Join us at the . . .

11th Conference of the Fachgruppe Computeralgebra

Join us at the . . .

11th Conference of the Fachgruppe Computeralgebra

- June 2 to 4, 2025, MPI, Leipzig

Join us at the . . .

11th Conference of the Fachgruppe Computeralgebra

- June 2 to 4, 2025, MPI, Leipzig
- Details: www.mis.mpg.de/computeralgebra

Join us at the . . .

11th Conference of the Fachgruppe Computeralgebra

- June 2 to 4, 2025, MPI, Leipzig
- Details: www.mis.mpg.de/computeralgebra
- Main speakers: Marta Panizzut, Milena Wrobel, Clemens Hofstadler, Thomas Kahle, Lorenz Panny

Join us at the . . .

11th Conference of the Fachgruppe Computeralgebra

- June 2 to 4, 2025, MPI, Leipzig
- Details: www.mis.mpg.de/computeralgebra
- Main speakers: Marta Panizzut, Milena Wrobel, Clemens Hofstadler, Thomas Kahle, Lorenz Panny
- Theme talk on research data management by Christiane Görgen

Join us at the . . .

11th Conference of the Fachgruppe Computeralgebra

- June 2 to 4, 2025, MPI, Leipzig
- Details: www.mis.mpg.de/computeralgebra
- Main speakers: Marta Panizzut, Milena Wrobel, Clemens Hofstadler, Thomas Kahle, Lorenz Panny
- Theme talk on research data management by Christiane Görgen
- Submit contributed talks until April 15

Join us at the . . .

11th Conference of the Fachgruppe Computeralgebra

- June 2 to 4, 2025, MPI, Leipzig
- Details: www.mis.mpg.de/computeralgebra
- Main speakers: Marta Panizzut, Milena Wrobel, Clemens Hofstadler, Thomas Kahle, Lorenz Panny
- Theme talk on research data management by Christiane Görgen
- Submit contributed talks until April 15
- Young researchers *highly* welcome!

Join us at the . . .

11th Conference of the Fachgruppe Computeralgebra

- June 2 to 4, 2025, MPI, Leipzig
- Details: www.mis.mpg.de/computeralgebra
- Main speakers: Marta Panizzut, Milena Wrobel, Clemens Hofstadler, Thomas Kahle, Lorenz Panny
- Theme talk on research data management by Christiane Görgen
- Submit contributed talks until April 15
- Young researchers *highly* welcome!
- Young Talent Award for best contributed talk: € 500 plus invitation as main speaker for next conference

Fachgruppe Computeralgebra

Die Fachgruppe Computeralgebra fördert Lehre, Forschung, Entwicklung, Anwendungen, Informationsaustausch und Zusammenarbeit auf dem Gebiet der Computeralgebra in Deutschland.

fachgruppe-computeralgebra.de

Fachgruppe der GI in Kooperation mit der DMV und GAMM

Gesellschaft für Informatik

Deutsche Mathematiker-Vereinigung

Gesellschaft für Angewandte Mathematik und Mechanik

The state of OSCAR

John Abbott, Thomas Breuer, Lars Göttgens, Max Horn

March 26, 2025

Overview

The state of OSCAR
Max Horn

S-characters
Thomas Breuer

Algebraic Lie Theory
Lars Göttgens

Practical Fast Linear Algebra Over $\mathbb{Z}$
John Abbott

Summing it up
Max Horn

The state of OSCAR

Max Horn

What happened so far (abridged version)

- 01/2017: OSCAR Kickoff Meeting in Kaiserslautern

What happened so far (abridged version)

- 01/2017: OSCAR Kickoff Meeting in Kaiserslautern
- 09/2019: Main OSCAR repository created

What happened so far (abridged version)

- 01/2017: OSCAR Kickoff Meeting in Kaiserslautern
- 09/2019: Main OSCAR repository created
- 03/2024: OSCAR v1.0.0 released!

What happened so far (abridged version)

- 01/2017: OSCAR Kickoff Meeting in Kaiserslautern
- 09/2019: Main OSCAR repository created
- 03/2024: OSCAR v1.0.0 released!
- 11/2024: SFB extended for another (final) four years

What happened so far (abridged version)

- 01/2017: OSCAR Kickoff Meeting in Kaiserslautern
- 09/2019: Main OSCAR repository created
- 03/2024: OSCAR v1.0.0 released!
- 11/2024: SFB extended for another (final) four years
- 02/2025: relaunch of oscar-system.org

What happened so far (abridged version)

- 01/2017: OSCAR Kickoff Meeting in Kaiserslautern
- 09/2019: Main OSCAR repository created
- 03/2024: OSCAR v1.0.0 released!
- 11/2024: SFB extended for another (final) four years
- 02/2025: relaunch of oscar-system.org
- 03/2025: OSCAR v1.3.1 released

What happened so far (abridged version)

