Group Recognition

Max Neunhöffer

University of St Andrews

GAC 2010, Allahabad

Constructive recognition

Problem

Let $\mathbb{G}$ be some ambient group and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$ and
- an algorithm that, given $M \in \mathbb{G}$,
  - decides, whether or not $M \in G$, and,
  - if so, expresses M as an SLP in the M_i.

Constructive recognition

Problem

Let $\mathbb{G}$ be some ambient group and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$ and
- an algorithm that, given $M \in \mathbb{G}$,
  - decides, whether or not $M \in G$, and,
  - if so, expresses M as an SLP in the M_i.
- The runtime should be bounded from above by a polynomial in the input size.

Constructive recognition

Problem

Let $\mathbb{G}$ be some ambient group and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$ and
- an algorithm that, given $M \in \mathbb{G}$,
  - decides, whether or not $M \in G$, and,
  - if so, expresses M as an SLP in the M_i.
- The runtime should be bounded from above by a polynomial in the input size.
- A Monte Carlo Algorithmus is enough.

Constructive recognition

Problem

Let $\mathbb{G}$ be some ambient group and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$ and
- an algorithm that, given $M \in \mathbb{G}$,
  - decides, whether or not $M \in G$, and,
  - if so, expresses M as an SLP in the M_i.
- The runtime should be bounded from above by a polynomial in the input size.
- A Monte Carlo Algorithmus is enough. (Verification!)

Constructive recognition

Problem

Let $\mathbb{G}$ be some ambient group and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$ and
- an algorithm that, given $M \in \mathbb{G}$,
  - decides, whether or not $M \in G$, and,
  - if so, expresses M as an SLP in the M_i.
- The runtime should be bounded from above by a polynomial in the input size.
- A Monte Carlo Algorithmus is enough. (Verification!)

If this problem is solved, we call

$\langle M_1, \ldots, M_k \rangle$ recognised constructively.

GAP examples

see other window

What is a reduction?

Let $G := \langle M_1, \ldots, M_k \rangle \le \mathbb{G}$.

What is a reduction?

Let $G := \langle M_1, \ldots, M_k \rangle \leq \mathbb{G}$.

A reduction is a group homomorphism

$$\begin{array}{cccc} \varphi : & G & \to & H \\ & M_i & \mapsto & P_i \end{array} \qquad \text{for all } i$$

with the following properties:

What is a reduction?

Let $G := \langle M_1, \ldots, M_k \rangle \leq \mathbb{G}$.

A reduction is a group homomorphism

$$\begin{array}{cccccl} \varphi & : & G & \to & H & \\ & & M_i & \mapsto & P_i & \quad \text{for all } i \end{array}$$

with the following properties:

- $\varphi(M)$ is explicitly computable for all $M \in G$

What is a reduction?

Let $G := \langle M_1, \ldots, M_k \rangle \leq \mathbb{G}$.

A reduction is a group homomorphism

$$\begin{array}{cccccl} \varphi & : & G & \to & H & \\ & & M_i & \mapsto & P_i & \qquad \text{for all } i \end{array}$$

with the following properties:

- $\varphi(M)$ is explicitly computable for all $M \in G$
- φ is surjective: $H = \langle P_1, \ldots, P_k \rangle$

What is a reduction?

Let $G := \langle M_1, \ldots, M_k \rangle \leq \mathbb{G}$.

A reduction is a group homomorphism

$$\begin{array}{cccccc} \varphi & : & G & \to & H & \\ & & M_i & \mapsto & P_i & \quad \text{for all } i \end{array}$$

with the following properties:

- $\varphi(M)$ is explicitly computable for all $M \in G$
- φ is surjective: $H = \langle P_1, \ldots, P_k \rangle$
- H is in some sense "smaller"
- or at least "easier to recognise constructively"

What is a reduction?

Let $G := \langle M_1, \ldots, M_k \rangle \leq \mathbb{G}$.

A reduction is a group homomorphism

$$\begin{array}{cccc} \varphi : & G & \to & H \\ & M_i & \mapsto & P_i \end{array} \qquad \text{for all } i$$

with the following properties:

- $\varphi(M)$ is explicitly computable for all $M \in G$
- φ is surjective: $H = \langle P_1, \ldots, P_k \rangle$
- H is in some sense "smaller"
- or at least "easier to recognise constructively"
- e.g. $H \leq \Sigma_m$ or $H \leq \mathrm{GL}_n(\mathbb{F}_q)$ with smaller m or n, q respectively.

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

1. Generate a (pseudo-) random element $M \in G$,

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

1. Generate a (pseudo-) random element $M \in G$,
2. map it with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

1. Generate a (pseudo-) random element $M \in G$,
2. map it with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
3. express $\varphi(M)$ as SLP in the P_i,

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

1. Generate a (pseudo-) random element $M \in G$,
2. map it with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
3. express $\varphi(M)$ as SLP in the the P_i,
4. evaluate the same SLP in the M_i,

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

1. Generate a (pseudo-) random element $M \in G$,
2. map it with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
3. express $\varphi(M)$ as SLP in the P_i,
4. evaluate the same SLP in the M_i,
5. get an element $M' \in G$ with $M \cdot M'^{-1} \in N$.

