Three termination problems

Three termination problems

Patrick Dehornoy

Three termination problems

Patrick Dehornoy

Laboratoire de Mathématiques
Nicolas Oresme, Université de Caen

Three termination problems

Patrick Dehornoy

Laboratoire Preuves, Programmes, Systèmes
Université Paris-Diderot

Three termination problems

Patrick Dehornoy

Laboratoire Preuves, Programmes, Systèmes
Université Paris-Diderot

- Three unrelated termination problems :

Three termination problems

Patrick Dehornoy

Laboratoire Preuves, Programmes, Systèmes
Université Paris-Diderot

- Three unrelated termination problems : partial specific answers known,

Three termination problems

Patrick Dehornoy

Laboratoire Preuves, Programmes, Systèmes
Université Paris-Diderot

- Three unrelated termination problems : partial specific answers known,
  but no global understanding:

Three termination problems

Patrick Dehornoy

Laboratoire Preuves, Programmes, Systèmes
Université Paris-Diderot

- Three unrelated termination problems : partial specific answers known,
  but no global understanding: can some general tools be useful?

- Plan :

- Plan :

  1. The Polish Algorithm for Left-Selfdistributivity

- Plan :

  1. The Polish Algorithm for Left-Selfdistributivity
  2. Handle reduction of braids

- Plan :

    1. The Polish Algorithm for Left-Selfdistributivity
    2. Handle reduction of braids
    3. Subword reversing for positively presented groups

- Plan :
  1. The Polish Algorithm for Left-Selfdistributivity
  2. Handle reduction of braids
  3. Subword reversing for positively presented groups

1. The Polish Algorithm for Left-Selfdistributivity
2. Handle reduction of braids
3. Subword reversing for positively presented groups

- A "bi-term rewrite system"

- A "bi-term rewrite system" (????)

- A "bi-term rewrite system" (????)

- The associativity law

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)
  Example: In Polish, associativity is $x\,y\,z ** = x\,y * z\,*$.

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,

  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)

  Example: In Polish, associativity is $x\,y\,z * * = x\,y * z *$.

- Definition.— The Polish Algorithm for A:

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)
  Example: In Polish, associativity is $x\,y\,z * * = x\,y * z *$.

- Definition.— The Polish Algorithm for A: starting with two terms $\boldsymbol{t}, \boldsymbol{t}'$ (in Polish):

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: t, t' are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)
  Example: In Polish, associativity is $x\, y\, z * * = x\, y * z *$.

- Definition.— The Polish Algorithm for A: starting with two terms $\boldsymbol{t}, \boldsymbol{t}'$ (in Polish):
  - while $\boldsymbol{t} \neq \boldsymbol{t}'$ do
    - $\boldsymbol{p} :=$ first clash between $\boldsymbol{t}$ and $\boldsymbol{t}'$ (pth letter of $\boldsymbol{t} \neq p$th letter of $\boldsymbol{t}'$)

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)
  Example: In Polish, associativity is $x\,y\,z * * = x\,y * z *$.

- Definition.— The Polish Algorithm for A: starting with two terms $\boldsymbol{t}, \boldsymbol{t}'$ (in Polish):
  - while $\boldsymbol{t} \neq \boldsymbol{t}'$ do
    - $\boldsymbol{p} :=$ first clash between $\boldsymbol{t}$ and $\boldsymbol{t}'$ (pth letter of $\boldsymbol{t} \neq p$th letter of $\boldsymbol{t}'$)
    - case type of $\boldsymbol{p}$ of
      - "variable *vs.* blank" : return NO;

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)
  Example: In Polish, associativity is $x\,y\,z * * = x\,y * z *$.

- Definition.— The Polish Algorithm for A: starting with two terms $\boldsymbol{t}, \boldsymbol{t}'$ (in Polish):
  - while $\boldsymbol{t} \neq \boldsymbol{t}'$ do
    - $\boldsymbol{p} :=$ first clash between $\boldsymbol{t}$ and $\boldsymbol{t}'$ (pth letter of $\boldsymbol{t} \neq p$th letter of $\boldsymbol{t}'$)
    - case type of $\boldsymbol{p}$ of
      - "variable *vs.* blank" : return NO;
      - "blank *vs.* variable" : return NO;

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)
  Example: In Polish, associativity is $x\,y\,z * * = x\,y * z *$.

- Definition.— The Polish Algorithm for A: starting with two terms $\boldsymbol{t}, \boldsymbol{t}'$ (in Polish):
  - while $\boldsymbol{t} \neq \boldsymbol{t}'$ do
    - $\boldsymbol{p} :=$ first clash between $\boldsymbol{t}$ and $\boldsymbol{t}'$ (pth letter of $\boldsymbol{t} \neq p$th letter of $\boldsymbol{t}'$)
    - case type of $\boldsymbol{p}$ of
      - "variable *vs.* blank" : return NO;
      - "blank *vs.* variable" : return NO;
      - "variable *vs.* variable" : return NO;

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,

  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: $\boldsymbol{t}, \boldsymbol{t}'$ are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)

  Example: In Polish, associativity is $x\, y\, z * * = x\, y * z *$.

- Definition.— The Polish Algorithm for A: starting with two terms $\boldsymbol{t}, \boldsymbol{t}'$ (in Polish):
  - while $\boldsymbol{t} \neq \boldsymbol{t}'$ do
    - $\boldsymbol{p} :=$ first clash between $\boldsymbol{t}$ and $\boldsymbol{t}'$ (pth letter of $\boldsymbol{t} \neq p$th letter of $\boldsymbol{t}'$)
    - case type of $\boldsymbol{p}$ of
      - "variable *vs.* blank" : return NO;
      - "blank *vs.* variable" : return NO;
      - "variable *vs.* variable" : return NO;
      - "variable *vs.* $*$" : apply A^+ to $\boldsymbol{t}$; $(\boldsymbol{t}_1 \boldsymbol{t}_2 \boldsymbol{t}_3 * * \to \boldsymbol{t}_1 \boldsymbol{t}_2 * \boldsymbol{t}_3 *)$

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: t, t' are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)
  Example: In Polish, associativity is $x\, y\, z * * = x\, y * z\, *$.

- Definition.— The Polish Algorithm for A: starting with two terms $\boldsymbol{t}, \boldsymbol{t}'$ (in Polish):
  - while $\boldsymbol{t} \neq \boldsymbol{t}'$ do
    - $\boldsymbol{p} :=$ first clash between $\boldsymbol{t}$ and $\boldsymbol{t}'$ (pth letter of $\boldsymbol{t} \neq p$th letter of $\boldsymbol{t}'$)
    - case type of $\boldsymbol{p}$ of
      - "variable *vs.* blank" : return NO;
      - "blank *vs.* variable" : return NO;
      - "variable *vs.* variable" : return NO;
      - "variable *vs.* $*$" : apply A^+ to $\boldsymbol{t}$; $(t_1 t_2 \boldsymbol{t}_3 * * \to t_1 t_2 * t_3 *)$
      - "$*$ *vs.* variable" : apply A^+ to $\boldsymbol{t}'$; $(t_1 t_2 \boldsymbol{t}_3 * * \to t_1 t_2 * t_3 *)$

- A "bi-term rewrite system" (????)

- The associativity law $(\boldsymbol{A})$: $x * (y * z) = (x * y) * z$,
  ... and the corresponding Word Problem:

  Given two terms $\boldsymbol{t}, \boldsymbol{t}'$, decide whether $\boldsymbol{t}$ and $\boldsymbol{t}'$ are $\boldsymbol{A}$-equivalent.

- A trivial problem: t, t' are A-equivalent iff become equal when brackets are removed.

- (Right-) Polish expression of a term: "$\boldsymbol{t}_1 \boldsymbol{t}_2 *$" for $\boldsymbol{t}_1 * \boldsymbol{t}_2$ (no bracket needed)
  Example: In Polish, associativity is $x\,y\,z * * = x\,y * z *$.

- Definition.— The Polish Algorithm for A: starting with two terms $\boldsymbol{t}, \boldsymbol{t}'$ (in Polish):
  - while $\boldsymbol{t} \neq \boldsymbol{t}'$ do
    - $\boldsymbol{p} :=$ first clash between $\boldsymbol{t}$ and $\boldsymbol{t}'$ (pth letter of $\boldsymbol{t} \neq p$th letter of $\boldsymbol{t}'$)
    - case type of $\boldsymbol{p}$ of
      - "variable *vs.* blank" : return NO;
      - "blank *vs.* variable" : return NO;
      - "variable *vs.* variable" : return NO;
      - "variable *vs.* $*$" : apply A^+ to $\boldsymbol{t}$; $(t_1 t_2 \boldsymbol{t}_3 * * \rightarrow t_1 t_2 * t_3 *)$
      - "$*$ *vs.* variable" : apply A^+ to $\boldsymbol{t}'$; $(t_1 t_2 \boldsymbol{t}_3 * * \rightarrow t_1 t_2 * t_3 *)$
  - return YES.

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z \ast \ast \\ x\,y \ast z \ast \end{cases}$.

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z * * \\ x\,y * z * \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$,

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z**\\ x\,y*z* \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\boldsymbol{t}_0 = x\,x\,x\,x***$$
$$\boldsymbol{t}'_0 = x\,x*x*x*$$

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z * * \\ x\,y * z * \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x\,x * * * \\ \boldsymbol{t}'_0 &= x\,x * x * x * \end{aligned}$$

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z**\\ x\,y*z* \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x\,x*** \\ \boldsymbol{t}'_0 &= x\,x*x*x* \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_1 &= x\,x*x\,x** \\ \boldsymbol{t}'_1 &= x\,x*x*x* \end{aligned}$$

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z * * \\ x\,y * z * \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x\,x * * * \\ \boldsymbol{t}'_0 &= x\,x * x * x * \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_1 &= x\,x * x\,x * * \\ \boldsymbol{t}'_1 &= x\,x * x * x * \end{aligned}$$

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z * * \\ x\,y * z * \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\boldsymbol{t}_0 = x\,x\,x\,x * * *$$
$$\boldsymbol{t}'_0 = x\,x * x * x *$$

$$\boldsymbol{t}_1 = x\,x * x\,x * *$$
$$\boldsymbol{t}'_1 = x\,x * x * x *$$

$$\boldsymbol{t}_2 = x\,x * x * x *$$
$$\boldsymbol{t}'_2 = x\,x * x * x *$$

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z*\,* \\ x\,y*z\,* \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x\,x*\,*\,* \\ \boldsymbol{t}'_0 &= x\,x*x*x* \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_1 &= x\,x*x\,x*\,* \\ \boldsymbol{t}'_1 &= x\,x*x*x* \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_2 &= x\,x*x*x* \\ \boldsymbol{t}'_2 &= x\,x*x*x* \end{aligned}$$

So $\boldsymbol{t}_2 = \boldsymbol{t}'_2$, hence $\boldsymbol{t}_0$ and $\boldsymbol{t}'_0$ are A-equivalent.

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z **\\ x\,y * z * \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x\,x*** \\ \boldsymbol{t}'_0 &= x\,x*x*x* \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_1 &= x\,x*x\,x** \\ \boldsymbol{t}'_1 &= x\,x*x*x* \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_2 &= x\,x*x*x* \\ \boldsymbol{t}'_2 &= x\,x*x*x* \end{aligned}$$

So $\boldsymbol{t}_2 = \boldsymbol{t}'_2$, hence $\boldsymbol{t}_0$ and $\boldsymbol{t}'_0$ are A-equivalent.

- "Theorem".—

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z**\\ x\,y*z*\end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\begin{aligned}\boldsymbol{t}_0 &= x\,x\,x\,x***\\ \boldsymbol{t}'_0 &= x\,x*x*x*\end{aligned}$$

$$\begin{aligned}\boldsymbol{t}_1 &= x\,x*x\,x**\\ \boldsymbol{t}'_1 &= x\,x*x*x*\end{aligned}$$

$$\begin{aligned}\boldsymbol{t}_2 &= x\,x*x*x*\\ \boldsymbol{t}'_2 &= x\,x*x*x*\end{aligned}$$

So $\boldsymbol{t}_2 = \boldsymbol{t}'_2$, hence $\boldsymbol{t}_0$ and $\boldsymbol{t}'_0$ are A-equivalent.

- "Theorem".— The Polish Algorithm works for associativity.

- Remember : in Polish, associativity is $\begin{cases} x\,y\,z * * \\ x\,y * z * \end{cases}$.

- Example: $\boldsymbol{t} = x*(x*(x*x))$, $\boldsymbol{t}' = ((x*x)*x)*x$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x\,x * * * \\ \boldsymbol{t}'_0 &= x\,x * x * x * \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_1 &= x\,x * x\,x * * \\ \boldsymbol{t}'_1 &= x\,x * x * x * \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_2 &= x\,x * x * x * \\ \boldsymbol{t}'_2 &= x\,x * x * x * \end{aligned}$$

So $\boldsymbol{t}_2 = \boldsymbol{t}'_2$, hence $\boldsymbol{t}_0$ and $\boldsymbol{t}'_0$ are A-equivalent.

- "Theorem".— The Polish Algorithm works for associativity.
(In particular, it terminates.)

