# Conjugacy in Higman-Thompson groups

David Robertson

Newcastle University, UK

2nd July, 2015

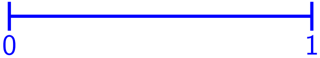
With hyperlinks!

# The plan

- Introduce these groups & their friends
- Higman's solution to the conjugacy problem
- Implementation and future work

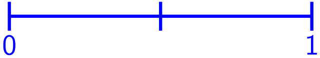
# A recipe for Thompson's group $F$

- Take the interval $[0, 1]$.



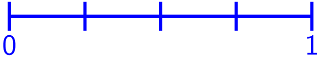
# A recipe for Thompson's group $F$

- Take the interval $[0, 1]$.
- Chop it in half.



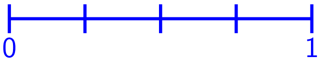
# A recipe for Thompson's group $F$

- Take the interval [0, 1].
- Chop it in half.
- If you like, chop some of the halves in half.



# A recipe for Thompson's group $F$

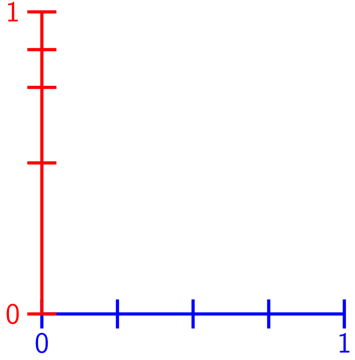
- Take the interval [0, 1].
- Chop it in half.
- If you like, chop some of the halves in half.
- Continue until you get bored.



# A recipe for Thompson's group $F$

- Take the interval [0, 1].
- Chop it in half.
- If you like, chop some of the halves in half.
- Continue until you get bored.

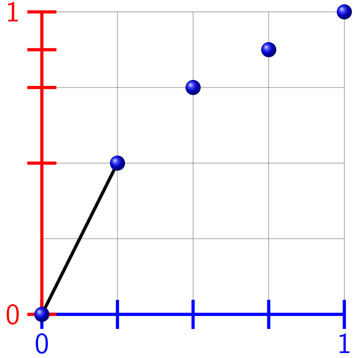
- Do the same to a second copy of $[0, 1]$. Same number of chops!



# A recipe for Thompson's group $F$

- Take the interval [0, 1].
- Chop it in half.
- If you like, chop some of the halves in half.
- Continue until you get bored.

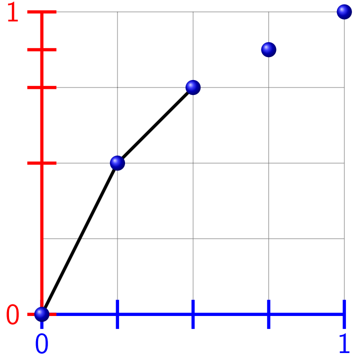
- Do the same to a second copy of $[0, 1]$. Same number of chops!
- Map subintervals to subintervals in order (linear & increasing).



## A recipe for Thompson's group $F$

- Take the interval [0, 1].
- Chop it in half.
- If you like, chop some of the halves in half.
- Continue until you get bored.

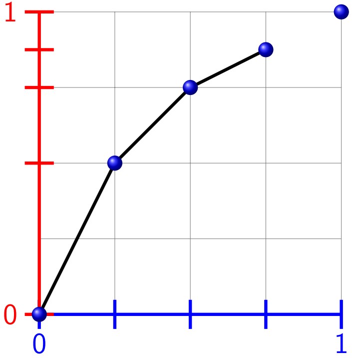
- Do the same to a second copy of $[0, 1]$. Same number of chops!
- Map subintervals to subintervals in order (linear & increasing).



# A recipe for Thompson's group $F$

- Take the interval [0, 1].
- Chop it in half.
- If you like, chop some of the halves in half.
- Continue until you get bored.

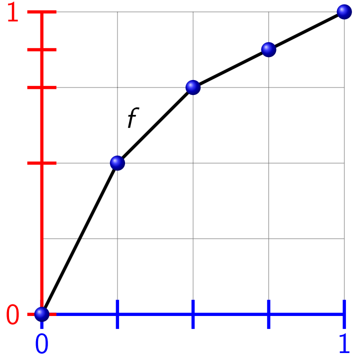
- Do the same to a second copy of $[0, 1]$. Same number of chops!
- Map subintervals to subintervals in order (linear & increasing).