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

1. Generate a (pseudo-) random element $M \in G$,
2. map it with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
3. express $\varphi(M)$ as SLP in the P_i,
4. evaluate the same SLP in the M_i,
5. get an element $M' \in G$ with $M \cdot M'^{-1} \in N$.
6. If M is uniformly distributed in G then $M \cdot M'^{-1}$ is uniformly distributed in N

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

1. Generate a (pseudo-) random element $M \in G$,
2. map it with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
3. express $\varphi(M)$ as SLP in the P_i,
4. evaluate the same SLP in the M_i,
5. get an element $M' \in G$ with $M \cdot M'^{-1} \in N$.
6. If M is uniformly distributed in G then $M \cdot M'^{-1}$ is uniformly distributed in N
7. Repeat.

Computing the kernel

Let $\varphi : G \to H$ be a reduction and assume that H is already recognised constructively.

Then we can compute the kernel N of φ:

1. Generate a (pseudo-) random element $M \in G$,
2. map it with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
3. express $\varphi(M)$ as SLP in the P_i,
4. evaluate the same SLP in the M_i,
5. get an element $M' \in G$ with $M \cdot M'^{-1} \in N$.
6. If M is uniformly distributed in G
   then $M \cdot M'^{-1}$ is uniformly distributed in N
7. Repeat.

$\to$ Monte Carlo algorithm to compute N

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Then we have recognised G constructively:

$$|G| = |H| \cdot |N|.$$

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Then we have recognised G constructively:

$|G| = |H| \cdot |N|$. And for $M \in \mathbb{G}$:

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Then we have recognised G constructively:

$|G| = |H| \cdot |N|$. And for $M \in \mathbb{G}$:

1. map M with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Then we have recognised G constructively:

$|G| = |H| \cdot |N|$. And for $M \in \mathbb{G}$:

1. map M with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
2. express $\varphi(M)$ as SLP in the P_i,

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Then we have recognised G constructively:

$|G| = |H| \cdot |N|$. And for $M \in \mathbb{G}$:

1. map M with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
2. express $\varphi(M)$ as SLP in the P_i,
3. evaluate the same SLP in the M_i,

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Then we have recognised G constructively:

$|G| = |H| \cdot |N|$. And for $M \in \mathbb{G}$:

1. map M with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
2. express $\varphi(M)$ as SLP in the P_i,
3. evaluate the same SLP in the M_i,
4. get an element $M' \in G$ such that $M \cdot M'^{-1} \in N$,

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Then we have recognised G constructively:

$$|G| = |H| \cdot |N|. \text{ And for } M \in \mathbb{G}\text{:}$$

1. map M with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
2. express $\varphi(M)$ as SLP in the P_i,
3. evaluate the same SLP in the M_i,
4. get an element $M' \in G$ such that $M \cdot M'^{-1} \in N$,
5. express $M \cdot M'^{-1}$ as SLP in the N_j,

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

Then we have recognised G constructively:

$$|G| = |H| \cdot |N|. \text{ And for } M \in \mathbb{G}:$$

1. map M with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
2. express $\varphi(M)$ as SLP in the P_i,
3. evaluate the same SLP in the M_i,
4. get an element $M' \in G$ such that $M \cdot M'^{-1} \in N$,
5. express $M \cdot M'^{-1}$ as SLP in the N_j,
6. get M as SLP in the M_i and N_j:
   $M' = \prod$ in the M_i, $\quad M \cdot M'^{-1} = \prod$ in the N_j
   $$\Rightarrow M = \left(\mathrm{SLP}(\{N_j\})\right) \cdot \left(\mathrm{SLP}(\{M_i\})\right).$$

Recognising image and kernel suffices

Let $\varphi : G \to H$ be a reduction and assume that both H and the kernel $N = \langle N_1, \ldots, N_m \rangle$ of φ are already recognised constructively.

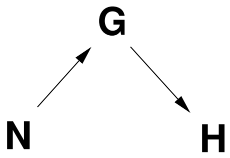
Then we have recognised G constructively:

$$|G| = |H| \cdot |N|. \text{ And for } M \in \mathbb{G}:$$

1. map M with φ onto $\varphi(M) \in H = \langle P_1, \ldots, P_k \rangle$,
2. express $\varphi(M)$ as SLP in the P_i,
3. evaluate the same SLP in the M_i,
4. get an element $M' \in G$ such that $M \cdot M'^{-1} \in N$,
5. express $M \cdot M'^{-1}$ as SLP in the N_j,
6. get M as SLP in the M_i and N_j:
   $M' = \prod$ in the M_i, $\quad M \cdot M'^{-1} = \prod$ in the N_j
   $$\Rightarrow M = (\mathrm{SLP}(\{N_j\})) \cdot (\mathrm{SLP}(\{M_i\})).$$
7. If $M \notin G$, then at least one step does not work.

