Des solutions au problème d'isotopie des tresses

Des solutions au problème d'isotopie des tresses

- **Grand nombre de solutions basées sur des approches variées, algèbre, topologie, géométrie, combinatoire,**

Des solutions au problème d'isotopie des tresses

- Grand nombre de solutions basées sur des approches variées,
  algèbre, topologie, géométrie, combinatoire,
  dont certaines sont très efficaces algorihmiquement.

Des solutions au problème d'isotopie des tresses

- **Grand nombre de solutions basées sur des approches variées,**
  algèbre, topologie, géométrie, combinatoire,
  dont certaines sont très efficaces algorihmiquement.

- **Intérêt : variété des solutions :**

Des solutions au problème d'isotopie des tresses

- Grand nombre de solutions basées sur des approches variées,
  algèbre, topologie, géométrie, combinatoire,
  dont certaines sont très efficaces algorihmiquement.

- Intérêt : variété des solutions :
  $\rightsquigarrow$ richesse de la structure des groupes de tresses,

Des solutions au problème d'isotopie des tresses

- Grand nombre de solutions basées sur des approches variées,
  algèbre, topologie, géométrie, combinatoire,
  dont certaines sont très efficaces algorihmiquement.

- Intérêt : variété des solutions :
  $\rightsquigarrow$ richesse de la structure des groupes de tresses,
  $\rightsquigarrow$ grand nombre de généralisations possibles.

Solution : Représentation d'Artin

Solution : Représentation d'Artin

- **Domaine : topologie algébrique**

Solution : Représentation d'Artin

- **Domaine :** topologie algébrique
- **Point de vue :** tresse $=$ homéomorphisme

Solution : Représentation d'Artin

- Domaine : topologie algébrique
- Point de vue : tresse = homéomorphisme
- Méthode : invariant complet

Solution : Représentation d'Artin

- Domaine : topologie algébrique
- Point de vue : tresse = homéomorphisme
- Méthode : invariant complet
- Auteur : Artin '47

Solution : Représentation d'Artin

- **Domaine :** topologie algébrique
- **Point de vue :** tresse = homéomorphisme
- **Méthode :** invariant complet
- **Auteur :** Artin '47
- Mots-clés : disque avec points marqués, groupe fondamental

Solution : Représentation d'Artin

- **Domaine :** topologie algébrique
- **Point de vue :** tresse = homéomorphisme
- **Méthode :** invariant complet
- **Auteur :** Artin '47
- **Mots-clés :** disque avec points marqués, groupe fondamental
- **Arrière-plan :** théorie des surfaces, «truc d'Alexander»

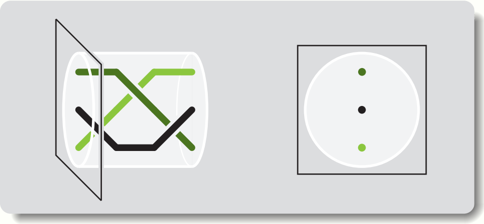
Solution : Représentation d'Artin

- **Domaine :** topologie algébrique
- **Point de vue :** tresse = homéomorphisme
- **Méthode :** invariant complet
- **Auteur :** Artin '47
- Mots-clés : disque avec points marqués, groupe fondamental
- Arrière-plan : théorie des surfaces, «truc d'Alexander»
- Extensions : mapping class groups, tresses de surface, ...

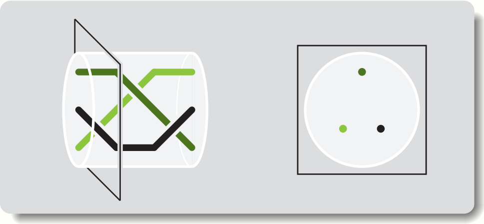
- **Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :**

- **Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :**

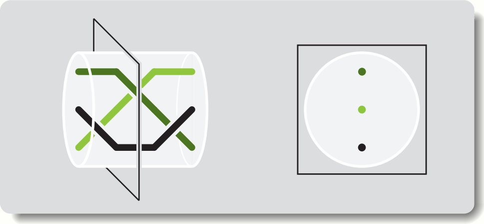
- Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :



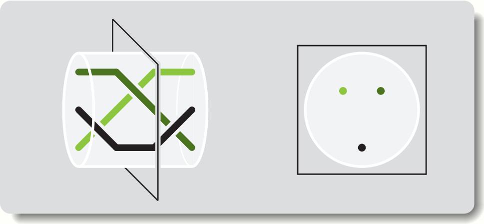
- Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :



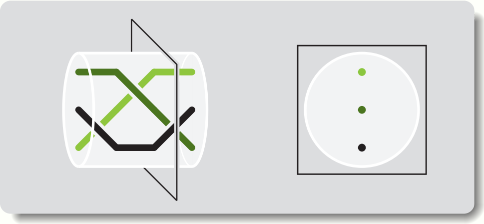
- **Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :**



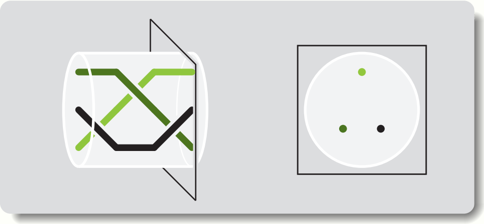
- **Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :**



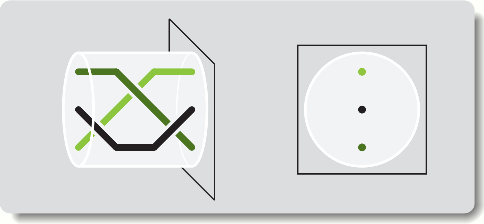
- **Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :**



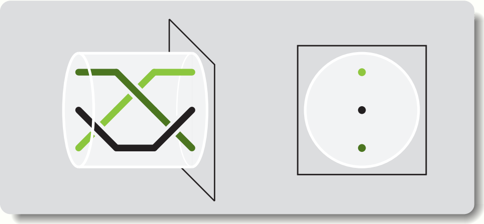
- **Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :**



- **Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :**

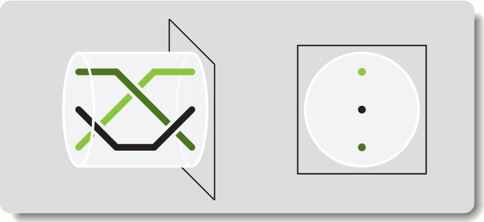


- Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :



… $\rightsquigarrow$ homéomorphisme de D_n laissant ∂D_n fixe

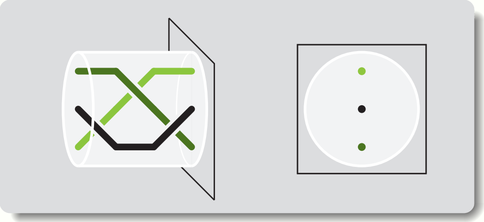
- Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :



... $\rightsquigarrow$ homéomorphisme de D_n laissant ∂D_n fixe

disque avec n points marqués — bord de D_n

- Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :

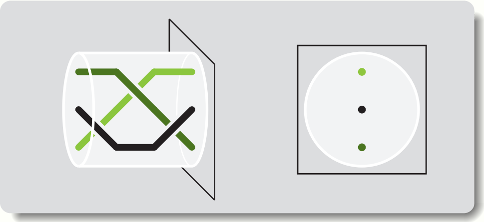


... $\rightsquigarrow$ homéomorphisme de D_n laissant ∂D_n fixe

disque avec n points marqués — bord de D_n

- Donc : $B_n \simeq$ groupe des classes d'isotopie d'homéomorphismes de D_n laissant ∂D_n fixe

- **Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :**



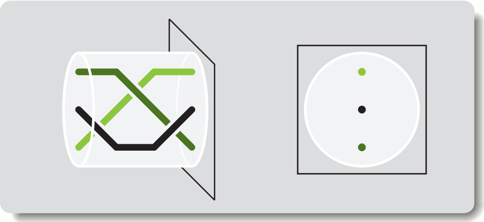
… $\rightsquigarrow$ homéomorphisme de D_n laissant ∂D_n fixe

disque avec n points marqués   bord de D_n

- **Donc : $B_n \simeq$ groupe des classes d'isotopie d'homéomorphismes de D_n laissant ∂D_n fixe**

«**mapping class group**» de D_n

- Diagramme de tresse à n brins $\rightsquigarrow$ mouvement de n points :



… $\rightsquigarrow$ homéomorphisme de D_n laissant ∂D_n fixe

disque avec n points marqués — bord de D_n

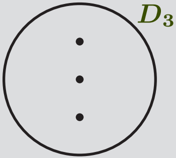
- Donc : $B_n \simeq$ groupe des classes d'isotopie d'homéomorphismes de D_n laissant ∂D_n fixe

«mapping class group» de D_n
(ou groupe de difféotopie de D_n)

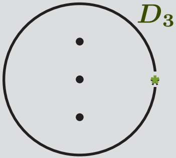
- **Tresse = homéomorphisme du disque troué D_n**

- **Tresse = homéomorphisme du disque troué D_n**
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n,

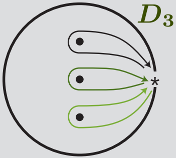
- **Tresse = homéomorphisme du disque troué D_n**
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.



- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.

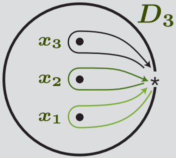


- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.



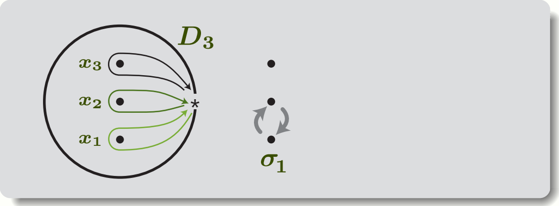
- **Tresse = homéomorphisme du disque troué D_n**

  $\rightsquigarrow$ **action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.**



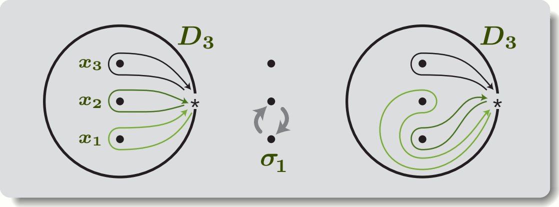
- Tresse = homéomorphisme du disque troué D_n

$\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.

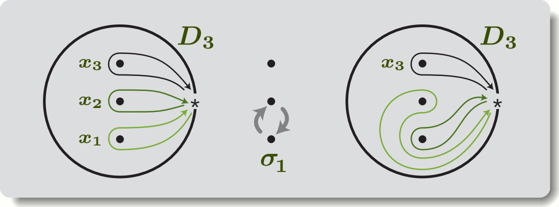


- **Tresse = homéomorphisme du disque troué D_n**

$\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n,
qui est un groupe libre de rang n.



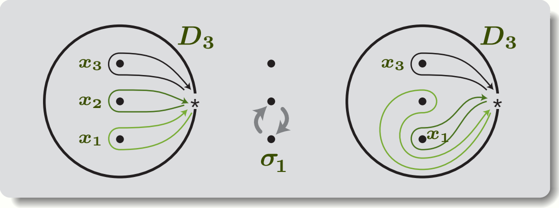
- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.



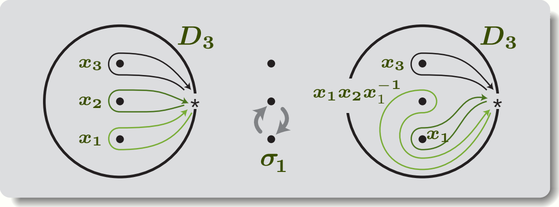
- **Tresse = homéomorphisme du disque troué D_n**

  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n,

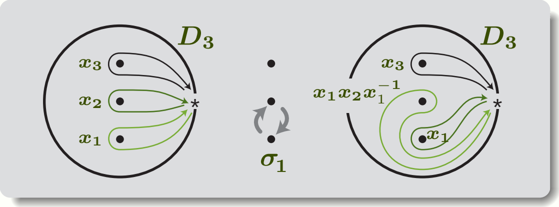
  qui est un groupe libre de rang n.



- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.

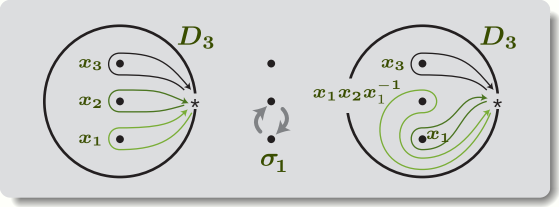


- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n,
  qui est un groupe libre de rang n.



- De là, homomorphisme ρ de B_n dans $\mathrm{Aut}(F_n)$:

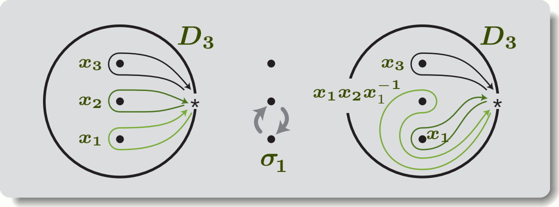
- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n,
  qui est un groupe libre de rang n.



- De là, homomorphisme ρ de B_n dans $\mathrm{Aut}(F_n)$:

$$\rho(\sigma_i) : \begin{cases} x_i \mapsto x_i x_{i+1} x_i^{-1}, \\ \\ \end{cases}$$

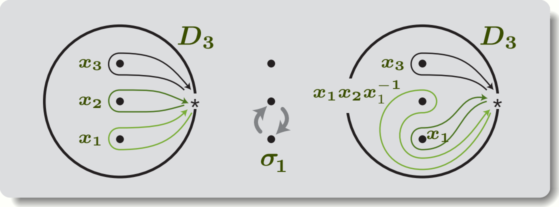
- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.



- De là, homomorphisme ρ de B_n dans $\mathrm{Aut}(F_n)$:

$$\rho(\sigma_i) : \begin{cases} x_i \mapsto x_i x_{i+1} x_i^{-1}, \\ x_{i+1} \mapsto x_i, \\ \end{cases}$$

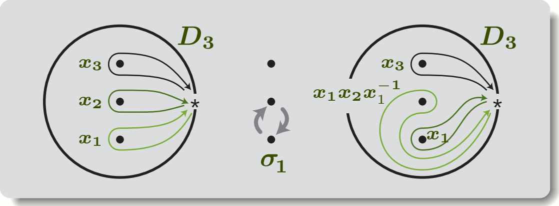
- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.



- De là, homomorphisme ρ de B_n dans $\mathrm{Aut}(F_n)$:

$$\rho(\sigma_i) : \begin{cases} x_i \mapsto x_i x_{i+1} x_i^{-1}, \\ x_{i+1} \mapsto x_i, \\ x_k \mapsto x_k \text{ pour } k \neq i, i+1. \end{cases}$$

- Tresse = homéomorphisme du disque troué D_n
  $\rightsquigarrow$ action de B_n sur le groupe fondamental de D_n, qui est un groupe libre de rang n.



- De là, homomorphisme ρ de B_n dans $\mathrm{Aut}(F_n)$:

$$\rho(\sigma_i) : \begin{cases} x_i \mapsto x_i x_{i+1} x_i^{-1}, \\ x_{i+1} \mapsto x_i, \\ x_k \mapsto x_k \text{ pour } k \neq i, i+1. \end{cases}$$

Théorème (Artin) : L'homomorphisme ρ est injectif.

Algorithme : Partant de w mot de tresse à n brins,

Algorithme : Partant de w mot de tresse à n brins,

- (i) Calculer, pour $i = 1, ..., n$, l'image de x_i par $\rho(w)$;

Algorithme : Partant de w mot de tresse à n brins,

- (i) Calculer, pour $i = 1, ..., n$, l'image de x_i par $\rho(w)$;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\rho(w)(x_i) = x_i$ pour tout i.

Algorithme : Partant de w mot de tresse à n brins,
- (i) Calculer, pour $i = 1, ..., n$, l'image de x_i par $\rho(w)$;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\rho(w)(x_i) = x_i$ pour tout i.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.

Algorithme : Partant de w mot de tresse à n brins,

- (i) Calculer, pour $i = 1, ..., n$, l'image de x_i par $\rho(w)$;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\rho(w)(x_i) = x_i$ pour tout i.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.

- (i) $\rho(w)(x_1) = x_1 x_2 x_1^{-1} x_3 x_1 x_2^{-1} x_1^{-1} x_3^{-1} x_1 x_3 x_1 x_2$
  $\dots\ x_1^{-1} x_3^{-1} x_1 x_2^{-1} x_1^{-1}$;

Algorithme : Partant de w mot de tresse à n brins,

- (i) Calculer, pour $i = 1, ..., n$, l'image de x_i par $\rho(w)$;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\rho(w)(x_i) = x_i$ pour tout i.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.

- (i) $\rho(w)(x_1) = x_1x_2x_1^{-1}x_3x_1x_2^{-1}x_1^{-1}x_3^{-1}x_1x_3x_1x_2$ ... $x_1^{-1}x_3^{-1}x_1x_2^{-1}x_1^{-1}$;
- (ii) ... $= x_1$? non, donc $w \not\equiv \varepsilon$.

Algorithme : Partant de w mot de tresse à n brins,
- (i) Calculer, pour $i = 1, ..., n$, l'image de x_i par $\rho(w)$;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\rho(w)(x_i) = x_i$ pour tout i.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.

- (i) $\rho(w)(x_1) = x_1 x_2 x_1^{-1} x_3 x_1 x_2^{-1} x_1^{-1} x_3^{-1} x_1 x_3 x_1 x_2$ ... $x_1^{-1} x_3^{-1} x_1 x_2^{-1} x_1^{-1}$;
- (ii) ... $= x_1$? non, donc $w \not\equiv \varepsilon$.

• Complexité exponentielle : toujours aussi calamiteux ;

Algorithme : Partant de w mot de tresse à n brins,
- (i) Calculer, pour $i = 1, ..., n$, l'image de x_i par $\rho(w)$;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\rho(w)(x_i) = x_i$ pour tout i.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.

- (i) $\rho(w)(x_1) = x_1 x_2 x_1^{-1} x_3 x_1 x_2^{-1} x_1^{-1} x_3^{-1} x_1 x_3 x_1 x_2$
  $\ldots\ x_1^{-1} x_3^{-1} x_1 x_2^{-1} x_1^{-1}$;
- (ii) $\ldots\ = x_1$? non, donc $w \not\equiv \varepsilon$.

- Complexité exponentielle : toujours aussi calamiteux ;
- Lien avec le coloriage des tresses ;

Algorithme : Partant de w mot de tresse à n brins,
- (i) Calculer, pour $i = 1, ..., n$, l'image de x_i par $\rho(w)$;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\rho(w)(x_i) = x_i$ pour tout i.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$.
- (i) $\rho(w)(x_1) = x_1x_2x_1^{-1}x_3x_1x_2^{-1}x_1^{-1}x_3^{-1}x_1x_3x_1x_2$ ... $x_1^{-1}x_3^{-1}x_1x_2^{-1}x_1^{-1}$;
- (ii) ... $= x_1$? non, donc $w \not\equiv \varepsilon$.

- Complexité exponentielle : toujours aussi calamiteux ;
- Lien avec le coloriage des tresses ;
- Calcul différentiel libre $\rightsquigarrow$ représentation de Burau, non fidèle.