# A recipe for Thompson's group $F$

- Take the interval [0, 1].
- Chop it in half.
- If you like, chop some of the halves in half.
- Continue until you get bored.

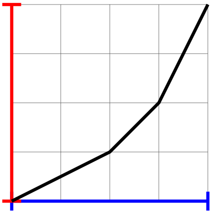
- Do the same to a second copy of $[0, 1]$. Same number of chops!
- Map subintervals to subintervals in order (linear & increasing).



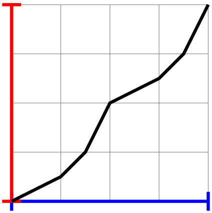

Functions $f$ like this are the elements of Thompson's group $F$.

# Example elements of $F$

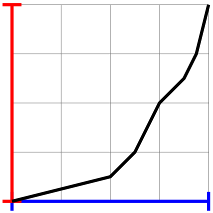
$F$ is the group of all such functions under function composition.
Some more examples:



$f^{-1}$



$g$



$g \circ f^{-1}$

# People *really* like $F$

- Provides interesting examples & counter-examples
- Easy to describe:

$$F = \left\langle A, B \,\middle|\, [AB^{-1}, B^{A}] = [AB^{-1}, B^{A^2}] = 1 \right\rangle$$

yet can be tricky to work with.

# People *really* like $F$

- Provides interesting examples & counter-examples
- Easy to describe:

$$F = \left\langle A, B \;\middle|\; [AB^{-1}, B^{A}] = [AB^{-1}, B^{A^2}] = 1 \right\rangle$$

  yet can be tricky to work with.
- Connections to topology ('homotopy idempotents'), associative laws, logic, ...
- Is $F$ amenable? automatic? autostackable? ...

# People *really* like $F$

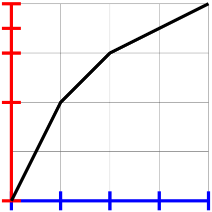
- Provides interesting examples & counter-examples
- Easy to describe:

$$F = \left\langle A, B \,\middle|\, [AB^{-1}, B^{A}] = [AB^{-1}, B^{A^2}] = 1 \right\rangle$$

  yet can be tricky to work with.
- Connections to topology ('homotopy idempotents'), associative laws, logic, ...
- Is $F$ amenable? automatic? autostackable? ...

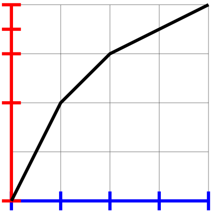
For more details, see Jim Belk's thesis; Cannon, Floyd & Parry; or John Meier's presentation.

# $F$'s friends $F < T < V$

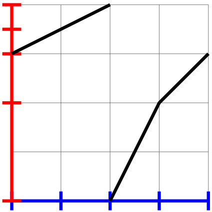


$F$: preserve order of subintervals

# $F$'s friends $F < T < V$

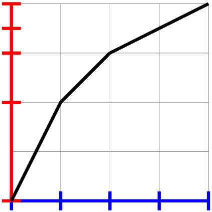


$F$: preserve order of subintervals

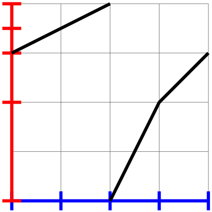


$T$: allow cyclic shifts

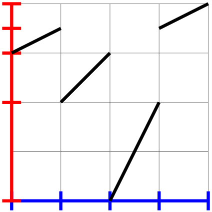
# $F$'s friends $F < T < V$



$F$: preserve order of subintervals

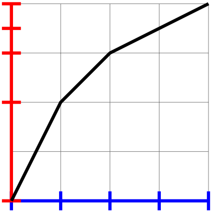


$T$: allow cyclic shifts

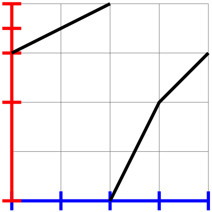


$V$: allow any rearrangement

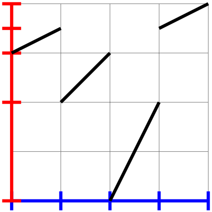
# $F$'s friends $F < T < V$



$F$: preserve order of subintervals



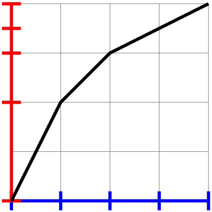
$T$: allow cyclic shifts



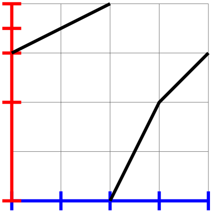
$V$: allow any rearrangement

- New stuff: e.g. torsion.
- $F$ has no torsion (except 1).
- As for the others:

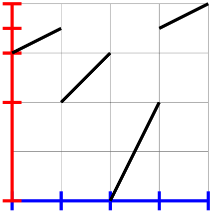
# $F$'s friends $F < T < V$



$F$: preserve order of subintervals

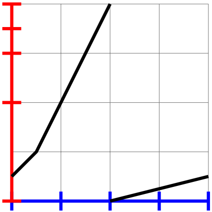


$T$: allow cyclic shifts



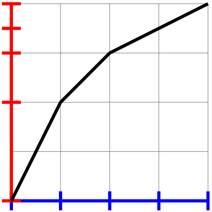
$V$: allow any rearrangement

- New stuff: e.g. torsion.
- $F$ has no torsion (except 1).
- As for the others:

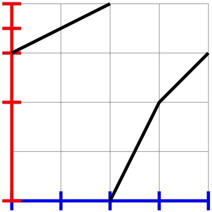


order 4 in $T$

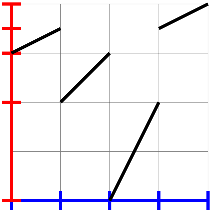
# $F$'s friends $F < T < V$



$F$: preserve order of subintervals

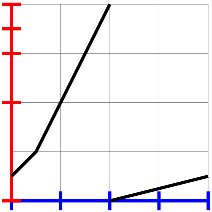


$T$: allow cyclic shifts

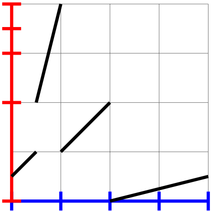


$V$: allow any rearrangement

- New stuff: e.g. torsion.
- $F$ has no torsion (except 1).
- As for the others:



order 4 in $T$



order 3 in $V$

# $F$'s friends are interesting too

$V$ contains a copy of every finite group $G$:

- $G \hookrightarrow \mathcal{S}_n \hookrightarrow \mathcal{S}_{2^m}$
- Chop interval into halves, quarters, eights, ..., $2^m$ths.
- Realise $\sigma \in \mathcal{S}_{2^m}$ as a permutation of the subintervals.

# $F$'s friends are interesting too

$V$ contains a copy of every finite group $G$:

- $G \hookrightarrow \mathcal{S}_n \hookrightarrow \mathcal{S}_{2^m}$
- Chop interval into halves, quarters, eights, ..., $2^m$ths.
- Realise $\sigma \in \mathcal{S}_{2^m}$ as a permutation of the subintervals.

$T$ and $V$ are finitely presented, infinite simple groups.

More exotic friends lead to more f.p. inf. simple groups:

# $F$'s friends are interesting too

$V$ contains a copy of every finite group $G$:

- $G \hookrightarrow \mathcal{S}_n \hookrightarrow \mathcal{S}_{2^m}$
- Chop interval into halves, quarters, eights, ..., $2^m$ths.
- Realise $\sigma \in \mathcal{S}_{2^m}$ as a permutation of the subintervals.

$T$ and $V$ are finitely presented, infinite simple groups.

More exotic friends lead to more f.p. inf. simple groups:

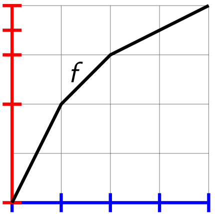
- Brin: $2V, 3V, \ldots, nV, \ldots$
- Repeatedly halve $[0,1]^2$, $[0,1]^3$, etc. and rearrange
- Higman: $F_{n,r}$, $T_{n,r}$, $V_{n,r}$
- Repeatedly chop $[0,1]$ into $n$ pieces and rearrange

# Conjugacy in $V$

- Many authors with many approaches to conjugacy in $V$:
  - 1974 Higman: automorphisms of an algebra
  - 2007 Belk, Matucci: strand diagrams
  - 2010 Salazar-Díaz: revealing tree pairs
  - 2011 Bleak *et al.*: train tracks and flow graphs
- More approaches for $F$ and $T$ & friends

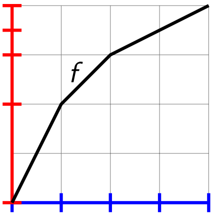
I'll try to explain Higman's approach, and comment on how we got a computer to implement it.