Recursion: composition trees

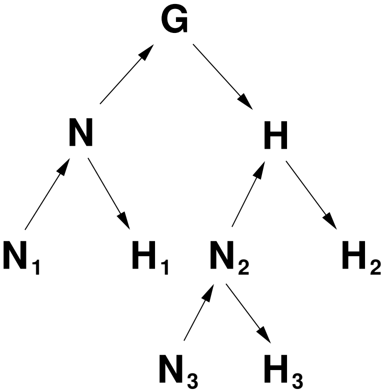
We get a tree:



Up arrows: inclusions
Down arrows: homomorphisms

Recursion: composition trees

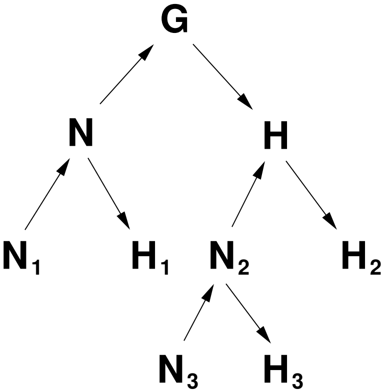
We get a tree:



Up arrows: inclusions
Down arrows: homomorphisms

Recursion: composition trees

We get a tree:



Up arrows: inclusions
Down arrows: homomorphisms

Old idea, improvements are still being made.

Example: invariant subspace

Let $V = \mathbb{F}_q^{1\times d}$ and $G \leq \mathrm{GL}_d(\mathbb{F}_q)$, then G acts on V.
Let $W \leq V$ be an invariant subspace, i.e.:

$$WM = W \quad \text{for all } M \in G$$

Example: invariant subspace

Let $V = \mathbb{F}_q^{1\times d}$ and $G \leq \mathrm{GL}_d(\mathbb{F}_q)$, then G acts on V.
Let $W \leq V$ be an invariant subspace, i.e.:

$$WM = W \quad \text{for all } M \in G$$

Choose basis $(w_1, \ldots, w_e)$ of W and extend to a basis

$$(w_1, \ldots, w_e, w_{e+1}, \ldots, w_d)$$

of V. After a base change the matrices in G look like this:

$$\left[\begin{array}{c|c} A & \mathbf{0} \\ \hline C & D \end{array}\right] \qquad \text{with } A \in \mathbb{F}_q^{e\times e}, C \in \mathbb{F}_q^{(d-e)\times e}, D \in \mathbb{F}_q^{(d-e)\times(d-e)}$$

Example: invariant subspace

Let $V = \mathbb{F}_q^{1\times d}$ and $G \leq \mathrm{GL}_d(\mathbb{F}_q)$, then G acts on V.
Let $W \leq V$ be an invariant subspace, i.e.:

$$WM = W \quad \text{for all } M \in G$$

Choose basis $(w_1, \ldots, w_e)$ of W and extend to a basis

$$(w_1, \ldots, w_e, w_{e+1}, \ldots, w_d)$$

of V. After a base change the matrices in G look like this:

$$\left[\begin{array}{c|c} A & \mathbf{0} \\ \hline C & D \end{array}\right] \quad \text{with } A \in \mathbb{F}_q^{e\times e}, C \in \mathbb{F}_q^{(d-e)\times e}, D \in \mathbb{F}_q^{(d-e)\times(d-e)}$$

and

$$G \to \mathrm{GL}_{d-e}(\mathbb{F}_q), \left[\begin{array}{cc} A & \mathbf{0} \\ C & D \end{array}\right] \mapsto D$$

is a homomorphism of groups.

Example: invariant subspace

$$G \to \mathrm{GL}_{d-e}(\mathbb{F}_q), \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \mapsto D$$

is a homomorphism of groups, its kernel is

$$N := \left\{ \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \in G \mid D = \mathbf{1} \right\}.$$

Example: invariant subspace

$$G \to \mathrm{GL}_{d-e}(\mathbb{F}_q), \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \mapsto D$$

is a homomorphism of groups, its kernel is

$$N := \left\{ \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \in G \mid D = \mathbf{1} \right\}.$$

The mapping

$$N \to \mathrm{GL}_e(\mathbb{F}_q), \begin{bmatrix} A & \mathbf{0} \\ C & \mathbf{1} \end{bmatrix} \mapsto A$$

also is a homomorphism of groups and has kernel

$$N_2 := \left\{ \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \in G \mid A = D = \mathbf{1} \right\}.$$

Example: invariant subspace

$$G \to \mathrm{GL}_{d-e}(\mathbb{F}_q), \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \mapsto D$$

is a homomorphism of groups, its kernel is

$$N := \left\{ \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \in G \mid D = \mathbf{1} \right\}.$$