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,z*\,* \end{cases}$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,z*\,* \end{cases}$ compare with associativity $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,z*\,* \end{cases}$ compare with associativity $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$,

- Left-selfdistributivity (LD) : $x * (y * z) = (x * y) * (x * z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z * * \\ x\,y * x\,z * * \end{cases}$ compare with associativity $\begin{cases} x\,y\,z * * \\ x\,y * x * \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x * ((x * x) * (x * x))$, $\boldsymbol{t'} = (x * x) * (x * (x * x))$, i.e., in Polish,

  $\boldsymbol{t}_0 = x\,x\,x * x\,x * * *$
  $\boldsymbol{t}'_0 = x\,x * x\,x\,x * * *$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,z*\,* \end{cases}$ compare with associativity $\begin{cases} x\,y\,z*\,* \\ x\,y*x* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

  $\boldsymbol{t}_0 = x\,x\,x*x\,x***$
  $\boldsymbol{t}'_0 = x\,x*x\,x\,x***$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,z*\,* \end{cases}$ compare with associativity $\begin{cases} x\,y\,z*\,* \\ x\,y*x* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

$$\boldsymbol{t}_0 = x\,x\,x*x\,x***$$
$$\boldsymbol{t}'_0 = x\,x*x\,x\,x***$$

$$\boldsymbol{t}_1 = x\,x*x\,x**x\,x\,x***$$
$$\boldsymbol{t}'_1 = x\,x*x\,x\,x*** \qquad (= \boldsymbol{t}'_0)$$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z**\\ x\,y*x\,z** \end{cases}$ compare with associativity $\begin{cases} x\,y\,z**\\ x\,y*x* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x*x\,x*** \\ \boldsymbol{t}'_0 &= x\,x*x\,x\,x*** \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_1 &= x\,x*x\,x**x\,x\,x*** \\ \boldsymbol{t}'_1 &= x\,x*x\,x\,x*** \qquad (= \boldsymbol{t}'_0) \end{aligned}$$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z**\\ x\,y*x\,z** \end{cases}$ compare with associativity $\begin{cases} x\,y\,z**\\ x\,y*x* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x*x\,x*** \\ \boldsymbol{t}'_0 &= x\,x*x\,x\,x*** \\[1ex] \boldsymbol{t}_1 &= x\,x*x\,x**x\,x\,x*** \\ \boldsymbol{t}'_1 &= x\,x*x\,x\,x*** && (= \boldsymbol{t}'_0) \\[1ex] \boldsymbol{t}_2 &= x\,x*x\,x**x\,x\,x*** && (= \boldsymbol{t}_1) \\ \boldsymbol{t}'_2 &= x\,x*x\,x*x\,x** \end{aligned}$$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z**\\ x\,y*x\,z** \end{cases}$ compare with associativity $\begin{cases} x\,y\,z**\\ x\,y*x* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

$$\begin{aligned} \boldsymbol{t}_0 &= x\,x\,x*x\,x*** \\ \boldsymbol{t}'_0 &= x\,x*x\,x\,x*** \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_1 &= x\,x*x\,x**x\,x\,x*** \\ \boldsymbol{t}'_1 &= x\,x*x\,x\,x*** && (= \boldsymbol{t}'_0) \end{aligned}$$

$$\begin{aligned} \boldsymbol{t}_2 &= x\,x*x\,x**x\,x\,x*** && (= \boldsymbol{t}_1) \\ \boldsymbol{t}'_2 &= x\,x*x\,x*x\,x** \end{aligned}$$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,z*\,* \end{cases}$ compare with associativity $\begin{cases} x\,y\,z*\,* \\ x\,y*x\,* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

$$\begin{aligned}
\boldsymbol{t}_0 &= x\,x\,x*x\,x*\,*\,* \\
\boldsymbol{t}'_0 &= x\,x*x\,x\,x*\,*\,* \\[1ex]
\boldsymbol{t}_1 &= x\,x*x\,x*\,*x\,x\,x*\,*\,* \\
\boldsymbol{t}'_1 &= x\,x*x\,x\,x*\,*\,* && (= \boldsymbol{t}'_0) \\[1ex]
\boldsymbol{t}_2 &= x\,x*x\,x*\,*x\,x\,x*\,*\,* && (= \boldsymbol{t}_1) \\
\boldsymbol{t}'_2 &= x\,x*x\,x*x\,x*\,* \\[1ex]
\boldsymbol{t}_3 &= x\,x*x\,x*\,*x\,x\,x*\,*\,* && (= \boldsymbol{t}_2) \\
\boldsymbol{t}'_3 &= x\,x*x\,x*\,*x\,x*x\,x*\,*\,*
\end{aligned}$$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z**\\ x\,y*x\,z** \end{cases}$ compare with associativity $\begin{cases} x\,y\,z**\\ x\,y*x* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

$$\begin{aligned}
\boldsymbol{t}_0 &= xxx*xx*** \\
\boldsymbol{t}'_0 &= xx*xxx*** \\[1ex]
\boldsymbol{t}_1 &= xx*xx**xxx*** \\
\boldsymbol{t}'_1 &= xx*xxx*** && (= \boldsymbol{t}'_0) \\[1ex]
\boldsymbol{t}_2 &= xx*xx**xxx*** && (= \boldsymbol{t}_1) \\
\boldsymbol{t}'_2 &= xx*xx*xx** \\[1ex]
\boldsymbol{t}_3 &= xx*xx**xxx*** && (= \boldsymbol{t}_2) \\
\boldsymbol{t}'_3 &= xx*xx**xx*xx***
\end{aligned}$$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z**\\ x\,y*x\,z** \end{cases}$ compare with associativity $\begin{cases} x\,y\,z**\\ x\,y*x* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

$$
\begin{aligned}
\boldsymbol{t}_0 &= x\,x\,x*x\,x***\\
\boldsymbol{t}'_0 &= x\,x*x\,x\,x***\\[1ex]
\boldsymbol{t}_1 &= x\,x*x\,x**x\,x\,x***\\
\boldsymbol{t}'_1 &= x\,x*x\,x\,x*** && (= \boldsymbol{t}'_0)\\[1ex]
\boldsymbol{t}_2 &= x\,x*x\,x**x\,x\,x*** && (= \boldsymbol{t}_1)\\
\boldsymbol{t}'_2 &= x\,x*x\,x*x\,x**\\[1ex]
\boldsymbol{t}_3 &= x\,x*x\,x**x\,x\,x*** && (= \boldsymbol{t}_2)\\
\boldsymbol{t}'_3 &= x\,x*x\,x**x\,x*x\,x***\\[1ex]
\boldsymbol{t}_4 &= x\,x*x\,x**x\,x*x\,x***\\
\boldsymbol{t}'_4 &= x\,x*x\,x**x\,x*x\,x*** && (= \boldsymbol{t}'_3)
\end{aligned}
$$

- Left-selfdistributivity (LD) : $x*(y*z) = (x*y)*(x*z)$,

  i.e., in Polish, $\begin{cases} x\,y\,z**\\ x\,y*x\,z** \end{cases}$ compare with associativity $\begin{cases} x\,y\,z**\\ x\,y*x* \end{cases}$

- Polish Algorithm: the same as for associativity.

- Example: $\boldsymbol{t} = x*((x*x)*(x*x))$, $\boldsymbol{t}' = (x*x)*(x*(x*x))$, i.e., in Polish,

$$\begin{aligned}
\boldsymbol{t}_0 &= xxx*xx***\\
\boldsymbol{t}'_0 &= xx*xxx***\\[1ex]
\boldsymbol{t}_1 &= xx*xx**xxx***\\
\boldsymbol{t}'_1 &= xx*xxx*** && (= \boldsymbol{t}'_0)\\[1ex]
\boldsymbol{t}_2 &= xx*xx**xxx*** && (= \boldsymbol{t}_1)\\
\boldsymbol{t}'_2 &= xx*xx*xx**\\[1ex]
\boldsymbol{t}_3 &= xx*xx**xxx*** && (= \boldsymbol{t}_2)\\
\boldsymbol{t}'_3 &= xx*xx**xx*xx***\\[1ex]
\boldsymbol{t}_4 &= xx*xx**xx*xx***\\
\boldsymbol{t}'_4 &= xx*xx**xx*xx*** && (= \boldsymbol{t}'_3)
\end{aligned}$$

So $\boldsymbol{t}_4 = \boldsymbol{t}'_4$, hence $\boldsymbol{t}_0$ and $\boldsymbol{t}'_0$ are LD-equivalent.

- Conjecture.— The Polish Algorithm works for left-selfdistributivity.

- Conjecture.— The Polish Algorithm works for left-selfdistributivity.

- Known.— (i) If it terminates, the Polish Algorithm works for left-selfdistributivity.

- Conjecture.— The Polish Algorithm works for left-selfdistributivity.

- Known.— (i) If it terminates, the Polish Algorithm works for left-selfdistributivity.
  (ii) The smallest counter-example to termination (if any) is huge.

1. The Polish Algorithm for Left-Selfdistributivity
2. Handle reduction of braids
3. Subword reversing for positively presented groups

- A true (but infinite) rewrite system.

- A true (but infinite) rewrite system.
- Alphabet: `a`, `b`, `A`, `B`

- A true (but infinite) rewrite system.
- Alphabet: `a`, `b`, `A`, `B` (think of `A` as an inverse of `a`, etc.)

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of A as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$,

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$,

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,

  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$,

- A true (but infinite) rewrite system.

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)

- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.

- A true (but infinite) rewrite system.

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)

- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.
- Example: $\boldsymbol{w}_0 = \mathtt{aabAbbAA}$

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

  and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.
- Example: $$w_0 = \mathtt{a}\underline{\mathtt{abA}}\mathtt{bbAA}$$

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

  and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.
- Example:

$$\begin{aligned} \boldsymbol{w}_0 &= \mathtt{a}\underline{\mathtt{abA}}\mathtt{bbAA} \\ \boldsymbol{w}_1 &= \mathtt{aBabbbAA} \end{aligned}$$

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.
- Example:

$$\begin{aligned} \boldsymbol{w}_0 &= \mathtt{a}\underline{\mathtt{abA}}\mathtt{bbAA} \\ \boldsymbol{w}_1 &= \mathtt{aB}\underline{\mathtt{abbbA}}\mathtt{A} \end{aligned}$$

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.
- Example:

$$\begin{aligned} \boldsymbol{w}_0 &= \mathtt{a}\underline{\mathtt{abA}}\mathtt{bbAA} \\ \boldsymbol{w}_1 &= \mathtt{aB}\underline{\mathtt{abbbA}}\mathtt{A} \\ \boldsymbol{w}_2 &= \mathtt{aBBaaabA} \end{aligned}$$

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

  and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.
- Example:

$$\begin{aligned} \boldsymbol{w}_0 &= \mathtt{a}\underline{\mathtt{abA}}\mathtt{bbAA} \\ \boldsymbol{w}_1 &= \mathtt{aB}\underline{\mathtt{abbbA}}\mathtt{A} \\ \boldsymbol{w}_2 &= \mathtt{aBBaa}\underline{\mathtt{abA}} \end{aligned}$$

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.
- Example:

$$\begin{aligned}
\boldsymbol{w}_0 &= \mathtt{a}\underline{\mathtt{abA}}\mathtt{bbAA}\\
\boldsymbol{w}_1 &= \mathtt{aB}\underline{\mathtt{abbbA}}\mathtt{A}\\
\boldsymbol{w}_2 &= \mathtt{aBBaa}\underline{\mathtt{abA}}\\
\boldsymbol{w}_3 &= \mathtt{aBBaaBab},
\end{aligned}$$

- A true (but infinite) rewrite system.
- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (so far trivial: "free group reduction")
  - $\mathtt{abA} \longrightarrow \mathtt{Bab}$, $\mathtt{aBA} \longrightarrow \mathtt{BAb}$, $\mathtt{Aba} \longrightarrow \mathtt{baB}$, $\mathtt{ABa} \longrightarrow \mathtt{bAB}$,

and, more generally,
  - $\mathtt{ab}^i\mathtt{A} \longrightarrow \mathtt{Ba}^i\mathtt{b}$, $\mathtt{aB}^i\mathtt{A} \longrightarrow \mathtt{BA}^i\mathtt{b}$, $\mathtt{Ab}^i\mathtt{a} \longrightarrow \mathtt{ba}^i\mathtt{B}$, $\mathtt{AB}^i\mathtt{a} \longrightarrow \mathtt{bA}^i\mathtt{B}$ for $i \geqslant 1$.
- Aim: obtain a word that does not contain both $\mathtt{a}$ and $\mathtt{A}$.
- Example:

$$\begin{aligned} w_0 &= \mathtt{a}\underline{\mathtt{abA}}\mathtt{bbAA} \\ w_1 &= \mathtt{aB}\underline{\mathtt{abbbA}}\mathtt{A} \\ w_2 &= \mathtt{aBBaa}\underline{\mathtt{abA}} \\ w_3 &= \mathtt{aBBaaBab}, \end{aligned} \qquad \rightsquigarrow \text{a word without } \mathtt{A}$$

- **Theorem.**— The process terminates in quadratic time.

- Theorem.— The process terminates in quadratic time.

- Proof: (Length does not increase, but could cycle.)

• Theorem.— The process terminates in quadratic time.

• Proof: (Length does not increase, but could cycle.)
Associate with the sequence of reductions a rectangular grid (quadratic area).

• Theorem.— The process terminates in quadratic time.

• Proof: (Length does not increase, but could cycle.)
Associate with the sequence of reductions a rectangular grid (quadratic area).

For the example:

$$\begin{aligned} \boldsymbol{w}_0 &= \texttt{aabAbbAA} \\ \boldsymbol{w}_1 &= \texttt{aBabbbAA} \\ \boldsymbol{w}_2 &= \texttt{aBBaaabA} \\ \boldsymbol{w}_3 &= \texttt{aBBaaBab} \end{aligned}$$

draw the grid:

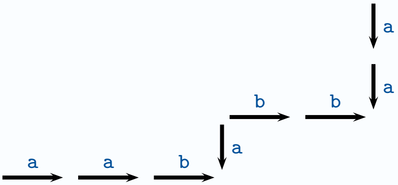
- **Theorem.—** The process terminates in quadratic time.

- Proof: (Length does not increase, but could cycle.)
Associate with the sequence of reductions a rectangular grid (quadratic area).

For the example:

$$\begin{aligned} w_0 &= \texttt{aabAbbAA} \\ w_1 &= \texttt{aBabbbAA} \\ w_2 &= \texttt{aBBaaabA} \\ w_3 &= \texttt{aBBaaBab} \end{aligned}$$

draw the grid:



- **Theorem.—** The process terminates in quadratic time.

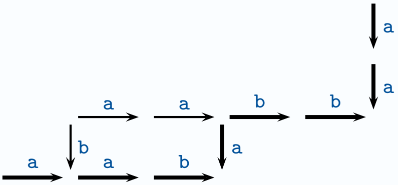
- Proof: (Length does not increase, but could cycle.)

Associate with the sequence of reductions a rectangular grid (quadratic area).

For the example:

$$\begin{aligned} \boldsymbol{w}_0 &= \texttt{aabAbbAA} \\ \boldsymbol{w}_1 &= \texttt{aBabbbAA} \\ \boldsymbol{w}_2 &= \texttt{aBBaaabA} \\ \boldsymbol{w}_3 &= \texttt{aBBaaBab} \end{aligned}$$

draw the grid:



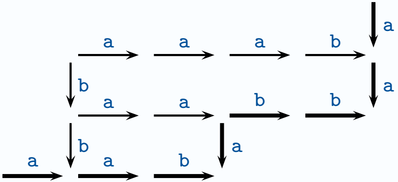
- Theorem.— The process terminates in quadratic time.

- Proof: (Length does not increase, but could cycle.)
Associate with the sequence of reductions a rectangular grid (quadratic area).

For the example:

$$\begin{aligned} \boldsymbol{w}_0 &= \texttt{aabAbbAA} \\ \boldsymbol{w}_1 &= \texttt{aBabbbAA} \\ \boldsymbol{w}_2 &= \texttt{aBBaaabA} \\ \boldsymbol{w}_3 &= \texttt{aBBaaBab} \end{aligned}$$

draw the grid:



- Theorem.— The process terminates in quadratic time.