Solution : Représentation linéaire

Solution : Représentation linéaire

- **Domaine :** algèbre linéaire

Solution : Représentation linéaire

- Domaine : algèbre linéaire
- Point de vue : tresse = matrice

Solution : Représentation linéaire

- **Domaine : algèbre linéaire**
- **Point de vue : tresse = matrice**
- **Méthode : représentation**

Solution : Représentation linéaire

- Domaine : algèbre linéaire
- Point de vue : tresse = matrice
- Méthode : représentation
- Auteurs : Burau ’36, Krammer ’00, Bigelow ’00

Solution : Représentation linéaire

- Domaine : algèbre linéaire
- Point de vue : tresse = matrice
- Méthode : représentation
- Auteurs : Burau '36, Krammer '00, Bigelow '00
- Mots-clés : coloriage de tresse, opérations auto-distributives

Solution : Représentation linéaire

- Domaine : algèbre linéaire
- Point de vue : tresse = matrice
- Méthode : représentation
- Auteurs : Burau '36, Krammer '00, Bigelow '00
- Mots-clés : coloriage de tresse, opérations auto-distributives
- Arrière-plan : théorie de Coxeter

Solution : Représentation linéaire

- Domaine : algèbre linéaire
- Point de vue : tresse = matrice
- Méthode : représentation
- Auteurs : Burau '36, Krammer '00, Bigelow '00
- Mots-clés : coloriage de tresse, opérations auto-distributives
- Arrière-plan : théorie de Coxeter
- Extensions : groupes et monoïdes d'Artin–Tits

- **Colorier un diagramme : propager les couleurs depuis la gauche ;**

Coloriage de tresses

- **Colorier un diagramme : propager les couleurs depuis la gauche ;**
- **Règle 1 :** 

- **Colorier un diagramme : propager les couleurs depuis la gauche ;**
- **Règle 1 :** 

$\rightsquigarrow$ **projection de B_n sur $\mathfrak{S}_n$**

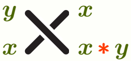
- **Colorier** un diagramme : propager les couleurs depuis la gauche ;

- Règle 1 : 

$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$

  pas de solution au problème d'isotopie

- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

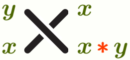
$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

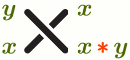
- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

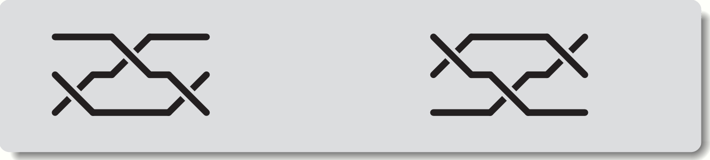
Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.

- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

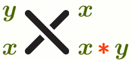
$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.

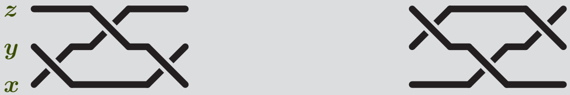


- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

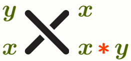
$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.

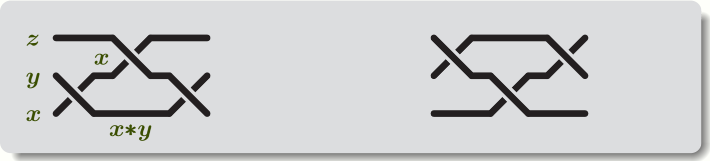


- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

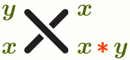
$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.

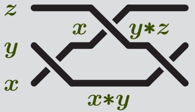


- Colorier un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

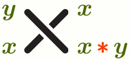
$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.

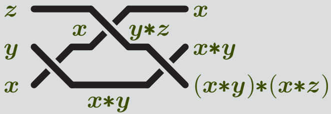


- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

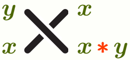
$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.

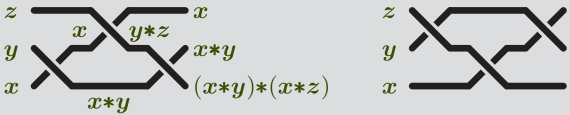


- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

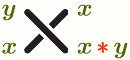
$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.

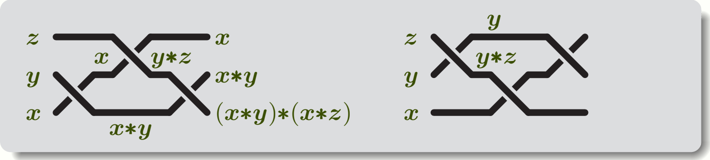


- **Colorier** un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

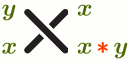
$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.

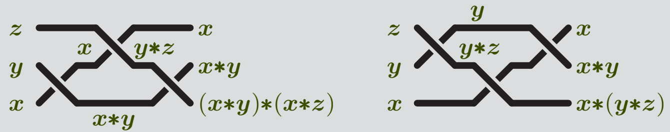


- Colorier un diagramme : propager les couleurs depuis la gauche ;
- Règle 1 : 

$\rightsquigarrow$ projection de B_n sur $\mathfrak{S}_n$
pas de solution au problème d'isotopie
- Règle 2 : 

$\rightsquigarrow$ compatibilité avec relations de tresse ?

Lemme : La règle 2 donne un coloriage de B_n^+ si et seulement si l'opération $*$ satisfait la loi $x * (y * z) = (x * y) * (x * z)$.



- **Opérations satisfaisant** $x * (y * z) = (x * y) * (x * z)$ **?**

- **Opérations satisfaisant** $x * (y * z) \underset{\uparrow}{=} (x * y) * (x * z)$ **?**

« **autodistributive** »

- **Opérations satisfaisant $x * (y * z) \underset{\uparrow}{=} (x * y) * (x * z)$?**

  « **autodistributive** »

- Opération triviale : $x * y = y$ (= règle 1) :
  $\rightsquigarrow$ projection sur une permutation — pas solution au pb. d'isot.

- **Opérations satisfaisant** $x * (y * z) \underset{\substack{\uparrow \\ \text{« autodistributive »}}}{=} (x * y) * (x * z)$ **?**

- **Opération triviale :** $x * y = y$ **(= règle 1) :**
  $\rightsquigarrow$ **projection sur une permutation — pas solution au pb. d'isot.**

- **Conjugaison sur un groupe :** $x * y = xyx^{-1}$ **;**

- **Opérations satisfaisant** $x * (y * z) = (x * y) * (x * z)$ **?**

  « **autodistributive** »

- Opération triviale : $x * y = y$ (= règle 1) :
  ⇝ projection sur une permutation — pas solution au pb. d'isot.

- **Conjugaison** sur un groupe : $x * y = xyx^{-1}$;

  Pour F_n (groupe libre) : $B_n \to \mathrm{Aut}(F_n)$:
  ⇝ représentation d'Artin

- Opérations satisfaisant $x * (y * z) = (x * y) * (x * z)$?

  $\uparrow$

  « **autodistributive** »

- Opération triviale : $x * y = y$ (= règle 1) :

  $\rightsquigarrow$ projection sur une permutation — pas solution au pb. d'isot.

- Conjugaison sur un groupe : $x * y = xyx^{-1}$;

  Pour F_n (groupe libre) : $B_n \to \mathrm{Aut}(F_n)$:

  $\rightsquigarrow$ représentation d'Artin — solution du problème d'isotopie.

- Opérations satisfaisant $x * (y * z) = (x * y) * (x * z)$?

  « **autodistributive** »

- Opération triviale : $x * y = y$ (= règle 1) :
  ⇝ projection sur une permutation — pas solution au pb. d'isot.

- **Conjugaison** sur un groupe : $x * y = xyx^{-1}$;
  Pour F_n (groupe libre) : $B_n \to \mathrm{Aut}(F_n)$:
  ⇝ représentation d'Artin — solution du problème d'isotopie.

- **Moyenne** sur un $\mathbb{Z}[t]$-module : $x * y = (1 - t)x + ty$.

- Opérations satisfaisant $x * (y * z) \underset{\uparrow}{=} (x * y) * (x * z)$?

  « autodistributive »

- Opération triviale : $x * y = y$ (= règle 1) :
  ⇝ projection sur une permutation — pas solution au pb. d'isot.

- Conjugaison sur un groupe : $x * y = xyx^{-1}$;

  Pour F_n (groupe libre) : $B_n \to \mathrm{Aut}(F_n)$:
  ⇝ représentation d'Artin — solution du problème d'isotopie.

- Moyenne sur un $\mathbb{Z}[t]$-module : $x * y = (1 - t)x + ty$.

  Alors sorties = combinaison linéaire des entrées, donc matrice :

- Opérations satisfaisant $x * (y * z) = (x * y) * (x * z)$?

  « autodistributive »

- Opération triviale : $x * y = y$ (= règle 1) :
  ⇝ projection sur une permutation — pas solution au pb. d'isot.

- Conjugaison sur un groupe : $x * y = xyx^{-1}$;

  Pour F_n (groupe libre) : $B_n \to \mathrm{Aut}(F_n)$:
  ⇝ représentation d'Artin — solution du problème d'isotopie.

- Moyenne sur un $\mathbb{Z}[t]$-module : $x * y = (1 - t)x + ty$.

  Alors sorties = combinaison linéaire des entrées, donc matrice :
  ⇝ représentation de Burau : $B_n \to \mathrm{GL}_n(\mathbb{Z}[t, t^{-1}])$.

- **La représentation de Burau résout-elle le problème d'isotopie ?**

- **La représentation de Burau résout-elle le problème d'isotopie ?**
  Est-ce une représentation fidèle (= injective) de B_n ?
- **Oui pour $n = 3$, mais**

- La représentation de Burau résout-elle le problème d'isotopie ?
  Est-ce une représentation fidèle (= injective) de B_n ?
- Oui pour $n = 3$, mais

Théorème (Moody '91, ...) La représentation de Burau de B_n n'est pas fidèle pour $n \geqslant 5$.

- **La représentation de Burau résout-elle le problème d'isotopie ?**
  Est-ce une représentation fidèle (= injective) de B_n ?

- Oui pour $n = 3$, mais

Théorème (**Moody** '91, ...) La représentation de Burau de B_n n'est pas fidèle pour $n \geqslant 5$.

- **Autres représentations ?**

- **La représentation de Burau résout-elle le problème d'isotopie ?**
  Est-ce une représentation fidèle (= injective) de B_n ?
- Oui pour $n = 3$, mais

Théorème (Moody '91, ...) La représentation de Burau de B_n n'est pas fidèle pour $n \geqslant 5$.

- **Autres représentations ?**

Théorème (Krammer '00, Bigelow '00) : Il existe une représentation linéaire fidèle de B_n dans $\mathrm{GL}_{n(n-1)/2}(\mathbb{Z}[t, t^{-1}, q, q^{-1}])$.

- La représentation de Burau résout-elle le problème d'isotopie ?

  Est-ce une représentation fidèle (= injective) de B_n ?

- Oui pour $n = 3$, mais

Théorème (Moody '91, ...) La représentation de Burau de B_n n'est pas fidèle pour $n \geqslant 5$.

- Autres représentations ?

Théorème (Krammer '00, Bigelow '00) : Il existe une représentation linéaire fidèle de B_n dans $\mathrm{GL}_{n(n-1)/2}(\mathbb{Z}[t, t^{-1}, q, q^{-1}])$.

- Donc solution au problème d'isotopie.

- La représentation de Burau résout-elle le problème d'isotopie ?

  Est-ce une représentation fidèle (= injective) de B_n ?

- Oui pour $n = 3$, mais

Théorème (Moody '91, ...) La représentation de Burau de B_n n'est pas fidèle pour $n \geqslant 5$.

- Autres représentations ?

Théorème (Krammer '00, Bigelow '00) : Il existe une représentation linéaire fidèle de B_n dans $\mathrm{GL}_{n(n-1)/2}(\mathbb{Z}[t, t^{-1}, q, q^{-1}])$.

- Donc solution au problème d'isotopie.

  — complexité polynomiale, mais inefficace en pratique.

Solution : « greedy normal form »

Solution : « greedy normal form »

- **Domaine :** combinatoire du groupe symétrique

Solution : « greedy normal form »

- **Domaine :** combinatoire du groupe symétrique
- **Point de vue :** tresse $=$ suite de permutations

Solution : « greedy normal form »

- Domaine : combinatoire du groupe symétrique
- Point de vue : tresse = suite de permutations
- Méthode : forme normale

Solution : « greedy normal form »

- **Domaine :** combinatoire du groupe symétrique
- **Point de vue :** tresse = suite de permutations
- **Méthode :** forme normale
- **Auteurs :** Adjan ’84, Thurston ’88, Morton–El Rifai ’88

Solution : « greedy normal form »

- Domaine : combinatoire du groupe symétrique
- Point de vue : tresse = suite de permutations
- Méthode : forme normale
- Auteurs : Adjan '84, Thurston '88, Morton–El Rifai '88
- Mots-clés : permutation, descentes, divisibilité, pgcd

Solution : « greedy normal form »

- Domaine : combinatoire du groupe symétrique
- Point de vue : tresse = suite de permutations
- Méthode : forme normale
- Auteurs : Adjan '84, Thurston '88, Morton–El Rifai '88
- Mots-clés : permutation, descentes, divisibilité, pgcd
- Arrière-plan : théorie des groupes automatiques

Solution : « greedy normal form »

- **Domaine :** combinatoire du groupe symétrique
- **Point de vue :** tresse = suite de permutations
- **Méthode :** forme normale
- **Auteurs :** Adjan '84, Thurston '88, Morton–El Rifai '88
- Mots-clés : permutation, descentes, divisibilité, pgcd
- Arrière-plan : théorie des groupes automatiques
- Extensions : groupes de Garside, ...

- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?

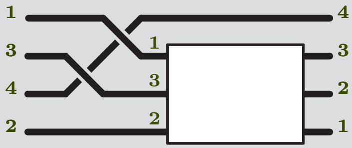
- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  = choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  = choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

Exemple : $f = (2, 4, 3, 1)$:

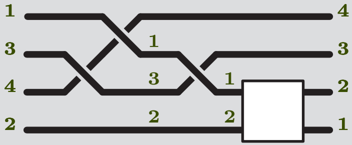
- Section pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  = choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

Exemple : $f = (2, 4, 3, 1)$:



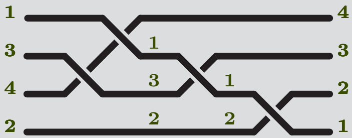
- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  = choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

Exemple : $f = (2, 4, 3, 1)$:



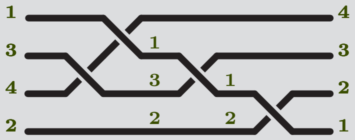
- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  = choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

Exemple : $f = (2, 4, 3, 1)$:



- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  = choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

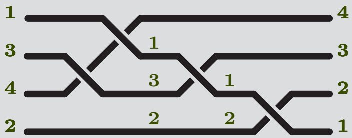
Exemple : $f = (2, 4, 3, 1)$:



$\rightsquigarrow \quad [f] = \sigma_2\sigma_3\sigma_2\sigma_1$.

- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  $=$ choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

Exemple : $f = (2, 4, 3, 1)$:

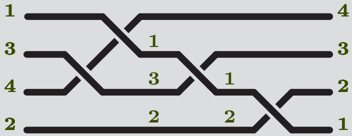


$\rightsquigarrow \quad [f] = \sigma_2\sigma_3\sigma_2\sigma_1.$

- Alors $[f]$ est dans B_n^+, et longueur de $[f] = \#$ inversions de f.

- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  = choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

Exemple : $f = (2, 4, 3, 1)$:



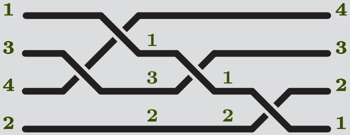
$\rightsquigarrow \quad [f] = \sigma_2\sigma_3\sigma_2\sigma_1.$

- Alors $[f]$ est dans B_n^+, et longueur de $[f] = \#$ inversions de f.

Proposition : Les $n!$ tresses de permutation dans B_n^+ sont les diviseurs de Δ_n dans le monoïde B_n^+.

- **Section** pour la projection de B_n sur le groupe symétrique $\mathfrak{S}_n$?
  $=$ choisir pour chaque permutation f une tresse $[f]$ qui la réalise.

Exemple : $f = (2, 4, 3, 1)$:



$\rightsquigarrow$ $[f] = \sigma_2\sigma_3\sigma_2\sigma_1$.

- Alors $[f]$ est dans B_n^+, et longueur de $[f]$ $=$ # inversions de f.

Proposition : Les $n!$ tresses de permutation dans B_n^+ sont les diviseurs de Δ_n dans le monoïde B_n^+.

$\uparrow$

x divise y à gauche s'il existe z vérifiant $y = xz$

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

↑
existence de ppcm et de pgcd

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

↑
existence de ppcm et de pgcd

• Corollaire : Pour tout β dans B_n^+, il existe une unique tresse de permutation maximale divisant β à gauche, à savoir $\mathrm{pgcd}(\beta, \Delta_n)$.

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

↑
existence de ppcm et de pgcd

- Corollaire : Pour tout β dans B_n^+, il existe une unique tresse de permutation maximale divisant β à gauche, à savoir $\mathrm{pgcd}(\beta, \Delta_n)$.

- Décomposition distinguée en produit de tresses de permutation :

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

↑
existence de ppcm et de pgcd

- Corollaire : Pour tout β dans B_n^+, il existe une unique tresse de permutation maximale divisant β à gauche, à savoir $\mathrm{pgcd}(\beta, \Delta_n)$.

- Décomposition distinguée en produit de tresses de permutation :

$$\beta = [f_1].\beta'$$

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

↑
existence de ppcm et de pgcd

- Corollaire : Pour tout β dans B_n^+, il existe une unique tresse de permutation maximale divisant β à gauche, à savoir $\mathrm{pgcd}(\beta, \Delta_n)$.

- Décomposition distinguée en produit de tresses de permutation :

$$\beta = [f_1].\beta' = [f_1][f_2].\beta''$$

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

↑
existence de ppcm et de pgcd

- Corollaire : Pour tout β dans B_n^+, il existe une unique tresse de permutation maximale divisant β à gauche, à savoir $\mathrm{pgcd}(\beta, \Delta_n)$.

- Décomposition distinguée en produit de tresses de permutation :

$$\beta = [f_1].\beta' = [f_1][f_2].\beta'' = ... = [f_1][f_2]...[f_d].$$

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

$\uparrow$
existence de ppcm et de pgcd

- Corollaire : Pour tout β dans B_n^+, il existe une unique tresse de permutation maximale divisant β à gauche, à savoir $\mathrm{pgcd}(\beta, \Delta_n)$.

- Décomposition distinguée en produit de tresses de permutation :

$$\beta = [f_1].\beta' = [f_1][f_2].\beta'' = \ldots = [f_1][f_2]\ldots[f_d].$$

Proposition (Thurston, Morton ... '88) : Toute tresse de B_n admet une unique expression $\Delta_n^p[f_1][f_2]\ldots[f_d]$ t.q. $f_1 \neq (n, \ldots, 1)$, $f_d \neq \mathrm{id}$, et, pour chaque r, tout recul de f_{r+1} est une descente de f_r.

Théorème (Garside '67) : Le monoïde B_n^+ équipé de la relation de divisibilité à gauche est un treillis.

↑
existence de ppcm et de pgcd

- **Corollaire :** Pour tout β dans B_n^+, il existe une unique tresse de permutation maximale divisant β à gauche, à savoir $\mathrm{pgcd}(\beta, \Delta_n)$.

- Décomposition distinguée en produit de tresses de permutation :

$$\beta = [f_1].\beta' = [f_1][f_2].\beta'' = \ldots = [f_1][f_2]\ldots[f_d].$$

Proposition (Thurston, Morton ... '88) : Toute tresse de B_n admet une unique expression $\Delta_n^p[f_1][f_2]\ldots[f_d]$ t.q. $f_1 \neq (n, \ldots, 1)$, $f_d \neq \mathrm{id}$, et, pour chaque r, tout recul de f_{r+1} est une descente de f_r.

↑
$f_{r+1}^{-1}(i) > f_{r+1}^{-1}(i+1)$ entraîne $f_r(i) > f_r(i+1)$

- **Nombre fini de permutations + normalité locale (descentes)**
  $\rightsquigarrow$ $\mathbf{NF}(w\sigma_i^{\pm 1})$ **obtenu depuis** $\mathbf{NF}(w)$ **et** $\sigma_i^{\pm 1}$ **par transducteur :**

- **Nombre fini de permutations $+$ normalité locale (descentes)**
  $\rightsquigarrow$ $\mathbf{NF}(w\boldsymbol{\sigma_i^{\pm 1}})$ obtenu depuis $\mathbf{NF}(w)$ et $\boldsymbol{\sigma_i^{\pm 1}}$ par transducteur :
  structure automatique du groupe B_n.

