Computable and Non-Computable 2-groups

Andrew W Swan

University of Ljubljana

April 16, 2025

Computable (1-)groups

Definition (Rabin)

A *computable group* is that group G whose carrier set is a computable subset of $\mathbb{N}$ and whose group multiplication is computable.

Theorem (Rabin)

A finitely generated group is computable iff it has soluble word problem.

Theorem (Boone, Novikov, Rabin)

There are examples of finitely generated groups with insoluble word problem: e.g. the group $\langle x, y, u, t \mid u^e x u^{-e} = t^e y t^{-e}, \varphi_e(e)\downarrow \rangle$ *is finitely generated, has computably enumerable relations but does not have soluble word problem, so is not computable.*

Overall plan

We will do the same thing for 2-groups, the simplest higher dimensional generalisation of group.

- We will use a simple definition of 2-group from homotopy type theory (HoTT).
- We will use ideas from synthetic computability to simplify what it means for a 2-group to be computable.
- Using the cubical assemblies model of HoTT we can relate the synthetic definition back to a concrete definition explicitly mentioning computable functions.
- Using a result due to Owen Milner (a HoTT version of a theorem due to Sính) we can more explicitly describe the algebraic structure associated to a computable 2-group.

Synthetic computability theory

In classical computability theory, the set of functions $\mathbb{N} \to \mathbb{N}$ contains both computable and non computable functions, and e.g. the halting set provides an example of a function $\mathbb{N} \to 2$ that is not computable.

Synthetic computability theory

In classical computability theory, the set of functions $\mathbb{N} \to \mathbb{N}$ contains both computable and non computable functions, and e.g. the halting set provides an example of a function $\mathbb{N} \to 2$ that is not computable.

In synthetic computability theory, everything is "computable by default," Formally, we assume the axiom of Church's thesis, which states *all* functions $\mathbb{N} \to \mathbb{N}$ are computable, justified by it holding in realizability models.

Synthetic computability theory

In classical computability theory, the set of functions $\mathbb{N} \to \mathbb{N}$ contains both computable and non computable functions, and e.g. the halting set provides an example of a function $\mathbb{N} \to 2$ that is not computable.

In synthetic computability theory, everything is "computable by default," Formally, we assume the axiom of Church's thesis, which states *all* functions $\mathbb{N} \to \mathbb{N}$ are computable, justified by it holding in realizability models.

Hence we need a non trivial definition to formulate what a non computable function is. This can be done e.g. using ∇, the $\neg\neg$-sheafification modality. Think of ∇A as "A with computational data stripped away." Elements of ∇A still need to be uniquely defined, but we don't have to compute them.

Synthetic computability theory

In classical computability theory, the set of functions $\mathbb{N} \to \mathbb{N}$ contains both computable and non computable functions, and e.g. the halting set provides an example of a function $\mathbb{N} \to 2$ that is not computable.

In synthetic computability theory, everything is "computable by default," Formally, we assume the axiom of Church's thesis, which states *all* functions $\mathbb{N} \to \mathbb{N}$ are computable, justified by it holding in realizability models.

Hence we need a non trivial definition to formulate what a non computable function is. This can be done e.g. using ∇, the $\neg\neg$-sheafification modality. Think of ∇A as "A with computational data stripped away." Elements of ∇A still need to be uniquely defined, but we don't have to compute them.

The halting set is an example of a function $\mathbb{N} \to \nabla 2$ that does not extend to any function $\mathbb{N} \to 2$.

Computable 1-groups in synthetic computability theory

Definition

A *synthetic computable group* is a group whose carrier set is a decidable subset of $\mathbb{N}$

Observation

Assuming **CT** *a group is synthetically computable iff it is computable.*

- The synthetic definition is simpler.
- Group structures on $\mathbb{N}$ are "computable by default," but we can still talk about non-computable group structures on $\mathbb{N}$ e.g. by considering group structures on $\nabla\mathbb{N}$.

Groups-as-spaces

Definition

A *pointed type* is a type A together with an element $a_0 : A$. The *loop space* $\Omega(A, a_0)$ is the identity type $a_0 =_A a_0$, i.e. proofs that a_0 is equal to itself.

Theorem (Buchholtz, Van Doorn, Rijke)

Every group G is of the form $\Omega(BG, *_{BG})$ for a unique pointed, connected, 1-truncated type $(BG, *_{BG})$. We call BG the classifying space of G.

Groups-as-spaces

Definition

A *pointed type* is a type A together with an element $a_0 : A$. The *loop space* $\Omega(A, a_0)$ is the identity type $a_0 =_A a_0$, i.e. proofs that a_0 is equal to itself.

Theorem (Buchholtz, Van Doorn, Rijke)

Every group G is of the form $\Omega(BG, *_{BG})$ for a unique pointed, connected, 1-truncated type $(BG, *_{BG})$. We call BG the classifying space of G.