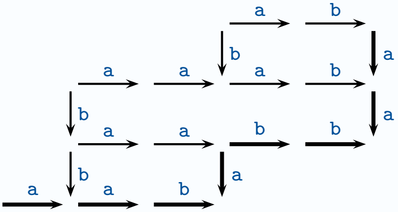
- Proof: (Length does not increase, but could cycle.)

Associate with the sequence of reductions a rectangular grid (quadratic area).

For the example:

$$\begin{aligned} w_0 &= \texttt{aabAbbAA} \\ w_1 &= \texttt{aBabbbAA} \\ w_2 &= \texttt{aBBaaabA} \\ w_3 &= \texttt{aBBaaBab} \end{aligned}$$

draw the grid:



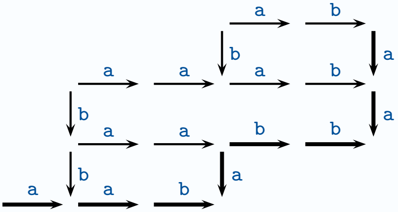
• **Theorem.—** The process terminates in quadratic time.

• Proof: (Length does not increase, but could cycle.)
Associate with the sequence of reductions a rectangular grid (quadratic area).

For the example:

$$
\begin{aligned}
\boldsymbol{w}_0 &= \texttt{aabAbbAA}\\
\boldsymbol{w}_1 &= \texttt{aBabbbAA}\\
\boldsymbol{w}_2 &= \texttt{aBBaaabA}\\
\boldsymbol{w}_3 &= \texttt{aBBaaBab}
\end{aligned}
$$

draw the grid:



□

- This is the braid handle reduction procedure;

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$,

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$, (as above)

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$, (as above)
  - for w in $\{\mathtt{b}, \mathtt{c}, \mathtt{C}\}^*$ or $\{\mathtt{B}, \mathtt{c}, \mathtt{C}\}^*$:

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$, (as above)
  - for w in $\{\mathtt{b}, \mathtt{c}, \mathtt{C}\}^*$ or $\{\mathtt{B}, \mathtt{c}, \mathtt{C}\}^*$: $\mathtt{a}w\mathtt{A} \longrightarrow \phi_{\mathtt{a}}(w)$,

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$, (as above)
  - for w in $\{\mathtt{b}, \mathtt{c}, \mathtt{C}\}^*$ or $\{\mathtt{B}, \mathtt{c}, \mathtt{C}\}^*$: $\mathtt{a}w\mathtt{A} \longrightarrow \phi_{\mathtt{a}}(w)$, $\mathtt{A}w\mathtt{a} \longrightarrow \phi_{\mathtt{A}}(w)$,

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \to \varepsilon$, $\mathtt{Aa} \to \varepsilon$, $\mathtt{bB} \to \varepsilon$, $\mathtt{Bb} \to \varepsilon$, $\mathtt{cC} \to \varepsilon$, $\mathtt{Cc} \to \varepsilon$, (as above)
  - for w in $\{\mathtt{b}, \mathtt{c}, \mathtt{C}\}^*$ or $\{\mathtt{B}, \mathtt{c}, \mathtt{C}\}^*$: $\mathtt{a}w\mathtt{A} \to \phi_{\mathtt{a}}(w)$, $\mathtt{A}w\mathtt{a} \to \phi_{\mathtt{A}}(w)$,
    with $\phi_{\mathtt{a}}(w)$ obtained from w by $\mathtt{b} \to \mathtt{Bab}$ and $\mathtt{B} \to \mathtt{BAb}$,

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$, (as above)
  - for w in $\{\mathtt{b}, \mathtt{c}, \mathtt{C}\}^*$ or $\{\mathtt{B}, \mathtt{c}, \mathtt{C}\}^*$: $\mathtt{a}w\mathtt{A} \longrightarrow \phi_{\mathtt{a}}(w)$, $\mathtt{A}w\mathtt{a} \longrightarrow \phi_{\mathtt{A}}(w)$,
    with $\phi_{\mathtt{a}}(w)$ obtained from w by $\mathtt{b} \longrightarrow \mathtt{Bab}$ and $\mathtt{B} \longrightarrow \mathtt{BAb}$,
    and $\phi_{\mathtt{A}}(w)$ obtained from w by $\mathtt{b} \longrightarrow \mathtt{baB}$ and $\mathtt{B} \longrightarrow \mathtt{bAB}$,

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$, (as above)
  - for w in $\{\mathtt{b}, \mathtt{c}, \mathtt{C}\}^*$ or $\{\mathtt{B}, \mathtt{c}, \mathtt{C}\}^*$: $\mathtt{a}w\mathtt{A} \longrightarrow \phi_{\mathtt{a}}(w)$, $\mathtt{A}w\mathtt{a} \longrightarrow \phi_{\mathtt{A}}(w)$,
    with $\phi_{\mathtt{a}}(w)$ obtained from w by $\mathtt{b} \longrightarrow \mathtt{Bab}$ and $\mathtt{B} \longrightarrow \mathtt{BAb}$,
    and $\phi_{\mathtt{A}}(w)$ obtained from w by $\mathtt{b} \longrightarrow \mathtt{baB}$ and $\mathtt{B} \longrightarrow \mathtt{bAB}$,
  - for w in $\{\mathtt{c}\}^*$ or $\{\mathtt{C}\}^*$: $\mathtt{b}w\mathtt{B} \longrightarrow \phi_{\mathtt{b}}(w)$, $\mathtt{B}w\mathtt{b} \longrightarrow \phi_{\mathtt{B}}(w)$,

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$, (as above)
  - for w in $\{\mathtt{b}, \mathtt{c}, \mathtt{C}\}^*$ or $\{\mathtt{B}, \mathtt{c}, \mathtt{C}\}^*$: $\mathtt{a}w\mathtt{A} \longrightarrow \phi_{\mathtt{a}}(w)$, $\mathtt{A}w\mathtt{a} \longrightarrow \phi_{\mathtt{A}}(w)$,
    with $\phi_{\mathtt{a}}(w)$ obtained from w by $\mathtt{b} \longrightarrow \mathtt{Bab}$ and $\mathtt{B} \longrightarrow \mathtt{BAb}$,
    and $\phi_{\mathtt{A}}(w)$ obtained from w by $\mathtt{b} \longrightarrow \mathtt{baB}$ and $\mathtt{B} \longrightarrow \mathtt{bAB}$,
  - for w in $\{\mathtt{c}\}^*$ or $\{\mathtt{C}\}^*$: $\mathtt{b}w\mathtt{B} \longrightarrow \phi_{\mathtt{b}}(w)$, $\mathtt{B}w\mathtt{b} \longrightarrow \phi_{\mathtt{B}}(w)$,
    with $\phi_{\mathtt{b}}(w)$ obtained from w by $\mathtt{c} \longrightarrow \mathtt{Cbc}$ and $\mathtt{C} \longrightarrow \mathtt{CBc}$,
    and $\phi_{\mathtt{B}}(w)$ obtained from w by $\mathtt{c} \longrightarrow \mathtt{cbC}$ and $\mathtt{C} \longrightarrow \mathtt{cBC}$.

- This is the braid handle reduction procedure;
  so far: case of "3-strand" braids; now: case of "4-strand" braids
  (case of "n strand" braids entirely similar for every n).

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{A}, \mathtt{B}, \mathtt{C}$.
- Rewrite rules:
  - $\mathtt{aA} \longrightarrow \varepsilon$, $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{bB} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, $\mathtt{cC} \longrightarrow \varepsilon$, $\mathtt{Cc} \longrightarrow \varepsilon$, (as above)
  - for w in $\{\mathtt{b}, \mathtt{c}, \mathtt{C}\}^*$ or $\{\mathtt{B}, \mathtt{c}, \mathtt{C}\}^*$: $\mathtt{a}w\mathtt{A} \longrightarrow \phi_{\mathtt{a}}(w)$, $\mathtt{A}w\mathtt{a} \longrightarrow \phi_{\mathtt{A}}(w)$,
    with $\phi_{\mathtt{a}}(w)$ obtained from w by $\mathtt{b} \longrightarrow \mathtt{Bab}$ and $\mathtt{B} \longrightarrow \mathtt{BAb}$,
    and $\phi_{\mathtt{A}}(w)$ obtained from w by $\mathtt{b} \longrightarrow \mathtt{baB}$ and $\mathtt{B} \longrightarrow \mathtt{bAB}$,
  - for w in $\{\mathtt{c}\}^*$ or $\{\mathtt{C}\}^*$: $\mathtt{b}w\mathtt{B} \longrightarrow \phi_{\mathtt{b}}(w)$, $\mathtt{B}w\mathtt{b} \longrightarrow \phi_{\mathtt{B}}(w)$,
    with $\phi_{\mathtt{b}}(w)$ obtained from w by $\mathtt{c} \longrightarrow \mathtt{Cbc}$ and $\mathtt{C} \longrightarrow \mathtt{CBc}$,
    and $\phi_{\mathtt{B}}(w)$ obtained from w by $\mathtt{c} \longrightarrow \mathtt{cbC}$ and $\mathtt{C} \longrightarrow \mathtt{cBC}$.

- Remark.— $\mathtt{ab}^i\mathtt{A} \longrightarrow (\mathtt{Bab})^i \longrightarrow \mathtt{Ba}^i\mathtt{b}$: extends the 3-strand case.

- Example:

- Example:

  `abcbABABCBA`

- Example:

    `abcbABABCBA`

- Example:

  ```
  abcbABABCBA
  BabcBabBABCBA
  ```

- Example:

  abcbABABCBA
  BabcBabBABCBA
  BabcBaABCBA

- Example:

  `abcbABABCBA`
  `BabcBabBABCBA`
  `BabcBaABCBA`
  `BabcBBCBA`

- Example:

  `abcbABABCBA`
  `BabcBabBABCBA`
  `BabcBaABCBA`
  `BabcBBCBA`
  `BaCbcBCBA`

- Example:

  ```
  abcbABABCBA
  BabcBabBABCBA
  BabcBaABCBA
  BabcBBCBA
  BaCbcBCBA
  BaCCbcCBA
  ```

- Example:

  abcbABABCBA
  BabcBabBABCBA
  BabcBaABCBA
  BabcBBCBA
  BaCbcBCBA
  BaCCbcCBA
  BaCCbBA

- Example:

  abcbABABCBA
  BabcBabBABCBA
  BabcBaABCBA
  BabcBBCBA
  BaCbcBCBA
  BaCCbcCBA
  BaCCbBA
  BaCCA

- Example:

  abcbABABCBA
  BabcBabBABCBA
  BabcBaABCBA
  BabcBBCBA
  BaCbcBCBA
  BaCCbcCBA
  BaCCbBA
  BaCCA
  BCC

- Example:

  `abcbABABCBA`
  `BabcBabBABCBA`
  `BabcBaABCBA`
  `BabcBBCBA`
  `BaCbcBCBA`
  `BaCCbcCBA`
  `BaCCbBA`
  `BaCCA`
  `BCC`

⇝ Terminates: the final word does not contain both `a` and `A`

- Example:

  `abcbABABCBA`
  `BabcBabBABCBA`
  `BabcBaABCBA`
  `BabcBBCBA`
  `BaCbcBCBA`
  `BaCCbcCBA`
  `BaCCbBA`
  `BaCCA`
  `BCC`

⇝ Terminates: the final word does not contain both `a` and `A`
(by the way: contains neither `a` nor `A`, and not both `b` and `B`.)

- Example:

  `abcbABABCBA`
  `BabcBabBABCBA`
  `BabcBaABCBA`
  `BabcBBCBA`
  `BaCbcBCBA`
  `BaCCbcCBA`
  `BaCCbBA`
  `BaCCA`
  `BCC`

$\rightsquigarrow$ Terminates: the final word does not contain both `a` and `A`
(by the way: contains neither `a` nor `A`, and not both `b` and `B`.)

- Theorem.— Handle reduction always terminates in exponential time

- Example:

  $\underline{\mathtt{abcbA}}\mathtt{BABCBA}$
  $\mathtt{BabcBa}\underline{\mathtt{bB}}\mathtt{ABCBA}$
  $\mathtt{BabcB}\underline{\mathtt{aA}}\mathtt{BCBA}$
  $\mathtt{Ba}\underline{\mathtt{bcB}}\mathtt{BCBA}$
  $\mathtt{BaC}\underline{\mathtt{bcB}}\mathtt{CBA}$
  $\mathtt{BaCCb}\underline{\mathtt{cC}}\mathtt{BA}$
  $\mathtt{BaCC}\underline{\mathtt{bB}}\mathtt{A}$
  $\mathtt{B}\underline{\mathtt{aCCA}}$
  $\mathtt{BCC}$

$\rightsquigarrow$ Terminates: the final word does not contain both `a` and `A`
(by the way: contains neither `a` nor `A`, and not both `b` and `B`.)

- **Theorem.**— Handle reduction always terminates in exponential time
(and *id.* for n-strand version).

- Example:

    abcbABABCBA
    BabcBabBABCBA
    BabcBaABCBA
    BabcBBCBA
    BaCbcBCBA
    BaCCbcCBA
    BaCCbBA
    BaCCA
    BCC

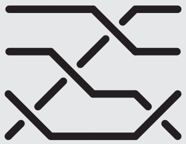
$\rightsquigarrow$ Terminates: the final word does not contain both a and A
(by the way: contains neither a nor A, and not both b and B.)

- Theorem.— Handle reduction always terminates in exponential time
(and *id.* for n-strand version).

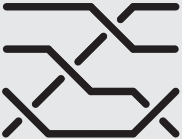
- Experimental evidence.— It terminates in quadratic time (for every n).