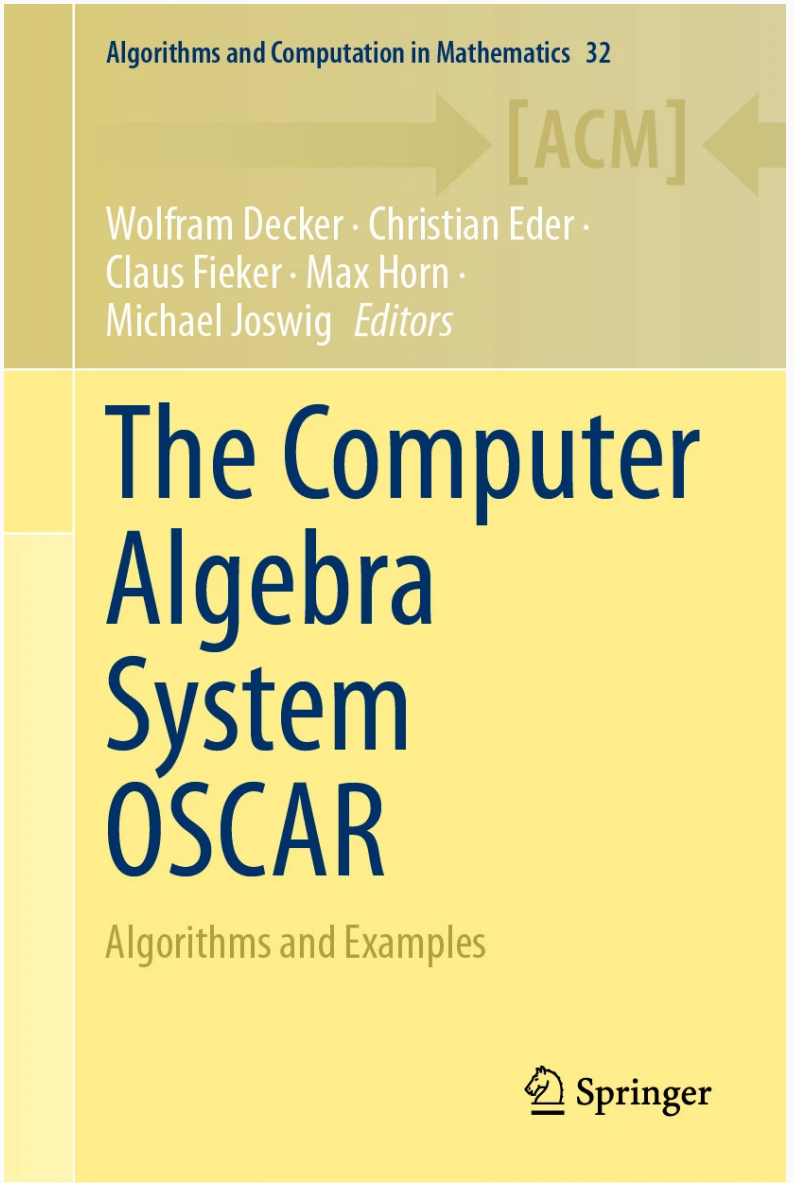
- 01/2017: OSCAR Kickoff Meeting in Kaiserslautern
- 09/2019: Main OSCAR repository created
- 03/2024: OSCAR v1.0.0 released!
- 11/2024: SFB extended for another (final) four years
- 02/2025: relaunch of oscar-system.org
- 03/2025: OSCAR v1.3.1 released
- In the past 12 months, 47 people contributed code to our main code repository

The OSCAR book

The OSCAR book is out!

Get it while it's hot

See book.oscar-system.org



And now to something different

The state of OSCAR
Max Horn

S-characters
Thomas Breuer

Algebraic Lie Theory
Lars Göttgens

Practical Fast Linear Algebra Over $\mathbb{Z}$
John Abbott

Summing it up
Max Horn

S-characters

Thomas Breuer

S-characters

- joint work with Michael Joswig and Gunter Malle
- `https://arxiv.org/abs/2408.16785`
- `https://github.com/oscar-system/FriendsOfOscar/tree/main/Zeros%2Bof%2BS-Characters`

S-characters: Definitions

G	finite group
$g_1 = 1, g_2, \ldots, g_n$	conjugacy class representatives
$\mathrm{Irr}(G) = \{\chi_1 = 1_G, \chi_2, \ldots, \chi_n\}$	irreducible characters
$A = [\chi_i(g_j)]_{i,j} \in \mathbb{C}^{n \times n}$	character table

S-characters: Definitions

G finite group

$g_1 = 1, g_2, \ldots, g_n$ conjugacy class representatives

$\mathrm{Irr}(G) = \{\chi_1 = 1_G, \chi_2, \ldots, \chi_n\}$ irreducible characters

$A = [\chi_i(g_j)]_{i,j} \in \mathbb{C}^{n \times n}$ character table

$$\mathrm{SCoord}(G) = \left\{x \in \mathbb{Z}^n; x \cdot A \in (\mathbb{R}^n)^{\geq 0}, x_1 = 1\right\}$$

$\mathrm{SChar}(G) = \{\sum_{i=1}^n x_i \chi_i; x \in \mathrm{SCoord}(G)\}$ S-characters

(Zhmud' 1995)

Example: S-characters of A_5

$G = A_5$

1A	2A	3A	5A	5B
1	1	1	1	1
3	-1	.	-b5	*
3	-1	.	*	-b5
4	.	1	-1	-1
5	1	-1	.	.

$\mathtt{b5} = \frac{1}{2}\left(-1+\sqrt{5}\right)$

x_1	x_2	x_3	x_4	x_5	1A	2A	3A	5A	5B
1	−1	−1	0	1	0	4	0	0	0
1	0	0	0	0	1	1	1	1	1
1	0	0	0	1	6	2	0	1	1
1	0	0	1	−1	0	0	3	0	0
1	0	0	1	0	5	1	2	0	0
1	0	0	1	1	10	2	1	0	0
1	0	0	1	2	15	3	0	0	0
1	0	1	−1	0	0	0	0	b5+3	-b5+2
1	0	1	0	0	4	0	1	b5+2	-b5+1
1	0	1	0	1	9	1	0	b5+2	-b5+1
1	1	0	−1	0	0	0	0	-b5+2	b5+3
1	1	0	0	0	4	0	1	-b5+1	b5+2
1	1	0	0	1	9	1	0	-b5+1	b5+2
1	1	1	0	1	12	0	0	2	2
1	1	1	1	1	16	0	1	1	1
1	1	1	1	2	21	1	0	1	1
1	1	1	2	1	20	0	2	0	0
1	1	1	2	2	25	1	1	0	0
1	1	1	2	3	30	2	0	0	0
1	1	2	1	2	24	0	0	b5+2	-b5+1
1	2	1	1	2	24	0	0	-b5+1	b5+2
1	2	2	2	3	36	0	0	1	1
1	2	2	3	3	40	0	1	0	0
1	2	2	3	4	45	1	0	0	0
1	3	3	4	5	60	0	0	0	0

Zeros of S-characters

- Each S-character except 1_G has a zero.
- If $H \leq G$ then 1_H^G is an S-character.
  (Permutation character)
- If $H < G$ then 1_H^G is zero at an element of prime power order.
  (Fein-Kantor-Schacher 1981)

Zeros of S-characters

- Each S-character except 1_G has a zero.
- If $H \leq G$ then 1_H^G is an S-character.
  (Permutation character)
- If $H < G$ then 1_H^G is zero at an element of prime power order.
  (Fein-Kantor-Schacher 1981)

Question (J.-P. Serre, 2023): Does this hold also for S-characters?