The mapping

$$N \to \mathrm{GL}_e(\mathbb{F}_q), \begin{bmatrix} A & \mathbf{0} \\ C & \mathbf{1} \end{bmatrix} \mapsto A$$

also is a homomorphism of groups and has kernel

$$N_2 := \left\{ \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \in G \mid A = D = \mathbf{1} \right\}.$$

This group is a p-group for $q = p^f$:

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} \\ C & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ C' & \mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ C + C' & \mathbf{1} \end{bmatrix}.$$

Example: invariant subspace

$$G \to \mathrm{GL}_{d-e}(\mathbb{F}_q), \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \mapsto D$$

is a homomorphism of groups, its kernel is

$$N := \left\{ \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \in G \mid D = \mathbf{1} \right\}.$$

The mapping

$$N \to \mathrm{GL}_e(\mathbb{F}_q), \begin{bmatrix} A & \mathbf{0} \\ C & \mathbf{1} \end{bmatrix} \mapsto A$$

also is a homomorphism of groups and has kernel

$$N_2 := \left\{ \begin{bmatrix} A & \mathbf{0} \\ C & D \end{bmatrix} \in G \mid A = D = \mathbf{1} \right\}.$$

This group is a p-group for $q = p^f$:

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} \\ C & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ C' & \mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ C + C' & \mathbf{1} \end{bmatrix}.$$

Together with a reduction additional information is gained!

Long SLPs

Typical examples:

(1) $G := (2 \times 2^{1+8}) : U_4(2) : 2 < \mathrm{GL}_{78}(2)$

(7th maximal subgroup of the sporadic simple group Fi_{22})

Long SLPs

Typical examples:

(1) $$G := (2 \times 2^{1+8}) : U_4(2) : 2 < \mathrm{GL}_{78}(2)$$

(7th maximal subgroup of the sporadic simple group Fi_{22})

G has 53 084 160 elements, generated by 2 elements.

Long SLPs

Typical examples:

(1) $G := (2 \times 2^{1+8}) : U_4(2) : 2 < \mathrm{GL}_{78}(2)$

(7th maximal subgroup of the sporadic simple group Fi_{22})

G has 53 084 160 elements, generated by 2 elements.

Composition tree of depth 8 with 3 non-trivial leaves.

Long SLPs

Typical examples:

(1) $G := (2 \times 2^{1+8}) : U_4(2) : 2 < \mathrm{GL}_{78}(2)$

(7th maximal subgroup of the sporadic simple group Fi_{22})

G has 53 084 160 elements, generated by 2 elements.

Composition tree of depth 8 with 3 non-trivial leaves.

Typical elements in G give SLPs of length $\approx$ 900.

Long SLPs

Typical examples:

(1) $G := (2 \times 2^{1+8}) : U_4(2) : 2 < \mathrm{GL}_{78}(2)$

(7th maximal subgroup of the sporadic simple group Fi_{22})

G has 53 084 160 elements, generated by 2 elements.

Composition tree of depth 8 with 3 non-trivial leaves.

Typical elements in G give SLPs of length $\approx$ 900.

(2) $W := \Sigma_{12} \wr \Sigma_5 < \Sigma_{60}$

Long SLPs

Typical examples:

(1) $G := (2 \times 2^{1+8}) : U_4(2) : 2 < \mathrm{GL}_{78}(2)$

(7th maximal subgroup of the sporadic simple group Fi_{22})

G has 53 084 160 elements, generated by 2 elements.

Composition tree of depth 8 with 3 non-trivial leaves.

Typical elements in G give SLPs of length $\approx$ 900.

(2) $W := \Sigma_{12} \wr \Sigma_5 < \Sigma_{60}$

W has 3 025 980 091 991 082 565 958 286 705 898 291 200 000 000 000 elements and is generated by 12 elements.

Long SLPs

Typical examples:

(1) $G := (2 \times 2^{1+8}) : U_4(2) : 2 < \mathrm{GL}_{78}(2)$

(7th maximal subgroup of the sporadic simple group Fi_{22})

G has 53 084 160 elements, generated by 2 elements.

Composition tree of depth 8 with 3 non-trivial leaves.

Typical elements in G give SLPs of length $\approx$ 900.

(2) $W := \Sigma_{12} \wr \Sigma_5 < \Sigma_{60}$

W has 3 025 980 091 991 082 565 958 286 705 898 291 200 000 000 000 elements and is generated by 12 elements.

Composition tree of depth 4 with 6 non-trivial leaves.

Long SLPs

Typical examples:

(1) $G := (2 \times 2^{1+8}) : U_4(2) : 2 < \mathrm{GL}_{78}(2)$

(7th maximal subgroup of the sporadic simple group Fi_{22})

G has 53 084 160 elements, generated by 2 elements.

Composition tree of depth 8 with 3 non-trivial leaves.

Typical elements in G give SLPs of length $\approx$ 900.