- A 4-strand braid diagram

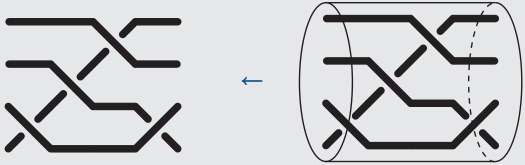
- A 4-strand braid diagram



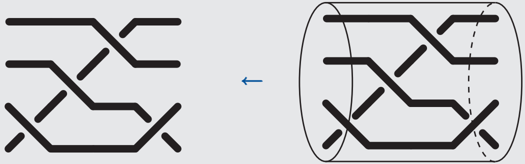
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



- A 4-strand braid diagram = 2D-projection of a 3D-figure:

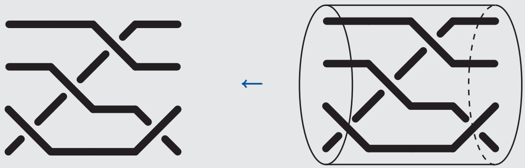


- A 4-strand braid diagram = 2D-projection of a 3D-figure:

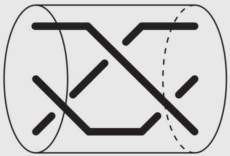


- isotopy = move the strands but keep the ends fixed:

- A 4-strand braid diagram = 2D-projection of a 3D-figure:

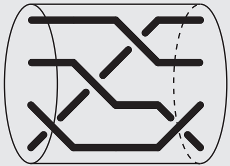


- isotopy = move the strands but keep the ends fixed:

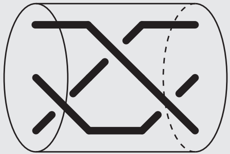


- A $\mathbf{4}$-strand braid diagram = 2D-projection of a 3D-figure:

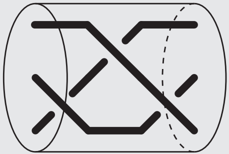
$\leftarrow$



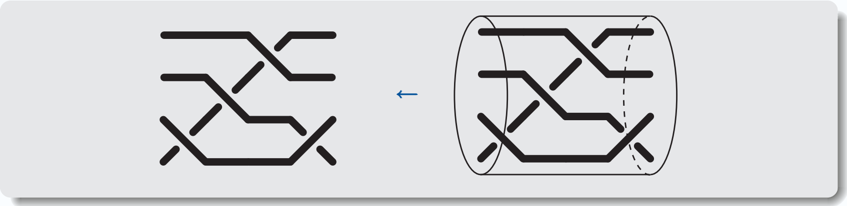
- isotopy = move the strands but keep the ends fixed:



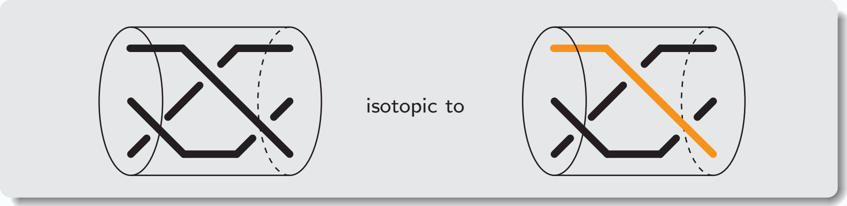
isotopic to



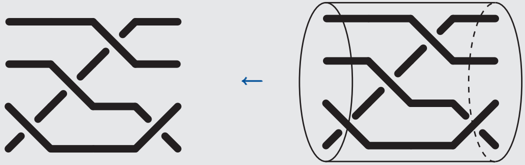
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



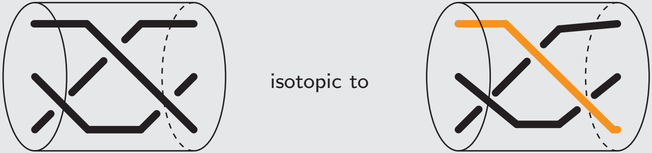
- isotopy = move the strands but keep the ends fixed:



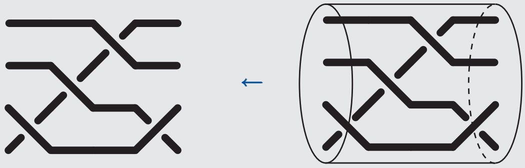
- A $\mathbf{4}$-strand braid diagram = 2D-projection of a 3D-figure:



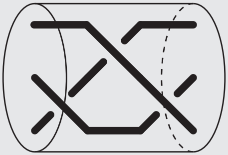
- isotopy = move the strands but keep the ends fixed:



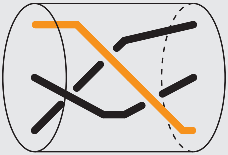
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



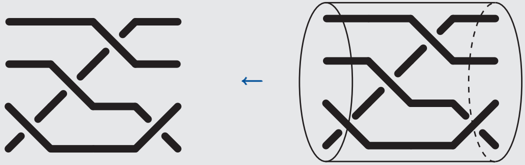
- isotopy = move the strands but keep the ends fixed:



isotopic to

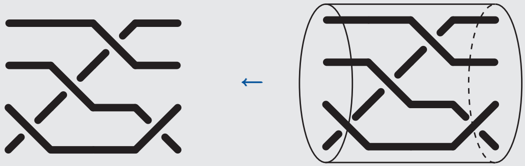


- A 4-strand braid diagram = 2D-projection of a 3D-figure:

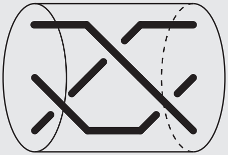


- isotopy = move the strands but keep the ends fixed:

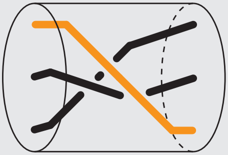
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



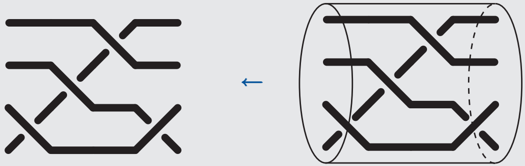
- isotopy = move the strands but keep the ends fixed:



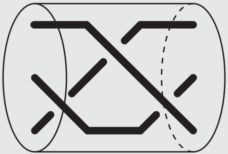
isotopic to



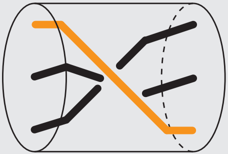
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



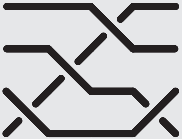
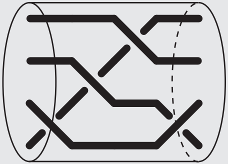
- isotopy = move the strands but keep the ends fixed:



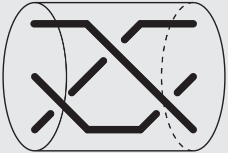
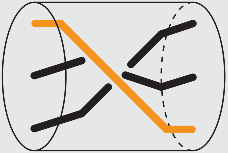
isotopic to



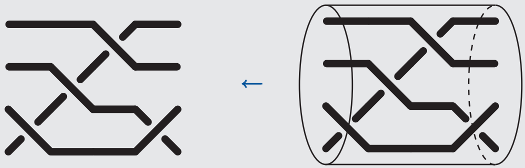
- A 4-strand braid diagram = 2D-projection of a 3D-figure:


←


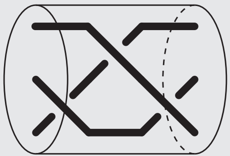
- isotopy = move the strands but keep the ends fixed:


isotopic to


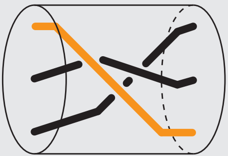
- A $\mathbf{4}$-strand braid diagram = 2D-projection of a 3D-figure:



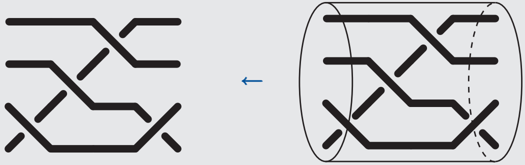
- isotopy = move the strands but keep the ends fixed:



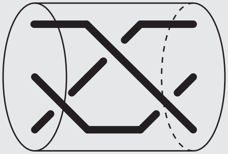
isotopic to



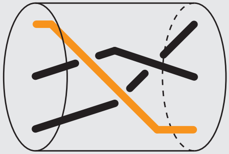
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



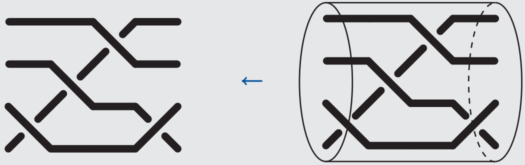
- isotopy = move the strands but keep the ends fixed:



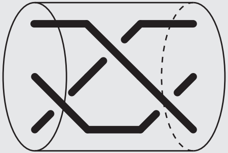
isotopic to



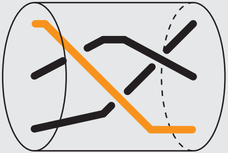
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



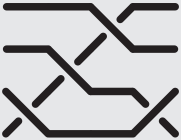
- isotopy = move the strands but keep the ends fixed:



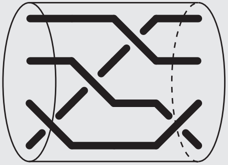
isotopic to



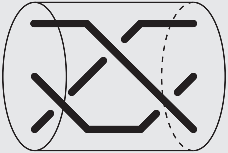
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



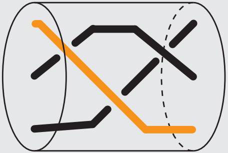
←



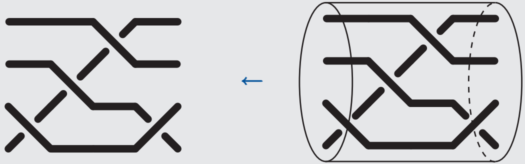
- isotopy = move the strands but keep the ends fixed:



isotopic to

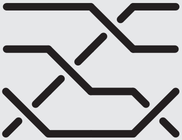


- A $\mathbf{4}$-strand braid diagram $=$ 2D-projection of a 3D-figure:

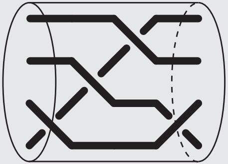


- isotopy $=$ move the strands but keep the ends fixed:

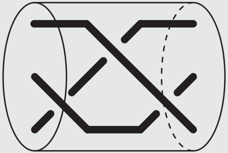
- A $\mathbf{4}$-strand braid diagram = 2D-projection of a 3D-figure:



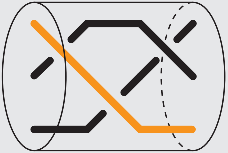
$\longleftarrow$



- isotopy = move the strands but keep the ends fixed:

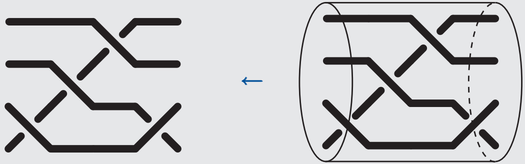


isotopic to



- a braid := an isotopy class $\rightsquigarrow$ represented by 2D-diagram,

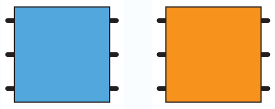
- A 4-strand braid diagram = 2D-projection of a 3D-figure:



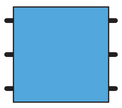
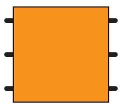
- isotopy = move the strands but keep the ends fixed:

- a braid := an isotopy class $\rightsquigarrow$ represented by 2D-diagram,
  but different 2D-diagrams may give rise to the same braid.

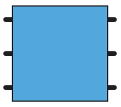
- Product of two braids:



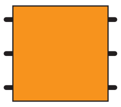
- Product of two braids:

 $*$  $:=$

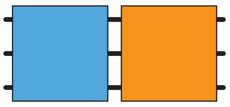
- Product of two braids:



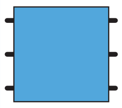
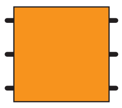
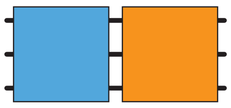
$*$



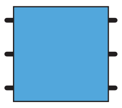
$:=$



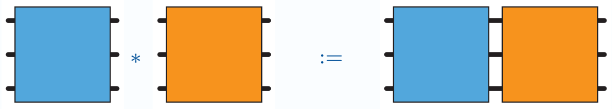
- Product of two braids:

 $*$  $:=$ 

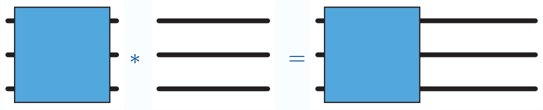
- Then well-defined with respect to isotopy), associative, admits a unit:

 $*$  $=$

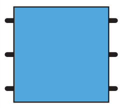
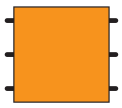
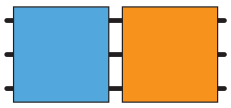
- Product of two braids:



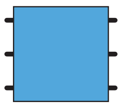
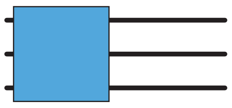
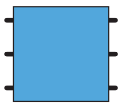
- Then well-defined with respect to isotopy), associative, admits a unit:



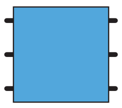
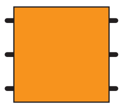
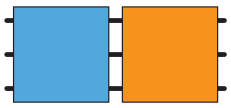
- Product of two braids:

 $*$  $:=$ 

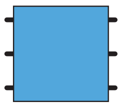
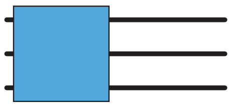
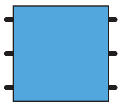
- Then well-defined with respect to isotopy), associative, admits a unit:

 $*$  $=$  $\approx$ 

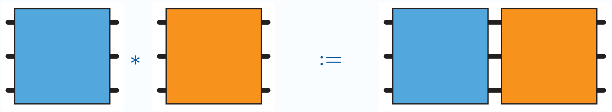
- Product of two braids:

 $*$  $:=$ 

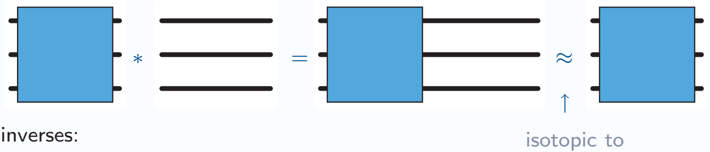
- Then well-defined with respect to isotopy), associative, admits a unit:

 $*$  $=$  $\underset{\substack{\uparrow \\ \text{isotopic to}}}{\approx}$ 

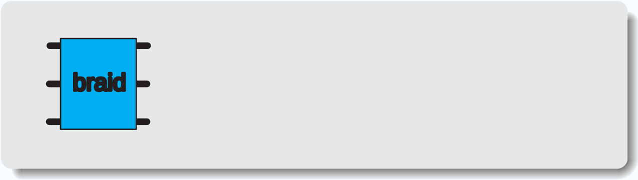
- Product of two braids:



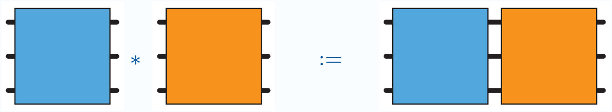
- Then well-defined with respect to isotopy), associative, admits a unit:



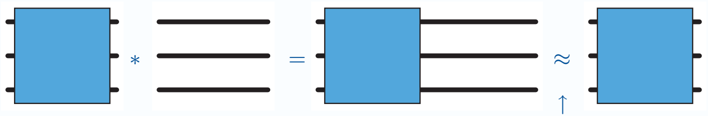
and inverses:



- Product of two braids:

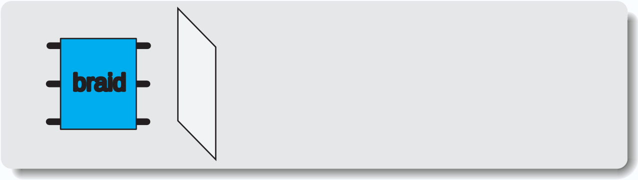


- Then well-defined with respect to isotopy), associative, admits a unit:

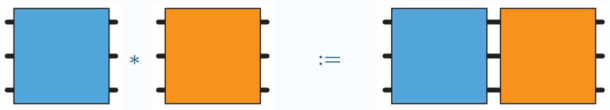


isotopic to

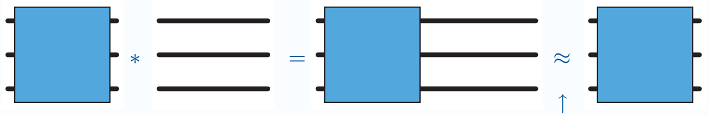
and inverses:



- Product of two braids:

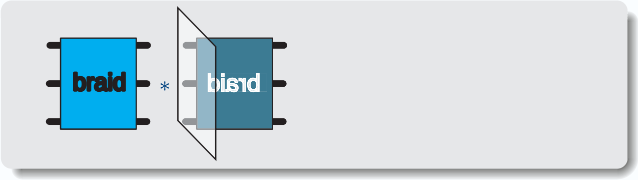


- Then well-defined with respect to isotopy), associative, admits a unit:

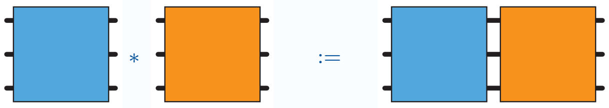


and inverses:

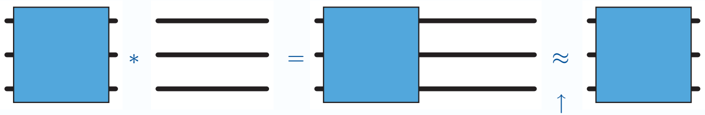
isotopic to



- Product of two braids:

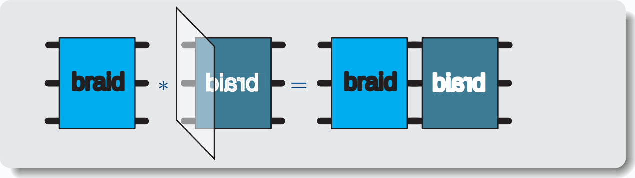


- Then well-defined with respect to isotopy), associative, admits a unit:

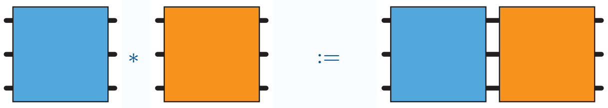


isotopic to

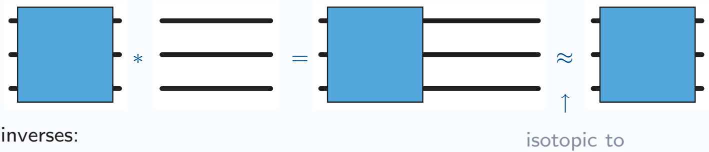
and inverses:



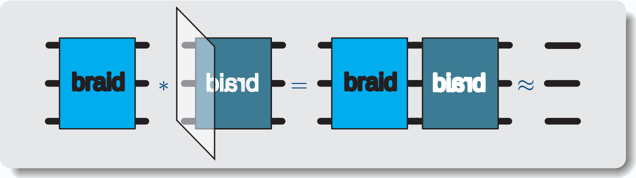
- Product of two braids:



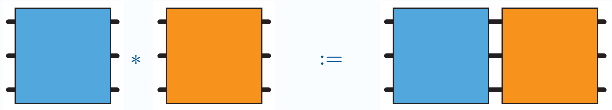
- Then well-defined with respect to isotopy), associative, admits a unit:



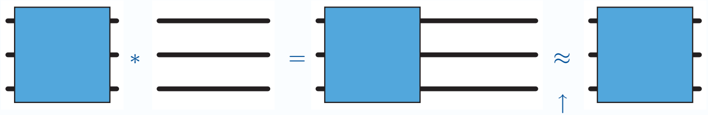
and inverses:



- Product of two braids:

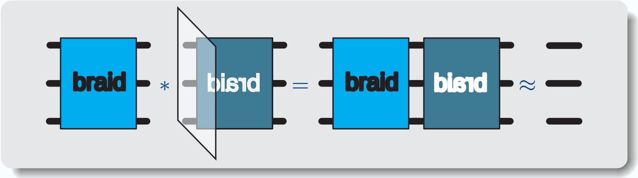


- Then well-defined with respect to isotopy), associative, admits a unit:



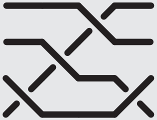
isotopic to

and inverses:

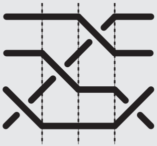


⇝ For each n, the group B_n of n-strand braids (E.Artin, 1925).

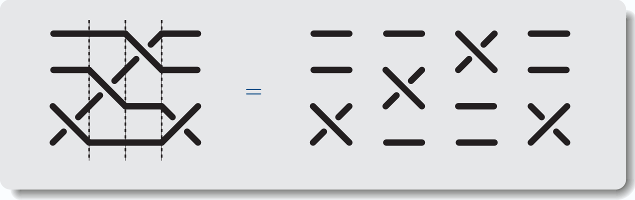
- Artin generators of B_n:



- Artin generators of B_n:

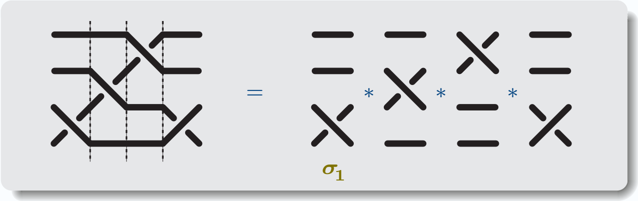


- Artin generators of B_n:

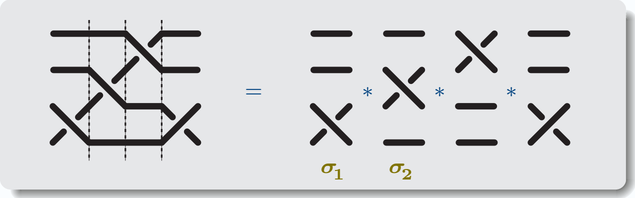


- Artin generators of B_n:

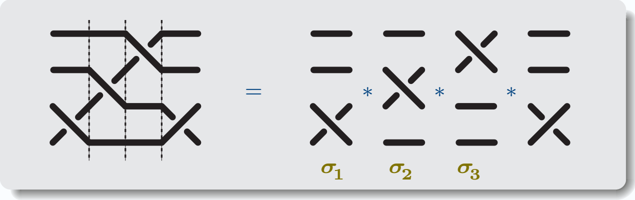
- Artin generators of B_n:



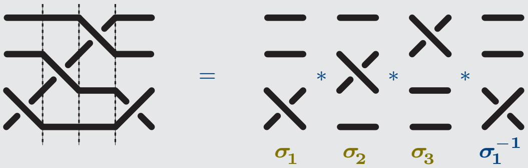
- Artin generators of B_n:



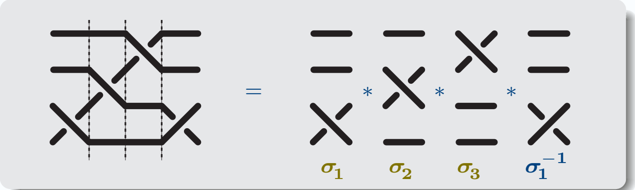
- Artin generators of B_n:



- Artin generators of B_n:

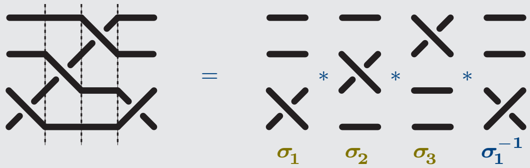


- Artin generators of B_n:



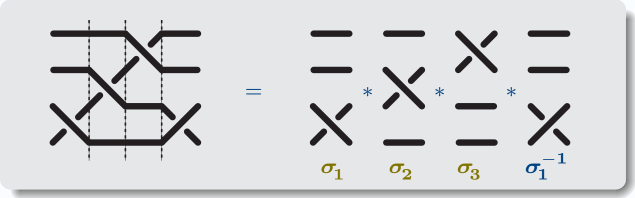
- Theorem (Artin): The group B_n is generated by $\sigma_1, \ldots, \sigma_{n-1}$,

- Artin generators of B_n:



- Theorem (Artin): The group B_n is generated by $\sigma_1, \ldots, \sigma_{n-1}$, subject to $\left\{ \begin{array}{ll} \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{for } |i-j| = 1, \\ & \end{array} \right.$

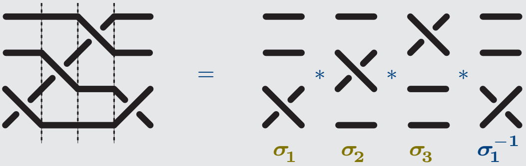
- Artin generators of B_n:



- Theorem (Artin): The group B_n is generated by $\sigma_1, \ldots, \sigma_{n-1}$,

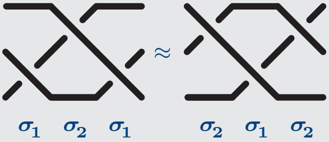
$$\text{subject to} \begin{cases} \sigma_i \sigma_j \sigma_i = \sigma_j \sigma_i \sigma_j & \text{for } |i-j| = 1, \\ \sigma_i \sigma_j = \sigma_j \sigma_i & \text{for } |i-j| \geqslant 2. \end{cases}$$

- Artin generators of B_n:

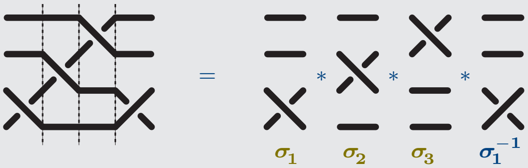


- Theorem (Artin): The group B_n is generated by $\sigma_1, \ldots, \sigma_{n-1}$,

subject to $\begin{cases} \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{for } |i-j| = 1, \\ \sigma_i\sigma_j = \sigma_j\sigma_i & \text{for } |i-j| \geqslant 2. \end{cases}$

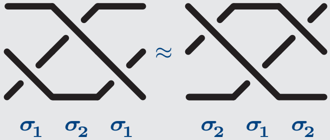


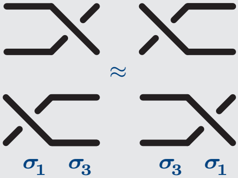
- Artin generators of B_n:



- Theorem (Artin): The group B_n is generated by $\sigma_1, \ldots, \sigma_{n-1}$,

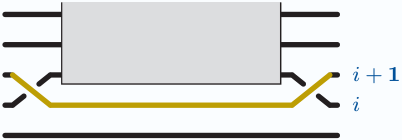
subject to $\begin{cases} \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{for } |i-j| = 1, \\ \sigma_i\sigma_j = \sigma_j\sigma_i & \text{for } |i-j| \geqslant 2. \end{cases}$



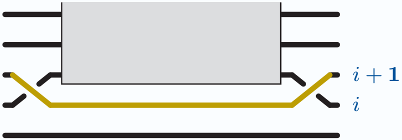


- A $\boldsymbol{\sigma}_i$-handle:

- A $\boldsymbol{\sigma}_i$-handle:

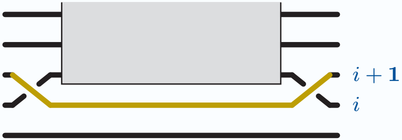


- A $\boldsymbol{\sigma}_i$-handle:

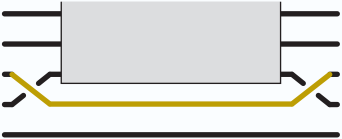


- Reducing a handle:

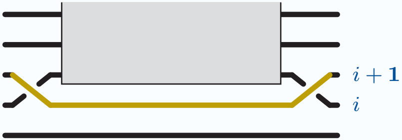
- A σ_i-handle:



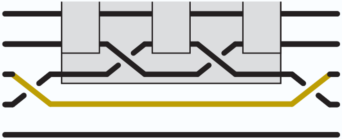
- Reducing a handle:



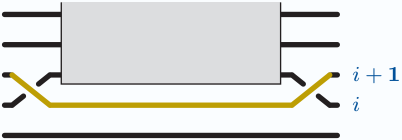
- A σ_i-handle:



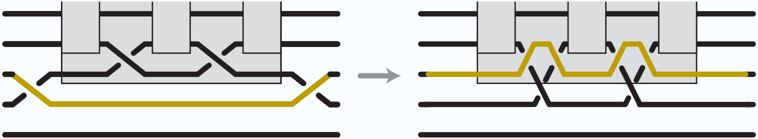
- Reducing a handle:



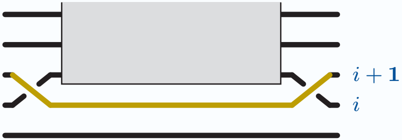
- A σ_i-handle:



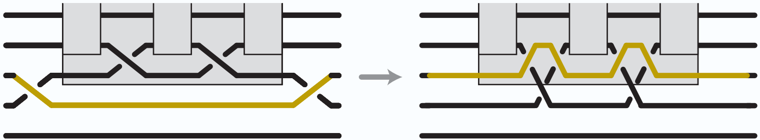
- Reducing a handle:



- A σ_i-handle:

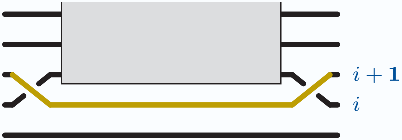


- Reducing a handle:

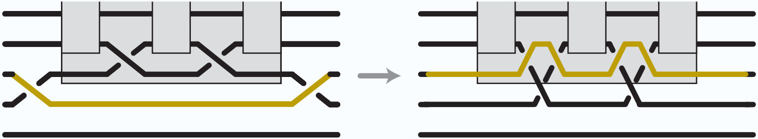


- Handle reduction is an isotopy;

- A σ_i-handle:

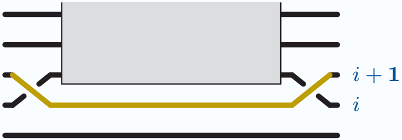


- Reducing a handle:

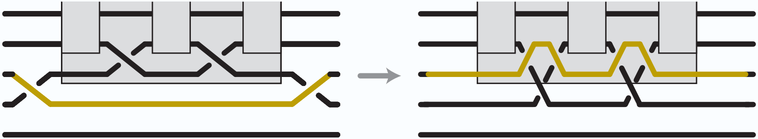


- Handle reduction is an isotopy; It extends free group reduction;

- A $\boldsymbol{\sigma}_i$-handle:

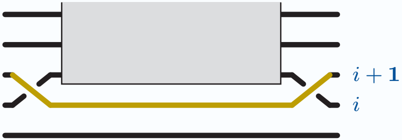


- Reducing a handle:

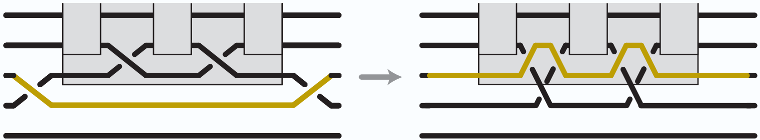


- Handle reduction is an isotopy; It extends free group reduction;
  Terminal words cannot contain both $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_1^{-1}$.