Zeros of S-characters

- Each S-character except 1_G has a zero.
- If $H \leq G$ then 1_H^G is an S-character.
  (Permutation character)
- If $H < G$ then 1_H^G is zero at an element of prime power order.
  (Fein-Kantor-Schacher 1981)

Question (J.-P. Serre, 2023): Does this hold also for S-characters?

$\mathrm{SChar}^{ppp}(G) = \{\psi \in \mathrm{SChar}(G); \psi(g) > 0 \text{ if } |g| = 1 \text{ or prime power}\}$

Zeros of S-characters

- Each S-character except 1_G has a zero.
- If $H \leq G$ then 1_H^G is an S-character.
  (Permutation character)
- If $H < G$ then 1_H^G is zero at an element of prime power order.
  (Fein-Kantor-Schacher 1981)

Question (J.-P. Serre, 2023): Does this hold also for S-characters?

$\mathrm{SChar}^{ppp}(G) = \{\psi \in \mathrm{SChar}(G); \psi(g) > 0 \text{ if } |g| = 1 \text{ or prime power}\}$

Question: $\mathrm{SChar}^{ppp}(G) \neq \{1_G\}$?

Compute $\mathrm{SChar}^{ppp}(G)$: Situation

$A = [\chi_i(g_j)]_{i,j}$ character table

Assume $A \in \mathbb{R}^{n \times n}$.

$\mathrm{SCoord}(G) = \{x \in \mathbb{Z}^n; x \cdot A \in (\mathbb{R}^n)^{\geq 0}, x_1 = 1\}$

Combinatorially:
$|x_i| \leq \chi_i(1)$ suffices

Compute $\mathrm{SChar}^{ppp}(G)$: Situation

$A = [\chi_i(g_j)]_{i,j}$ character table

Assume $A \in \mathbb{R}^{n \times n}$.

$$\mathrm{SCoord}(G) = \{x \in \mathbb{Z}^n; x \cdot A \in (\mathbb{R}^n)^{\geq 0}, x_1 = 1\}$$

Combinatorially:

$|x_i| \leq \chi_i(1)$ suffices

($G = A_5$: 4851 candidates,
$G = A_8$: $\approx 10^{20}$ candidates)

Compute $\mathrm{SChar}^{ppp}(G)$: Situation

$A = [\chi_i(g_j)]_{i,j}$ character table

Assume $A \in \mathbb{R}^{n \times n}$.

$$\mathrm{SCoord}(G) = \{x \in \mathbb{Z}^n; x \cdot A \in (\mathbb{R}^n)^{\geq 0}, x_1 = 1\}$$

Combinatorially:

$|x_i| \leq \chi_i(1)$ suffices

($G = A_5$: 4851 candidates,
$G = A_8$: $\approx 10^{20}$ candidates)

Geometrically:

$S(G) = \{x \in \mathbb{R}^n; x \cdot A \in (\mathbb{R}^n)^{\geq 0}, x_1 = 1\}$
is a bounded polytope,
$\mathrm{SCoord}(G)$ are its lattice points.

Compute $\mathrm{SChar}^{ppp}(G)$: Situation

$A = [\chi_i(g_j)]_{i,j}$ character table

Assume $A \in \mathbb{R}^{n \times n}$.

$$\mathrm{SCoord}(G) = \{x \in \mathbb{Z}^n; x \cdot A \in (\mathbb{R}^n)^{\geq 0}, x_1 = 1\}$$

Combinatorially:
$|x_i| \leq \chi_i(1)$ suffices

($G = A_5$: 4851 candidates,
$G = A_8$: $\approx 10^{20}$ candidates)

Geometrically:
$S(G) = \{x \in \mathbb{R}^n; x \cdot A \in (\mathbb{R}^n)^{\geq 0}, x_1 = 1\}$
is a bounded polytope,
$\mathrm{SCoord}(G)$ are its lattice points.

Note: All lattice points except 1_G lie on the boundary.

Compute $\mathrm{SChar}^{ppp}(G)$: Idea

- **Use OSCAR:**

  Create the polytope $S(G)$,
  enumerate its lattice points,
  check which of them belong to $\mathrm{SChar}^{ppp}(G)$.

Compute $\mathrm{SChar}^{ppp}(G)$: Idea

- **Use OSCAR:**

  Create the polytope $S(G)$,
  enumerate its lattice points,
  check which of them belong to $\mathrm{SChar}^{ppp}(G)$.

- **Challenge:**

  The polytope is defined over an embedded number field.

Compute $\mathrm{SChar}^{ppp}(G)$: Idea

- **Use OSCAR:**

  Create the polytope $S(G)$,
  enumerate its lattice points,
  check which of them belong to $\mathrm{SChar}^{ppp}(G)$.

- **Challenge:**

  The polytope is defined over an embedded number field.

- **Improvements:**

  Prescribe nonzero values a priori on rational columns.
  Use projections to factor groups where possible.