(2) $W := \Sigma_{12} \wr \Sigma_5 < \Sigma_{60}$

W has 3 025 980 091 991 082 565 958 286 705 898 291 200 000 000 000 elements and is generated by 12 elements.

Composition tree of depth 4 with 6 non-trivial leaves.

Typical elements in W give SLPs of length $\approx$ 10000.

Learning from base and strong generators

The same groups with stabiliser chains:

$$G := (2 \times 2^{1+8}) : U_4(2) : 2 < \Sigma_{3510}$$

(7th maximal subgroup of the sporadic simple group Fi_{22})

Learning from base and strong generators

The same groups with stabiliser chains:

$$G := (2 \times 2^{1+8}) : U_4(2) : 2 < \Sigma_{3510}$$

(7th maximal subgroup of the sporadic simple group Fi_{22})

Stabiliser chain of length 4 with 14 strong generators.

Learning from base and strong generators

The same groups with stabiliser chains:

$$G := (2 \times 2^{1+8}) : U_4(2) : 2 < \Sigma_{3510}$$

(7th maximal subgroup of the sporadic simple group Fi_{22})

Stabiliser chain of length 4 with 14 strong generators.

Typical elements in G give SLPs of length $\approx$ 15.

Learning from base and strong generators

The same groups with stabiliser chains:

$$G := (2 \times 2^{1+8}) : U_4(2) : 2 < \Sigma_{3510}$$

(7th maximal subgroup of the sporadic simple group Fi_{22})

Stabiliser chain of length 4 with 14 strong generators.

Typical elements in G give SLPs of length $\approx$ 15.

$$W := \Sigma_{12} \wr \Sigma_5 < \Sigma_{60}$$

Learning from base and strong generators

The same groups with stabiliser chains:

$$G := (2 \times 2^{1+8}) : U_4(2) : 2 < \Sigma_{3510}$$

(7th maximal subgroup of the sporadic simple group Fi_{22})

Stabiliser chain of length 4 with 14 strong generators.

Typical elements in G give SLPs of length $\approx$ 15.

$$W := \Sigma_{12} \wr \Sigma_5 < \Sigma_{60}$$

Stabiliser chain of length 55 with 434 strong generators.

Learning from base and strong generators

The same groups with stabiliser chains:

$$G := (2 \times 2^{1+8}) : U_4(2) : 2 < \Sigma_{3510}$$

(7th maximal subgroup of the sporadic simple group Fi_{22})

Stabiliser chain of length 4 with 14 strong generators.

Typical elements in G give SLPs of length $\approx$ 15.

$$W := \Sigma_{12} \wr \Sigma_5 < \Sigma_{60}$$

Stabiliser chain of length 55 with 434 strong generators.

Typical elements in W give SLPs of length $\approx$ 500.

Comparison

We compare lengths of SLPs:

	Stabiliser chain		Composition tree	
	in strong			in gens
G	15			900
$\Sigma_{12} \wr \Sigma_5$	500			10000

Comparison

We compare lengths of SLPs:

	Stabiliser chain		Composition tree	
	in strong	in gens		in gens
G	15	290		900
$\Sigma_{12} \wr \Sigma_5$	500	4300		10000

Comparison

We compare lengths of SLPs:

	Stabiliser chain		Composition tree	
	in strong	in gens		in gens
G	15	290		900
$\Sigma_{12} \wr \Sigma_5$	500	4300		10000

We want to change the generating system!

$\Longrightarrow$ "nice generators"

Comparison

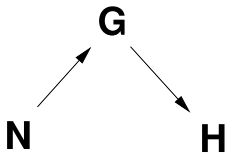
We compare lengths of SLPs:

	Stabiliser chain		Composition tree	
	in strong	in gens	in nice	in gens
G	15	290	15	900
$\Sigma_{12} \wr \Sigma_5$	500	4300	300	10000

We want to change the generating system!

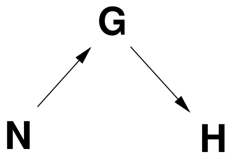
$\Longrightarrow$ “nice generators”

Problems with recursion



Recall: Generators of H were images of those of G.

Problems with recursion



Recall: Generators of H were images of those of G.

Having changed the generators in H,

we can no longer find preimages!

Problems with recursion

$$\begin{tikzcd} & \mathbf{G} \arrow[dr] & \\ \mathbf{N} \arrow[ur] & & \mathbf{H} \end{tikzcd}$$

Recall: Generators of H were images of those of G.

Having changed the generators in H,

we can no longer find preimages!

Solution: Nice generators of G are

- preimages of the nice generators of H together with
- nice generators of N.

Problems with recursion

$$\begin{tikzcd} & \mathbf{G} \arrow[dr] & \\ \mathbf{N} \arrow[ur] & & \mathbf{H} \end{tikzcd}$$

Recall: Generators of H were images of those of G.

Having changed the generators in H,

we can no longer find preimages!

Solution: Nice generators of G are

- preimages of the nice generators of H together with
- nice generators of N.