Key idea: We can just work with the classifying spaces, rather than groups themselves.

- Basic group theory goes through surprisingly smoothly.
- Some results are easier to prove this way.
- It is much easier to generalise from groups to higher groups.

Groups-as-spaces

Definition

A *pointed type* is a type A together with an element $a_0 : A$. The *loop space* $\Omega(A, a_0)$ is the identity type $a_0 =_A a_0$, i.e. proofs that a_0 is equal to itself.

Theorem (Buchholtz, Van Doorn, Rijke)

Every group G is of the form $\Omega(BG, *_{BG})$ for a unique pointed, connected, 1-truncated type $(BG, *_{BG})$. We call BG the classifying space of G.

Key idea: We can just work with the classifying spaces, rather than groups themselves.

- Basic group theory goes through surprisingly smoothly.
- Some results are easier to prove this way.
- It is much easier to generalise from groups to higher groups.

Definition (Buchholtz, Van Doorn, Rijke)

A 2-*group* is a pointed, connected, 2-truncated type.

Theorem (Milner (following Sính))

A 2-group $(BG, *)$ *can be decomposed as*

1. *A 1-group* G_0
2. *An abelian 1-group* G_1
3. *An action of* G_0 *on* G_1
4. *A pointed section of the family of pointed types* $u : BG \vdash K(G_1(u), 3)$*, the "untruncated" reduced cohomology group* $\tilde{H}^3(G_0, G_1)$ *(the "Sính invariant")*

Synthetic computable 2-groups

Definition

A *computable* 2-group is a pointed, connected, 2-truncated type $(BG, *_{BG})$ such that

1. the set of loops $\|\Omega(BG, *_{BG})\|_0$ is in bijection with a decidable subset of $\mathbb{N}$, and
2. the set of homotopies $\Omega^2(BG, *_{BG})$ is in bijection with a decidable subset of $\mathbb{N}$.

Synthetic computable 2-groups

Definition

A *computable* 2-group is a pointed, connected, 2-truncated type $(BG, *_{BG})$ such that

1. the set of loops $\|\Omega(BG, *_{BG})\|_0$ is in bijection with a decidable subset of $\mathbb{N}$, and
2. the set of homotopies $\Omega^2(BG, *_{BG})$ is in bijection with a decidable subset of $\mathbb{N}$.

By Church's thesis, any algebraic structure we can derive on $\Omega(BG, *_{BG})$ and $\Omega^2(BG, *_{BG})$ is automatically computable.

- G_0 and G_1 are both computable groups.
- The action of G_0 on G_1 is a computable operation.
- From the Sính invariant we can extract a computable "normalised 3-cocycle" operation $G_0^3 \to G_1$.

A 2-group with insoluble word problem

Theorem

There is a finitely generated 2-group G with soluble 1-word problem but insoluble 2-word problem. Hence, the underlying 1-group is computable, but the 2-group itself is not.

A 2-group with insoluble word problem

Theorem

There is a finitely generated 2*-group* G *with soluble* 1*-word problem but insoluble* 2*-word problem. Hence, the underlying* 1*-group is computable, but the* 2*-group itself is not.*

BG is generated by a basepoint $*_{BG}$, two paths $s, t : \Omega(BG, *_{BG})$, and a homotopy $\alpha \in \Omega^2(BG, *_{BG})$. We add a relation $(s^{-e} \cdot t \cdot s^{e}) \bullet \alpha = \alpha$ whenever $\varphi_e(e)\downarrow$.

A 2-group with insoluble word problem

Theorem

There is a finitely generated 2*-group* G *with soluble* 1*-word problem but insoluble* 2*-word problem. Hence, the underlying* 1*-group is computable, but the* 2*-group itself is not.*

BG is generated by a basepoint $*_{BG}$, two paths $s, t : \Omega(BG, *_{BG})$, and a homotopy $\alpha \in \Omega^2(BG, *_{BG})$. We add a relation $(s^{-e} \cdot t \cdot s^e) \bullet \alpha = \alpha$ whenever $\varphi_e(e)\downarrow$.

Note that to 1-truncate we can just erase the generating homotopy, leaving BF_2, which has decidable word problem.

A 2-group with insoluble word problem

Theorem

There is a finitely generated 2*-group* G *with soluble* 1*-word problem but insoluble* 2*-word problem. Hence, the underlying* 1*-group is computable, but the* 2*-group itself is not.*

BG is generated by a basepoint $*_{BG}$, two paths $s, t : \Omega(BG, *_{BG})$, and a homotopy $\alpha \in \Omega^2(BG, *_{BG})$. We add a relation $(s^{-e} \cdot t \cdot s^e) \bullet \alpha = \alpha$ whenever $\varphi_e(e)\downarrow$.

Note that to 1-truncate we can just erase the generating homotopy, leaving BF_2, which has decidable word problem.