Compute $\mathrm{SChar}^{ppp}(\mathrm{A}_8)$

```
julia> using S_characters, Oscar

julia> tbls = map(character_table, ["A8", "2.A8"]);

julia> @time length(s_characters(tbls[1]))
  2.462459 seconds (27.24 M allocations: 416.007 MiB)
3635

julia> @time length(s_characters(tbls[1], ppow_nonzero = true))
  0.081231 seconds (543.64 k allocations: 4.496 MiB)
1
```

Compute $\mathrm{SChar}^{ppp}(\mathrm{A}_8)$

```
julia> using S_characters, Oscar

julia> tbls = map(character_table, ["A8", "2.A8"]);

julia> @time length(s_characters(tbls[1]))
  2.462459 seconds (27.24 M allocations: 416.007 MiB)
3635

julia> @time length(s_characters(tbls[1], ppow_nonzero = true))
  0.081231 seconds (543.64 k allocations: 4.496 MiB)
1

julia> @time length(s_characters(tbls[2], ppow_nonzero = true))
  0.308319 seconds (3.79 M allocations: 30.041 MiB)
3

julia> @time length(s_characters_via_factor_group(tbls[2],
                        [1, 2], ppow_nonzero = true))
  0.186391 seconds (656.97 k allocations: 6.412 MiB)
3
```

Results: $\mathrm{SChar}^{ppp}(G) \neq \{1_G\}$

G	#classes	#real	#rat.	#SChar^{ppp}	#virt. SChar^{ppp}
A_8	14	12	12	1	0
$2.\mathrm{A}_8$	23	18	18	2	1
$2^6:\mathrm{A}_8$	41	39	39	8	1
A_9	18	17	17	3	0
$2.\mathrm{A}_9$	30	26	26	10	7
A_{10}	24	24	23	1	0
$2.\mathrm{A}_{10}$	39	38	35	3	2
A_{11}	31	30	29	6	0
$2.\mathrm{A}_{11}$	49	47	43	53	26
M_{12}	15	14	14	1	0
$M_{12}.2$	21	21	19	1	0
M_{24}	26	21	21	223	4
J_1	15	15	10	1	0
J_2	21	21	16	155	8
$2.J_2$	38	37	28	2571	1280
$J_2.2$	27	27	26	12	1
J_3	21	20	14	5	0
M^cL	24	18	18	2588	65
HS	24	22	22	93	2
$2.HS$	42	34	33	2211	1094

Results: $\mathrm{SChar}^{ppp}(G) \neq \{1_G\}$

G	#classes	#real	#rat.	#SChar^{ppp}	#virt. SChar^{ppp}
$\mathrm{L}_4(3)$	29	26	25	6	0
$\mathrm{L}_5(2)$	27	20	18	11	0
$2^5.\mathrm{L}_5(2)$	41	29	27	2696	118
$\mathrm{S}_4(4)$	27	27	18	133	10
$\mathrm{S}_4(4).2$	30	29	28	5	1
$\mathrm{S}_4(5)$	34	34	25	76	11
$\mathrm{S}_6(2)$	30	30	30	1	1
$\mathrm{U}_4(2)$	20	15	15	2	1
$2.\mathrm{U}_4(2)$	34	24	24	1	1
${}^2F_4(2)'$	22	19	16	71	2
${}^2F_4(2)'.2$	29	23	23	7	3
${}^2G_2(27)$	35	32	13	9	0
$G_2(3)$	23	22	21	24	0

Results: $\mathrm{SChar}^{ppp}(G) \neq \{1_G\}$

Our examples of $\psi \in \mathrm{SChar}^{ppp}(G) \setminus \{1_G\}$:

- All known ψ are rational-valued.
  (Are there non-rational examples?)
- $G = \mathrm{A}_8$ is the smallest such Atlas group.
- Checked simple groups by increasing conjugacy class number:

  The computation did not finish for
  $O'N$, $G_2(4)$, He, ${}^3D_4(2)$, Ru, $O_8^-(2)$, Co_3, A_{12}
  (with 128 GB RAM).

Results: $\mathrm{SChar}^{ppp}(G) \neq \{1_G\}$

Our examples of $\psi \in \mathrm{SChar}^{ppp}(G) \setminus \{1_G\}$:

- All known ψ are rational-valued.
  (Are there non-rational examples?)
- $G = \mathrm{A}_8$ is the smallest such Atlas group.
- Checked simple groups by increasing conjugacy class number:

  The computation did not finish for
  $O'N$, $G_2(4)$, He, ${}^3D_4(2)$, Ru, $O_8^-(2)$, Co_3, A_{12}
  (with 128 GB RAM).
- Checked for *rational* ψ in G with at most 40 rational classes:

  No example ψ known for $|G| \leq 1535$.
  No example ψ known for solvable G.

  (Are there solvable examples?
  We can show:
  Solvable groups do not have *proper characters* as examples.)

Future?

S-characters are . . .

Future?

S-characters are . . .

. . . virtual characters and real.

Future?

S-characters are . . .

. . . virtual characters and real.

Virtual Reality?

Algebraic Lie Theory

Lars Göttgens

Feature overview

- (generalized) Cartan matrices,
- Root systems,
- Weyl groups,
- Lie algebras,
- Lie subalgebras, ideals
- Lie algebra modules
- Morphisms of Lie algebras, morphisms of modules
- Representation theory helpers (L semisimple, $\operatorname{char} K = 0$):
  - dimension formula, character formula, character formula for Demazure modules

Feature overview

- (generalized) Cartan matrices,
- Root systems,
- Weyl groups,
- Lie algebras,
- Lie subalgebras, ideals
- Lie algebra modules
- Morphisms of Lie algebras, morphisms of modules
- Representation theory helpers (L semisimple, $\operatorname{char} K = 0$):
  - dimension formula, character formula, character formula for Demazure modules

Non-trivial contributions by: A. Della Vecchia, LG, M. Horn, J. Nohilly, J. Peters, F. Röhrich, L. Voggesberger, T. Wiedemann

Feature overview (graduated from `experimental`)

- **(generalized) Cartan matrices,**
- **Root systems,**
- **Weyl groups,**
- Lie algebras,
- Lie subalgebras, ideals
- Lie algebra modules
- Morphisms of Lie algebras, morphisms of modules
- Representation theory helpers (L semisimple, char $K = 0$):
  - dimension formula, character formula, character formula for Demazure modules

Non-trivial contributions by: A. Della Vecchia, LG, M. Horn, J. Nohilly, J. Peters, F. Röhrich, L. Voggesberger, T. Wiedemann

Feature overview (this talk)

- **(generalized) Cartan matrices,**
- **Root systems,**
- **Weyl groups,**
- **Lie algebras,**
- Lie subalgebras, ideals
- Lie algebra modules
- Morphisms of Lie algebras, morphisms of modules
- Representation theory helpers (L semisimple, $\operatorname{char} K = 0$):
  - dimension formula, character formula, character formula for Demazure modules

Non-trivial contributions by: A. Della Vecchia, LG, M. Horn, J. Nohilly, J. Peters, F. Röhrich, L. Voggesberger, T. Wiedemann

Root systems: construction

Only *reduced*, *cristallographic* root systems are supported.