- Nombre fini de permutations + normalité locale (descentes)
  $\rightsquigarrow$ $\mathrm{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathrm{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,

- Nombre fini de permutations + normalité locale (descentes)
  $\rightsquigarrow$ $\mathbf{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathbf{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,
- (i) Calculer la forme normale $\mathbf{NF}(w)$ de w incrémentalement ;

- Nombre fini de permutations $+$ normalité locale (descentes)
  $\rightsquigarrow$ $\mathbf{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathbf{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,
- (i) Calculer la forme normale $\mathbf{NF}(w)$ de w incrémentalement ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\mathbf{NF}(w) = [\mathrm{id}]$.

- Nombre fini de permutations + normalité locale (descentes)
  $\rightsquigarrow$ $\mathrm{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathrm{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,
- (i) Calculer la forme normale $\mathrm{NF}(w)$ de w incrémentalement ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\mathrm{NF}(w) = [\mathrm{id}]$.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.

- Nombre fini de permutations + normalité locale (descentes)
  $\rightsquigarrow$ $\mathrm{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathrm{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,
- (i) Calculer la forme normale $\mathrm{NF}(w)$ de w incrémentalement ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\mathrm{NF}(w) = [\mathrm{id}]$.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.
- (i) $\mathrm{NF}(\sigma_2^{-1}) = \Delta_3^{-1}.[3, 1, 2]$,

- Nombre fini de permutations + normalité locale (descentes)
  $\rightsquigarrow$ $\mathrm{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathrm{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,
- (i) Calculer la forme normale $\mathrm{NF}(w)$ de w incrémentalement ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\mathrm{NF}(w) = [\mathrm{id}]$.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.
- (i) $\mathrm{NF}(\sigma_2^{-1}) = \Delta_3^{-1}.[3,1,2]$, $\quad \mathrm{NF}(\sigma_2^{-2}) = ...,$

- Nombre fini de permutations + normalité locale (descentes)
  ⇝ $\mathrm{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathrm{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,
- (i) Calculer la forme normale $\mathrm{NF}(w)$ de w incrémentalement ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\mathrm{NF}(w) = [\mathrm{id}]$.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$.
- (i) $\mathrm{NF}(\sigma_2^{-1}) = \Delta_3^{-1}.[3,1,2]$, $\quad \mathrm{NF}(\sigma_2^{-2}) = \ldots$,

  ... $\mathrm{NF}(w) = \Delta_3^{-3}.[3,1,2].[2,1,3].[2,3,1].[1,3,2].[3,1,2].[2,1,3]$.
- (ii) ... $= [1,2,3]$? non, donc $w \not\equiv \varepsilon$.

- Nombre fini de permutations + normalité locale (descentes)
  $\rightsquigarrow$ $\mathrm{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathrm{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,
- (i) Calculer la forme normale $\mathrm{NF}(w)$ de w incrémentalement ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\mathrm{NF}(w) = [\mathrm{id}]$.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$.
- (i) $\mathrm{NF}(\sigma_2^{-1}) = \Delta_3^{-1}.[3,1,2]$, $\quad \mathrm{NF}(\sigma_2^{-2}) = \dots$,

  ... $\mathrm{NF}(w) = \Delta_3^{-3}.[3,1,2].[2,1,3].[2,3,1].[1,3,2].[3,1,2].[2,1,3]$.
- (ii) ... $= [1,2,3]$? non, donc $w \not\equiv \varepsilon$.

- Complexité quadratique à n fixé — donc utilisable.

- Nombre fini de permutations + normalité locale (descentes)
  $\rightsquigarrow$ $\mathrm{NF}(w\sigma_i^{\pm 1})$ obtenu depuis $\mathrm{NF}(w)$ et $\sigma_i^{\pm 1}$ par transducteur :
  structure automatique du groupe B_n.

Algorithme : Partant de w mot de tresse,
- (i) Calculer la forme normale $\mathrm{NF}(w)$ de w incrémentalement ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $\mathrm{NF}(w) = [\mathrm{id}]$.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$.
- (i) $\mathrm{NF}(\sigma_2^{-1}) = \Delta_3^{-1}.[3,1,2]$, $\mathrm{NF}(\sigma_2^{-2}) = \ldots$,

  ... $\mathrm{NF}(w) = \Delta_3^{-3}.[3,1,2].[2,1,3].[2,3,1].[1,3,2].[3,1,2].[2,1,3]$.
- (ii) ... $= [1,2,3]$? non, donc $w \not\equiv \varepsilon$.

- Complexité quadratique à n fixé — donc utilisable.
- Existe en version symétrique : deux suites de permutations.

Solution : Retournement de mot

Solution : Retournement de mot

- **Domaine :** système de récriture

Solution : Retournement de mot

- **Domaine :** système de récriture
- **Point de vue :** tresse = diagramme de van Kampen

Solution : Retournement de mot

- **Domaine :** système de récriture
- **Point de vue :** tresse $=$ diagramme de van Kampen
- **Méthode :** réduction

Solution : Retournement de mot

- Domaine : système de récriture
- Point de vue : tresse = diagramme de van Kampen
- Méthode : réduction
- Auteur : '91 $\rightsquigarrow$

Solution : Retournement de mot

- **Domaine :** système de récriture
- **Point de vue :** tresse = diagramme de van Kampen
- **Méthode :** réduction
- **Auteur :** ’91 $\rightsquigarrow$
- **Mots-clés :** confluence, terminaison, complétude

Solution : Retournement de mot

- **Domaine :** système de récriture
- **Point de vue :** tresse = diagramme de van Kampen
- **Méthode :** réduction
- **Auteur :** '91 $\rightsquigarrow$
- Mots-clés : confluence, terminaison, complétude
- Arrière-plan : structure de Garside

Solution : Retournement de mot

- **Domaine :** système de récriture
- **Point de vue :** tresse = diagramme de van Kampen
- **Méthode :** réduction
- **Auteur :** '91 $\rightsquigarrow$
- Mots-clés : confluence, terminaison, complétude
- Arrière-plan : structure de Garside
- Extensions : semigroupes

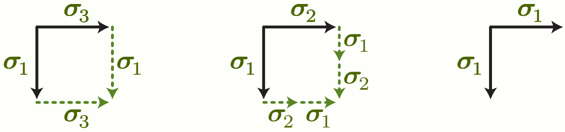
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles

$$\begin{tikzcd} \cdot \arrow[r, "\sigma_3"] \arrow[d, "\sigma_1"'] & {} \\ {} & \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[r, "\sigma_2"] \arrow[d, "\sigma_1"'] & {} \\ {} & \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[r, "\sigma_1"] \arrow[d, "\sigma_1"'] & {} \\ {} & \end{tikzcd}$$

- **Associer à un mot de tresse un escalier :** $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- **Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles**

$$\begin{tikzcd} \cdot \arrow[r, "\sigma_3"] \arrow[d, "\sigma_1"'] & \cdot \arrow[d, dashed, "\sigma_1"] \\ \cdot \arrow[r, dashed, "\sigma_3"'] & \cdot \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[r, "\sigma_2"] \arrow[d, "\sigma_1"'] & \cdot \\ \cdot & \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[r, "\sigma_1"] \arrow[d, "\sigma_1"'] & \cdot \\ \cdot & \end{tikzcd}$$

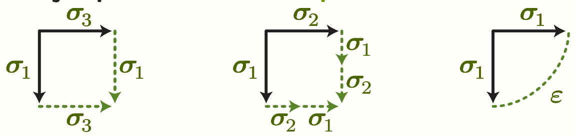
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



- **Associer à un mot de tresse un escalier :** $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \Big\downarrow \sigma_i$.
- **Compléter jusqu'à obtenir une équerre** $\lrcorner$ **à l'aide des règles**

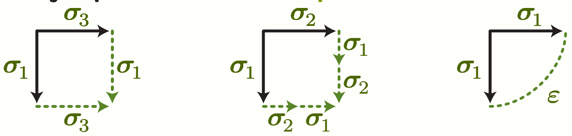
$$\begin{tikzcd} \bullet \arrow[r, "\sigma_3"] \arrow[d, "\sigma_1"'] & \bullet \arrow[d, dashed, "\sigma_1"] \\ \bullet \arrow[r, dashed, "\sigma_3"'] & \bullet \end{tikzcd} \qquad \begin{tikzcd} \bullet \arrow[rr, "\sigma_2"] \arrow[dd, "\sigma_1"'] & & \bullet \arrow[d, dashed, "\sigma_1"] \\ & & \bullet \arrow[d, dashed, "\sigma_2"] \\ \bullet \arrow[r, dashed, "\sigma_2"'] & \bullet \arrow[r, dashed, "\sigma_1"'] & \bullet \end{tikzcd} \qquad \begin{tikzcd} \bullet \arrow[r, "\sigma_1"] \arrow[d, "\sigma_1"'] & \bullet \arrow[dl, dashed, bend left, no head, "\varepsilon"] \\ \bullet & \end{tikzcd}$$

- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \Big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre $\lrcorner$ à l'aide des règles



- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$

- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$

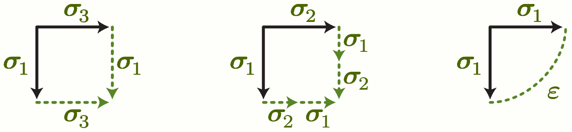
- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** $\lrcorner$ à l'aide des règles

$$\begin{tikzcd} \cdot \arrow[r, "\sigma_3"] \arrow[d, "\sigma_1"'] & \cdot \arrow[d, dashed, "\sigma_1"] \\ \cdot \arrow[r, dashed, "\sigma_3"'] & \cdot \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[rr, "\sigma_2"] \arrow[dd, "\sigma_1"'] & & \cdot \arrow[d, dashed, "\sigma_1"] \\ & & \cdot \arrow[d, dashed, "\sigma_2"] \\ \cdot \arrow[r, dashed, "\sigma_2"'] & \cdot \arrow[r, dashed, "\sigma_1"'] & \cdot \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[r, "\sigma_1"] \arrow[d, "\sigma_1"'] & \cdot \\ \cdot \arrow[ur, dashed, no head, bend right, "\varepsilon"'] & \end{tikzcd}$$

- **Exemple** :
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$

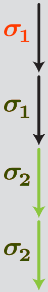
$$\begin{tikzcd} \cdot \arrow[d, "\sigma_1"'] \\ \cdot \arrow[d, "\sigma_2"'] \\ \cdot \arrow[d, "\sigma_2"'] \\ \cdot \end{tikzcd}$$

- **Associer à un mot de tresse un escalier :** $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- **Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles**

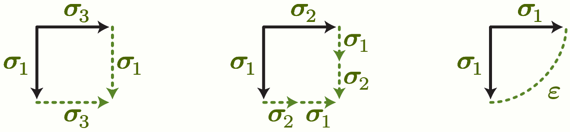


- **Exemple :**

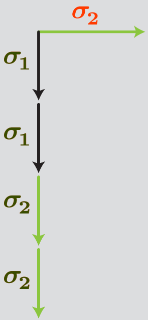
$w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$



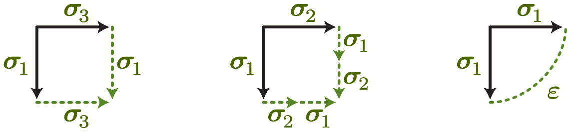
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



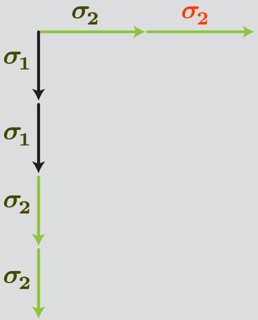
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



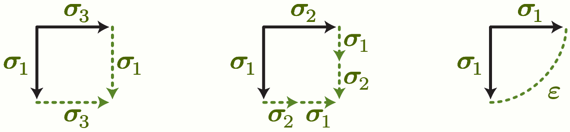
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre $\lrcorner$ à l'aide des règles



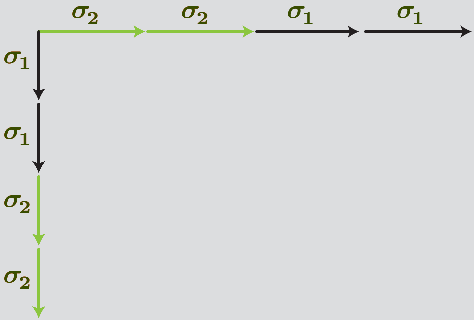
- Exemple :
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



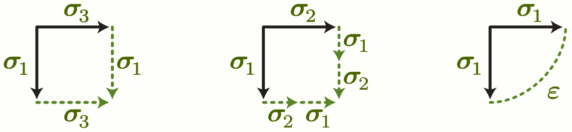
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



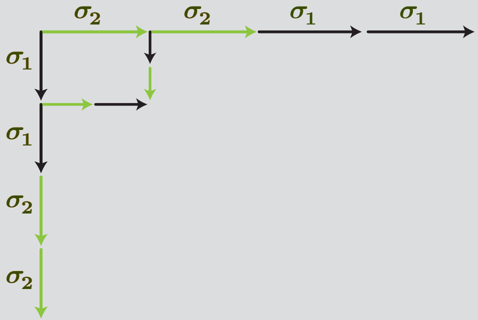
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$



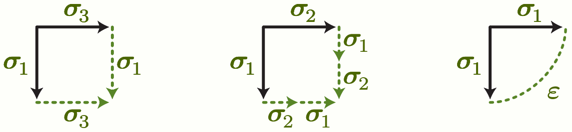
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



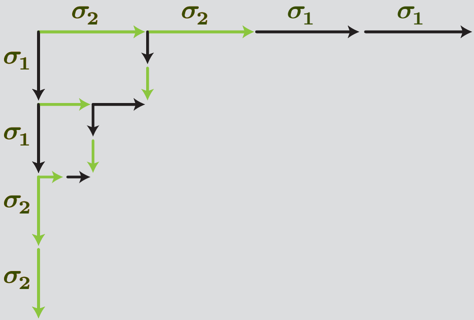
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



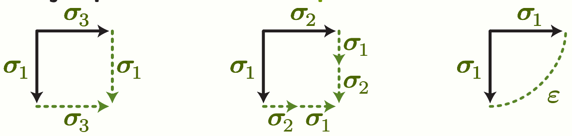
- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** ⌟ à l'aide des règles



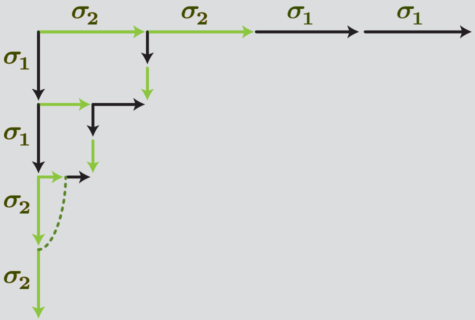
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$



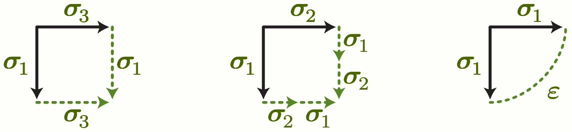
- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** ⌟ à l'aide des règles



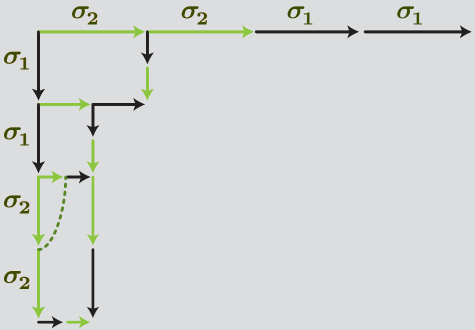
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



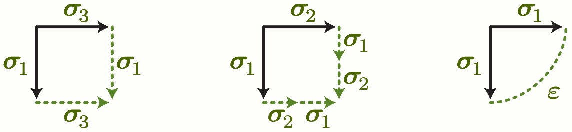
- **Associer à un mot de tresse un escalier :** $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- **Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles**



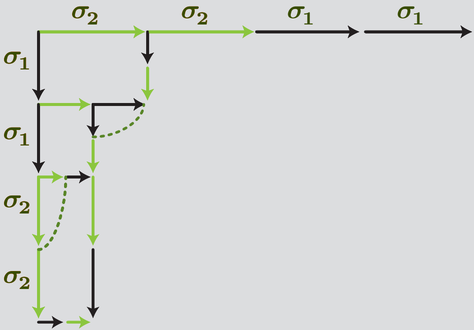
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



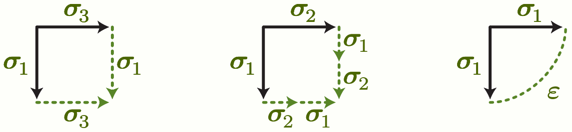
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \downarrow\sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



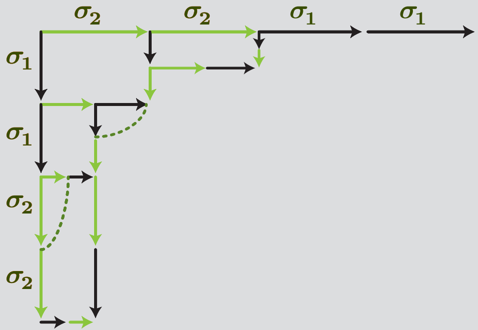
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



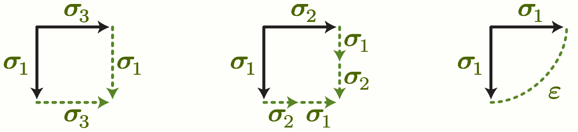
- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** ⌟ à l'aide des règles



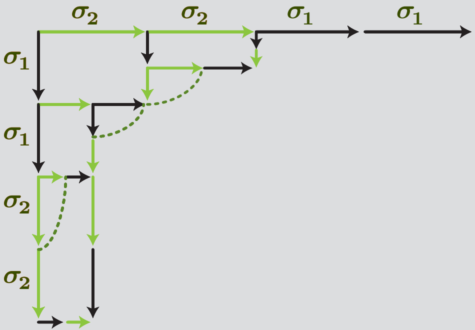
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



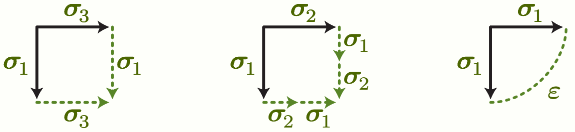
- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** ⌟ à l'aide des règles



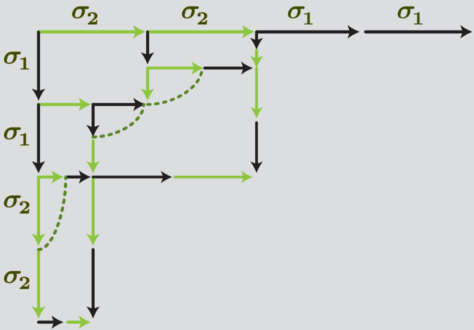
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



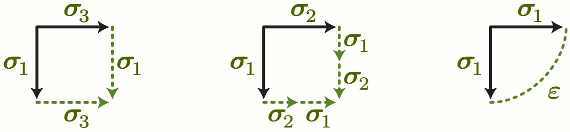
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



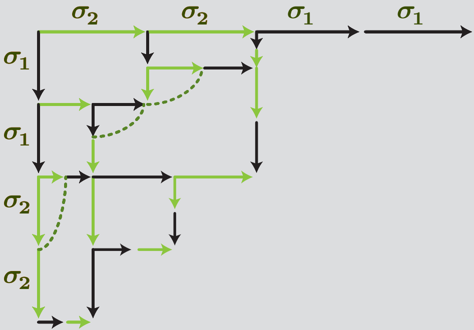
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



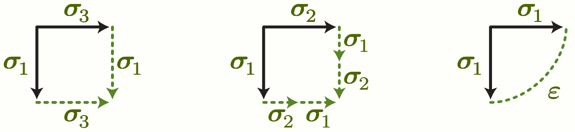
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



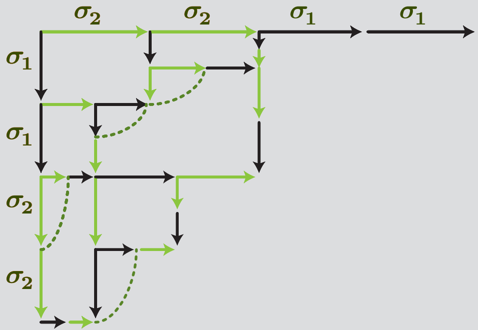
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