- A σ_i-handle:



- Reducing a handle:



- Handle reduction is an isotopy; It extends free group reduction;
  Terminal words cannot contain both σ_1 and σ_1^{-1}.

- Theorem.— Every sequence of handle reductions terminates.

1. The Polish Algorithm for Left-Selfdistributivity
2. Handle reduction of braids
3. Subword reversing for positively presented groups

- This time: a truly true rewrite system...
- Alphabet: `a`, `b`, `A`, `B`

- This time: a truly true rewrite system...
- Alphabet: `a`, `b`, `A`, `B` (think of `A` as an inverse of a, etc.)

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$,

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$, $\mathtt{Ba} \longrightarrow \mathtt{aB}$.

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$, $\mathtt{Ba} \longrightarrow \mathtt{aB}$. ("reverse $-+$ patterns into $+-$ patterns")

- This time: a truly true rewrite system...

- Alphabet: a, b, A, B (think of A as an inverse of a, etc.)
- Rewrite rules:
  - Aa $\longrightarrow \varepsilon$, Bb $\longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - Ab $\longrightarrow$ bA, Ba $\longrightarrow$ aB. ("reverse $-+$ patterns into $+-$ patterns")

- Aim: transforming an arbitrary signed word into a positive–negative word.

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$, $\mathtt{Ba} \longrightarrow \mathtt{aB}$. ("reverse $-+$ patterns into $+-$ patterns")

- Aim: transforming an arbitrary signed word into a positive–negative word.

- Example: $\mathtt{BBAbabb}$

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$, $\mathtt{Ba} \longrightarrow \mathtt{aB}$. ("reverse $-+$ patterns into $+-$ patterns")

- Aim: transforming an arbitrary signed word into a positive–negative word.

- Example: $\mathtt{B}\underline{\mathtt{BA}}\mathtt{babb}$

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$, $\mathtt{Ba} \longrightarrow \mathtt{aB}$. ("reverse $-+$ patterns into $+-$ patterns")

- Aim: transforming an arbitrary signed word into a positive–negative word.

- Example: $\mathtt{B}\underline{\mathtt{Ab}}\mathtt{abb} \longrightarrow \mathtt{B}\underline{\mathtt{Bb}}\mathtt{Aabb}$

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$, $\mathtt{Ba} \longrightarrow \mathtt{aB}$. ("reverse $-+$ patterns into $+-$ patterns")

- Aim: transforming an arbitrary signed word into a positive–negative word.

- Example: $\mathtt{B}\underline{\mathtt{BAb}}\mathtt{abb} \longrightarrow \mathtt{B}\underline{\mathtt{Bb}}\mathtt{Aabb} \longrightarrow \mathtt{B}\underline{\mathtt{Aa}}\mathtt{bb}$

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of $\mathtt{A}$ as an inverse of $\mathtt{a}$, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$, $\mathtt{Ba} \longrightarrow \mathtt{aB}$. ("reverse $-+$ patterns into $+-$ patterns")

- Aim: transforming an arbitrary signed word into a positive–negative word.

- Example: $\mathtt{B}\underline{\mathtt{BAb}}\mathtt{abb} \longrightarrow \mathtt{B}\underline{\mathtt{Bb}}\mathtt{Aabb} \longrightarrow \mathtt{B}\underline{\mathtt{Aa}}\mathtt{bb} \longrightarrow \underline{\mathtt{Bb}}\mathtt{b}$

- This time: a truly true rewrite system...

- Alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$ (think of A as an inverse of a, etc.)
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ ("free group reduction" as usual, but only one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{bA}$, $\mathtt{Ba} \longrightarrow \mathtt{aB}$. ("reverse $-+$ patterns into $+-$ patterns")

- Aim: transforming an arbitrary signed word into a positive–negative word.

- Example: $\mathtt{B}\mathtt{B}\underline{\mathtt{Ab}}\mathtt{abb} \longrightarrow \mathtt{B}\underline{\mathtt{Bb}}\mathtt{Aabb} \longrightarrow \mathtt{B}\underline{\mathtt{Aa}}\mathtt{bb} \longrightarrow \underline{\mathtt{Bb}}\mathtt{b} \longrightarrow \mathtt{b}$.

- "Theorem".— It terminates in quadratic time.

- "Theorem".— It terminates in quadratic time.

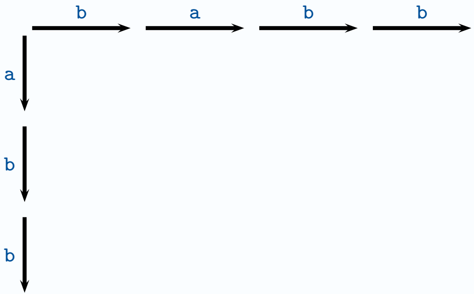
- Proof: (obvious).

- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:

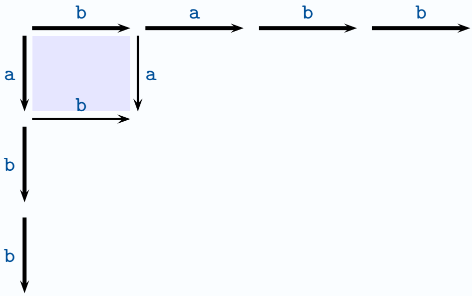
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



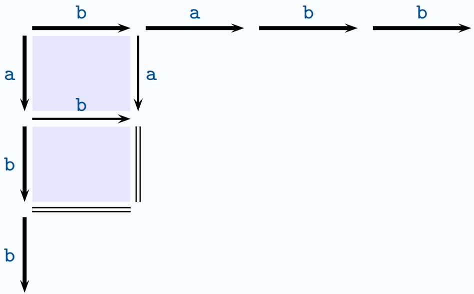
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



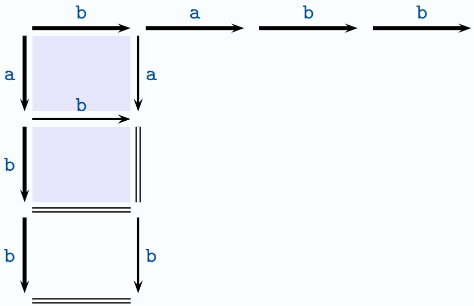
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



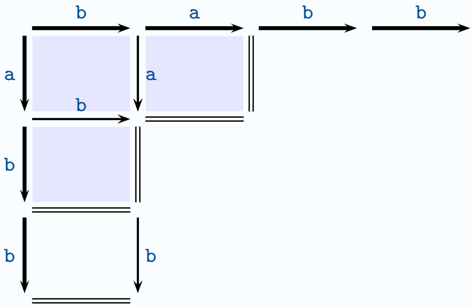
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



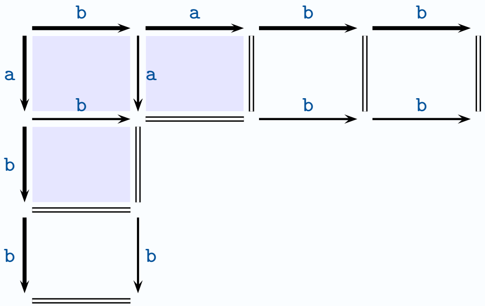
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



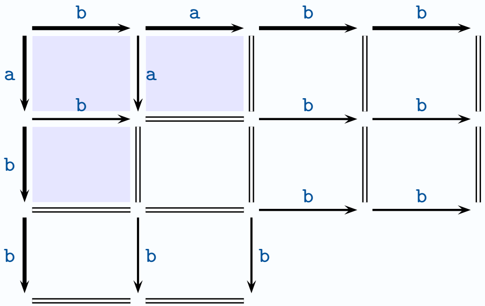
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



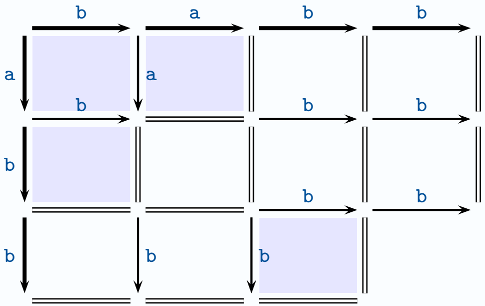
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



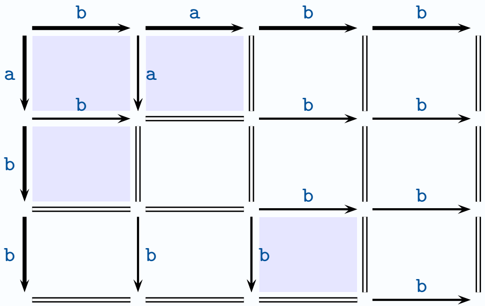
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



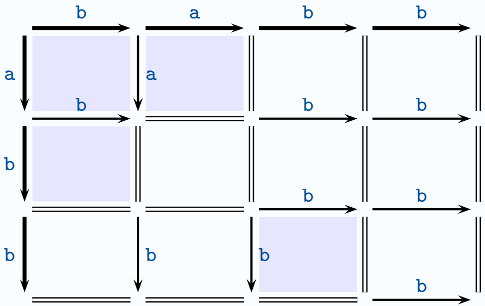
- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



- "Theorem".— It terminates in quadratic time.

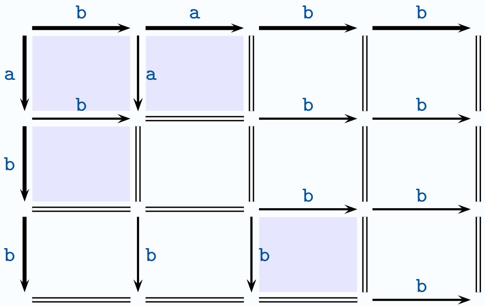
- Proof: (obvious). Construct a reversing grid:



$\rightsquigarrow$ Clear that reversing terminates with quadratic time upper bound (and linear space upper bound).

- "Theorem".— It terminates in quadratic time.

- Proof: (obvious). Construct a reversing grid:



⇝ Clear that reversing terminates with quadratic time upper bound (and linear space upper bound).

- Obviously: *id.* for any number of letters.

- Example 2:
- Same alphabet: `a`, `b`, `A`, `B`

- Example 2:
- Same alphabet: $\texttt{a}, \texttt{b}, \texttt{A}, \texttt{B}$

- Rewrite rules:
  - $\texttt{Aa} \longrightarrow \varepsilon$, $\texttt{Bb} \longrightarrow \varepsilon$

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$,

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$.

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

- Example: $\mathtt{BBAbabb}$

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

- Example: $\mathtt{BB\underline{Ab}abb}$

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$
- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.
- Termination? Not clear: length may increase...
- Example: $\mathtt{BB}\underline{\mathtt{Ab}}\mathtt{abb} \longrightarrow \mathtt{B}\underline{\mathtt{Bb}}\mathtt{aBAabb}$

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

- Example: $\mathtt{BB\underline{Ab}abb} \longrightarrow \mathtt{B\underline{Bb}aBAabb} \longrightarrow \mathtt{\underline{Ba}BAabb}$

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

- Example: $\mathtt{BB\underline{Ab}abb} \longrightarrow \mathtt{B\underline{Bb}aBAabb} \longrightarrow \mathtt{\underline{Ba}BAabb}$
  $\longrightarrow \mathtt{abABB\underline{Aa}bb}$

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

- Example: $\mathtt{BB\underline{Ab}abb} \longrightarrow \mathtt{B\underline{Bb}aBAabb} \longrightarrow \mathtt{\underline{Ba}BAabb}$

  $\longrightarrow \mathtt{abABB\underline{Aa}bb} \longrightarrow \mathtt{abAB\underline{Bb}b}$

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

- Example: $\mathtt{B}\underline{\mathtt{Ab}}\mathtt{abb}$ $\longrightarrow$ $\mathtt{B}\underline{\mathtt{Bb}}\mathtt{aBAabb}$ $\longrightarrow$ $\underline{\mathtt{Ba}}\mathtt{BAabb}$
  $\longrightarrow$ $\mathtt{abAB}\underline{\mathtt{Aa}}\mathtt{bb}$ $\longrightarrow$ $\mathtt{abA}\underline{\mathtt{Bb}}\mathtt{b}$ $\longrightarrow$ $\mathtt{abA}\underline{\mathtt{Bb}}$

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

- Example: $\mathtt{BB\underline{Ab}abb} \longrightarrow \mathtt{B\underline{Bb}aBAabb} \longrightarrow \mathtt{\underline{Ba}BAabb}$

  $\longrightarrow \mathtt{abABB\underline{Aa}bb} \longrightarrow \mathtt{abAB\underline{Bb}b} \longrightarrow \mathtt{abA\underline{Bb}} \longrightarrow \mathtt{abA}$.

- Example 2:
- Same alphabet: $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$

- Rewrite rules:
  - $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$ (free group reduction in one direction)
  - $\mathtt{Ab} \longrightarrow \mathtt{baBA}$, $\mathtt{Ba} \longrightarrow \mathtt{abAB}$. ("reverse $-+$ into $+-$", but different rule)

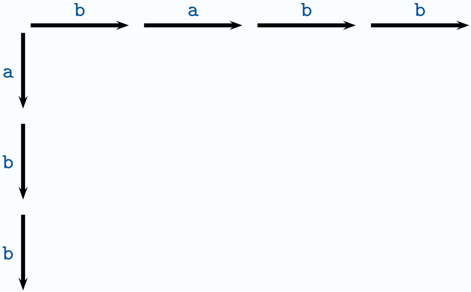
    $\rightsquigarrow$ Again: transforms an arbitrary signed word into a positive–negative word.

- Termination? Not clear: length may increase...