# Tree pair diagrams




How should we describe this function?

# Tree pair diagrams

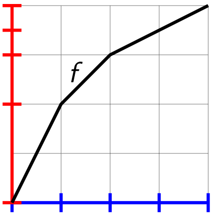



How should we describe this function?

$[0, 1]$

$[0, 1]$

# Tree pair diagrams



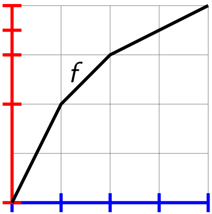

How should we describe this function?

$[0,1]$

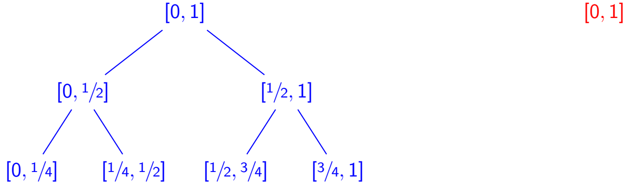
$[0, 1/2]$ $[1/2, 1]$

$[0,1]$

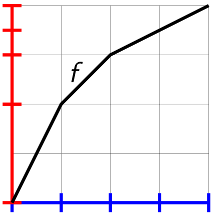
# Tree pair diagrams



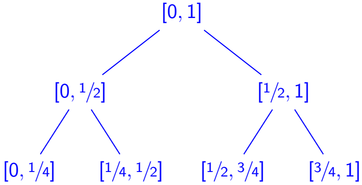

How should we describe this function?

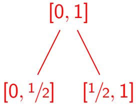


# Tree pair diagrams

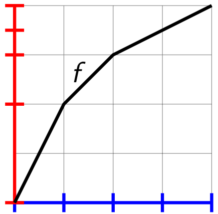



How should we describe this function?

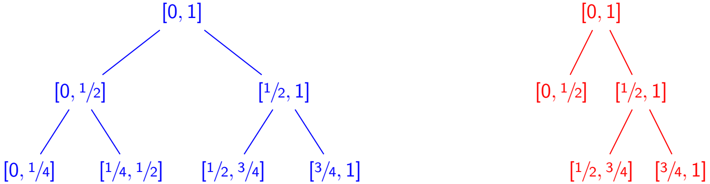


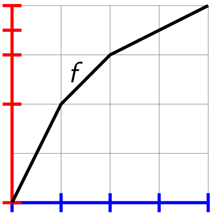
# Tree pair diagrams



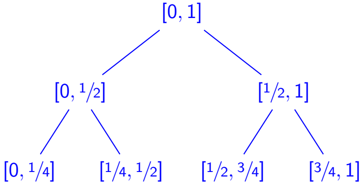

How should we describe this function?

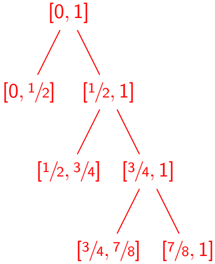


# Tree pair diagrams




How should we describe this function?




In $T$ and $V$, number the leaves to show how intervals get mapped.

# Trees $\to$ paths $\to$ words

Trees aren't a perfect description:

- Where does $f$ send $[{}^{34}/_{128}, {}^{35}/_{128}]$?

# Trees $\to$ paths $\to$ words

Trees aren't a perfect description:

- Where does $f$ send $[{}^{34}/_{128}, {}^{35}/_{128}]$?
- Trees aren't the friendliest data structure to work with
- Can only tell if you're at the root, a leaf or somewhere else

# Trees $\to$ paths $\to$ words

Trees aren't a perfect description:

- Where does $f$ send $[{}^{34}/_{128}, {}^{35}/_{128}]$?
- Trees aren't the friendliest data structure to work with
- Can only tell if you're at the root, a leaf or somewhere else
- Recursively delegate to children
- Quickly becomes confusing! for me, at least...