Construction via:

Root systems: construction

Only *reduced*, *cristallographic* root systems are supported.

Construction via:

- Cartan matrix (as `ZZMatrix` or `MatrixInt`)

```
julia> rs = root_system(matrix(ZZ, [2 -1; -1 2]))
Root system of rank 2
  of type A2
```

Root systems: construction

Only *reduced*, *cristallographic* root systems are supported.

Construction via:

- Cartan matrix (as `ZZMatrix` or `MatrixInt`)

```
julia> rs = root_system(matrix(ZZ, [2 -1; -1 2]))
Root system of rank 2
  of type A2
```

- Dynkin type, or list thereof

```
julia> rs = root_system([(:G, 2), (:E, 7)])
Root system of rank 9
  of type G2 x E7
```

Root systems: construction

Only *reduced*, *cristallographic* root systems are supported.

Construction via:

- Cartan matrix (as `ZZMatrix` or `MatrixInt`)

```
julia> rs = root_system(matrix(ZZ, [2 -1; -1 2]))
Root system of rank 2
  of type A2
```

- Dynkin type, or list thereof

```
julia> rs = root_system([(:G, 2), (:E, 7)])
Root system of rank 9
  of type G2 x E7
```

Running example:

```
julia> rs = root_system(:B, 3)
Root system of rank 3
  of type B3
```

Root systems: reflections at simple and arbitrary roots

```
julia> positive_roots(rs)
9-element Vector{RootSpaceElem}:
  α_1
  α_2
  α_3
  α_1 + α_2
  α_2 + α_3
  α_2 + 2*α_3
  α_1 + α_2 + α_3
  α_1 + α_2 + 2*α_3
  α_1 + 2*α_2 + 2*α_3

julia> reflect(root(rs, 1), 2) # σ_2(α_1)
α_1 + α_2

julia> reflect(root(rs, 1), root(rs, 4)) # σ_{α_1+α_2}(α_1)
-α_2
```

Root systems: root arithmetics

```
julia> positive_roots(rs)
9-element Vector{RootSpaceElem}:
  α_1
  α_2
  α_3
  α_1 + α_2
  α_2 + α_3
  α_2 + 2*α_3
  α_1 + α_2 + α_3
  α_1 + α_2 + 2*α_3
  α_1 + 2*α_2 + 2*α_3

julia> is_positive_root_with_index(root(rs, 5) + root(rs, 3))
(true, 6)

julia> is_positive_root_with_index(root(rs, 5) + 2*root(rs, 2))
(false, 0)
```

Weyl groups

Root systems and Weyl groups are intrinsically connected.

```
julia> W = weyl_group(rs)
Weyl group
  of root system of rank 3
    of type B3

julia> s1,s2,s3 = gens(W)
[s1, s2, s3]

julia> one(W)
id

julia> longest_element(W)
s1 * s2 * s1 * s3 * s2 * s1 * s3 * s2 * s3
```

Weyl groups: element representations

```
julia> x = W([3,2,1])
s3 * s2 * s1

julia> word(x)
UInt8[0x03, 0x02, 0x01]

julia> letters(x)
Int[3, 2, 1]

julia> syllables(x)
3-element Vector{Pair{Int64, ZZRingElem}}:
  3 => 1
  2 => 1
  1 => 1
```

All of these representations can also be passed to `W` to construct an element.

Weyl groups: normal form

Elements are always stored as a word in *short-lex normal form*, i.e., the lexicographic smallest word of minimal length.

```
julia> W([1,2,2,3])
s1 * s3
```

Weyl groups: normal form

Elements are always stored as a word in *short-lex normal form*, i.e., the lexicographic smallest word of minimal length.

```
julia> W([1,2,2,3])
s1 * s3

julia> W([3,2,1,2,3])
s1 * s3 * s2 * s1 * s3
```

Weyl groups: normal form

Elements are always stored as a word in *short-lex normal form*, i.e., the lexicographic smallest word of minimal length.

```
julia> W([1,2,2,3])
s1 * s3

julia> W([3,2,1,2,3])
s1 * s3 * s2 * s1 * s3

julia> x = s3 * s1 * s2 * s3 * s2 * s1
s1 * s2 * s3 * s2 * s1 * s3
```

Weyl groups: normal form

Elements are always stored as a word in *short-lex normal form*, i.e., the lexicographic smallest word of minimal length.

```
julia> W([1,2,2,3])
s1 * s3

julia> W([3,2,1,2,3])
s1 * s3 * s2 * s1 * s3

julia> x = s3 * s1 * s2 * s3 * s2 * s1
s1 * s2 * s3 * s2 * s1 * s3

julia> inv(x)
s1 * s2 * s3 * s2 * s1 * s3
```

Weyl groups: all reduced expressions of an element

```
julia> x = s3 * s1 * s2 * s3 * s2 * s1
s1 * s2 * s3 * s2 * s1 * s3

julia> reduced_expressions(x)
ReducedExpressionIterator(s1 * s2 * s3 * s2 * s1 * s3, false)
```

Weyl groups: all reduced expressions of an element

```
julia> x = s3 * s1 * s2 * s3 * s2 * s1
s1 * s2 * s3 * s2 * s1 * s3

julia> reduced_expressions(x)
ReducedExpressionIterator(s1 * s2 * s3 * s2 * s1 * s3, false)

julia> reduced_expressions(x) |> collect .|> Vector{Int}
4-element Vector{Vector{Int64}}:
  [1, 2, 3, 2, 1, 3]
  [1, 2, 3, 2, 3, 1]
  [1, 3, 2, 3, 2, 1]
  [3, 1, 2, 3, 2, 1]
```