Note: The first allows to compute N once H is recognised!

Constructive recognition revisited

Problem — new formulation

Let $\mathbb{G}$ be Σ_n or $\mathrm{GL}_n(\mathbb{F}_q)$ or $\mathrm{PGL}_n(\mathbb{F}_q)$ and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$,

Constructive recognition revisited

Problem — new formulation

Let $\mathbb{G}$ be Σ_n or $\mathrm{GL}_n(\mathbb{F}_q)$ or $\mathrm{PGL}_n(\mathbb{F}_q)$ and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$,
- new nice generators $G = \langle N_1, \ldots, N_m \rangle$ and

Constructive recognition revisited

Problem — new formulation

Let $\mathbb{G}$ be Σ_n or $\mathrm{GL}_n(\mathbb{F}_q)$ or $\mathrm{PGL}_n(\mathbb{F}_q)$ and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$,
- new nice generators $G = \langle N_1, \ldots, N_m \rangle$ and
- a procedure that, given $M \in \mathbb{G}$,
  - decides, whether or not $M \in G$ and
  - if so, expresses M as an SLP in the N_j and

Constructive recognition revisited

Problem — new formulation

Let $\mathbb{G}$ be Σ_n or $\mathrm{GL}_n(\mathbb{F}_q)$ or $\mathrm{PGL}_n(\mathbb{F}_q)$ and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$,
- new nice generators $G = \langle N_1, \ldots, N_m \rangle$ and
- a procedure that, given $M \in \mathbb{G}$,
  - decides, whether or not $M \in G$ and
  - if so, expresses M as an SLP in the N_j and
- another procedure that, given preimages $\hat{M}_1, \ldots, \hat{M}_k$ of the M_i under some homomorphism onto G, produces preimages of the nice generators.

Constructive recognition revisited

Problem — new formulation

Let $\mathbb{G}$ be Σ_n or $\mathrm{GL}_n(\mathbb{F}_q)$ or $\mathrm{PGL}_n(\mathbb{F}_q)$ and

$$M_1, \ldots, M_k \in \mathbb{G}.$$

Find for $G := \langle M_1, \ldots, M_k \rangle$:

- The group order $|G|$,
- new nice generators $G = \langle N_1, \ldots, N_m \rangle$ and
- a procedure that, given $M \in \mathbb{G}$,
  - decides, whether or not $M \in G$ and
  - if so, expresses M as an SLP in the N_j and
- another procedure that, given preimages $\hat{M}_1, \ldots, \hat{M}_k$ of the M_i under some homomorphism onto G, produces preimages of the nice generators.

If this problem is solved, we call

$\langle M_1, \ldots, M_k \rangle$ recognised constructively.

Recursion works again

Having recognised H in this sense, we can:

- ask H to generate preimages of its nice generators,

Recursion works again

Having recognised H in this sense, we can:

- ask H to generate preimages of its nice generators,
- compute generators for N,

Recursion works again

Having recognised H in this sense, we can:

- ask H to generate preimages of its nice generators,
- compute generators for N,
- recursively recognise N and

Recursion works again

Having recognised H in this sense, we can:

- ask H to generate preimages of its nice generators,
- compute generators for N,
- recursively recognise N and
- put together the nice generators for G.

Recursion works again

Having recognised H in this sense, we can:

- ask H to generate preimages of its nice generators,
- compute generators for N,
- recursively recognise N and
- put together the nice generators for G.

If we remember how we created the generators for N, then we have recognised G constructively:

- Using H and N we can test membership in G,

Recursion works again

Having recognised H in this sense, we can:

- ask H to generate preimages of its nice generators,
- compute generators for N,
- recursively recognise N and
- put together the nice generators for G.

If we remember how we created the generators for N, then we have recognised G constructively:

- Using H and N we can test membership in G,
- express elements as SLPs in the nice generators,

Recursion works again

Having recognised H in this sense, we can:

- ask H to generate preimages of its nice generators,
- compute generators for N,
- recursively recognise N and
- put together the nice generators for G.

If we remember how we created the generators for N, then we have recognised G constructively:

- Using H and N we can test membership in G,
- express elements as SLPs in the nice generators,
- and, given preimages of the original generators of G under some homomorphism, we can find preimages of the nice generators.

Recognising permutation groups

The same strategy is good for permutation groups.

Recognising permutation groups

The same strategy is good for permutation groups.

For a permutation group, we try one after another:

Recognising permutation groups

The same strategy is good for permutation groups.

For a permutation group, we try one after another:

1. If intransitive, then restrict to an orbit.

Recognising permutation groups

The same strategy is good for permutation groups.

For a permutation group, we try one after another:

1. If intransitive, then restrict to an orbit.
2. If imprimitive, then take block action.

Recognising permutation groups

The same strategy is good for permutation groups.

For a permutation group, we try one after another:

1. If intransitive, then restrict to an orbit.
2. If imprimitive, then take block action.
3. Check if it is a giant, i.e. Σ_n or A_n.
   If so, handle this case separately.