To verify that the 2-word problem is insoluble, we need to check that if $\varphi_e(e)$ does not halt, then $(s^{-e} \cdot t \cdot s^e) \bullet \alpha \neq \alpha$.

To do this, we define a non trivial 2-action of G on a groupoid. In HoTT this amounts to constructing a map from BG to the type of groupoids. Since BG is a HIT we just need to define what happens on each constructor.

To do this, we define a non trivial 2-action of G on a groupoid. In HoTT this amounts to constructing a map from BG to the type of groupoids. Since BG is a HIT we just need to define what happens on each constructor.

- We map $*_{BG}$ to $\mathbb{N} \times \nabla\mathbb{S}^1$.

To do this, we define a non trivial 2-action of G on a groupoid. In HoTT this amounts to constructing a map from BG to the type of groupoids. Since BG is a HIT we just need to define what happens on each constructor.

- We map $*_{BG}$ to $\mathbb{N} \times \nabla\mathbb{S}^1$.
- We map s and t to elements of the loop space of $\mathbb{N} \times \nabla\mathbb{S}^1$. In both cases, we obtain these by applying univalence to appropriate permutations of $\mathbb{N}$.

To do this, we define a non trivial 2-action of G on a groupoid. In HoTT this amounts to constructing a map from BG to the type of groupoids. Since BG is a HIT we just need to define what happens on each constructor.

- We map $*_{BG}$ to $\mathbb{N} \times \nabla\mathbb{S}^1$.
- We map s and t to elements of the loop space of $\mathbb{N} \times \nabla\mathbb{S}^1$. In both cases, we obtain these by applying univalence to appropriate permutations of $\mathbb{N}$.
- We need to map α to a homotopy from the reflexivity path on $\mathbb{N} \times \nabla\mathbb{S}^1$ to itself.

To do this, we define a non trivial 2-action of G on a groupoid. In HoTT this amounts to constructing a map from BG to the type of groupoids. Since BG is a HIT we just need to define what happens on each constructor.

- We map $*_{BG}$ to $\mathbb{N} \times \nabla\mathbb{S}^1$.
- We map s and t to elements of the loop space of $\mathbb{N} \times \nabla\mathbb{S}^1$. In both cases, we obtain these by applying univalence to appropriate permutations of $\mathbb{N}$.
- We need to map α to a homotopy from the reflexivity path on $\mathbb{N} \times \nabla\mathbb{S}^1$ to itself. By univalence, this means constructing a proof that the identity equivalence is equal to itself, which amounts to a choice of loop in $\nabla\mathbb{S}^1$ for each $n : \mathbb{N}$.

To do this, we define a non trivial 2-action of G on a groupoid. In HoTT this amounts to constructing a map from BG to the type of groupoids. Since BG is a HIT we just need to define what happens on each constructor.

- We map $*_{BG}$ to $\mathbb{N} \times \nabla\mathbb{S}^1$.
- We map s and t to elements of the loop space of $\mathbb{N} \times \nabla\mathbb{S}^1$. In both cases, we obtain these by applying univalence to appropriate permutations of $\mathbb{N}$.
- We need to map α to a homotopy from the reflexivity path on $\mathbb{N} \times \nabla\mathbb{S}^1$ to itself. By univalence, this means constructing a proof that the identity equivalence is equal to itself, which amounts to a choice of loop in $\nabla\mathbb{S}^1$ for each $n : \mathbb{N}$. We choose the nth loop to be `loop` if $\varphi_e(e)\downarrow$ and trivial otherwise.

To do this, we define a non trivial 2-action of G on a groupoid. In HoTT this amounts to constructing a map from BG to the type of groupoids. Since BG is a HIT we just need to define what happens on each constructor.

- We map $*_{BG}$ to $\mathbb{N} \times \nabla\mathbb{S}^1$.
- We map s and t to elements of the loop space of $\mathbb{N} \times \nabla\mathbb{S}^1$. In both cases, we obtain these by applying univalence to appropriate permutations of $\mathbb{N}$.
- We need to map α to a homotopy from the reflexivity path on $\mathbb{N} \times \nabla\mathbb{S}^1$ to itself. By univalence, this means constructing a proof that the identity equivalence is equal to itself, which amounts to a choice of loop in $\nabla\mathbb{S}^1$ for each $n : \mathbb{N}$. We choose the nth loop to be `loop` if $\varphi_e(e)\downarrow$ and trivial otherwise.

By choosing the actions of s and t appropriately we can ensure the action respects the relations, and that $(s^{-e} \cdot t \cdot s^e) \bullet \alpha$ and α act differently on $\mathbb{N} \times \nabla\mathbb{S}^1$ when $\varphi_e(e)\uparrow$. $\square$

This is work in progress. Still to do:

1. Details of correspondence between computable 2-groups and computable Sính triples
2. A 2-group with trivial action and non computable Sính invariant
3. More results on computable higher structures!

Thanks for you attention!