chains $(Q_1, \ldots, Q_N)$ of preorders on A such that Q_1 is total, Q_N is separated, any two elements are Q_i-comparable iff Q_{i-1}-equivalent. This set is itself finite, and partially ordered.
- There is code to recognise elements of $\mathrm{ICP}_N(A)$. This could easily be converted to a definition of $\mathrm{ICP}_N(A)$ as a dependent type in Agda/Coq/Isabelle. Architecture is designed with this in mind.
- There is code to define the partial order on $\mathrm{ICP}_N(A)$; also easily translatable. A small amount of extra work would give a formal proof of partial order axioms.

Chained preorders

- Example: given a finite set A and $N > 0$ let $\mathrm{ICP}_N(A)$ be the set of chains $(Q_1, \dots, Q_N)$ of preorders on A such that Q_1 is total, Q_N is separated, any two elements are Q_i-comparable iff Q_{i-1}-equivalent. This set is itself finite, and partially ordered.
- There is code to recognise elements of $\mathrm{ICP}_N(A)$. This could easily be converted to a definition of $\mathrm{ICP}_N(A)$ as a dependent type in Agda/Coq/Isabelle. Architecture is designed with this in mind.
- There is code to define the partial order on $\mathrm{ICP}_N(A)$; also easily translatable. A small amount of extra work would give a formal proof of partial order axioms.
- There is code to enumerate the elements of $\mathrm{ICP}_N(A)$ (for small (A, N)). Translation would implicitly involve a formal proof of correctness; probably substantial extra work.

Chained preorders

Chained preorders

- There is code to generate random elements of $\mathrm{ICP}_N(A)$ (tractable for larger (A, N)). The algorithm overweights special cases. This is extremely valuable for exploration and hypothesis testing. I do not know how to fit anything like that in standard proof assistants.

Chained preorders

- There is code to generate random elements of $\mathrm{ICP}_N(A)$ (tractable for larger (A, N)). The algorithm overweights special cases. This is extremely valuable for exploration and hypothesis testing. I do not know how to fit anything like that in standard proof assistants.
- We are interested in the homotopy theory of (the geometric realisation of) $\mathrm{ICP}_N(A)$ and related posets. There is code for various general constructions in this theory (shellings, discrete Morse theory, the geometric realisation itself).

Chained preorders

- There is code to generate random elements of $\mathrm{ICP}_N(A)$ (tractable for larger (A, N)). The algorithm overweights special cases. This is extremely valuable for exploration and hypothesis testing. I do not know how to fit anything like that in standard proof assistants.
- We are interested in the homotopy theory of (the geometric realisation of) $\mathrm{ICP}_N(A)$ and related posets. There is code for various general constructions in this theory (shellings, discrete Morse theory, the geometric realisation itself).
- The discipline of coding all posets, morphisms and operad structures helps ensure that everything is properly specified. We can test statements on a complete list or random selection of elements to ensure that edge cases are handled correctly.