# Trees $\to$ paths $\to$ words

Trees aren't a perfect description:

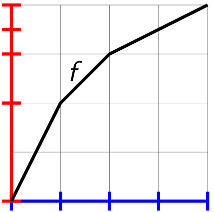
- Where does $f$ send $[^{34}/_{128}, ^{35}/_{128}]$?
- Trees aren't the friendliest data structure to work with
- Can only tell if you're at the root, a leaf or somewhere else
- Recursively delegate to children
- Quickly becomes confusing! for me, at least...

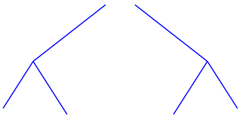
Higman described paths in the tree using an algebra. To write them down as words, he introduced some labels:

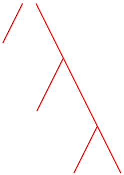
Root $\mapsto x$ left $\mapsto \alpha_1$ right $\mapsto \alpha_2$

# Trees $\to$ paths $\to$ words

Root $\mapsto x$ left $\mapsto \alpha_1$ right $\mapsto \alpha_2$

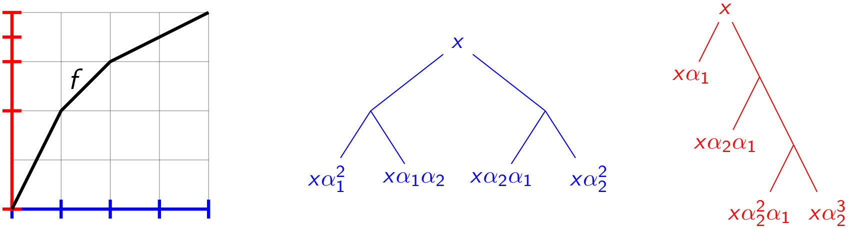




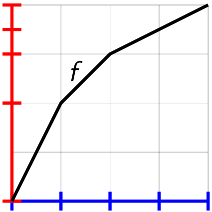
# Trees $\to$ paths $\to$ words

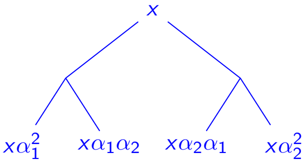
Root $\mapsto x$ left $\mapsto \alpha_1$ right $\mapsto \alpha_2$

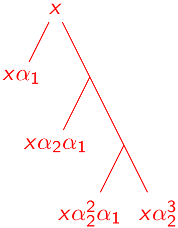


# Trees $\to$ paths $\to$ words

Root $\mapsto x$ left $\mapsto \alpha_1$ right $\mapsto \alpha_2$



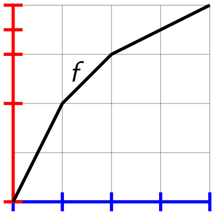




The function $f$ is completely specified by a mapping from domain to range words.

$$
\begin{aligned}
x\alpha_1^2 &\mapsto x\alpha_1 & \qquad x\alpha_2\alpha_1 &\mapsto x\alpha_2^2\alpha_1 \\
x\alpha_1\alpha_2 &\mapsto x\alpha_2\alpha_1 & \qquad x\alpha_2^2 &\mapsto x\alpha_2^3
\end{aligned}
$$

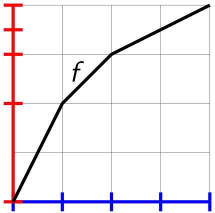
# Trees → paths → words




Can track repeated application of $f$, e.g. using $x\alpha_2^2 \mapsto x\alpha_2^3$.

$$\begin{array}{ccccccc} x\alpha_2^2 & \mapsto & x\alpha_2^3 & \mapsto & x\alpha_2^4 & \mapsto & \dots \\ [3/4, 1] & \mapsto & [7/8, 1] & \mapsto & [15/16, 1] & \mapsto & \dots \end{array}$$

# Trees $\to$ paths $\to$ words

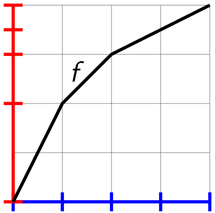



Can track repeated application of $f$, e.g. using $x\alpha_2^2 \mapsto x\alpha_2^3$.

$$\begin{array}{ccccccc} x\alpha_2^2 & \mapsto & x\alpha_2^3 & \mapsto & x\alpha_2^4 & \mapsto & \dots \\ [3/4, 1] & \mapsto & [7/8, 1] & \mapsto & [15/16, 1] & \mapsto & \dots \end{array}$$

- Where does $x\alpha_2^2 \approx [3/4, 1]$ come from?