Weyl groups: all reduced expressions of an element

```
julia> x = s3 * s1 * s2 * s3 * s2 * s1
s1 * s2 * s3 * s2 * s1 * s3

julia> reduced_expressions(x)
ReducedExpressionIterator(s1 * s2 * s3 * s2 * s1 * s3, false)

julia> reduced_expressions(x) |> collect .|> Vector{Int}
4-element Vector{Vector{Int64}}:
  [1, 2, 3, 2, 1, 3]
  [1, 2, 3, 2, 3, 1]
  [1, 3, 2, 3, 2, 1]
  [3, 1, 2, 3, 2, 1]

julia> reduced_expressions(x; up_to_commutation=true) |>
↪  collect .|> Vector{Int}
4-element Vector{Vector{Int64}}:
  [1, 2, 3, 2, 1, 3]
  [1, 3, 2, 3, 2, 1]
```

Weyl groups: operation on roots and weights

Warning: **RIGHT** action!

Weyl groups: operation on roots and weights

Warning: **RIGHT** action!

```
julia> a = positive_root(rs, 4)
α_1 + α_2

julia> a * s1
α_2
```

Weyl groups: operation on roots and weights

Warning: **RIGHT** action!

```
julia> a = positive_root(rs, 4)
α_1 + α_2

julia> a * s1
α_2

julia> w = fundamental_weight(rs, 1)
ω_1

julia> w * s1
-ω_1 + ω_2
```

Weyl groups: conversion to other group types

Challenge: Weyl groups are the first (widely-used, multiplicative) groups in OSCAR, that are not backed by GAP.

Weyl groups: conversion to other group types

Challenge: Weyl groups are the first (widely-used, multiplicative) groups in OSCAR, that are not backed by GAP.

$\Rightarrow$ not supported yet in all functions that take a group

Weyl groups: conversion to other group types

Challenge: Weyl groups are the first (widely-used, multiplicative) groups in OSCAR, that are not backed by GAP.

$\Rightarrow$ not supported yet in all functions that take a group

(temporary) solution: convert to a group of another type

Weyl groups: conversion to other group types

Challenge: Weyl groups are the first (widely-used, multiplicative) groups in OSCAR, that are not backed by GAP.

$\Rightarrow$ not supported yet in all functions that take a group

(temporary) solution: convert to a group of another type

```
julia> iso = isomorphism(PermGroup, W)
Map defined by a julia-function with inverse
  from Weyl group of root system of type B3
  to permutation group of degree 6 and order 48

julia> FPGroup(W)
Finitely presented group of order 48

julia> geometric_representation(W)
(Matrix group of degree 3 over ZZ, Map: W -> matrix group)
```

Weyl groups: conversion to other group types

Challenge: Weyl groups are the first (widely-used, multiplicative) groups in OSCAR, that are not backed by GAP.

$\Rightarrow$ not supported yet in all functions that take a group

(temporary) solution: convert to a group of another type

```
julia> iso = isomorphism(PermGroup, W)
Map defined by a julia-function with inverse
  from Weyl group of root system of type B3
  to permutation group of degree 6 and order 48

julia> FPGroup(W)
Finitely presented group of order 48

julia> geometric_representation(W)
(Matrix group of degree 3 over ZZ, Map: W -> matrix group)
```

A lot of ongoing effort by many people to make Weyl groups more usable, and add more features.

Lie algebras: construction

Fix a field F.

Construction via:

Lie algebras: construction

Fix a field F.
Construction via:

- basis of matrices, with shorthands for $\mathfrak{gl}_n$, $\mathfrak{sl}_n$, $\mathfrak{so}_n$, ...

Lie algebras: construction

Fix a field F.
Construction via:

- basis of matrices, with shorthands for $\mathfrak{gl}_n$, $\mathfrak{sl}_n$, $\mathfrak{so}_n$, ...
- structure constant table (as `Array{elem_type(F), 3}` or `Matrix{sparse_row_type(F)}`)

Lie algebras: construction

Fix a field F.
Construction via:

- basis of matrices, with shorthands for $\mathfrak{gl}_n$, $\mathfrak{sl}_n$, $\mathfrak{so}_n$, ...
- structure constant table (as `Array{elem_type(F), 3}` or `Matrix{sparse_row_type(F)}`)
- root system or Dynkin type

Lie algebras: construction

Fix a field F.
Construction via:

- basis of matrices, with shorthands for $\mathfrak{gl}_n$, $\mathfrak{sl}_n$, $\mathfrak{so}_n$, ...
- structure constant table (as `Array{elem_type(F), 3}` or `Matrix{sparse_row_type(F)}`)
- root system or Dynkin type

Running example:

```
julia> sc = fill(sparse_row(QQ), 14, 14);
julia> sc[1,3] = sparse_row(QQ, [3,8,10,11,13],
↪  [3,2,-4,-2,2]);
julia> sc[1,4] = sparse_row(QQ, [4,8,9,10,11,13],
↪  [5,4//3,-3,-8//3,8//3,-8//3]);
# ...
julia> L = lie_algebra(QQ, sc, ["b_$i" for i in 1:14])
Abstract Lie algebra
  of dimension 14
```

Lie algebras: properties

```
julia> dim(L)
14
```

Lie algebras: properties

```
julia> dim(L)
14

julia> is_nilpotent(L)
false
```

Lie algebras: properties

```
julia> dim(L)
14

julia> is_nilpotent(L)
false

julia> is_solvable(L)
false
```

Lie algebras: properties

```
julia> dim(L)
14

julia> is_nilpotent(L)
false

julia> is_solvable(L)
false

julia> is_perfect(L)
true
```

Lie algebras: properties

```
julia> dim(L)
14

julia> is_nilpotent(L)
false

julia> is_solvable(L)
false

julia> is_perfect(L)
true

julia> rank(killing_matrix(L))
14
```

Lie algebras: properties

```
julia> dim(L)
14

julia> is_nilpotent(L)
false

julia> is_solvable(L)
false

julia> is_perfect(L)
true

julia> rank(killing_matrix(L))
14

julia> is_semisimple(L)
true
```