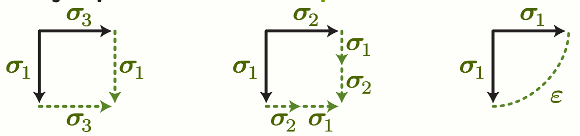
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



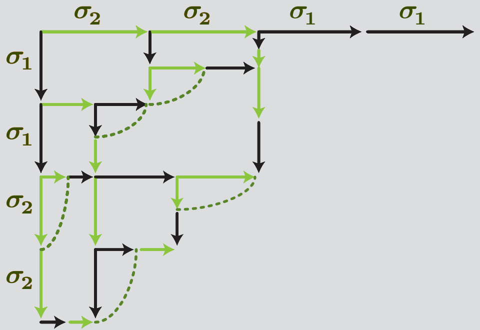
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



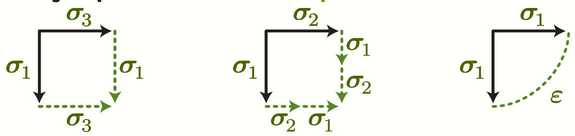
- **Associer à un mot de tresse un escalier :** $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \downarrow \sigma_i$.
- **Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles**



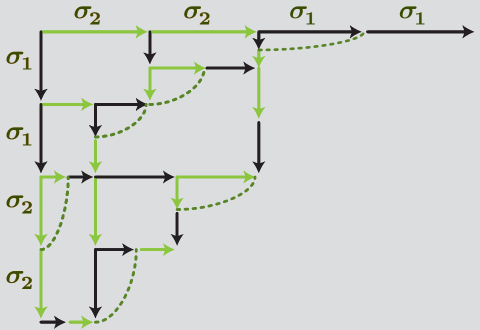
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



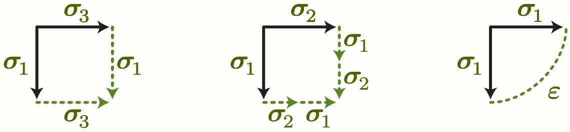
- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** ⌟ à l'aide des règles



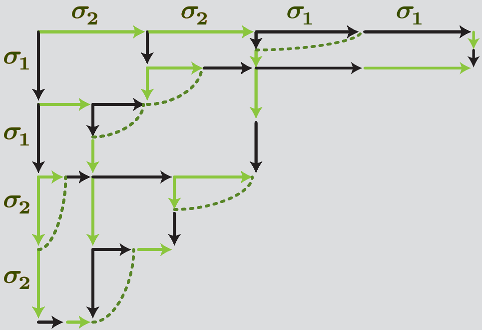
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



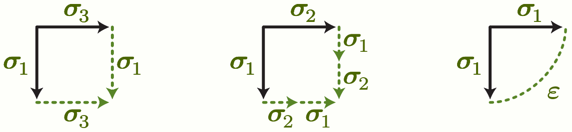
- **Associer à un mot de tresse un escalier :** $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \downarrow \sigma_i$.
- **Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles**



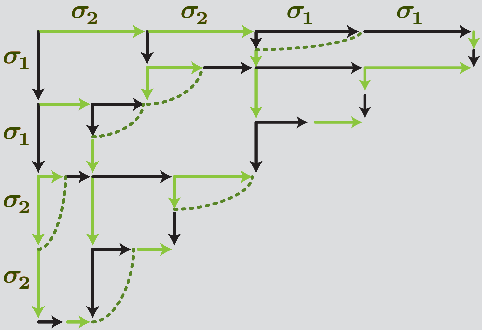
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$



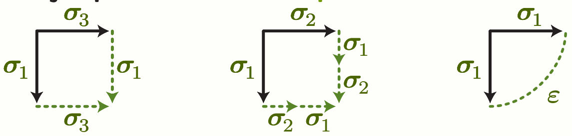
- Associer à un mot de tresse un escalier : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles



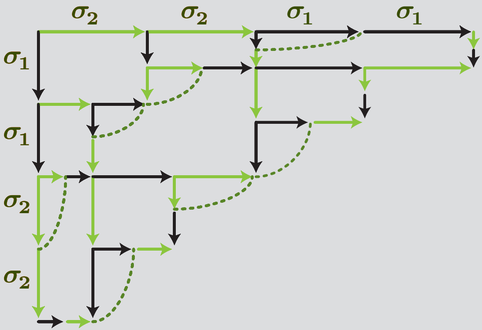
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



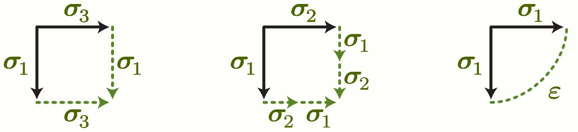
- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** ⌟ à l'aide des règles



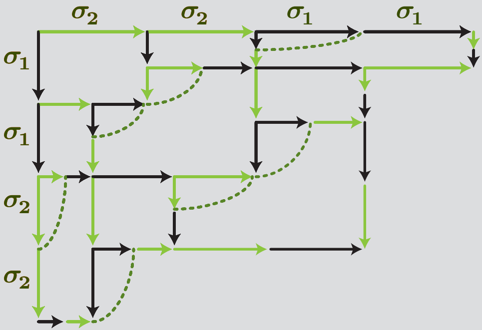
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$



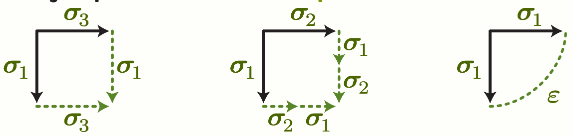
- **Associer à un mot de tresse un escalier :** $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- **Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles**



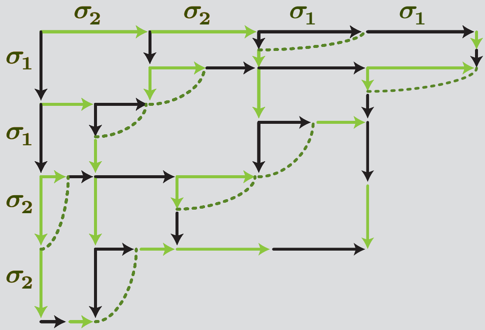
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



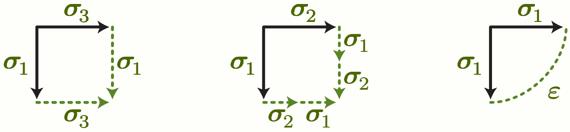
- **Associer à un mot de tresse un escalier :** $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \downarrow \sigma_i$.
- **Compléter jusqu'à obtenir une équerre ⌟ à l'aide des règles**



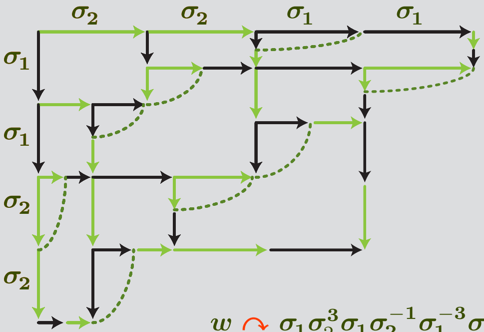
- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$



- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \Big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** ⌟ à l'aide des règles

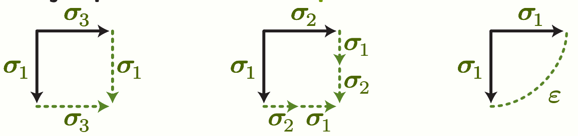


- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$

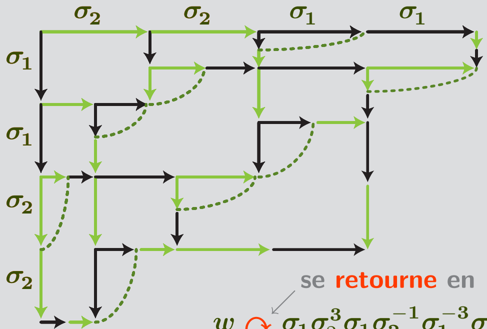


$$w \curvearrowright \sigma_1\sigma_2^3\sigma_1\sigma_2^{-1}\sigma_1^{-3}\sigma_2^{-1}$$

- Associer à un mot de tresse un **escalier** : $\sigma_i \mapsto \xrightarrow{\sigma_i}$, $\sigma_i^{-1} \mapsto \big\downarrow \sigma_i$.
- Compléter jusqu'à obtenir une **équerre** ⌟ à l'aide des règles



- **Exemple :**
  $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$



se retourne en

$w \curvearrowright \sigma_1\sigma_2^3\sigma_1\sigma_2^{-1}\sigma_1^{-3}\sigma_2^{-1}$

- **Où est la récriture ?**

- **Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...**

- **Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...**

  retourner les $-+$ en $+-$.

- **Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...**
  retourner les $-+$ en $+-$.
- **Mots terminaux $= uv^{-1}$ avec u, v positifs (équerres).**

- Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...

  retourner les $-+$ en $+-$.

- Mots terminaux $= uv^{-1}$ avec u, v positifs (équerres).
- Confluence (unicité d'un mot terminal)? Unicité du diagramme.

- Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...

  retourner les $-+$ en $+-$.

- Mots terminaux $= uv^{-1}$ avec u, v positifs (équerres).
- Confluence (unicité d'un mot terminal)? Unicité du diagramme.
- Terminaison (existence d'un mot terminal) ?

- Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...

  retourner les $-+$ en $+-$.

- Mots terminaux $= uv^{-1}$ avec u, v positifs (équerres).
- Confluence (unicité d'un mot terminal)? Unicité du diagramme.
- Terminaison (existence d'un mot terminal) ?

Proposition : Supposons
$$\begin{tikzcd} \cdot \arrow[r, "v"] \arrow[d, "u"'] \arrow[dr, phantom, "\curvearrowright"] & \cdot \arrow[d, "u'"] \\ \cdot \arrow[r, "v'"'] & \cdot \end{tikzcd}$$
et $u_1 \equiv^+ u$, $v_1 \equiv^+ v$.

- Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...

  retourner les $-+$ en $+-$.

- Mots **terminaux** $= uv^{-1}$ avec u, v positifs (équerres).
- **Confluence** (unicité d'un mot terminal)? Unicité du diagramme.
- **Terminaison** (existence d'un mot terminal) ?

Proposition : Supposons
$$\begin{tikzcd} \cdot \arrow[r, "v"] \arrow[d, "u"'] \arrow[dr, phantom, "\curvearrowright"] & \cdot \arrow[d, "u'"] \\ \cdot \arrow[r, "v'"'] & \cdot \end{tikzcd}$$
et $u_1 \equiv^+ u$, $v_1 \equiv^+ v$.

Alors il existe $u_1' \equiv^+ u'$, $v_1' \equiv^+ v'$ t.q.
$$\begin{tikzcd} \cdot \arrow[r, "v_1"] \arrow[d, "u_1"'] \arrow[dr, phantom, "\curvearrowright"] & \cdot \arrow[d, dashed, "u_1'"] \\ \cdot \arrow[r, dashed, "v_1'"'] & \cdot \end{tikzcd}$$
.

- Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...
  retourner les $-+$ en $+-$.

- Mots **terminaux** $= uv^{-1}$ avec u, v positifs (équerres).
- **Confluence** (unicité d'un mot terminal)? Unicité du diagramme.
- **Terminaison** (existence d'un mot terminal) ?

Proposition : Supposons $\begin{tikzcd} \cdot \arrow[r, "v"] \arrow[d, "u"'] \arrow[dr, phantom, "\curvearrowright"] & \cdot \arrow[d, "u'"] \\ \cdot \arrow[r, "v'"'] & \cdot \end{tikzcd}$ et $u_1 \equiv^+ u$, $v_1 \equiv^+ v$.

Alors il existe $u_1' \equiv^+ u'$, $v_1' \equiv^+ v'$ t.q. $\begin{tikzcd} \cdot \arrow[r, "v_1"] \arrow[d, "u_1"'] \arrow[dr, phantom, "\curvearrowright"] & \cdot \arrow[d, dashed, "u_1'"] \\ \cdot \arrow[r, dashed, "v_1'"'] & \cdot \end{tikzcd}$.

Corollaire 1 : Pour u, v positifs, $u \equiv^+ v$ entraîne $u^{-1}v \curvearrowright \varepsilon$.

- Où est la récriture ? Lire les étiquettes de Pau à Strasbourg...

  retourner les $-+$ en $+-$.

- Mots **terminaux** $= uv^{-1}$ avec u, v positifs (équerres).
- **Confluence** (unicité d'un mot terminal)? Unicité du diagramme.
- **Terminaison** (existence d'un mot terminal) ?

Proposition : Supposons $\begin{tikzcd} \cdot \arrow[r, "v"] \arrow[d, "u"'] \arrow[dr, phantom, "\curvearrowright"] & \cdot \arrow[d, "u'"] \\ \cdot \arrow[r, "v'"'] & \cdot \end{tikzcd}$ et $u_1 \equiv^+ u$, $v_1 \equiv^+ v$.

Alors il existe $u'_1 \equiv^+ u'$, $v'_1 \equiv^+ v'$ t.q. $\begin{tikzcd} \cdot \arrow[r, "v_1"] \arrow[d, "u_1"'] \arrow[dr, phantom, "\curvearrowright"] & \cdot \arrow[d, dashed, "u'_1"] \\ \cdot \arrow[r, dashed, "v'_1"'] & \cdot \end{tikzcd}$.

Corollaire 1 : Pour u, v positifs, $u \equiv^+ v$ entraîne $u^{-1}v \curvearrowright \varepsilon$.

Corollaire 2 : Pour tout w, il existe u, v positifs t.q. $w \curvearrowright uv^{-1}$.

Algorithme : Partant d'un mot de tresse w,

Algorithme : Partant d'un mot de tresse w,

- (i) Retourner w en uv^{-1} avec u, v positifs ;

Algorithme : Partant d'un mot de tresse w,

- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$w \equiv \varepsilon$

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- **Justification** :
  $w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon$

Algorithme : Partant d'un mot de tresse w,

- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :
  $w \equiv \varepsilon \iff uv^{-1} \equiv \varepsilon \iff u \equiv v$

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$$w \equiv \varepsilon \iff uv^{-1} \equiv \varepsilon \iff u \equiv v \iff u \equiv^{+} v$$

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- **Justification :**

$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon.$ $\square$

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon.$ $\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$:

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon. \qquad \square$$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$$w \equiv \varepsilon \iff uv^{-1} \equiv \varepsilon \iff u \equiv v \iff u \equiv^{+} v \iff v^{-1}u \curvearrowright \varepsilon. \qquad \square$$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) `BBAAbbaa`

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :
$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon.$ $\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) `BBAAbbaa` $\curvearrowright$

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon. \qquad \square$$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) $\mathtt{BBAAbbaa} \curvearrowright \mathtt{abbbaBAAAB}$;

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$$w \equiv \varepsilon \iff uv^{-1} \equiv \varepsilon \iff u \equiv v \iff u \equiv^{+} v \iff v^{-1}u \curvearrowright \varepsilon. \qquad \square$$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) BBAAbbaa $\curvearrowright$ abbbaBAAAB ;
- (ii) BAAAB

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon. \qquad \square$$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser **a** $= \sigma_1$, **b** $= \sigma_2$... **A** $= \sigma_1^{-1}$...
- (i) `BBAAbbaa` $\curvearrowright$ `abbbaBAAAB` ;
- (ii) `BAAABabbba`

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :
$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon.$ $\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser **a** $= \sigma_1$, **b** $= \sigma_2$... **A** $= \sigma_1^{-1}$...
- (i) `BBAAbbaa` $\curvearrowright$ `abbbaBAAAB` ;
- (ii) `BAAABabbba` $\curvearrowright$

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon. \qquad \square$$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) $\mathtt{BBAAbbaa} \curvearrowright \mathtt{abbbaBAAAB}$;
- (ii) $\mathtt{BAAABabbba} \curvearrowright \mathtt{BBAAbbaa}$

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon.$ $\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) BBAAbbaa $\curvearrowright$ abbbaBAAAB ;
- (ii) BAAABabbba $\curvearrowright$ BBAAbbaa $(= w)$;

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- **Justification :**
$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon.$ $\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$: poser a $= \sigma_1$, b $= \sigma_2$... A $= \sigma_1^{-1}$...
- (i) BBAAbbaa $\curvearrowright$ abbbaBAAAB ;
- (ii) BAAABabbba $\curvearrowright$ BBAAbbaa $(= w)$;
- (iii) BBAAbbaa $= \varepsilon$?

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- **Justification :**

$$w \equiv \varepsilon \iff uv^{-1} \equiv \varepsilon \iff u \equiv v \iff u \equiv^{+} v \iff v^{-1}u \curvearrowright \varepsilon. \qquad \square$$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) $\mathtt{BBAAbbaa} \curvearrowright \mathtt{abbbaBAAAB}$;
- (ii) $\mathtt{BAAABabbba} \curvearrowright \mathtt{BBAAbbaa}$ $(= w)$;
- (iii) $\mathtt{BBAAbbaa} = \varepsilon$? ... non, donc $w \not\equiv \varepsilon$.

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :

$$w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon. \qquad \square$$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) $\mathtt{BBAAbbaa} \curvearrowright \mathtt{abbbaBAAAB}$;
- (ii) $\mathtt{BAAABabbba} \curvearrowright \mathtt{BBAAbbaa}$ $(= w)$;
- (iii) $\mathtt{BBAAbbaa} = \varepsilon$? ... non, donc $w \not\equiv \varepsilon$.

- Complexité quadratique à n fixé,

Algorithme : Partant d'un mot de tresse w,
- (i) Retourner w en uv^{-1} avec u, v positifs ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $v^{-1}u$ se retourne en ε.

- Justification :
  $w \equiv \varepsilon \Longleftrightarrow uv^{-1} \equiv \varepsilon \Longleftrightarrow u \equiv v \Longleftrightarrow u \equiv^{+} v \Longleftrightarrow v^{-1}u \curvearrowright \varepsilon.$ $\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$: poser $\mathtt{a} = \sigma_1$, $\mathtt{b} = \sigma_2$... $\mathtt{A} = \sigma_1^{-1}$...
- (i) $\mathtt{BBAAbbaa} \curvearrowright \mathtt{abbbaBAAAB}$;
- (ii) $\mathtt{BAAABabbba} \curvearrowright \mathtt{BBAAbbaa}$ $(= w)$;
- (iii) $\mathtt{BBAAbbaa} = \varepsilon$? ... non, donc $w \not\equiv \varepsilon$.

- Complexité quadratique à n fixé, mais inconnue par rapport à n.