# Trees $\to$ paths $\to$ words

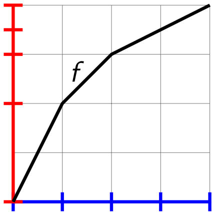



Can track repeated application of $f$, e.g. using $x\alpha_2^2 \mapsto x\alpha_2^3$.

$$\begin{array}{ccccccc} x\alpha_2^2 & \mapsto & x\alpha_2^3 & \mapsto & x\alpha_2^4 & \mapsto & \dots \\ [3/4, 1] & \mapsto & [7/8, 1] & \mapsto & [15/16, 1] & \mapsto & \dots \end{array}$$

- Where does $x\alpha_2^2 \approx [3/4, 1]$ come from?
- Answer: $x\alpha_2 \approx [1/2, 1]$. Where does this come from?

# Trees → paths → words

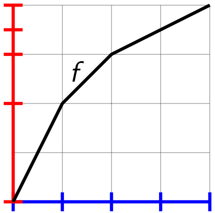



Can track repeated application of $f$, e.g. using $x\alpha_2^2 \mapsto x\alpha_2^3$.

$$\begin{array}{ccccccc} x\alpha_2^2 & \mapsto & x\alpha_2^3 & \mapsto & x\alpha_2^4 & \mapsto & \dots \\ [3/4, 1] & \mapsto & [7/8, 1] & \mapsto & [15/16, 1] & \mapsto & \dots \end{array}$$

- Where does $x\alpha_2^2 \approx [3/4, 1]$ come from?
- Answer: $x\alpha_2 \approx [1/2, 1]$. Where does this come from?
- Answer: [¼, 1]. Can't get that by repeatedly halving the interval!
- Only a 'partial action'. Workaround?

# Trees → paths → words




Can track repeated application of $f$, e.g. using $x\alpha_2^2 \mapsto x\alpha_2^3$.

$$\begin{array}{ccccccc} x\alpha_2^2 & \mapsto & x\alpha_2^3 & \mapsto & x\alpha_2^4 & \mapsto & \dots \\ [^3\!/\!_4, 1] & \mapsto & [^7\!/\!_8, 1] & \mapsto & [^{15}\!/\!_{16}, 1] & \mapsto & \dots \end{array}$$

- Where does $x\alpha_2^2 \approx [^3\!/\!_4, 1]$ come from?
- Answer: $x\alpha_2 \approx [^1\!/\!_2, 1]$. Where does this come from?
- Answer: $[^1\!/\!_4, 1]$. Can't get that by repeatedly halving the interval!
- Only a 'partial action'. Workaround?

$$\begin{aligned} [^1\!/\!_4, 1] &= [^1\!/\!_4, ^1\!/\!_2] \cup [^1\!/\!_2, 1] \\ &\approx x\alpha_1\alpha_2 \cup x\alpha_2 \\ &\approx (x\alpha_1\alpha_2)(x\alpha_2)\lambda \end{aligned}$$

# Components ($\approx$ orbits)

- Pick your favourite element/interval e.g. $w = x\alpha_1^2 \approx [0, 1/4]$.
- Compute (pre)images of $w$ until you can't any more:

$$\dots,\ f^{-2}(w),\ f^{-1}(w),\ w,\ f(w),\ f^2(w),\ \dots$$

# Components ($\approx$ orbits)

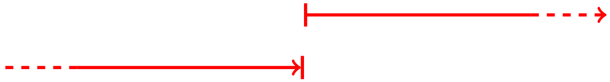
- Pick your favourite element/interval e.g. $w = x\alpha_1^2 \approx [0, 1/4]$.
- Compute (pre)images of $w$ until you can't any more:

$$\ldots,\ f^{-2}(w),\ f^{-1}(w),\ w,\ f(w),\ f^2(w),\ \ldots$$

# Components ($\approx$ orbits)

- Pick your favourite element/interval e.g. $w = x\alpha_1^2 \approx [0, 1/4]$.
- Compute (pre)images of $w$ until you can't any more:

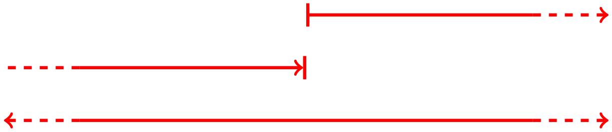
$$\ldots,\ f^{-2}(w),\ f^{-1}(w),\ w,\ f(w),\ f^2(w),\ \ldots$$



# Components ($\approx$ orbits)

- Pick your favourite element/interval e.g. $w = x\alpha_1^2 \approx [0, 1/4]$.
- Compute (pre)images of $w$ until you can't any more:

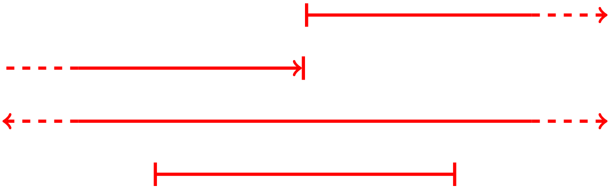
$$\ldots,\ f^{-2}(w),\ f^{-1}(w),\ w,\ f(w),\ f^2(w),\ \ldots$$



# Components ($\approx$ orbits)

- Pick your favourite element/interval e.g. $w = x\alpha_1^2 \approx [0, 1/4]$.
- Compute (pre)images of $w$ until you can't any more:

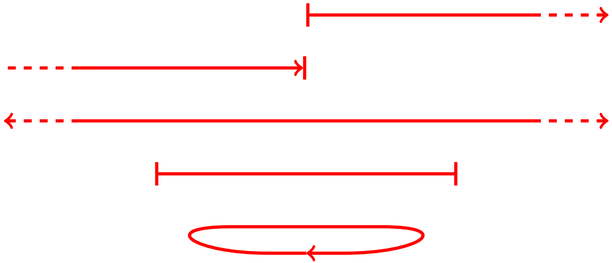
$$\ldots,\ f^{-2}(w),\ f^{-1}(w),\ w,\ f(w),\ f^2(w),\ \ldots$$



# Components ($\approx$ orbits)

- Pick your favourite element/interval e.g. $w = x\alpha_1^2 \approx [0, 1/4]$.
- Compute (pre)images of $w$ until you can't any more:

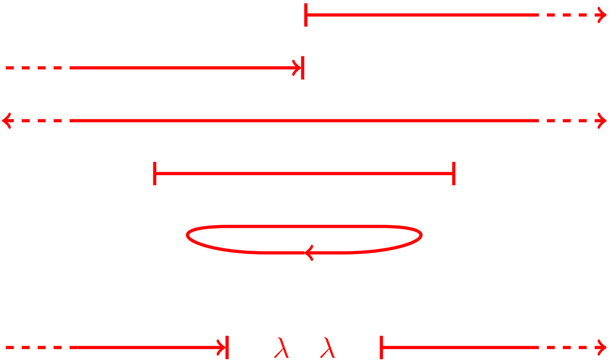
$$\dots,\ f^{-2}(w),\ f^{-1}(w),\ w,\ f(w),\ f^2(w),\ \dots$$



# Components ($\approx$ orbits)

- Pick your favourite element/interval e.g. $w = x\alpha_1^2 \approx [0, 1/4]$.
- Compute (pre)images of $w$ until you can't any more:

$$\ldots,\ f^{-2}(w),\ f^{-1}(w),\ w,\ f(w),\ f^2(w),\ \ldots$$



# Components and Conjugacy

Higman uses these components to find constraints on conjugators. Suppose $hfh^{-1} = g$.

- Can describe $f$ using finitely many components (same for $g$).

# Components and Conjugacy

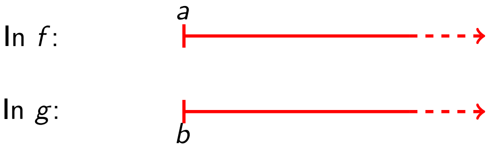
Higman uses these components to find constraints on conjugators. Suppose $hfh^{-1} = g$.