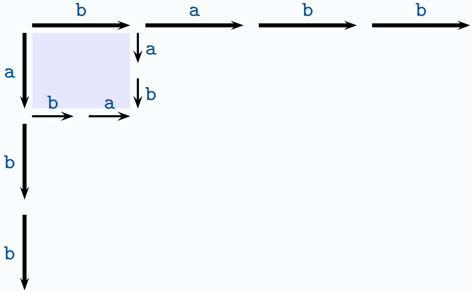
- Example: $\mathtt{BB\underline{Ab}abb} \longrightarrow \mathtt{B\underline{Bb}aBAabb} \longrightarrow \mathtt{\underline{Ba}BAabb}$
  $\longrightarrow \mathtt{abABB\underline{Aa}bb} \longrightarrow \mathtt{abAB\underline{Bb}b} \longrightarrow \mathtt{abA\underline{Bb}} \longrightarrow \mathtt{abA}$.

- Reversing grid:

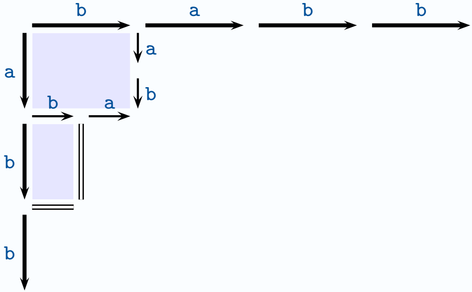
- Reversing grid: same, but possibly smaller and smaller arrows.



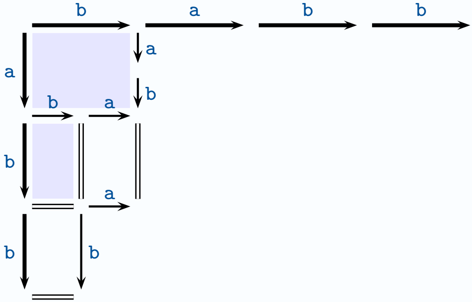
- Reversing grid: same, but possibly smaller and smaller arrows.



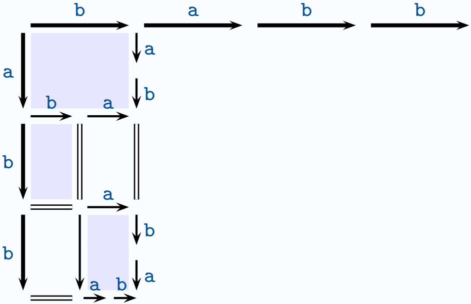
- Reversing grid: same, but possibly smaller and smaller arrows.



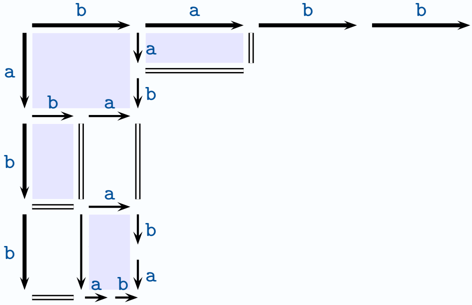
- Reversing grid: same, but possibly smaller and smaller arrows.



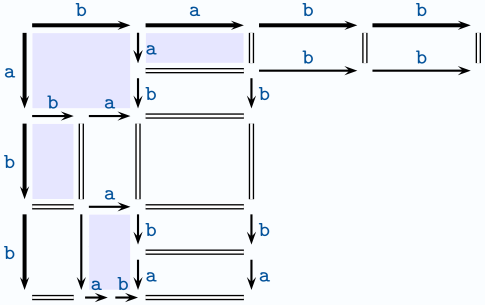
- Reversing grid: same, but possibly smaller and smaller arrows.



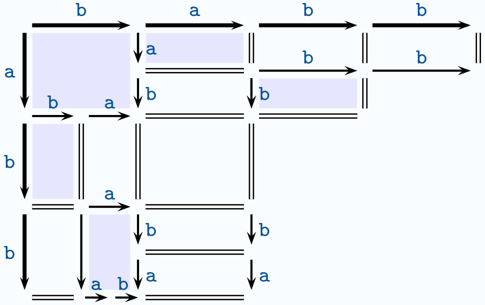
- Reversing grid: same, but possibly smaller and smaller arrows.



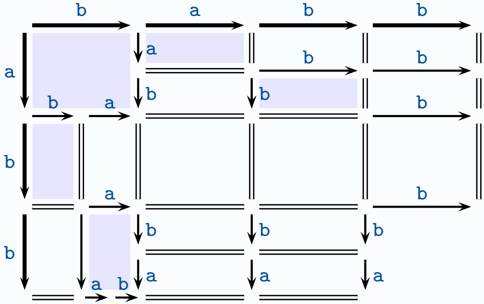
- Reversing grid: same, but possibly smaller and smaller arrows.



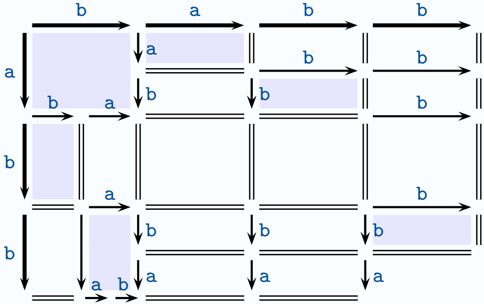
- Reversing grid: same, but possibly smaller and smaller arrows.



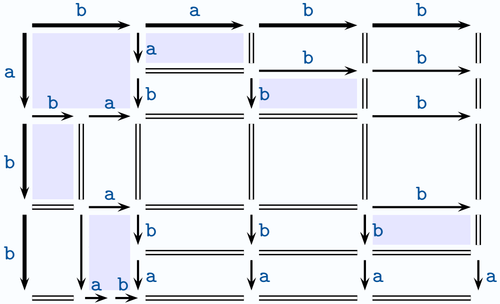
- Reversing grid: same, but possibly smaller and smaller arrows.



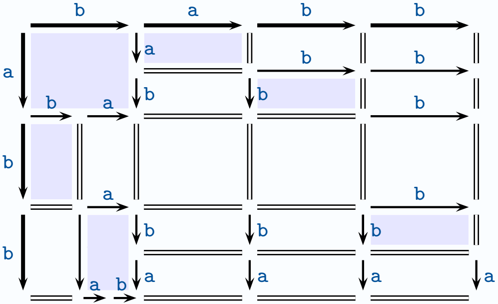
- Reversing grid: same, but possibly smaller and smaller arrows.



- Reversing grid: same, but possibly smaller and smaller arrows.



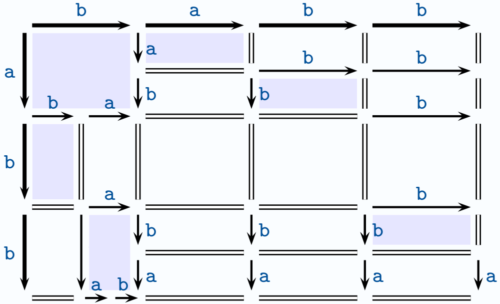
- Reversing grid: same, but possibly smaller and smaller arrows.



- Theorem.— Reversing terminates in quadratic time (in this specific case).

- Proof:

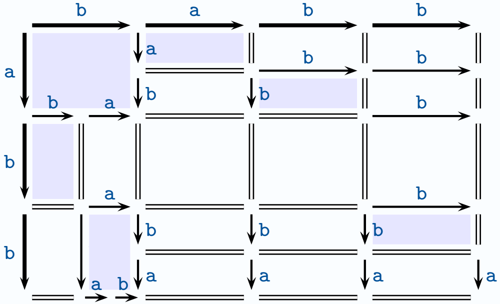
- Reversing grid: same, but possibly smaller and smaller arrows.



- Theorem.— Reversing terminates in quadratic time (in this specific case).

- Proof: Return to the baby case

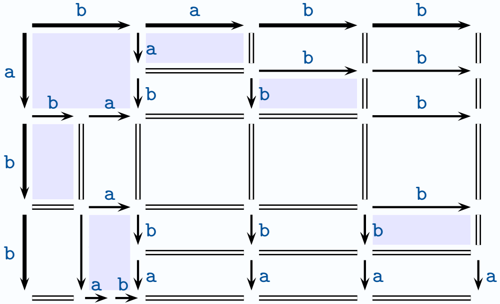
- Reversing grid: same, but possibly smaller and smaller arrows.



- Theorem.— Reversing terminates in quadratic time (in this specific case).

- Proof: Return to the baby case = find a (finite) set of words S that includes the alphabet and closed under reversing.

- Reversing grid: same, but possibly smaller and smaller arrows.



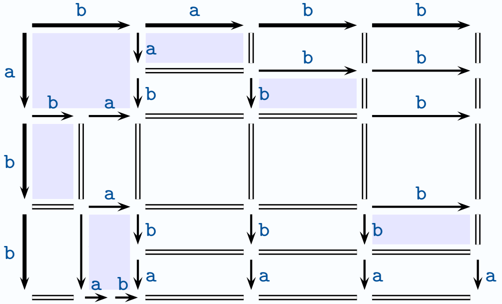
- Theorem.— Reversing terminates in quadratic time (in this specific case).

- Proof: Return to the baby case = find a (finite) set of words S that includes the alphabet and closed under reversing.

↑ for all u, v in S, exist u', v' in S s.t. $\exists$ reversing grid

$$\begin{tikzcd} \cdot \arrow[r, "v"] \arrow[d, "u"'] & \cdot \arrow[d, "u'"] \\ \cdot \arrow[r, "v'"'] & \cdot \end{tikzcd}$$

• Reversing grid: same, but possibly smaller and smaller arrows.



• Theorem.— Reversing terminates in quadratic time (in this specific case).

• Proof: Return to the baby case = find a (finite) set of words $\boldsymbol{S}$ that includes the alphabet and closed under reversing.

for all $\boldsymbol{u}, \boldsymbol{v}$ in $\boldsymbol{S}$, exist $\boldsymbol{u}', \boldsymbol{v}'$ in $\boldsymbol{S}$ s.t. $\exists$ reversing grid
$$\begin{tikzcd} \cdot \arrow[r, "\boldsymbol{v}"] \arrow[d, "\boldsymbol{u}"'] & \cdot \arrow[d, "\boldsymbol{u}'"] \\ \cdot \arrow[r, "\boldsymbol{v}'"'] & \cdot \end{tikzcd}$$

Here: works with $\boldsymbol{S} = \{\mathtt{a}, \mathtt{b}, \mathtt{ab}, \mathtt{ba}\}$. $\square$

- Always like that?

- Always like that? Not really...

- Example 3:

Alphabet `a`, `b`, `A`, `B`,

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$,

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m \text{ letters}}\underbrace{\ldots\mathtt{BABA}}_{m \text{ letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m \text{ letters}}\underbrace{\ldots\mathtt{ABAB}}_{m \text{ letters}}$.

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$,

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$,

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$:

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\text{ letters}}\underbrace{\ldots\mathtt{BABA}}_{m\text{ letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\text{ letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\text{ letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underline{\mathtt{Ba}}\mathtt{b}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underline{\mathtt{Ba}}\mathtt{b} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba...}}_{m \text{ letters}}\underbrace{\mathtt{...BABA}}_{m \text{ letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab...}}_{m \text{ letters}}\underbrace{\mathtt{...ABAB}}_{m \text{ letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underline{\mathtt{Ba}}\mathtt{b} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underline{\mathtt{Ba}}\mathtt{b} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\dots}_{m\,\text{letters}}\underbrace{\dots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\dots}_{m\,\text{letters}}\underbrace{\dots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underline{\mathtt{Ba}}\mathtt{b} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba...}}_{m\,\text{letters}}\underbrace{\mathtt{...BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab...}}_{m\,\text{letters}}\underbrace{\mathtt{...ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underline{\mathtt{Ba}}\mathtt{b} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underline{\mathtt{Ba}}\mathtt{b} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Ba}}\mathtt{bAAA}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underline{\mathtt{Ba}}\mathtt{b} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Ba}}\mathtt{bAAA} \longrightarrow \mathtt{aaaaB}\underline{\mathtt{Ab}}\mathtt{AAA}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underset{\uparrow\atop w}{\underline{\mathtt{Ba}}\mathtt{b}} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \underset{\uparrow\atop \mathtt{a}w\mathtt{A}}{\mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA}} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \underset{\uparrow\atop \mathtt{a}^2w\mathtt{A}^2}{\mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA}} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Ba}}\mathtt{bAAA} \longrightarrow \mathtt{aaaaB}\underline{\mathtt{Ab}}\mathtt{AAA}$

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\text{ letters}}\underbrace{\ldots\mathtt{BABA}}_{m\text{ letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\text{ letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\text{ letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underbrace{\underline{\mathtt{Ba}}\mathtt{b}}_{w} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \underbrace{\mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA}}_{\mathtt{a}w\mathtt{A}} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \underbrace{\mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA}}_{\mathtt{a}^2 w \mathtt{A}^2} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Ba}}\mathtt{bAAA} \longrightarrow \mathtt{aaaaB}\underline{\mathtt{Ab}}\mathtt{AAA}$

$\rightsquigarrow$ Here : non-terminating

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underset{\overset{\uparrow}{w}}{\underline{\mathtt{Ba}}\mathtt{b}} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \underset{\overset{\uparrow}{\mathtt{a}w\mathtt{A}}}{\mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA}} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \underset{\overset{\uparrow}{\mathtt{a}^2 w \mathtt{A}^2}}{\mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA}} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Ba}}\mathtt{bAAA} \longrightarrow \mathtt{aaaaB}\underline{\mathtt{Ab}}\mathtt{AAA}$

$\rightsquigarrow$ Here : non-terminating

- Example 5:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$,

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underset{\substack{\uparrow \\ w}}{\underline{\mathtt{Ba}}\mathtt{b}} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \underset{\substack{\uparrow \\ \mathtt{a}w\mathtt{A}}}{\mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA}} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \underset{\substack{\uparrow \\ \mathtt{a}^2 w\mathtt{A}^2}}{\mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA}} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Ba}}\mathtt{bAAA} \longrightarrow \mathtt{aaaaB}\underline{\mathtt{Ab}}\mathtt{AAA}$

$\rightsquigarrow$ Here : non-terminating

- Example 5:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$,

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\text{ letters}}\underbrace{\ldots\mathtt{BABA}}_{m\text{ letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\text{ letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\text{ letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underset{\uparrow\, w}{\underline{\mathtt{Bab}}} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \underset{\uparrow\, \mathtt{a}w\mathtt{A}}{\mathtt{a}\underline{\mathtt{Bab}}\mathtt{A}} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \underset{\uparrow\, \mathtt{a}^2 w\mathtt{A}^2}{\mathtt{aa}\underline{\mathtt{Bab}}\mathtt{AA}} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Bab}}\mathtt{AAA} \longrightarrow \mathtt{aaaaB}\underline{\mathtt{Ab}}\mathtt{AAA}$

$\rightsquigarrow$ Here : non-terminating

- Example 5:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ba} \longrightarrow \mathtt{abab}^2\mathtt{ab}^2\mathtt{abab}$,

$\mathtt{Ba} \longrightarrow \mathtt{BABAB}^2\mathtt{AB}^2\mathtt{ABA}$,

- Always like that? Not really...