Solution : Réduction des poignées

Solution : Réduction des poignées

- **Domaine** : géométrie + théorie combinatoire des groupes

Solution : Réduction des poignées

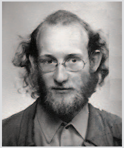
- **Domaine :** géométrie + théorie combinatoire des groupes
- **Point de vue :** tresse = chemin dans le graphe de Cayley

Solution : Réduction des poignées

- **Domaine :** géométrie + théorie combinatoire des groupes
- **Point de vue :** tresse = chemin dans le graphe de Cayley
- **Méthode :** réduction

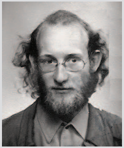
Solution : Réduction des poignées

- Domaine : géométrie + théorie combinatoire des groupes
- Point de vue : tresse = chemin dans le graphe de Cayley
- Méthode : réduction
- Auteur : '95



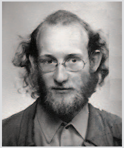
Solution : Réduction des poignées

- **Domaine :** géométrie + théorie combinatoire des groupes
- **Point de vue :** tresse = chemin dans le graphe de Cayley
- **Méthode :** réduction
- **Auteur :** '95
- Mots-clés : tresse σ-positive, monotonicité



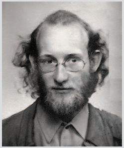
Solution : Réduction des poignées

- **Domaine :** géométrie + théorie combinatoire des groupes
- **Point de vue :** tresse = chemin dans le graphe de Cayley
- **Méthode :** réduction
- **Auteur :** '95
- Mots-clés : tresse σ-positive, monotonicité
- Arrière-plan : ordre des tresses



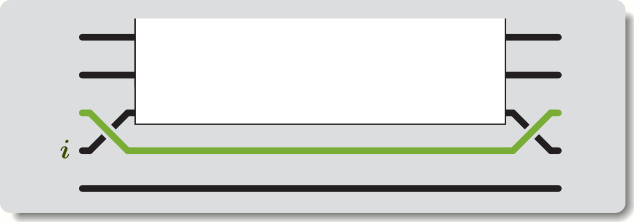
Solution : Réduction des poignées

- **Domaine :** géométrie + théorie combinatoire des groupes
- **Point de vue :** tresse = chemin dans le graphe de Cayley
- **Méthode :** réduction
- **Auteur :** '95
- Mots-clés : tresse σ-positive, monotonicité
- Arrière-plan : ordre des tresses
- Extensions : ??? (spécifique aux tresses)

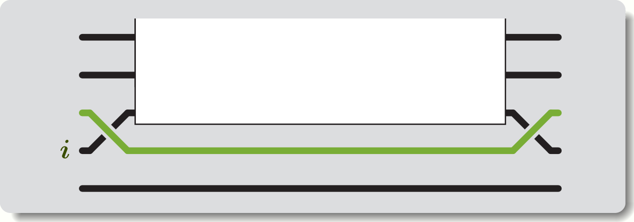


- **Une σ_i-poignée dans un diagramme de tresse :**

- Une σ_i-poignée dans un diagramme de tresse :

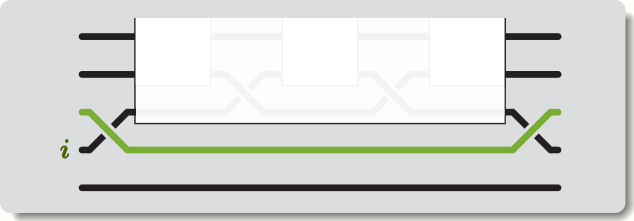


- Une σ_i-poignée dans un diagramme de tresse :



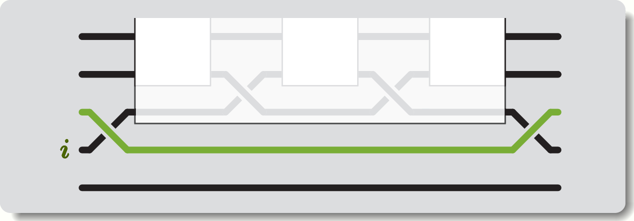
- La réduction d'une poignée :

- Une σ_i-poignée dans un diagramme de tresse :



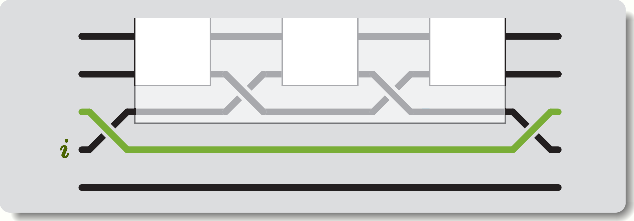
- La réduction d'une poignée :

- Une σ_i-poignée dans un diagramme de tresse :



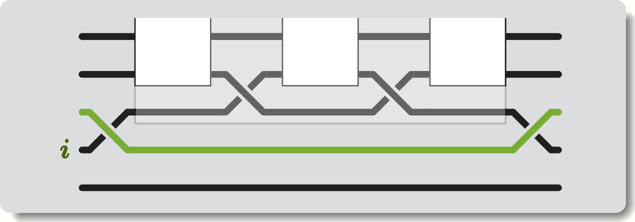
- La réduction d'une poignée :

- Une σ_i-poignée dans un diagramme de tresse :



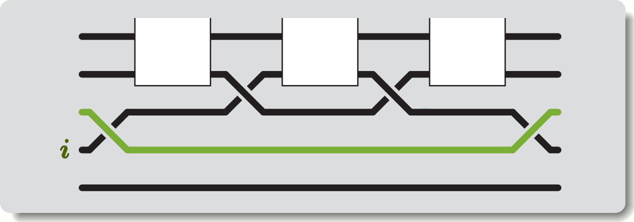
- La réduction d'une poignée :

- Une σ_i-poignée dans un diagramme de tresse :



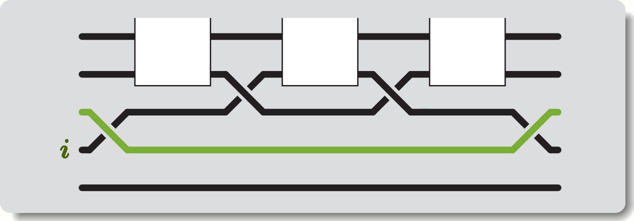
- La réduction d'une poignée :

- Une σ_i-poignée dans un diagramme de tresse :

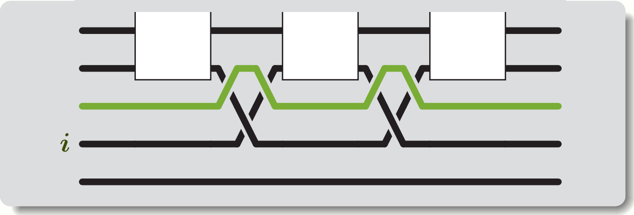


- La réduction d'une poignée :

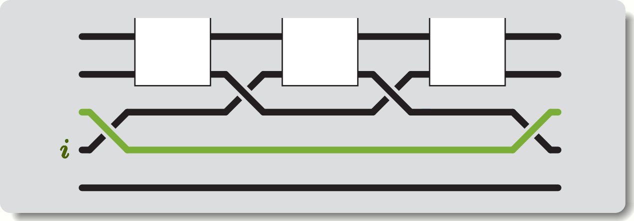
- Une σ_i-poignée dans un diagramme de tresse :



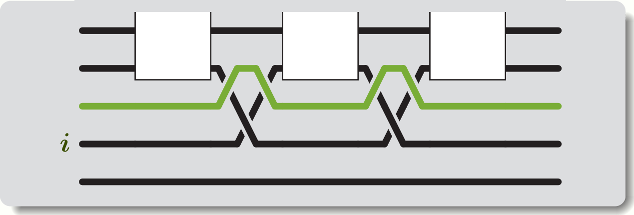
- La réduction d'une poignée :



- Une σ_i-poignée dans un diagramme de tresse :



- La réduction d'une poignée :



- **En termes de mots :**

- **En termes de mots :**

σ_i-poignée = $\sigma_i^e v \sigma_i^{-e}$, avec $e = \pm 1$ et que des $\sigma_j^{\pm 1}$, $j \geqslant i$, dans v,

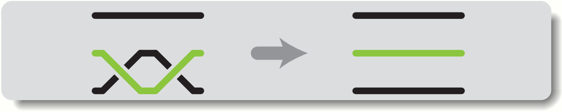
- **En termes de mots :**

σ_i-**poignée** $= \sigma_i^e v \sigma_i^{-e}$, avec $e = \pm 1$ et que des $\sigma_j^{\pm 1}$, $j \geqslant i$, dans v,
réduire $=$ supprimer $\sigma_i^{\pm 1}$, remplacer chaque σ_{i+1}^d par $\sigma_{i+1}^{-e}\sigma_i^d\sigma_{i+1}^e$.

- **En termes de mots :**

σ_i-poignée $= \sigma_i^e v \sigma_i^{-e}$, avec $e = \pm 1$ et que des $\sigma_j^{\pm 1}$, $j \geqslant i$, dans v,
réduire $=$ supprimer $\sigma_i^{\pm 1}$, remplacer chaque σ_{i+1}^d par $\sigma_{i+1}^{-e}\sigma_i^d\sigma_{i+1}^e$.

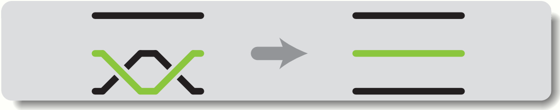
- La réduction des poignées étend la **réduction libre** $\sigma_i\sigma_i^{-1} \mapsto \varepsilon$:



- **En termes de mots :**

σ_i-poignée $= \sigma_i^e v \sigma_i^{-e}$, avec $e = \pm 1$ et que des $\sigma_j^{\pm 1}$, $j \geqslant i$, dans v,
réduire $=$ supprimer $\sigma_i^{\pm 1}$, remplacer chaque σ_{i+1}^d par $\sigma_{i+1}^{-e} \sigma_i^d \sigma_{i+1}^e$.

- La réduction des poignées étend la **réduction libre** $\sigma_i \sigma_i^{-1} \mapsto \varepsilon$:

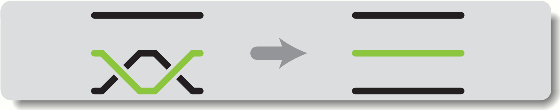


- **Réduire donne un mot** équivalent (donc $w \mapsto \varepsilon$ entraîne $w \equiv \varepsilon$).

- **En termes de mots :**

σ_i-poignée $= \sigma_i^e v \sigma_i^{-e}$, avec $e = \pm 1$ et que des $\sigma_j^{\pm 1}$, $j \geqslant i$, dans v,
réduire $=$ supprimer $\sigma_i^{\pm 1}$, remplacer chaque σ_{i+1}^d par $\sigma_{i+1}^{-e} \sigma_i^d \sigma_{i+1}^e$.

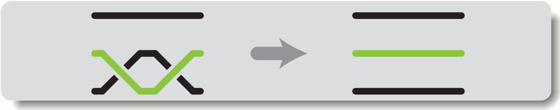
- La réduction des poignées étend la réduction libre $\sigma_i \sigma_i^{-1} \mapsto \varepsilon$:



- Réduire donne un mot équivalent (donc $w \mapsto \varepsilon$ entraîne $w \equiv \varepsilon$).
- Mots terminaux $=$ mots sans poignée ($=$ où le σ_i de plus petit indice n'apparaît qu'avec un seul signe).

• En termes de mots :
σ_i-poignée $= \sigma_i^e v \sigma_i^{-e}$, avec $e = \pm 1$ et que des $\sigma_j^{\pm 1}$, $j \geqslant i$, dans v,
réduire = supprimer $\sigma_i^{\pm 1}$, remplacer chaque σ_{i+1}^d par $\sigma_{i+1}^{-e}\sigma_i^d\sigma_{i+1}^e$.

• La réduction des poignées étend la réduction libre $\sigma_i\sigma_i^{-1} \mapsto \varepsilon$:



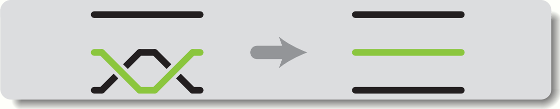
• Réduire donne un mot équivalent (donc $w \mapsto \varepsilon$ entraîne $w \equiv \varepsilon$).
• Mots terminaux = mots sans poignée (= où le σ_i de plus petit indice n'apparaît qu'avec un seul signe).

Théorème : (i) Un mot sans poignée n'est jamais trivial.
(ii) Toute suite de réductions est finie.

- En termes de mots :

σ_i-poignée $= \sigma_i^e v \sigma_i^{-e}$, avec $e = \pm 1$ et que des $\sigma_j^{\pm 1}$, $j \geqslant i$, dans v,
réduire $=$ supprimer $\sigma_i^{\pm 1}$, remplacer chaque σ_{i+1}^d par $\sigma_{i+1}^{-e}\sigma_i^d\sigma_{i+1}^e$.

- La réduction des poignées étend la réduction libre $\sigma_i\sigma_i^{-1} \mapsto \varepsilon$:



- Réduire donne un mot équivalent (donc $w \mapsto \varepsilon$ entraîne $w \equiv \varepsilon$).
- Mots terminaux $=$ mots sans poignée ($=$ où le σ_i de plus petit indice n'apparaît qu'avec un seul signe).

Théorème : (i) Un mot sans poignée n'est jamais trivial.
(ii) Toute suite de réductions est finie.

Algorithme : Partant d'un mot de tresse w,

Algorithme : Partant d'un mot de tresse w,

- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

Algorithme : Partant d'un mot de tresse w,

- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

• Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.□

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$:

Algorithme : Partant d'un mot de tresse w,

- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser **a** $= \sigma_1$, **b** $= \sigma_2$... **A** $= \sigma_1^{-1}$...

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser a $= \sigma_1$, b $= \sigma_2$... A $= \sigma_1^{-1}$...
- BBAAbbaa

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w, et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser a $= \sigma_1$, b $= \sigma_2$... A $= \sigma_1^{-1}$...
- BBAAbbaa
  - BBAbaBbaBa

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser a $= \sigma_1$, b $= \sigma_2$... A $= \sigma_1^{-1}$...
- BBAAbbaa
  - BBAbaBbaBa
    - BBAbaaBa

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w,
et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser a $= \sigma_1$, b $= \sigma_2$... A $= \sigma_1^{-1}$...
- BBAAbbaa
  - BBAbaBbaBa
    - BBAbaaBa
      - BBbaBaBa

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$: poser **a** $= \sigma_1$, **b** $= \sigma_2$... **A** $= \sigma_1^{-1}$...
- BBAAbbaa
  - BBAbaBbaBa
    - BBAbaaBa
      - BBbaBaBa
        - BaBaBa,

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w, et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.□

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser **a** $= \sigma_1$, **b** $= \sigma_2$... **A** $= \sigma_1^{-1}$...
- BBAAbbaa
  - BBAbaBbaBa
    - BBAbaaBa
      - BBbaBaBa
        - BaBaBa, sans poignée et non vide, donc $w \not\equiv \varepsilon$.

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w,
  et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^{2}\sigma_1^{2}$: poser **a** $= \sigma_1$, **b** $= \sigma_2$... **A** $= \sigma_1^{-1}$...
- BBAAbbaa
  - BBAbaBbaBa
    - BBAbaaBa
      - BBbaBaBa
        - BaBaBa, sans poignée et non vide, donc $w \not\equiv \varepsilon$.

- Stratégies de réduction, par exemple diviser pour régner.

Algorithme : Partant d'un mot de tresse w,
- (i) Réduire la première poignée de w, et itérer jusqu'à obtenir w' sans poignée ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si $w' = \varepsilon$.

- Justification : Pour w' sans poignée, $w' \equiv \varepsilon$ équivaut à $w' = \varepsilon$.$\square$

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$: poser **a** $= \sigma_1$, **b** $= \sigma_2$... **A** $= \sigma_1^{-1}$...
- BBAAbbaa
  - BBAbaBbaBa
    - BBAbaaBa
      - BBbaBaBa
        - BaBaBa, sans poignée et non vide, donc $w \not\equiv \varepsilon$.

- Stratégies de réduction, par exemple diviser pour régner.
- **Complexité** : inconnue ($\leqslant$exponentielle, conjecturée quadratique)

- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

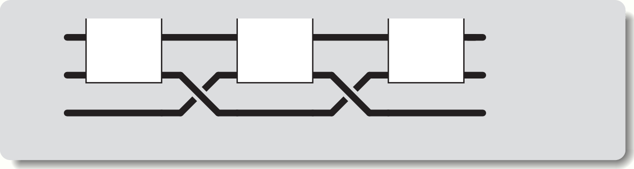
- **Système autodistributif ordonné :** $(S, \underset{\substack{\uparrow\\ \text{opération autodistributive}}}{*}, \underset{\substack{\uparrow\\ \text{ordre total t.q. } x < x * y}}{<})$

- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

opération autodistributive ordre total t.q. $x < x * y$

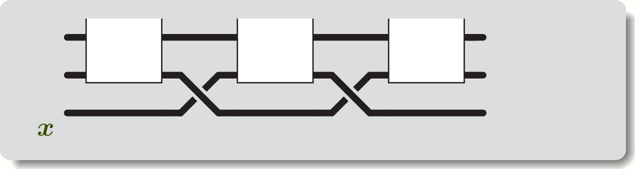


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

opération autodistributive ordre total t.q. $x < x * y$

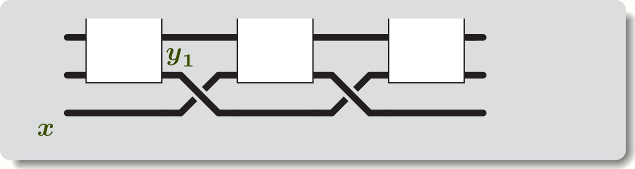


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

opération autodistributive   ordre total t.q. $x < x * y$

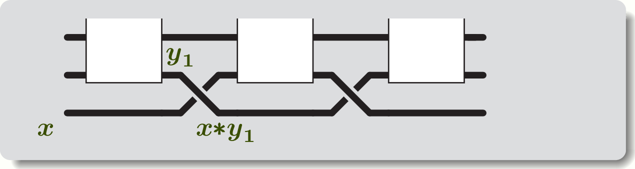


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

$\uparrow$ opération autodistributive $\uparrow$ ordre total t.q. $x < x * y$

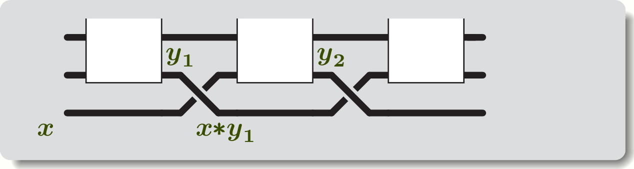


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

$\uparrow$ opération autodistributive $\uparrow$ ordre total t.q. $x < x * y$

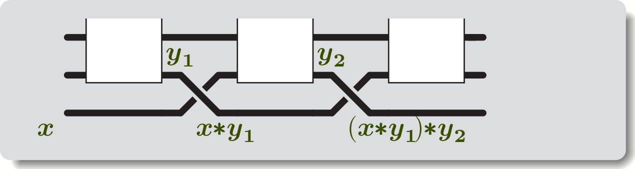


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

opération autodistributive ordre total t.q. $x < x * y$

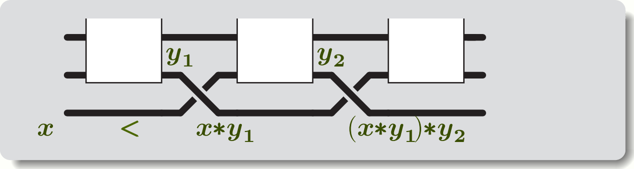


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

opération autodistributive ordre total t.q. $x < x * y$

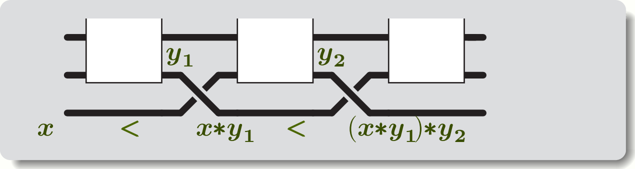


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

opération autodistributive ($*$) ordre total t.q. $x < x * y$ ($<$)

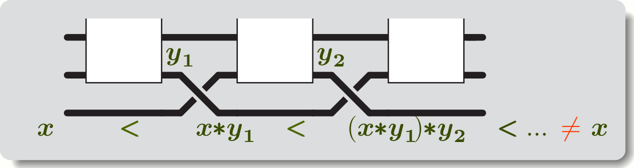


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

$*$: opération autodistributive ; $<$: ordre total t.q. $x < x * y$

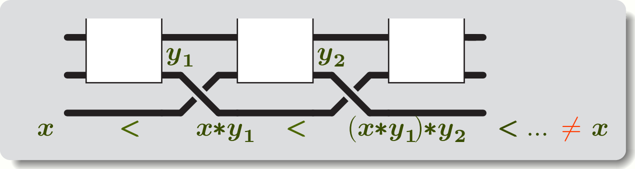


- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

$*$: opération autodistributive ; $<$: ordre total t.q. $x < x * y$



- **Il existe des systèmes autodistributifs ordonnés :**

- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

$\uparrow$ opération autodistributive $\uparrow$ ordre total t.q. $x < x * y$

$x \quad < \quad x{*}y_1 \quad < \quad (x{*}y_1){*}y_2 \quad < \dots \neq x$

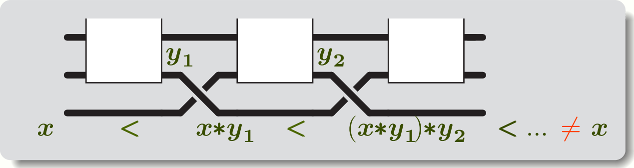
- **Il existe des systèmes autodistributifs ordonnés :**
  - (Laver '89) avec des grands cardinaux (th. des ensembles),

- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, \underset{\uparrow}{*}, \underset{\uparrow}{<})$

opération autodistributive ordre total t.q. $x < x * y$



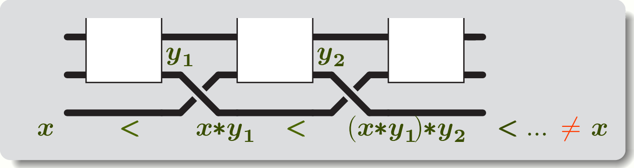
- **Il existe des systèmes autodistributifs ordonnés :**
  - (Laver '89) avec des grands cardinaux (th. des ensembles),
  - (D. '92) sans théorie des ensembles. $\square$

- **Démonstration du point (i) :**

« Un mot de tresse contenant σ_1 et et pas σ_1^{-1} n'est pas trivial. »

- **Système autodistributif ordonné :** $(S, *, <)$

$\uparrow$ opération autodistributive (pour $*$) $\uparrow$ ordre total t.q. $x < x * y$ (pour $<$)



- **Il existe des systèmes autodistributifs ordonnés :**
  - (Laver '89) avec des grands cardinaux (th. des ensembles),
  - (D. '92) sans théorie des ensembles. $\square$

- **Graphe de Cayley** de B_n :

- **Graphe de Cayley** de B_n :

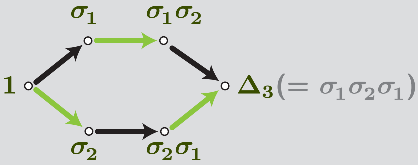
  sommets = tresses ;

- **Graphe de Cayley** de B_n :
  sommets = tresses ; arête $\overset{\beta}{\circ}\xrightarrow{\sigma_i}\overset{\beta'}{\circ}$ pour $\beta' = \beta\sigma_i$.