Recognising permutation groups

The same strategy is good for permutation groups.

For a permutation group, we try one after another:

1. If intransitive, then restrict to an orbit.
2. If imprimitive, then take block action.
3. Check if it is a giant, i.e. Σ_n or A_n.
   If so, handle this case separately.
4. Check if it is a giant acting on k-sets.
   If so, handle this case separately.

Recognising permutation groups

The same strategy is good for permutation groups.

For a permutation group, we try one after another:

1. If intransitive, then restrict to an orbit.
2. If imprimitive, then take block action.
3. Check if it is a giant, i.e. Σ_n or A_n.
   If so, handle this case separately.
4. Check if it is a giant acting on k-sets.
   If so, handle this case separately.
5. If all this fails, then compute a stabiliser chain.

Recognising permutation groups

The same strategy is good for permutation groups.

For a permutation group, we try one after another:

1. If intransitive, then restrict to an orbit.
2. If imprimitive, then take block action.
3. Check if it is a giant, i.e. Σ_n or A_n.
   If so, handle this case separately.
4. Check if it is a giant acting on k-sets.
   If so, handle this case separately.
5. If all this fails, then compute a stabiliser chain.

This approach implements the asymptotically best known algorithms for permutation groups in the composition tree framework.

The Aschbacher approach

The Aschbacher approach

Aschbacher has defined classes C1 to C8 of subgroups of $\mathrm{GL}_n(\mathbb{F}_q)$.

The Aschbacher approach

Aschbacher has defined classes C1 to C8 of subgroups of $\mathrm{GL}_n(\mathbb{F}_q)$.

Theorem (Aschbacher, 1984)

Let $G \leq \mathrm{GL}_n(\mathbb{F}_q)$ and $Z := G \cap Z(\mathrm{GL}_n(\mathbb{F}_q))$ the subgroup of scalar matrices. Then G lies in at least one of the classes C1 to C8 or we have:

- *$T \subseteq G/Z \subseteq Aut(T)$ for a non-abelian simple group T, and*
- *G acts absolutely irreducibly on $V = \mathbb{F}_q^n$.*

The Aschbacher approach

Aschbacher has defined classes C1 to C8 of subgroups of $\mathrm{GL}_n(\mathbb{F}_q)$.

Theorem (Aschbacher, 1984)

Let $G \leq \mathrm{GL}_n(\mathbb{F}_q)$ and $Z := G \cap Z(\mathrm{GL}_n(\mathbb{F}_q))$ the subgroup of scalar matrices. Then G lies in at least one of the classes C1 to C8 or we have:

- $T \subseteq G/Z \subseteq Aut(T)$
  for a non-abelian simple group T, and
- *G acts absolutely irreducibly on $V = \mathbb{F}_q^n$.*

(This last case is called C9.)

The Aschbacher approach

Aschbacher has defined classes C1 to C8 of subgroups of $\mathrm{GL}_n(\mathbb{F}_q)$.

Theorem (Aschbacher, 1984)

Let $G \leq \mathrm{GL}_n(\mathbb{F}_q)$ and $Z := G \cap Z(\mathrm{GL}_n(\mathbb{F}_q))$ the subgroup of scalar matrices. Then G lies in at least one of the classes C1 to C8 or we have:

- $T \subseteq G/Z \subseteq Aut(T)$
  for a non-abelian simple group T, and
- *G acts absolutely irreducibly on $V = \mathbb{F}_q^n$.*

(This last case is called C9.)

The classes C1 to C7 are defined "geometrically" and promise some reduction.

The Aschbacher approach

Aschbacher has defined classes C1 to C8 of subgroups of $\mathrm{GL}_n(\mathbb{F}_q)$.

Theorem (Aschbacher, 1984)

Let $G \leq \mathrm{GL}_n(\mathbb{F}_q)$ and $Z := G \cap Z(\mathrm{GL}_n(\mathbb{F}_q))$ the subgroup of scalar matrices. Then G lies in at least one of the classes C1 to C8 or we have:

- $T \subseteq G/Z \subseteq Aut(T)$ *for a non-abelian simple group T, and*
- *G acts absolutely irreducibly on $V = \mathbb{F}_q^n$.*

(This last case is called C9.)

The classes C1 to C7 are defined “geometrically” and promise some reduction.

The classes C8 and C9 have to be dealt with as leaves of the composition tree.