- Example 3:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

$\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$

Start with $\mathtt{Bab}$: $\underset{\overset{\uparrow}{w}}{\underline{\mathtt{Ba}}\mathtt{b}} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \underset{\overset{\uparrow}{\mathtt{a}w\mathtt{A}}}{\mathtt{a}\underline{\mathtt{Ba}}\mathtt{bA}} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \underset{\overset{\uparrow}{\mathtt{a}^2 w\mathtt{A}^2}}{\mathtt{aa}\underline{\mathtt{Ba}}\mathtt{bAA}} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Ba}}\mathtt{bAAA} \longrightarrow \mathtt{aaaaB}\underline{\mathtt{Ab}}\mathtt{AAA}$

$\rightsquigarrow$ Here : non-terminating

- Example 5:

Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ba} \longrightarrow \mathtt{abab}^2\mathtt{ab}^2\mathtt{abab}$,

$\mathtt{Ba} \longrightarrow \mathtt{BABAB}^2\mathtt{AB}^2\mathtt{ABA}$,

$\rightsquigarrow$ Here : terminating in cubic time and quadratic space

- Always like that? Not really...

- Example 3:
Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \underbrace{\mathtt{baba}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{BABA}}_{m\,\text{letters}}$, $\mathtt{Ba} \longrightarrow \underbrace{\mathtt{abab}\ldots}_{m\,\text{letters}}\underbrace{\ldots\mathtt{ABAB}}_{m\,\text{letters}}$.

  $\rightsquigarrow$ Here : terminating in quadratic time and linear space

- Example 4:
Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ab} \longrightarrow \mathtt{abA}$, $\mathtt{Ba} \longrightarrow \mathtt{aBA}$
Start with $\mathtt{Bab}$: $\underbrace{\underline{\mathtt{Bab}}}_{w} \longrightarrow \mathtt{aB}\underline{\mathtt{Ab}} \longrightarrow \underbrace{\mathtt{a}\underline{\mathtt{Bab}}\mathtt{A}}_{\mathtt{a}w\mathtt{A}} \longrightarrow \mathtt{aaB}\underline{\mathtt{Ab}}\mathtt{A} \longrightarrow \underbrace{\mathtt{aa}\underline{\mathtt{Bab}}\mathtt{AA}}_{\mathtt{a}^2 w \mathtt{A}^2} \longrightarrow \mathtt{aaaB}\underline{\mathtt{Ab}}\mathtt{AA} \longrightarrow \mathtt{aaa}\underline{\mathtt{Bab}}\mathtt{AAA} \longrightarrow \mathtt{aaaaB}\underline{\mathtt{Ab}}\mathtt{AAA}$

  $\rightsquigarrow$ Here : non-terminating

- Example 5:
Alphabet $\mathtt{a}, \mathtt{b}, \mathtt{A}, \mathtt{B}$, rules $\mathtt{Aa} \longrightarrow \varepsilon$, $\mathtt{Bb} \longrightarrow \varepsilon$, plus $\mathtt{Ba} \longrightarrow \mathtt{abab}^2\mathtt{ab}^2\mathtt{abab}$,
$\mathtt{Ba} \longrightarrow \mathtt{BABAB}^2\mathtt{AB}^2\mathtt{ABA}$,

  $\rightsquigarrow$ Here : terminating in cubic time and quadratic space

- What are we doing?

- What are we doing? We are working with a semigroup presentation
  and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations,

- What are we doing? We are working with a semigroup presentation and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$.

- What are we doing? We are working with a semigroup presentation
  and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$. $\rightsquigarrow$ monoid $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle^+$,

- What are we doing? We are working with a semigroup presentation
  and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$. $\rightsquigarrow$ monoid $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle^{+}$, group $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$.

- What are we doing? We are working with a semigroup presentation and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$. $\rightsquigarrow$ monoid $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab}\rangle^{+}$, group $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab}\rangle$.

- Definition.— Assume (A, R) semigroup presentation and, for all $s \neq t$ in A, there is exactly one relation $s... = t...$ in R, say $sC(s,t) = tC(t,s)$. Then reversing is the rewrite system on $A \cup \overline{A}$ (a copy of A, here : capitalized letters) with rules $\overline{s}s \to \varepsilon$ and $\overline{s}t \to C(s,t)\overline{C(t,s)}$ for $s \neq t$ in A.

- Reversing does not change the element of the group that is represented;

- What are we doing? We are working with a semigroup presentation and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$. $\rightsquigarrow$ monoid $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab}\rangle^+$, group $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab}\rangle$.

- Definition.— Assume (A, R) semigroup presentation and, for all $s \neq t$ in A, there is exactly one relation $s... = t...$ in R, say $sC(s,t) = tC(t,s)$.
Then reversing is the rewrite system on $A \cup \overline{A}$ (a copy of A, here : capitalized letters) with rules $\overline{s}s \to \varepsilon$ and $\overline{s}t \to C(s,t)\overline{C(t,s)}$ for $s \neq t$ in A.

- Reversing does not change the element of the group that is represented;
$\rightsquigarrow$ if it terminates, every element of the group is a fraction fg^{-1} with f, g positive.

- Example 1 = reversing for the free Abelian group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{ab} = \mathtt{ba}\rangle$;

- What are we doing? We are working with a semigroup presentation
  and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$. $\rightsquigarrow$ monoid $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle^{+}$, group $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$.

- Definition.— Assume $(\boldsymbol{A}, \boldsymbol{R})$ semigroup presentation and, for all $\boldsymbol{s} \neq \boldsymbol{t}$ in $\boldsymbol{A}$, there is exactly one relation $\boldsymbol{s}... = \boldsymbol{t}...$ in $\boldsymbol{R}$, say $\boldsymbol{s}\boldsymbol{C}(\boldsymbol{s}, \boldsymbol{t}) = \boldsymbol{t}\boldsymbol{C}(\boldsymbol{t}, \boldsymbol{s})$.
Then reversing is the rewrite system on $\boldsymbol{A} \cup \overline{\boldsymbol{A}}$ (a copy of $\boldsymbol{A}$, here : capitalized letters) with rules $\overline{\boldsymbol{s}}\boldsymbol{s} \to \boldsymbol{\varepsilon}$ and $\overline{\boldsymbol{s}}\boldsymbol{t} \to \boldsymbol{C}(\boldsymbol{s}, \boldsymbol{t})\overline{\boldsymbol{C}(\boldsymbol{t}, \boldsymbol{s})}$ for $\boldsymbol{s} \neq \boldsymbol{t}$ in $\boldsymbol{A}$.

- Reversing does not change the element of the group that is represented;
  $\rightsquigarrow$ if it terminates, every element of the group is a fraction $\boldsymbol{f}\boldsymbol{g}^{-1}$ with $\boldsymbol{f}, \boldsymbol{g}$ positive.

- Example 1 = reversing for the free Abelian group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{ab} = \mathtt{ba} \rangle$;
- Example 2 = reversing for the 3-strand braid group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$;

- What are we doing? We are working with a semigroup presentation and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$. $\rightsquigarrow$ monoid $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle^{+}$, group $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$.

- Definition.— Assume (A, R) semigroup presentation and, for all $s \neq t$ in A, there is exactly one relation $s ... = t ...$ in R, say $sC(s,t) = tC(t,s)$.
Then reversing is the rewrite system on $A \cup \overline{A}$ (a copy of A, here : capitalized letters) with rules $\overline{s}s \to \varepsilon$ and $\overline{s}t \to C(s,t)\overline{C(t,s)}$ for $s \neq t$ in A.

- Reversing does not change the element of the group that is represented;
$\rightsquigarrow$ if it terminates, every element of the group is a fraction fg^{-1} with f, g positive.

- Example 1 = reversing for the free Abelian group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{ab} = \mathtt{ba} \rangle$;
- Example 2 = reversing for the 3-strand braid group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$;
- Example 3 = reversing for type $I_2(m+1)$ Artin group: $\langle \mathtt{a}, \mathtt{b} \mid \underbrace{\mathtt{abab...}}_{m+1} = \underbrace{\mathtt{baba...}}_{m+1} \rangle$;

- What are we doing? We are working with a semigroup presentation and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$. $\rightsquigarrow$ monoid $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle^+$, group $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$.

- Definition.— Assume (A, R) semigroup presentation and, for all $s \neq t$ in A, there is exactly one relation $s... = t...$ in R, say $sC(s,t) = tC(t,s)$.
Then reversing is the rewrite system on $A \cup \overline{A}$ (a copy of A, here : capitalized letters) with rules $\overline{s}s \to \varepsilon$ and $\overline{s}t \to C(s,t)\overline{C(t,s)}$ for $s \neq t$ in A.

- Reversing does not change the element of the group that is represented;
$\rightsquigarrow$ if it terminates, every element of the group is a fraction fg^{-1} with f, g positive.

- Example 1 = reversing for the free Abelian group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{ab} = \mathtt{ba} \rangle$;
- Example 2 = reversing for the 3-strand braid group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$;
- Example 3 = reversing for type $I_2(m+1)$ Artin group: $\langle \mathtt{a}, \mathtt{b} \mid \underbrace{\mathtt{abab}...}_{m+1} = \underbrace{\mathtt{baba}...}_{m+1} \rangle$;
- Example 4 = reversing for the Baumslag–Solitar group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{ab}^2 = \mathtt{ba} \rangle$;

- What are we doing? We are working with a semigroup presentation
and trying to represent the elements of the presented group by fractions.

- A semigroup presentation: list of generators (alphabet), plus list of relations, e.g., $\{\mathtt{a}, \mathtt{b}\}$, plus $\{\mathtt{aba} = \mathtt{bab}\}$. $\rightsquigarrow$ monoid $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle^+$, group $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$.

- Definition.— Assume (A, R) semigroup presentation and, for all $s \neq t$ in A,
there is exactly one relation $s... = t...$ in R, say $sC(s,t) = tC(t,s)$.
Then reversing is the rewrite system on $A \cup \overline{A}$ (a copy of A, here : capitalized letters)
with rules $\overline{s}s \to \varepsilon$ and $\overline{s}t \to C(s,t)\overline{C(t,s)}$ for $s \neq t$ in A.

- Reversing does not change the element of the group that is represented;
$\rightsquigarrow$ if it terminates, every element of the group is a fraction fg^{-1} with f, g positive.

- Example 1 = reversing for the free Abelian group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{ab} = \mathtt{ba} \rangle$;
- Example 2 = reversing for the 3-strand braid group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{aba} = \mathtt{bab} \rangle$;
- Example 3 = reversing for type $I_2(m+1)$ Artin group: $\langle \mathtt{a}, \mathtt{b} \mid \underbrace{\mathtt{abab}...}_{m+1} = \underbrace{\mathtt{baba}...}_{m+1} \rangle$;

- Example 4 = reversing for the Baumslag–Solitar group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{ab}^2 = \mathtt{ba} \rangle$;
- Example 5 = reversing for the ordered group: $\langle \mathtt{a}, \mathtt{b} \mid \mathtt{a} = \mathtt{babab}^2\mathtt{ab}^2\mathtt{abab} \rangle$.

- The only known facts:

- The only known facts:
  - reduction to the baby case $\Rightarrow$ termination;

- The only known facts:
  - reduction to the baby case $\Rightarrow$ termination;
  - self-reproducing pattern $\Rightarrow$ non-termination;

- The only known facts:
  - reduction to the baby case $\Rightarrow$ termination;
  - self-reproducing pattern $\Rightarrow$ non-termination;

  - if reversing is complete for $(\boldsymbol{A}, \boldsymbol{R})$, then it is terminating
    iff any two elements of the monoid $\langle \boldsymbol{A} \mid \boldsymbol{R} \rangle^{+}$ admit a common right-multiple.

- The only known facts:
  - reduction to the baby case $\Rightarrow$ termination;
  - self-reproducing pattern $\Rightarrow$ non-termination;
  - if reversing is complete for $(\boldsymbol{A}, \boldsymbol{R})$, then it is terminating iff any two elements of the monoid $\langle \boldsymbol{A} \mid \boldsymbol{R} \rangle^+$ admit a common right-multiple.

- Question.— What can YOU say about reversing?

For the Polish Algorithm:

• P. Dehornoy, Braids and selfdistributivity, Progress in math. vol 192, Birkhaüser 2000 (Chapter VIII)

• O. Deiser, Notes on the Polish Algorithm, deiser@tum.de (Technishe Universität München)

For the Polish Algorithm:

• P. Dehornoy, Braids and selfdistributivity, Progress in math. vol 192, Birkhaüser 2000 (Chapter VIII)

• O. Deiser, Notes on the Polish Algorithm, deiser@tum.de (Technishe Universität München)

For Handle Reduction of braids:

• P. Dehornoy, with I. Dynnikov, D. Rolfsen, B. Wiest, Braid ordering, Math. Surveys and Monographs vol. 148, Amer. Math. Soc. 2008 (Chapter V)

For the Polish Algorithm:

• P. Dehornoy, Braids and selfdistributivity, Progress in math. vol 192, Birkhaüser 2000 (Chapter VIII)

• O. Deiser, Notes on the Polish Algorithm, deiser@tum.de (Technishe Universität München)

For Handle Reduction of braids:

• P. Dehornoy, with I. Dynnikov, D. Rolfsen, B. Wiest, Braid ordering, Math. Surveys and Monographs vol. 148, Amer. Math. Soc. 2008 (Chapter V)

For reversing associated with a semigroup presentation:

• P. Dehornoy, with F. Digne, E. Godelle, D. Krammer, J. Michel, Foundations of Garside Theory, submitted www.math.unicaen.fr/∼dehornoy/ (Chapter II)

For the Polish Algorithm:

• P. Dehornoy, Braids and selfdistributivity, Progress in math. vol 192, Birkhaüser 2000 (Chapter VIII)

• O. Deiser, Notes on the Polish Algorithm, deiser@tum.de (Technishe Universität München)

For Handle Reduction of braids:

• P. Dehornoy, with I. Dynnikov, D. Rolfsen, B. Wiest, Braid ordering, Math. Surveys and Monographs vol. 148, Amer. Math. Soc. 2008 (Chapter V)

For reversing associated with a semigroup presentation:

• P. Dehornoy, with F. Digne, E. Godelle, D. Krammer, J. Michel, Foundations of Garside Theory, submitted www.math.unicaen.fr/∼dehornoy/ (Chapter II)

...venez au groupe de travail du vendredi !