- **Graphe de Cayley** de B_n :
  sommets = tresses ; arête $\overset{\beta}{\circ} \xrightarrow{\sigma_i} \overset{\beta'}{\circ}$ pour $\beta' = \beta\sigma_i$.
- $\Gamma(\Delta_n^d)$ = restriction du g. de Cayley aux diviseurs de Δ_n^d dans B_n^+.

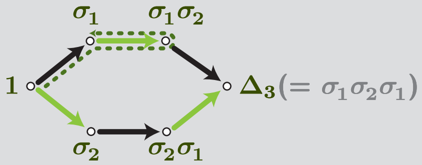
- **Graphe de Cayley** de B_n :
  sommets = tresses ; arête $\beta \xrightarrow{\sigma_i} \beta'$ pour $\beta' = \beta\sigma_i$.
- $\Gamma(\Delta_n^d)$ = restriction du g. de Cayley aux diviseurs de Δ_n^d dans B_n^+.

$\Gamma(\Delta_3)$:



- **Graphe de Cayley** de B_n :
  sommets = tresses ; arête $\beta \xrightarrow{\sigma_i} \beta'$ pour $\beta' = \beta\sigma_i$.
- $\Gamma(\Delta_n^d)$ = restriction du g. de Cayley aux diviseurs de Δ_n^d dans B_n^+.

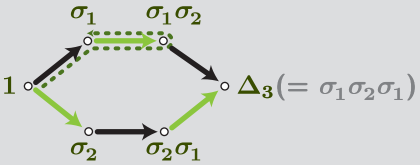
$\Gamma(\Delta_3)$:



- Mot **tracé** à partir de β dans $\Gamma(\Delta_n^d)$:

- **Graphe de Cayley** de B_n :
  sommets = tresses ; arête $\beta \xrightarrow{\sigma_i} \beta'$ pour $\beta' = \beta\sigma_i$.
- $\Gamma(\Delta_n^d)$ = restriction du g. de Cayley aux **diviseurs** de Δ_n^d dans B_n^+.

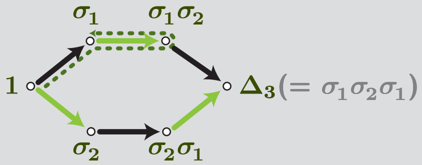
$\Gamma(\Delta_3)$:



- Mot **tracé** à partir de β dans $\Gamma(\Delta_n^d)$:
  $\sigma_1\sigma_2\sigma_2^{-1}$ tracé à partir de 1 dans $\Gamma(\Delta_3)$, mais σ_1^2 non tracé.

- **Graphe de Cayley** de B_n :
  sommets = tresses ; arête $\beta \xrightarrow{\sigma_i} \beta'$ pour $\beta' = \beta\sigma_i$.
- $\Gamma(\Delta_n^d)$ = restriction du g. de Cayley aux **diviseurs** de Δ_n^d dans B_n^+.

$\Gamma(\Delta_3)$:

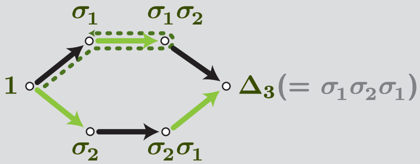


- Mot **tracé** à partir de β dans $\Gamma(\Delta_n^d)$:
  $\sigma_1\sigma_2\sigma_2^{-1}$ tracé à partir de 1 dans $\Gamma(\Delta_3)$, mais σ_1^2 non tracé.

Lemme : (i) Tout mot est tracé dans $\Gamma(\Delta_n^d)$ pour $n, d \gg 0$.

- **Graphe de Cayley** de B_n :
  sommets = tresses ; arête $\beta \xrightarrow{\sigma_i} \beta'$ pour $\beta' = \beta\sigma_i$.
- $\Gamma(\Delta_n^d)$ = restriction du g. de Cayley aux **diviseurs** de Δ_n^d dans B_n^+.

$\Gamma(\Delta_3)$:



- Mot **tracé** à partir de β dans $\Gamma(\Delta_n^d)$:
  $\sigma_1\sigma_2\sigma_2^{-1}$ tracé à partir de 1 dans $\Gamma(\Delta_3)$, mais σ_1^2 non tracé.

Lemme : (i) Tout mot est tracé dans $\Gamma(\Delta_n^d)$ pour $n, d \gg 0$.
(ii) L'ensemble des mots tracés dans $\Gamma(\Delta_n^d)$ est clos par réduction.

- **Démonstration du point (ii) :**

  « Toute suite de réductions est finie. »

- **Démonstration du point (ii) :**

  « Toute suite de réductions est finie. »

- **Soit $w_0, w_1, \ldots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$. Alors il existe $u = u_1 u_2 \ldots$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,**

- **Démonstration du point (ii) :**

  « Toute suite de réductions est finie. »

- **Soit $w_0, w_1, \dots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$.**
  Alors il existe $u = u_1 u_2 \dots$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,
  si w_r, w_{r+1} est la réduction de la première σ_1-poignée,

- **Démonstration du point (ii) :**

  « Toute suite de réductions est finie. »

- **Soit $w_0, w_1, \ldots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$.
  Alors il existe $u = u_1 u_2 \ldots$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,
  si w_r, w_{r+1} est la réduction de la première σ_1-poignée,
  alors u_r contient un σ_1 et zéro σ_1^{-1},**

- **Démonstration du point (ii) :**

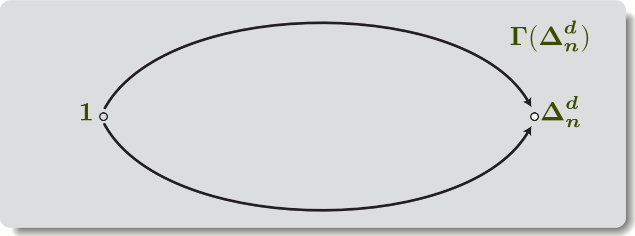
  « Toute suite de réductions est finie. »

- **Soit $w_0, w_1, \ldots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$. Alors il existe $u = u_1 u_2 \ldots$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,**
  si w_r, w_{r+1} est la réduction de la première σ_1-poignée,
  alors u_r contient un σ_1 et zéro σ_1^{-1}, sinon u_r contient zéro $\sigma_1^{\pm 1}$.

- Démonstration du point (ii) :

  « Toute suite de réductions est finie. »

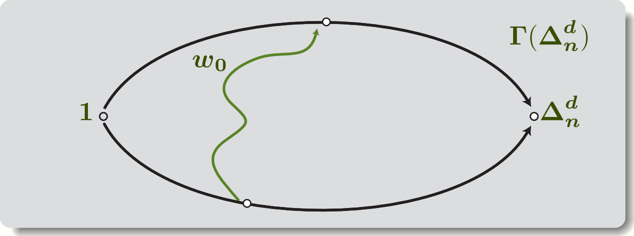
- Soit $w_0, w_1, \dots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$.
  Alors il existe $u = u_1 u_2 \dots$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,
  si w_r, w_{r+1} est la réduction de la première σ_1-poignée,
  alors u_r contient un σ_1 et zéro σ_1^{-1}, sinon u_r contient zéro $\sigma_1^{\pm 1}$.



- Démonstration du point (ii) :

  « Toute suite de réductions est finie. »

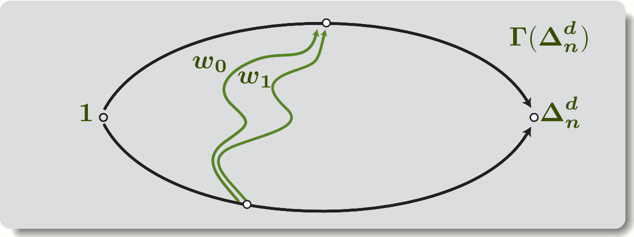
- Soit $w_0, w_1, \ldots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$.
  Alors il existe $u = u_1 u_2 \ldots$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,
  si w_r, w_{r+1} est la réduction de la première σ_1-poignée,
  alors u_r contient un σ_1 et zéro σ_1^{-1}, sinon u_r contient zéro $\sigma_1^{\pm 1}$.



- Démonstration du point (ii) :

« Toute suite de réductions est finie. »

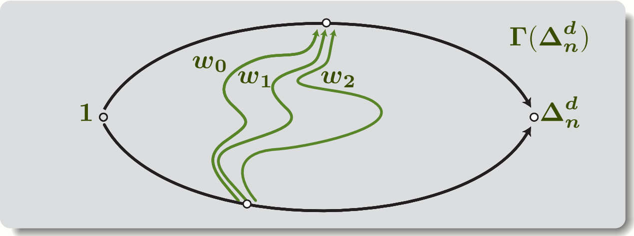
- Soit $w_0, w_1, \ldots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$. Alors il existe $u = u_1 u_2 \ldots$ tracé dans $\Gamma(\Delta_n^d)$ t.q., si w_r, w_{r+1} est la réduction de la première σ_1-poignée, alors u_r contient un σ_1 et zéro σ_1^{-1}, sinon u_r contient zéro $\sigma_1^{\pm 1}$.



- Démonstration du point (ii) :

« Toute suite de réductions est finie. »

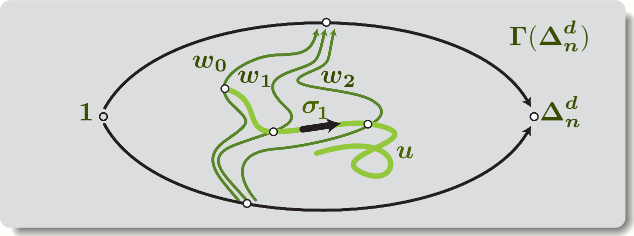
- Soit $w_0, w_1, \ldots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$. Alors il existe $u = u_1 u_2 \ldots$ tracé dans $\Gamma(\Delta_n^d)$ t.q., si w_r, w_{r+1} est la réduction de la première σ_1-poignée, alors u_r contient un σ_1 et zéro σ_1^{-1}, sinon u_r contient zéro $\sigma_1^{\pm 1}$.



- Démonstration du point (ii) :

  « Toute suite de réductions est finie. »

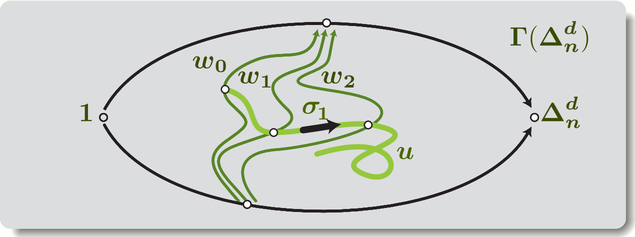
- Soit $w_0, w_1, ...$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$.
  Alors il existe $u = u_1 u_2 ...$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,
  si w_r, w_{r+1} est la réduction de la première σ_1-poignée,
  alors u_r contient un σ_1 et zéro σ_1^{-1}, sinon u_r contient zéro $\sigma_1^{\pm 1}$.



- Démonstration du point (ii) :

  « Toute suite de réductions est finie. »

- Soit $w_0, w_1, \ldots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$.
  Alors il existe $u = u_1 u_2 \ldots$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,
  si w_r, w_{r+1} est la réduction de la première σ_1-poignée,
  alors u_r contient un σ_1 et zéro σ_1^{-1}, sinon u_r contient zéro $\sigma_1^{\pm 1}$.

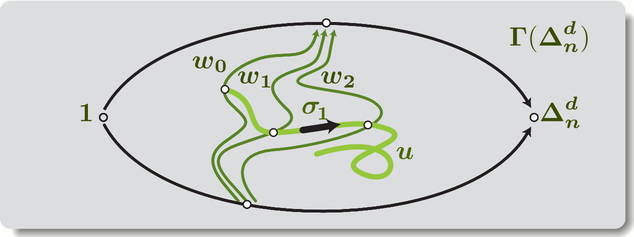


- Par (i), le mot u traverse au plus une fois chaque σ_1 de $\Gamma(\Delta_n^d)$,

- Démonstration du point (ii) :

  « Toute suite de réductions est finie. »

- Soit $w_0, w_1, \ldots$ une suite de réductions, les w_r tracés dans $\Gamma(\Delta_n^d)$.
  Alors il existe $u = u_1 u_2 \ldots$ tracé dans $\Gamma(\Delta_n^d)$ t.q.,
  si w_r, w_{r+1} est la réduction de la première σ_1-poignée,
  alors u_r contient un σ_1 et zéro σ_1^{-1}, sinon u_r contient zéro $\sigma_1^{\pm 1}$.



- Par (i), le mot u traverse au plus une fois chaque σ_1 de $\Gamma(\Delta_n^d)$,
  et il n'y a qu'un nombre fini d'arêtes σ_1 dans $\Gamma(\Delta_n^d)$. $\square$

Solution : Coordonnées de Dynnikov

Solution : Coordonnées de Dynnikov

- **Domaine : topologie**

Solution : Coordonnées de Dynnikov

- **Domaine :** topologie
- **Point de vue :** tresse = action sur lamination

Solution : Coordonnées de Dynnikov

- **Domaine : topologie**
- **Point de vue : tresse = action sur lamination**
- **Méthode : coordonnées**

Solution : Coordonnées de Dynnikov

- Domaine : topologie
- Point de vue : tresse = action sur lamination
- Méthode : coordonnées
- Auteur : Dynnikov ’00

Solution : Coordonnées de Dynnikov

- **Domaine :** topologie
- **Point de vue :** tresse = action sur lamination
- **Méthode :** coordonnées
- **Auteur :** Dynnikov '00
- **Mots-clés :** algèbre tropicale, triangulation, flip

Solution : Coordonnées de Dynnikov

- **Domaine :** topologie
- **Point de vue :** tresse = action sur lamination
- **Méthode :** coordonnées
- **Auteur :** Dynnikov '00
- Mots-clés : algèbre tropicale, triangulation, flip
- Arrière-plan : théorie des surfaces, théorie de Nielsen–Thurston
- Extensions : mapping class groups

- **Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$,**

- **Pour** $x \in \mathbb{Z}$**, on pose** $x^+ = \max(0, x)$**,** $x^- = \min(x, 0)$**, et** $F^+(x_1, y_1, x_2, y_2) =$

- **Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$, et**
  $F^+(x_1, y_1, x_2, y_2) =$
  $(x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+)$,

- **Pour** $x \in \mathbb{Z}$**, on pose** $x^+ = \max(0, x)$, $x^- = \min(x, 0)$**, et**

$$F^+(x_1, y_1, x_2, y_2) = (x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+),$$

$$F^-(x_1, y_1, x_2, y_2) = (x_1 - y_1^+ - (y_2^+ + z_2)^+, y_2 + z_2^-, x_2 - y_2^- - (y_1^- - z_2)^-, y_1 - z_2^-),$$

- Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$, et

$$F^+(x_1, y_1, x_2, y_2) = (x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+),$$

$$F^-(x_1, y_1, x_2, y_2) = (x_1 - y_1^+ - (y_2^+ + z_2)^+, y_2 + z_2^-, x_2 - y_2^- - (y_1^- - z_2)^-, y_1 - z_2^-),$$

où $z_1 = x_1 - y_1^- - x_2 + y_2^+$ et $z_2 = x_1 + y_1^- - x_2 - y_2^+$.

- Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$, et

$$F^+(x_1, y_1, x_2, y_2) = (x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+),$$

$$F^-(x_1, y_1, x_2, y_2) = (x_1 - y_1^+ - (y_2^+ + z_2)^+, y_2 + z_2^-, x_2 - y_2^- - (y_1^- - z_2)^-, y_1 - z_2^-),$$

où $z_1 = x_1 - y_1^- - x_2 + y_2^+$ et $z_2 = x_1 + y_1^- - x_2 - y_2^+$.

- On fait agir les mots de tresse à n brins sur $\mathbb{Z}^{2n}$ par

$$(a_1, b_1, ..., a_n, b_n) * \sigma_i^e = (a'_1, b'_1, ..., a'_n, b'_n)$$

- Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$, et
  $F^+(x_1, y_1, x_2, y_2) =$
  $(x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+)$,
  $F^-(x_1, y_1, x_2, y_2) =$
  $(x_1 - y_1^+ - (y_2^+ + z_2)^+, y_2 + z_2^-, x_2 - y_2^- - (y_1^- - z_2)^-, y_1 - z_2^-)$,
  où $z_1 = x_1 - y_1^- - x_2 + y_2^+$ et $z_2 = x_1 + y_1^- - x_2 - y_2^+$.
- On fait agir les mots de tresse à n brins sur $\mathbb{Z}^{2n}$ par
  $$(a_1, b_1, ..., a_n, b_n) * \sigma_i^e = (a'_1, b'_1, ..., a'_n, b'_n)$$
  avec $a'_k = a_k$ et $b'_k = b_k$ pour $k \neq i, i+1$, et

- Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$, et
  $F^+(x_1, y_1, x_2, y_2) =$
  $(x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+)$,
  $F^-(x_1, y_1, x_2, y_2) =$
  $(x_1 - y_1^+ - (y_2^+ + z_2)^+, y_2 + z_2^-, x_2 - y_2^- - (y_1^- - z_2)^-, y_1 - z_2^-)$,
  où $z_1 = x_1 - y_1^- - x_2 + y_2^+$ et $z_2 = x_1 + y_1^- - x_2 - y_2^+$.
- On fait agir les mots de tresse à n brins sur $\mathbb{Z}^{2n}$ par
  $$(a_1, b_1, ..., a_n, b_n) * \sigma_i^e = (a'_1, b'_1, ..., a'_n, b'_n)$$
  avec $a'_k = a_k$ et $b'_k = b_k$ pour $k \neq i, i+1$, et
  $$(a'_i, b'_i, a'_{i+1}, b'_{i+1}) = F^e(a_i, b_i, a_{i+1}, b_{i+1}).$$

- Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$, et
  $F^+(x_1, y_1, x_2, y_2) =$
  $(x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+)$,
  $F^-(x_1, y_1, x_2, y_2) =$
  $(x_1 - y_1^+ - (y_2^+ + z_2)^+, y_2 + z_2^-, x_2 - y_2^- - (y_1^- - z_2)^-, y_1 - z_2^-)$,
  où $z_1 = x_1 - y_1^- - x_2 + y_2^+$ et $z_2 = x_1 + y_1^- - x_2 - y_2^+$.
- On fait agir les mots de tresse à n brins sur $\mathbb{Z}^{2n}$ par
  $$(a_1, b_1, ..., a_n, b_n) * \sigma_i^e = (a'_1, b'_1, ..., a'_n, b'_n)$$
  avec $a'_k = a_k$ et $b'_k = b_k$ pour $k \neq i, i+1$, et
  $$(a'_i, b'_i, a'_{i+1}, b'_{i+1}) = F^e(a_i, b_i, a_{i+1}, b_{i+1}).$$
- Par définition, les coordonnées de w sont $(0, 1, 0, 1, ..., 0, 1) * w$.

- Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$, et
$$F^+(x_1, y_1, x_2, y_2) = \\ (x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+),$$
$$F^-(x_1, y_1, x_2, y_2) = \\ (x_1 - y_1^+ - (y_2^+ + z_2)^+, y_2 + z_2^-, x_2 - y_2^- - (y_1^- - z_2)^-, y_1 - z_2^-),$$
où $z_1 = x_1 - y_1^- - x_2 + y_2^+$ et $z_2 = x_1 + y_1^- - x_2 - y_2^+$.
- On fait agir les mots de tresse à n brins sur $\mathbb{Z}^{2n}$ par
$$(a_1, b_1, ..., a_n, b_n) * \sigma_i^e = (a'_1, b'_1, ..., a'_n, b'_n)$$
avec $a'_k = a_k$ et $b'_k = b_k$ pour $k \neq i, i+1$, et
$$(a'_i, b'_i, a'_{i+1}, b'_{i+1}) = F^e(a_i, b_i, a_{i+1}, b_{i+1}).$$
- Par définition, les coordonnées de w sont $(0, 1, 0, 1, ..., 0, 1) * w$.

Théorème (Dynnikov '00) : Les coordonnées de w ne dépendent que de la tresse représentée par w,

- Pour $x \in \mathbb{Z}$, on pose $x^+ = \max(0, x)$, $x^- = \min(x, 0)$, et

$$F^+(x_1, y_1, x_2, y_2) = (x_1 + y_1^+ + (y_2^+ - z_1)^+, y_2 - z_1^+, x_2 + y_2^- + (y_1^- + z_1)^-, y_1 + z_1^+),$$

$$F^-(x_1, y_1, x_2, y_2) = (x_1 - y_1^+ - (y_2^+ + z_2)^+, y_2 + z_2^-, x_2 - y_2^- - (y_1^- - z_2)^-, y_1 - z_2^-),$$

où $z_1 = x_1 - y_1^- - x_2 + y_2^+$ et $z_2 = x_1 + y_1^- - x_2 - y_2^+$.

- On fait agir les mots de tresse à n brins sur $\mathbb{Z}^{2n}$ par

$$(a_1, b_1, ..., a_n, b_n) * \sigma_i^e = (a'_1, b'_1, ..., a'_n, b'_n)$$

avec $a'_k = a_k$ et $b'_k = b_k$ pour $k \neq i, i+1$, et

$$(a'_i, b'_i, a'_{i+1}, b'_{i+1}) = F^e(a_i, b_i, a_{i+1}, b_{i+1}).$$

- Par définition, les coordonnées de w sont $(0, 1, 0, 1, ..., 0, 1) * w$.

Théorème (Dynnikov '00) : Les coordonnées de w ne dépendent que de la tresse représentée par w, et caractérisent celle-ci.

Algorithme : Partant d'un mot de tresse w,

Algorithme : Partant d'un mot de tresse w,

- (i) Calculer ses coordonnées par les formules de Dynnikov ;

Algorithme : Partant d'un mot de tresse w,

- (i) Calculer ses coordonnées par les formules de Dynnikov ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si on obtient $(0, 1, 0, 1, ..., 0, 1)$.

Algorithme : Partant d'un mot de tresse w,
- (i) Calculer ses coordonnées par les formules de Dynnikov ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si on obtient $(0, 1, 0, 1, ..., 0, 1)$.

Exemple : $w = \sigma_2^{-2} \sigma_1^{-2} \sigma_2^2 \sigma_1^2$;

Algorithme : Partant d'un mot de tresse w,
- (i) Calculer ses coordonnées par les formules de Dynnikov ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si on obtient $(0, 1, 0, 1, ..., 0, 1)$.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$; coordonnées :

$$(0, 1, 0, 1, ..., 0, 1) * w = (1, -19, -12, 9, 0, 13, 0, 1)$$

Algorithme : Partant d'un mot de tresse w,
- (i) Calculer ses coordonnées par les formules de Dynnikov ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si on obtient $(0, 1, 0, 1, ..., 0, 1)$.

Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$; coordonnées :

$$(0, 1, 0, 1, ..., 0, 1) * w = (1, -19, -12, 9, 0, 13, 0, 1)$$

$\neq (0, 1, 0, 1, ..., 0, 1)$, donc $w \not\equiv \varepsilon$.

Algorithme : Partant d'un mot de tresse w,
- (i) Calculer ses coordonnées par les formules de Dynnikov ;
- (ii) Alors $w \equiv \varepsilon$ si et seulement si on obtient $(0, 1, 0, 1, ..., 0, 1)$.

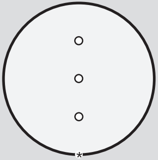
Exemple : $w = \sigma_2^{-2}\sigma_1^{-2}\sigma_2^2\sigma_1^2$; coordonnées :

$$(0, 1, 0, 1, ..., 0, 1) * w = (1, -19, -12, 9, 0, 13, 0, 1)$$

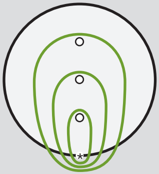
$$\neq (0, 1, 0, 1, ..., 0, 1), \text{ donc } w \not\equiv \varepsilon.$$

- Complexité : quadratique en temps, linéaire en espace, indépendamment de n.

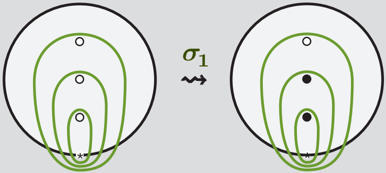
- **Tresse=homéomorphisme ⇝ agit sur les courbes tracées sur D_n.**



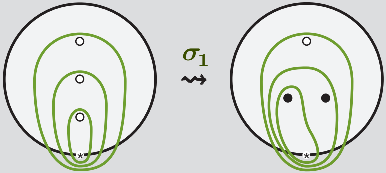
- **Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.**



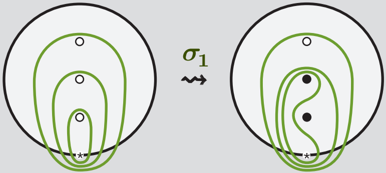
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



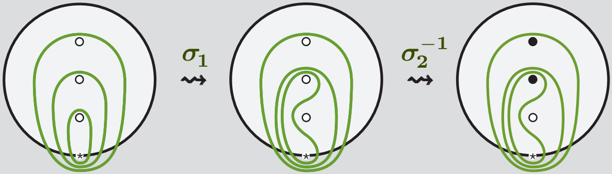
- **Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.**



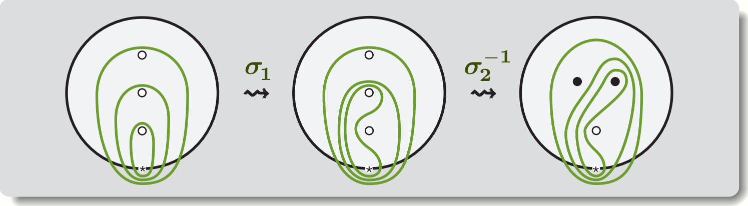
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



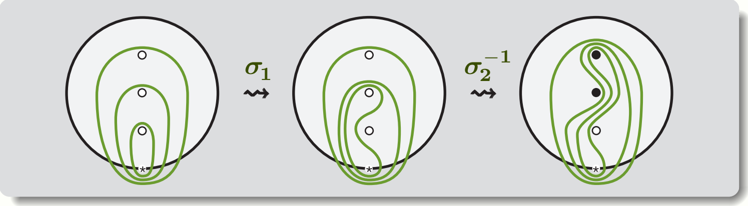
- **Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.**



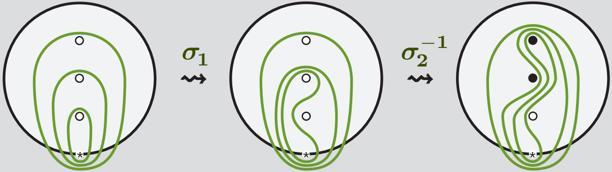
- **Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.**



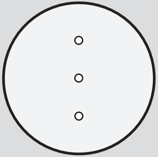
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



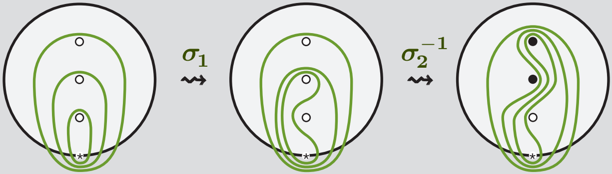
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



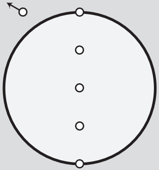
- Compter les intersections avec une triangulation fixée :



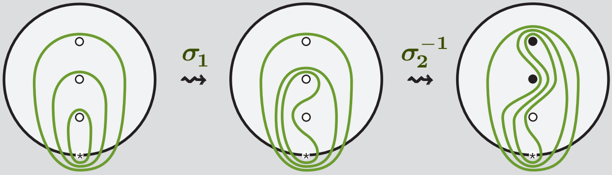
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



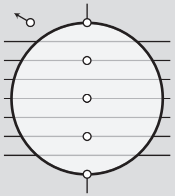
- Compter les intersections avec une triangulation fixée :



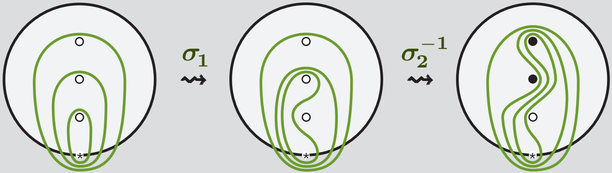
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



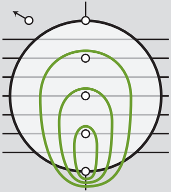
- Compter les intersections avec une triangulation fixée :



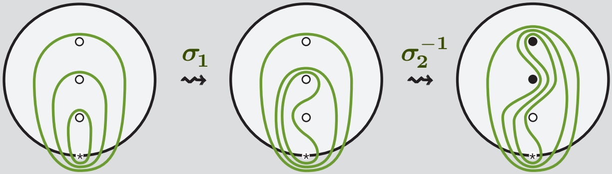
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



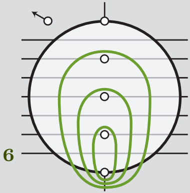
- Compter les intersections avec une triangulation fixée :



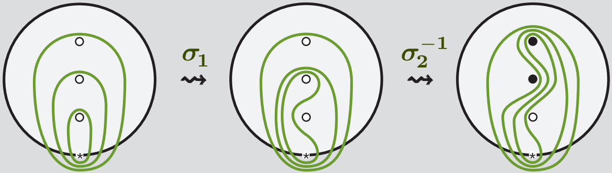
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



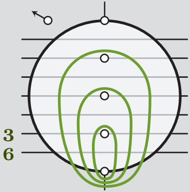
- Compter les intersections avec une triangulation fixée :



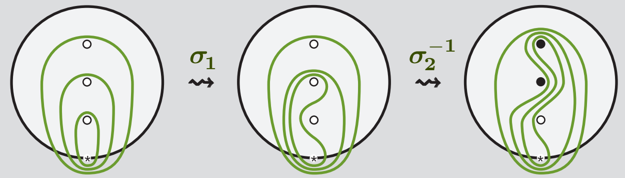
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



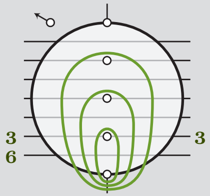
- Compter les intersections avec une triangulation fixée :



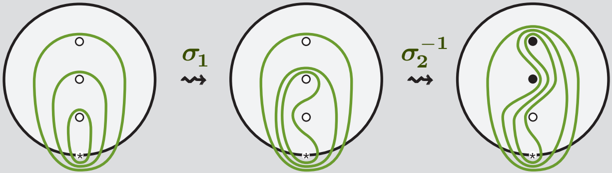
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



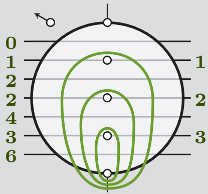
- Compter les intersections avec une triangulation fixée :



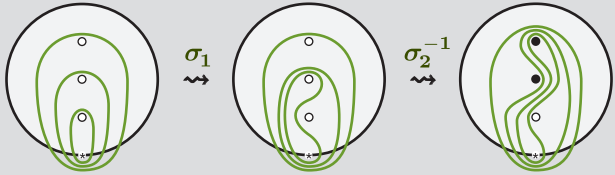
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



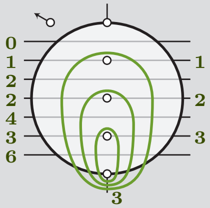
- Compter les intersections avec une triangulation fixée :



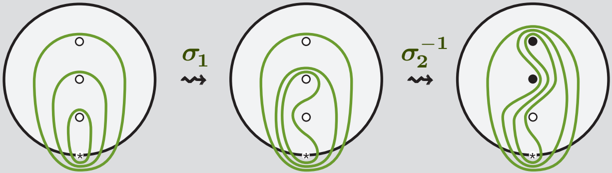
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



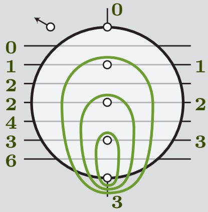
- Compter les intersections avec une triangulation fixée :



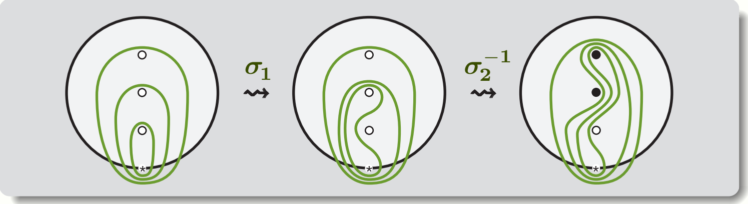
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



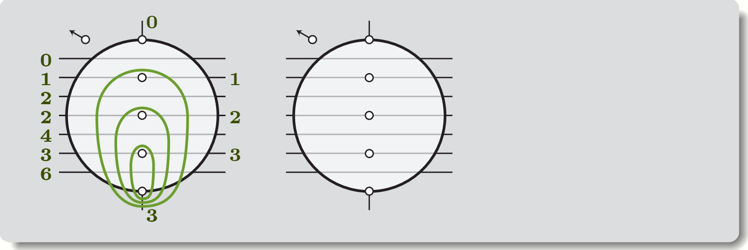
- Compter les intersections avec une triangulation fixée :



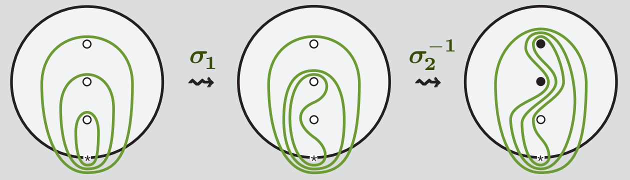
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



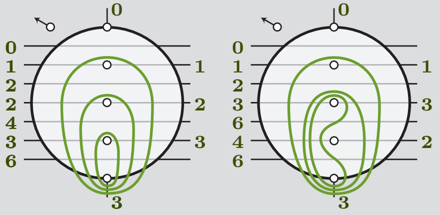
- Compter les intersections avec une triangulation fixée :



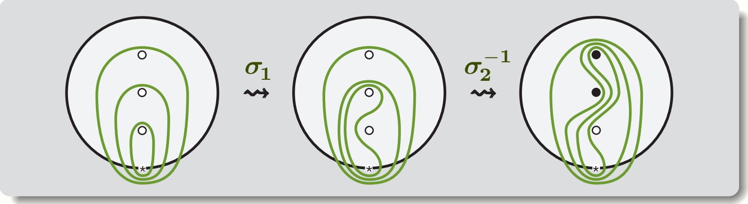
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



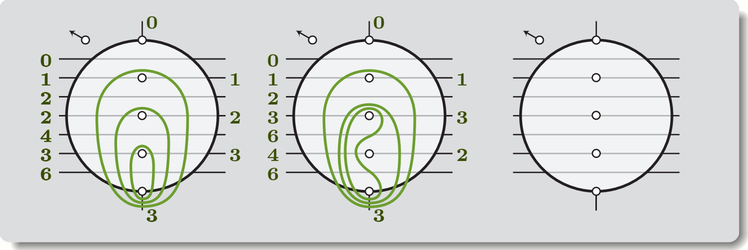
- Compter les intersections avec une triangulation fixée :



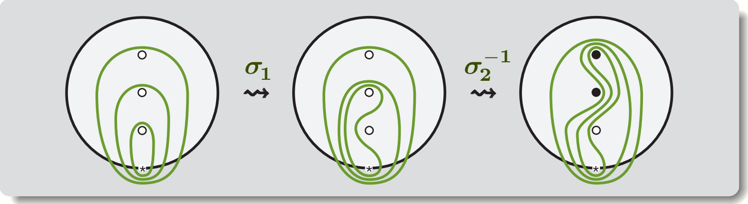
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



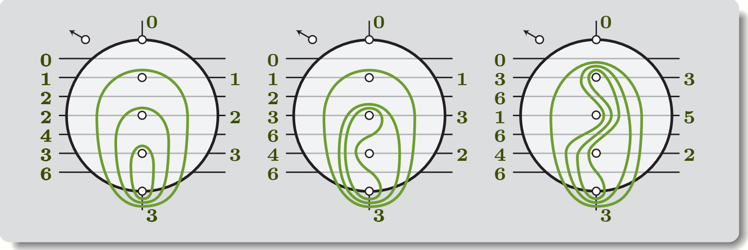
- Compter les intersections avec une triangulation fixée :



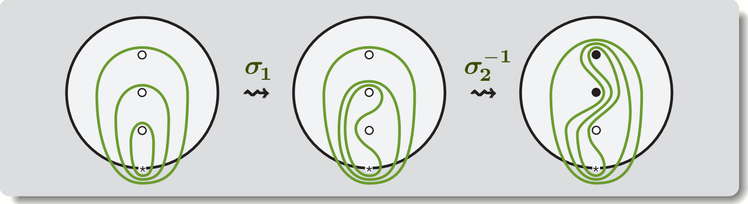
- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



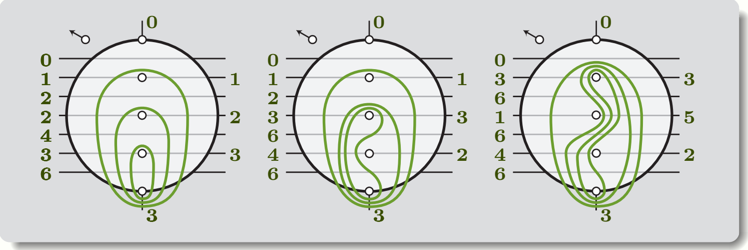
- Compter les intersections avec une triangulation fixée :



- Tresse=homéomorphisme $\rightsquigarrow$ agit sur les courbes tracées sur D_n.



- Compter les intersections avec une triangulation fixée :



$\rightsquigarrow$ $3n+3$ nombres, déterminant la tresse

- **Coordonnées = demi-différences entre nombres d'intersection ;**
  (passage de $3n+3$ à $2n$ entiers)

- Coordonnées = demi-différences entre nombres d'intersection ;
  (passage de $3n + 3$ à $2n$ entiers)

Question : Coordonnées de $\beta\sigma_i$ à partir de celles de β et de i ?

- **Coordonnées = demi-différences entre nombres d'intersection ;**

  (passage de $3n+3$ à $2n$ entiers)

Question : Coordonnées de $\beta\sigma_i$ à partir de celles de β et de i ?

= comparer les intersections de C et $\sigma_i(C)$ avec la triangulation T

courbe(s) fermée(s)

- Coordonnées = demi-différences entre nombres d'intersection ;

(passage de $3n+3$ à $2n$ entiers)

Question : Coordonnées de $\beta\sigma_i$ à partir de celles de β et de i ?