Current status

In the GAP implementation, we have

- a package recogbase providing a framework to implement recognition algorithms and composition trees (Ákos Seress, N.),

Current status

In the GAP implementation, we have

- a package recogbase providing a framework to implement recognition algorithms and composition trees (Ákos Seress, N.),
- a package recog collecting methods to find reductions and recognise leafs constructively,

Current status

In the GAP implementation, we have

- a package recogbase providing a framework to implement recognition algorithms and composition trees (Ákos Seress, N.),
- a package recog collecting methods to find reductions and recognise leafs constructively, Authors (currently): P. Brooksbank, F. Celler, S. Howe, M. Law, S. Linton, G. Malle, N., A. Niemeyer, E. O'Brien, C. Roney-Dougal, Á. Seress,

Current status

In the GAP implementation, we have

- a package recogbase providing a framework to implement recognition algorithms and composition trees (Ákos Seress, N.),
- a package recog collecting methods to find reductions and recognise leafs constructively, Authors (currently): P. Brooksbank, F. Celler, S. Howe, M. Law, S. Linton, G. Malle, N., A. Niemeyer, E. O'Brien, C. Roney-Dougal, Á. Seress,
- complete asymptotically best methods to handle permutation groups,

Current status

In the GAP implementation, we have

- a package recogbase providing a framework to implement recognition algorithms and composition trees (Ákos Seress, N.),
- a package recog collecting methods to find reductions and recognise leafs constructively, Authors (currently): P. Brooksbank, F. Celler, S. Howe, M. Law, S. Linton, G. Malle, N., A. Niemeyer, E. O'Brien, C. Roney-Dougal, Á. Seress,
- complete asymptotically best methods to handle permutation groups,
- methods for all Aschbacher classes for matrix groups and projective groups (some improved algorithms still needed),

Current status

In the GAP implementation, we have

- a package recogbase providing a framework to implement recognition algorithms and composition trees (Ákos Seress, N.),
- a package recog collecting methods to find reductions and recognise leafs constructively, Authors (currently): P. Brooksbank, F. Celler, S. Howe, M. Law, S. Linton, G. Malle, N., A. Niemeyer, E. O'Brien, C. Roney-Dougal, Á. Seress,
- complete asymptotically best methods to handle permutation groups,
- methods for all Aschbacher classes for matrix groups and projective groups (some improved algorithms still needed),
- non-constructive recognition ("name the group"),

Current status

In the GAP implementation, we have

- a package recogbase providing a framework to implement recognition algorithms and composition trees (Ákos Seress, N.),
- a package recog collecting methods to find reductions and recognise leafs constructively, Authors (currently): P. Brooksbank, F. Celler, S. Howe, M. Law, S. Linton, G. Malle, N., A. Niemeyer, E. O'Brien, C. Roney-Dougal, Á. Seress,
- complete asymptotically best methods to handle permutation groups,
- methods for all Aschbacher classes for matrix groups and projective groups (some improved algorithms still needed),
- non-constructive recognition ("name the group"),
- not enough leaf methods,

Current status

In the GAP implementation, we have

- a package recogbase providing a framework to implement recognition algorithms and composition trees (Ákos Seress, N.),
- a package recog collecting methods to find reductions and recognise leafs constructively, Authors (currently): P. Brooksbank, F. Celler, S. Howe, M. Law, S. Linton, G. Malle, N., A. Niemeyer, E. O'Brien, C. Roney-Dougal, Á. Seress,
- complete asymptotically best methods to handle permutation groups,
- methods for all Aschbacher classes for matrix groups and projective groups (some improved algorithms still needed),
- non-constructive recognition ("name the group"),
- not enough leaf methods,
- not much verification.

The End

Bibliography

William M. Kantor and Ákos Seress.
Computing with matrix groups.
In *Groups, combinatorics & geometry (Durham, 2001)*, pages 123–137. World Sci. Publ., River Edge, NJ, 2003.

Charles R. Leedham-Green.
The computational matrix group project.
In *Groups and computation, III (Columbus, OH, 1999)*, volume 8 of *Ohio State Univ. Math. Res. Inst. Publ.*, pages 229–247. de Gruyter, Berlin, 2001.

C. R. Leedham-Green and E. A. O'Brien.
Constructive recognition of classical groups in odd characteristic.
J. Algebra, 322(3):833–881, 2009.

Bibliography

F. Lübeck, K. Magaard, and E. A. O'Brien.
Constructive recognition of $\mathrm{SL}_3(q)$.
J. Algebra, 316(2):619–633, 2007.

Martin W. Liebeck and E. A. O'Brien.
Finding the characteristic of a group of Lie type.
J. Lond. Math. Soc. (2), 75(3):741–754, 2007.

Kay Magaard, E. A. O'Brien, and Ákos Seress.
Recognition of small dimensional representations of general linear groups.
J. Aust. Math. Soc., 85(2):229–250, 2008.

Bibliography

Max Neunhöffer and Ákos Seress.
A data structure for a uniform approach to computations with finite groups.
In *ISSAC 2006*, pages 254–261. ACM, New York, 2006.

E. A. O'Brien.
Towards effective algorithms for linear groups.
In *Finite geometries, groups, and computation*, pages 163–190. Walter de Gruyter, Berlin, 2006.

Ákos Seress.
A unified approach to computations with permutation and matrix groups.
In *International Congress of Mathematicians. Vol. II*, pages 245–258. Eur. Math. Soc., Zürich, 2006.