Lie algebras: Cartan subalgebra

```
julia> csa = cartan_subalgebra(L)
Lie subalgebra
  of dimension 2
of abstract Lie algebra over QQ
```

Lie algebras: Cartan subalgebra

```
julia> csa = cartan_subalgebra(L)
Lie subalgebra
  of dimension 2
of abstract Lie algebra over QQ

julia> basis(csa)
2-element Vector{AbstractLieAlgebraElem{QQFieldElem}}:
  b_1
  b_2
```

Lie algebras: Cartan subalgebra

```
julia> csa = cartan_subalgebra(L)
Lie subalgebra
  of dimension 2
of abstract Lie algebra over QQ

julia> basis(csa)
2-element Vector{AbstractLieAlgebraElem{QQFieldElem}}:
  b_1
  b_2

julia> normalizer(L, csa) == csa
true

julia> is_nilpotent(lie_algebra(csa)[1])
true
```

Lie algebras: root system

```
julia> rs = root_system(L)
Root system of rank 2
  of type G2
```

Lie algebras: root system

```
julia> rs = root_system(L)
Root system of rank 2
  of type G2

julia> cartan_matrix(rs)
[ 2   -3]
[-1    2]
```

Lie algebras: root system

```
julia> rs = root_system(L)
Root system of rank 2
  of type G2

julia> cartan_matrix(rs)
[ 2   -3]
[-1    2]

julia> positive_roots(rs)
6-element Vector{RootSpaceElem}:
  α_1
  α_2
  α_1 + α_2
  2*α_1 + α_2
  3*α_1 + α_2
  3*α_1 + 2*α_2
```

Lie algebras: Chevalley basis

```
julia> xs = chevalley_basis(L)[1]
6-element Vector{AbstractLieAlgebraElem{QQFieldElem}}:
  -3//2*b_4 - 1//2*b_8 + 3//2*b_9 + b_10 - b_11 + b_13
  -2*b_8 + b_10 - b_11 + b_13
  9//4*b_7 + 3//4*b_8 - 3//2*b_10 + 3//2*b_11 - 3//2*b_13
  27//4*b_4 + 27//4*b_5 - 9//4*b_8 - 27//4*b_9 - 9//4*b_10 + 9//4*b_11
  27//8*b_8 + 27//8*b_10 + 27//4*b_11 - 81//8*b_12 - 27//4*b_13
  81//8*b_8 - 81//4*b_10 - 81//8*b_11 + 81//8*b_13

julia> ys = chevalley_basis(L)[2]
6-element Vector{AbstractLieAlgebraElem{QQFieldElem}}:
  -2//9*b_8 + 4//9*b_10 + 2//9*b_11 - 2//3*b_12 - 2//9*b_13 + 2//3*b_14
  [...]
  -4//243*b_3 - 4//729*b_8 + 8//729*b_10 + 4//729*b_11 - 4//729*b_13

julia> hs = chevalley_basis(L)[3]
2-element Vector{AbstractLieAlgebraElem{QQFieldElem}}:
  -2*b_1 - 3*b_2
  b_1 + 2*b_2
```

Give it a try yourself!

Get in touch!

We are looking forward to your questions,
comments, bug reports, code contributions, ...

Practical Fast Linear Algebra Over $\mathbb{Z}$

John Abbott

Practically fast linear algebra over the integers

Joint work with Claus Fieker

Outline:

- background & basics
- testing unimodularity
- solving linear systems (square, full-rank)
- computing (non-zero) determinants

Existing analyses assume A is *uniform*:

- most entries have similar size
- simplifies complexity analysis
- often an *unrealistic* assumption

Basics and Background

- Work with square matrix $A \in \mathrm{Mat}_{n\times n}(\mathbb{Z})$:
  - dense $\longrightarrow$ not many zero entries
  - uniform $\longrightarrow$ most entries same size as largest
  - full-rank $\longrightarrow$ $\det A \neq 0$
- Write $\|A\|_{max}$ for largest entry in A
- SNF $A = U_L\, S\, U_R$ with $S = \mathrm{diag}(s_1, \ldots, s_n)$ and $s_j | s_{j+1}$
- Hadamard bound for $|\det A|$:
  - $H = \prod |r_i|$ where r_i is i-th row
  - $H^2 \in \mathbb{Z}$ $\longleftarrow$ quick & easy to compute
  - for "random" matrices H is not too loose
- Cramer's rule:
  - When $Ax = b$ then $x_i = \det A^{(i)} / \det A$
  - so $|\det A|$ is common denom for solution x

    (best common denom is largest Smith invariant)
- Cross-over: boundary of regions where algm A, B, C or ... is best

Background about CRT

CRT abbr. for *chinese remaindering*

Example: CRT method for determinant:

- For (distinct) primes $p_1, p_2, p_3, \ldots$ do
  Compute $D_{p_j} \leftarrow \det(\phi_{p_j}(A))$ where ϕ_{p_j} is reduction $\mathbb{Z} \to \mathbb{Z}/p_j\mathbb{Z}$
- Use CRT to recombine $D_{p_1}, D_{p_2}, D_{p_3}, \ldots$

Stopping criterion: Hadamard's bound, or "stability" (fast, but unverified)

Complexity problem: *quadratic* in size of $\|A\|_{max}$
[each $\phi_{p_k}(A)$ has cost linear in $\|A\|_{max}$ and num. iterations is linear in $\|A\|_{max}$]

Solution: reduce A modulo blocks of k primes together
$\longrightarrow$ *soft linear* complexity; needs more memory (use symmetric residues!)

Further complication: number of iterations depends on *stopping criterion*

Testing for unimodularity

Developed from *Pauderis+Storjohann* (2012, 2013)

Unimodular matrix $\longleftrightarrow$ $\det(A) = \pm 1$

Two approaches:

(1) compute $\det(A)$ *e.g.* by CRT

(2) double-plus-one lifting:

(compute A^{-1} via cunning quadratic p-adic approximation)

Cross-over: when to use which method?