- Can describe $f$ using finitely many components (same for $g$).
- $h$ must map $f$-components to $g$-components of the same type.
  e.g. periodic $\mapsto$ periodic, with the same period

# Components and Conjugacy

Higman uses these components to find constraints on conjugators.
Suppose $hfh^{-1} = g$.

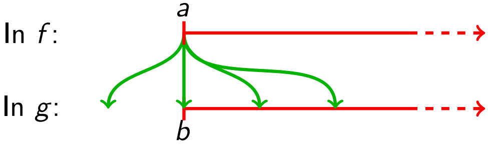
- Can describe $f$ using finitely many components (same for $g$).
- $h$ must map $f$-components to $g$-components of the same type.
  e.g. periodic $\mapsto$ periodic, with the same period
- Can impose some restrictions on $h$



# Components and Conjugacy

Higman uses these components to find constraints on conjugators. Suppose $hfh^{-1} = g$.

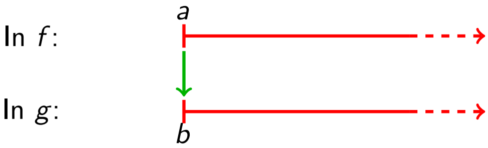
- Can describe $f$ using finitely many components (same for $g$).
- $h$ must map $f$-components to $g$-components of the same type.
  e.g. periodic $\mapsto$ periodic, with the same period
- Can impose some restrictions on $h$



## Components and Conjugacy

Higman uses these components to find constraints on conjugators. Suppose $hfh^{-1} = g$.

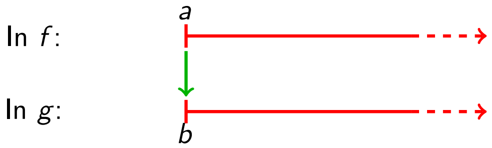
- Can describe $f$ using finitely many components (same for $g$).
- $h$ must map $f$-components to $g$-components of the same type.
  e.g. periodic $\mapsto$ periodic, with the same period
- Can impose some restrictions on $h$
  e.g. $h(\text{endpoint}) = \text{endpoint}$



# Components and Conjugacy

Higman uses these components to find constraints on conjugators. Suppose $hfh^{-1} = g$.

- Can describe $f$ using finitely many components (same for $g$).
- $h$ must map $f$-components to $g$-components of the same type.
  e.g. periodic $\mapsto$ periodic, with the same period
- Can impose some restrictions on $h$
  e.g. $h(\text{endpoint}) = \text{endpoint}$




- End up with a finite number of potential conjugators—test them all!

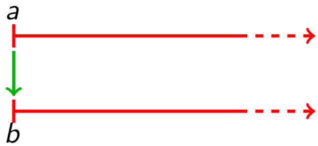
# Components and Conjugacy

Higman uses these components to find constraints on conjugators. Suppose $hfh^{-1} = g$.

- Can describe $f$ using finitely many components (same for $g$).
- $h$ must map $f$-components to $g$-components of the same type.
  e.g. periodic $\mapsto$ periodic, with the same period
- Can impose some restrictions on $h$
  e.g. $h(\text{endpoint}) = \text{endpoint}$

In $f$:

In $g$:




- End up with a finite number of potential conjugators—test them all!
- Some shortcuts e.g. conjugator has to preserve relations between components

# Implementation

- My job: get a computer to do this.
- Like programming a fancy calculator!
- Written in Python: something I knew and quick to work with.

# Implementation

- My job: get a computer to do this.
- Like programming a fancy calculator!
- Written in Python: something I knew and quick to work with.
- Summer project that was more like a semester project. . .

# Implementation

- My job: get a computer to do this.
- Like programming a fancy calculator!
- Written in Python: something I knew and quick to work with.
- Summer project that was more like a semester project...
- Other tools exist to do calculations in $V$, but not to solve the conjugacy problem.

Code is on GitHub and documentation online. Details written on the arXiv [with Nathan Barker & Andrew Duncan].
Time for a quick demo?

# Future Work

- Complexity?
- Simultaneous conjugacy?
  Given $x_1, \ldots, x_n; y_1, \ldots, y_n$ find a single conjugator $z$

$$z^{-1} x_i z = y_i, \quad \forall i$$

- Can we solve different kinds of equations in $V$ (and $V_{n,r}$)?
- Can we apply this to other Thompson-like groups?