- Quick & easy to estimate accurately cost of **(1)**
- Cannot quickly estimate accurately cost of **(2)**
  (but can estimate a range)

Testing for unimodularity (cont'd)

Testing on random unimodular matrices

Table of ratio of double-plus-one to determinant:

n	125 bits	250 bits	500 bits	1000 bits	2000 bits
25	0.58	0.51	0.46	0.44	0.36
50	0.32	0.31	0.32	0.30	0.29
100	0.16	0.15	0.16	0.15	0.16
200	0.08	0.08	0.08	0.08	0.07

Usually quick for non-unimodular: *e.g.* for 200×200 with 2000 bit entries

- $\sim 0.01s$ if not unimodular (typically)
- $\sim 4s$ if unimodular
- $\sim 50{-}60s$ to compute determinant (FLINT)

Q: But what is a "random unimodular matrix"?

Solving a linear system

Task: solve $Ax = b$ with A square, full-rank

Three approaches: (all are modular)

(1) solve via CRT
(2) solve via Dixon's method (linear p-adic approx; quadratic is no faster)
(3) double-plus-one lifting: compute $A^{-1}b$ via p-adic approx

Choice of stopping criterion: Hadamard or "stability"

Cross-over: when to use which method?

- (1) compute A^{-1} modulo each prime
- (2) compute A^{-1} modulo 1 prime; then matrix–vector prods
- (3) like (2) but cleverer/faster

Computing the determinant

CRT algorithm: Base reference method

ABM algorithm:

- Solve $Ax = b$ for $x \in \mathbb{Q}^n$ for random rhs $x \in \mathbb{Z}^n$ $\longleftarrow$ "fairly fast"
- Let d be least common denominator of x
- Compute $\det(A)/d$ using CRT

Works well when $|\det A|/d$ is small $\longleftrightarrow$ "generic SNF"

Refined by Pauderis+Storjohann:

- Use d and numerators of x to yield triang. matrix H
- Set $A_{new} \leftarrow AH^{-1} \in \mathrm{Mat}(\mathbb{Z})$, and iterate
- Result is $\det A = \det(A_{new}) \det(H)$

Better than ABM when SNF "a bit less generic"

Q: What to do when SNF "far from generic"?

Computing the determinant (cont'd)

New idea:

if d in ABM not too big, use P+S but with H being d-modular HNF

- Telescopes several P+S iterations into one step
- If d is max. Smith factor, we obtain $|\det A|$ directly
- If d not too big, computing d-modular HNF is quick
  (can also "guess" d from a few CRT iterations)

Cross-over: what is "not too big"?

Testing on "favourable" matrices

Table of ratio of new method to FLINT (standard ABM)

n	100 bits	200 bits	400 bits	800 bits
125	0.41	0.43	0.27	0.18
250	0.33	0.21	0.16	0.10
500	0.17	0.12	0.06	0.05
1000	0.14	0.14	0.06	0.03

Conclusion: det is much faster . . . . . (sometimes)

Summing it up

Max Horn

Much more to come

- OSCAR has picked up quite some momentum!

Much more to come

- OSCAR has picked up quite some momentum!
- More to come in this third and final period of the SFB

Much more to come

- OSCAR has picked up quite some momentum!
- More to come in this third and final period of the SFB
- . . . and beyond ($\rightsquigarrow$ discussion yesterday)

Much more to come

- OSCAR has picked up quite some momentum!
- More to come in this third and final period of the SFB
- . . . and beyond ($\rightsquigarrow$ discussion yesterday)
- We also need to "spread the word": OSCAR ambassadors, website overhaul, merchandise, documentation sprints, OSCAR teaching material and so on ($\rightsquigarrow$ the other discussion yesterday)

Much more to come

- OSCAR has picked up quite some momentum!
- More to come in this third and final period of the SFB
- . . . and beyond ($\rightsquigarrow$ discussion yesterday)
- We also need to "spread the word": OSCAR ambassadors, website overhaul, merchandise, documentation sprints, OSCAR teaching material and so on ($\rightsquigarrow$ the other discussion yesterday)
- We have much work ahead, not just technical


We need
YOU!

- We welcome collaborations, within the SFB and outside

- We welcome collaborations, within the SFB and outside
- Call us, email us, visit us, invite us – **talk to us here, today**

- We welcome collaborations, within the SFB and outside
- Call us, email us, visit us, invite us – **talk to us here, today**
- Need help? Try **Slack**: oscar-system.org/slack

- We welcome collaborations, within the SFB and outside
- Call us, email us, visit us, invite us – **talk to us here, today**
- Need help? Try **Slack**: oscar-system.org/slack
- We meet every Wednesday, 11:30 and Friday, 15:00 on Zoom, join us: oscar-system.org/zoom

When? September 15 - 19, 2025.

OSCAR Summer School 2025

When? September 15 - 19, 2025.

What? Learning to use and become better at using the OSCAR computer algebra system, for beginners and advanced users.

OSCAR Summer School 2025

When? September 15 - 19, 2025.

What? Learning to use and become better at using the OSCAR computer algebra system, for beginners and advanced users.

For whom? Primarily Master & PhD students, but just ask us

OSCAR Summer School 2025

When? September 15 - 19, 2025.

What? Learning to use and become better at using the OSCAR computer algebra system, for beginners and advanced users.

For whom? Primarily Master & PhD students, but just ask us

Where? Pfalzakademie Lambrecht, Rheinland-Pfalz

OSCAR Summer School 2025

When? September 15 - 19, 2025.

What? Learning to use and become better at using the OSCAR computer algebra system, for beginners and advanced users.

For whom? Primarily Master & PhD students, but just ask us

Where? Pfalzakademie Lambrecht, Rheinland-Pfalz

How? Register at oscar-system.org/school

OSCAR Summer School 2025

When? September 15 - 19, 2025.

What? Learning to use and become better at using the OSCAR computer algebra system, for beginners and advanced users.

For whom? Primarily Master & PhD students, but just ask us

Where? Pfalzakademie Lambrecht, Rheinland-Pfalz

How? Register at oscar-system.org/school

Thank you for your attention!