= comparer les intersections de C et $\sigma_i(C)$ avec la triangulation T

courbe(s) fermée(s)

- Or on a $$\#(\boldsymbol{\sigma_i}(\boldsymbol{C}) \cap \boldsymbol{T}) = \#(\boldsymbol{C} \cap \boldsymbol{\sigma_i^{-1}}(\boldsymbol{T}))$$

- Coordonnées = demi-différences entre nombres d'intersection ;

(passage de $3n+3$ à $2n$ entiers)

Question : Coordonnées de $\beta\sigma_i$ à partir de celles de β et de i ?

= comparer les intersections de C et $\sigma_i(C)$ avec la triangulation T

courbe(s) fermée(s)

- Or on a $$\#(\sigma_i(C) \cap T) = \#(C \cap \sigma_i^{-1}(T))$$

$\rightsquigarrow$ comparer les intersections de C avec T et $\sigma_i^{-1}(T)$.

- Coordonnées = demi-différences entre nombres d'intersection ;
  (passage de $3n+3$ à $2n$ entiers)

Question : Coordonnées de $\beta\sigma_i$ à partir de celles de β et de i ?

= comparer les intersections de C et $\sigma_i(C)$ avec la triangulation T
courbe(s) fermée(s)

- Or on a $$\#(\boldsymbol{\sigma_i}(\boldsymbol{C}) \cap \boldsymbol{T}) = \#(\boldsymbol{C} \cap \boldsymbol{\sigma_i}^{-1}(\boldsymbol{T}))$$

$\rightsquigarrow$ comparer les intersections de C avec T et $\sigma_i^{-1}(T)$.

Proposition : Si T, T' sont deux triangulations (singulières) d'une surface, on peut passer de T à T' par une suite finie de flips.

- Coordonnées = demi-différences entre nombres d'intersection ;
(passage de $3n+3$ à $2n$ entiers)

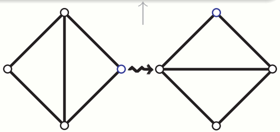
Question : Coordonnées de $\beta\sigma_i$ à partir de celles de β et de i ?

= comparer les intersections de C et $\sigma_i(C)$ avec la triangulation T
courbe(s) fermée(s)

- Or on a $$\boldsymbol{\#(\sigma_i(C)\cap T) = \#(C\cap \sigma_i^{-1}(T))}$$

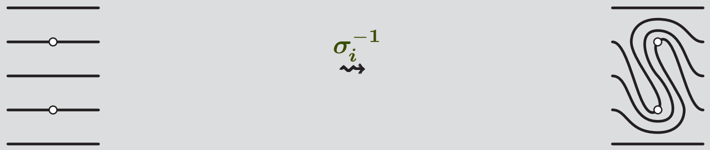
$\rightsquigarrow$ comparer les intersections de C avec T et $\sigma_i^{-1}(T)$.

Proposition : Si T, T' sont deux triangulations (singulières) d'une surface, on peut passer de T à T' par une suite finie de flips.

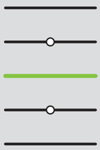


- **Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :**

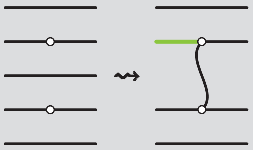
- Donc on **doit** passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



- **Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :**

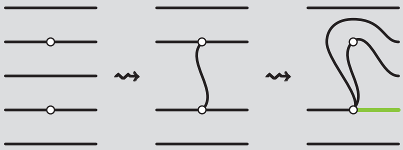


- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :

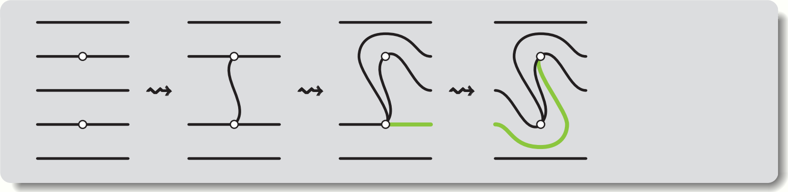


- Donc on **doit** passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :

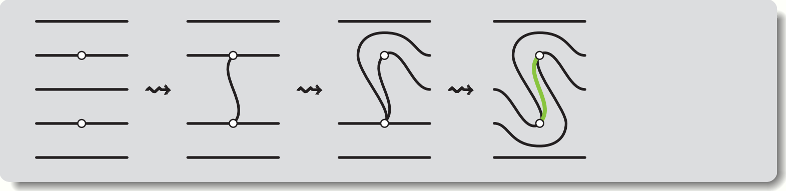
- Donc on **doit** passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



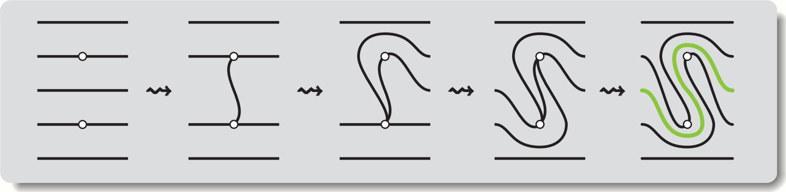
- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



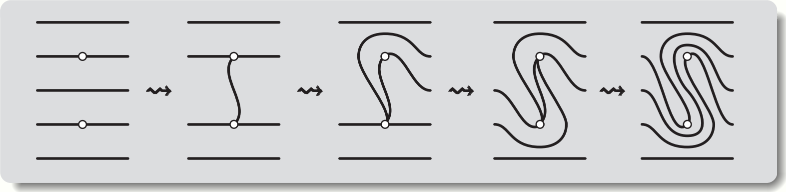
- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



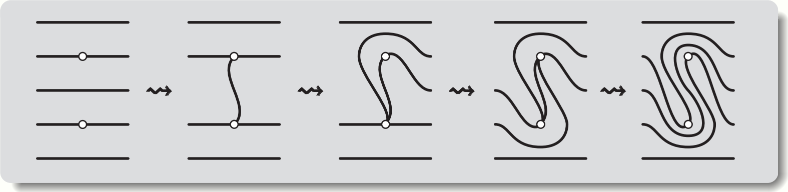
- Donc on **doit** passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



- Donc on **doit** passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :

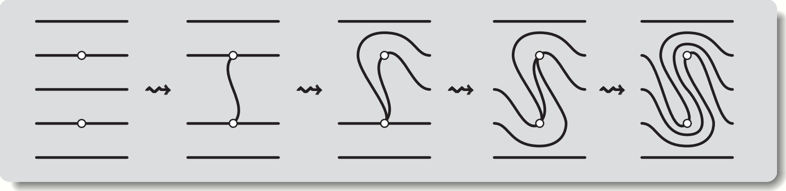


- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :

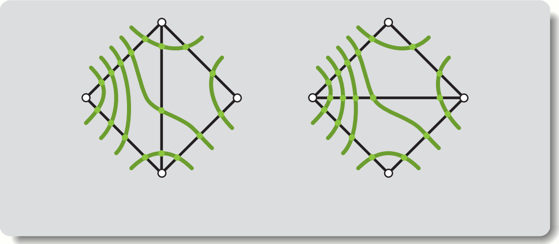


- Pour un flip, la formule est

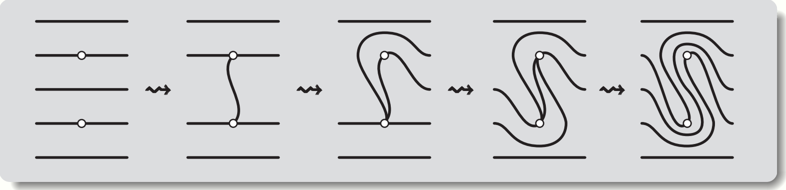
- Donc on **doit** passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



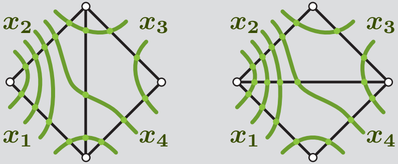
- Pour **un** flip, la formule est



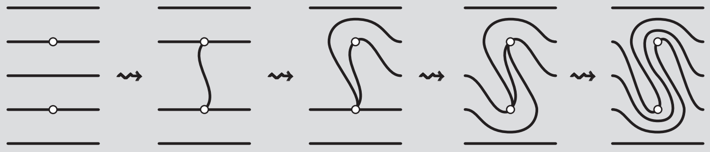
- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



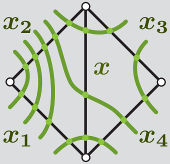
- Pour un flip, la formule est



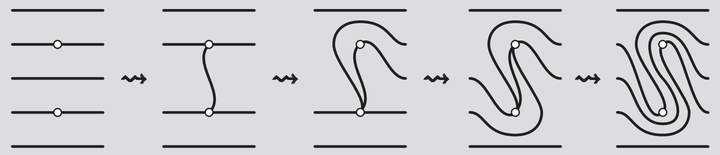
- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



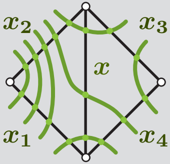
- Pour un flip, la formule est



- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :

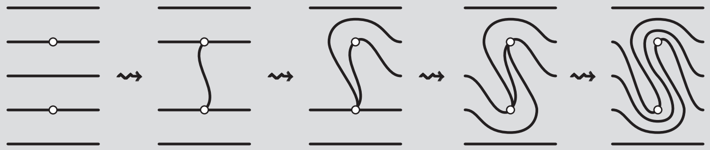


- Pour un flip, la formule est

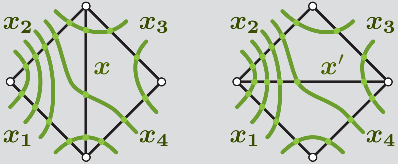


$x + x' =$

- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :

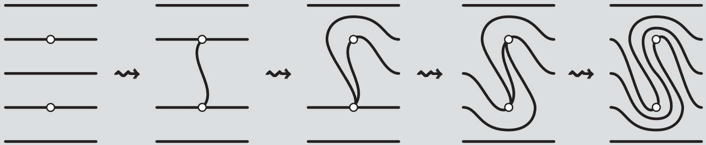


- Pour un flip, la formule est

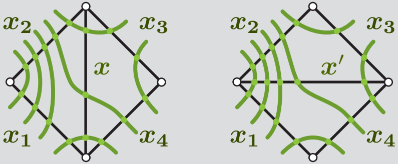


$$x + x' = \max(x_1 + x_3, x_2 + x_4)$$

- Donc on doit passer de T à $\sigma_i^{-1}(T)$ par une suite finie de flips :



- Pour un flip, la formule est



$$x + x' = \max(x_1 + x_3, x_2 + x_4)$$

$\rightsquigarrow$ formules de Dynnikov en itérant quatre fois.

Solution : Formes normales alternante et cyclante

Solution : Formes normales alternante et cyclante

- **Domaine :** algèbre / combinatoire

Solution : Formes normales alternante et cyclante

- **Domaine :** algèbre / combinatoire
- **Point de vue :** monoïde de tresse

Solution : Formes normales alternante et cyclante

- **Domaine :** algèbre / combinatoire
- **Point de vue :** monoïde de tresse
- **Méthode :** forme normale

Solution : Formes normales alternante et cyclante

- **Domaine :** algèbre / combinatoire
- **Point de vue :** monoïde de tresse
- **Méthode :** forme normale
- **Auteurs :** Burckel'94,

Solution : Formes normales alternante et cyclante

- Domaine : algèbre / combinatoire
- Point de vue : monoïde de tresse
- Méthode : forme normale
- Auteurs : Burckel'94, D.'07,

Solution : Formes normales alternante et cyclante

- Domaine : algèbre / combinatoire
- Point de vue : monoïde de tresse
- Méthode : forme normale
- Auteurs : Burckel'94, D.'07, Fromentin'07

Solution : Formes normales alternante et cyclante

- Domaine : algèbre / combinatoire
- Point de vue : monoïde de tresse
- Méthode : forme normale
- Auteurs : Burckel'94, D.'07, Fromentin'07
- Mots-clés : automorphisme, générateurs de Birman–Ko–Lee

Solution : Formes normales alternante et cyclante

- **Domaine :** algèbre / combinatoire
- **Point de vue :** monoïde de tresse
- **Méthode :** forme normale
- **Auteurs :** Burckel'94, D.'07, Fromentin'07
- Mots-clés : automorphisme, générateurs de Birman–Ko–Lee
- Arrière-plan : combinatoire des partitions non croisées

Solution : Formes normales alternante et cyclante

- **Domaine** : algèbre / combinatoire
- **Point de vue** : monoïde de tresse
- **Méthode** : forme normale
- **Auteurs** : Burckel'94, D.'07, Fromentin'07

- Mots-clés : automorphisme, générateurs de Birman–Ko–Lee
- Arrière-plan : combinatoire des partitions non croisées
- Extensions : catégories de Garside

- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

  «flip»: symétrie horizontale des diagrammes

- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

  «flip»: symétrie horizontale des diagrammes

- Toute tresse de B_n^+ admet une unique décomposition

$$\beta = \beta_1 \cdot \Phi_n(\beta_2) \cdot \ldots \cdot \Phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^+

- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

  «flip»: symétrie horizontale des diagrammes

- Toute tresse de B_n^+ admet une unique décomposition

$$\beta = \beta_1 \cdot \Phi_n(\beta_2) \cdot ... \cdot \Phi_n^{p-1}(\beta_p)$$

avec $\beta_1, ..., \beta_p$ dans B_{n-1}^+ t.q., pour tout $r \geqslant 2$,

aucun σ_i avec $i < n-1$ ne divise $\Phi_n(\beta_r) \cdot ... \cdot \Phi_n^{p-r+1}(\beta_p)$.

- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

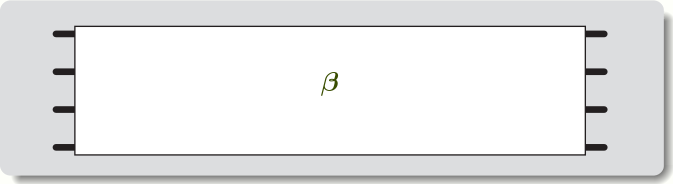
  «flip»: symétrie horizontale des diagrammes

- Toute tresse de B_n^+ admet une unique décomposition

$$\beta = \beta_1 \cdot \Phi_n(\beta_2) \cdot \ldots \cdot \Phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^+ t.q., pour tout $r \geqslant 2$,

aucun σ_i avec $i < n-1$ ne divise $\Phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

  «flip»: symétrie horizontale des diagrammes

- Toute tresse de B_n^+ admet une unique décomposition

$$\beta = \beta_1 \cdot \Phi_n(\beta_2) \cdot \ldots \cdot \Phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^+ t.q., pour tout $r \geqslant 2$,

aucun σ_i avec $i < n-1$ ne divise $\Phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.

- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

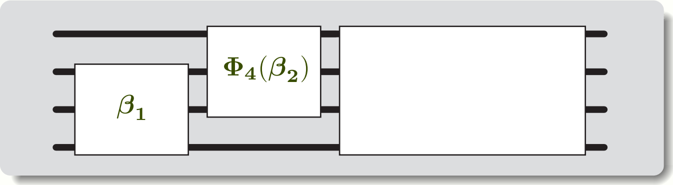
  «flip»: symétrie horizontale des diagrammes

- Toute tresse de B_n^+ admet une unique décomposition

$$\beta = \beta_1 \cdot \Phi_n(\beta_2) \cdot \ldots \cdot \Phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^+ t.q., pour tout $r \geqslant 2$,

aucun σ_i avec $i < n-1$ ne divise $\Phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

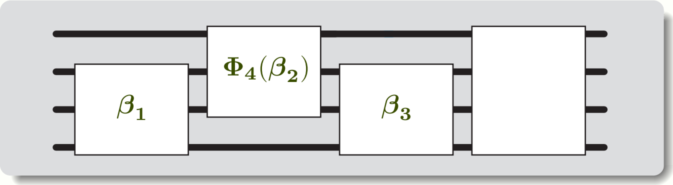
  «flip»: symétrie horizontale des diagrammes

- Toute tresse de B_n^+ admet une unique décomposition

$$\beta = \beta_1 \cdot \Phi_n(\beta_2) \cdot \ldots \cdot \Phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^+ t.q., pour tout $r \geqslant 2$,

aucun σ_i avec $i < n-1$ ne divise $\Phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

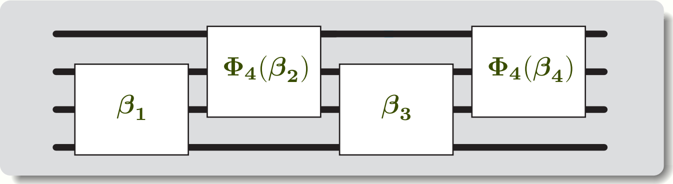
  «flip»: symétrie horizontale des diagrammes

- Toute tresse de B_n^+ admet une unique décomposition

$$\beta = \beta_1 \cdot \Phi_n(\beta_2) \cdot \ldots \cdot \Phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^+ t.q., pour tout $r \geqslant 2$,

aucun σ_i avec $i < n-1$ ne divise $\Phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- Φ_n : automorphisme de B_n échangeant σ_i et σ_{n-i} pour $i < n$.

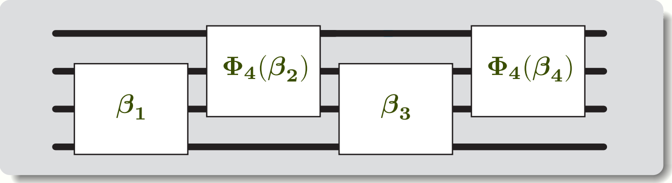
  «flip»: symétrie horizontale des diagrammes

- Toute tresse de B_n^+ admet une unique décomposition

$$\beta = \beta_1 \cdot \Phi_n(\beta_2) \cdot \ldots \cdot \Phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^+ t.q., pour tout $r \geqslant 2$,

aucun σ_i avec $i < n-1$ ne divise $\Phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- En itérant, forme normale

  — et solution quadratique au problème d'isotopie.

- (Birman–Ko–Lee '97) d'autres générateurs de B_n :

- **(Birman–Ko–Lee '97) d'autres générateurs de B_n :**

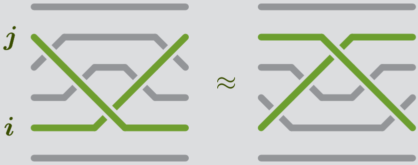
$$a_{i,j} = \sigma_{j-1}...\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1}...\sigma_{j-1}^{-1} \text{ pour } 1 \leqslant i < j \leqslant n.$$

- (Birman–Ko–Lee '97) d'autres générateurs de B_n :

$$a_{i,j} = \sigma_{j-1}...\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1}...\sigma_{j-1}^{-1} \text{ pour } 1 \leqslant i < j \leqslant n.$$

- (Birman–Ko–Lee '97) d'autres générateurs de B_n :

$$a_{i,j} = \sigma_{j-1}\ldots\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1}\ldots\sigma_{j-1}^{-1} \text{ pour } 1 \leqslant i < j \leqslant n.$$



- (**Birman–Ko–Lee** '97) d'autres générateurs de B_n :

$$a_{i,j} = \sigma_{j-1}...\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1}...\sigma_{j-1}^{-1} \text{ pour } 1 \leqslant i < j \leqslant n.$$

- (Birman–Ko–Lee '97) d'autres générateurs de B_n :

$$a_{i,j} = \sigma_{j-1} \dots \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1} \dots \sigma_{j-1}^{-1} \text{ pour } 1 \leqslant i < j \leqslant n.$$

- (Birman–Ko–Lee '97) d'autres générateurs de B_n :

$$a_{i,j} = \sigma_{j-1} ... \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1} ... \sigma_{j-1}^{-1} \text{ pour } 1 \leqslant i < j \leqslant n.$$

- Monoïde dual : le sous-monoïde B_n^{+*} de B_n engendré par les $a_{i,j}$.

- (Birman–Ko–Lee '97) d'autres générateurs de B_n :

$$a_{i,j} = \sigma_{j-1}...\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1}...\sigma_{j-1}^{-1} \text{ pour } 1 \leqslant i < j \leqslant n.$$

- Monoïde **dual** : le sous-monoïde B_n^{+*} de B_n engendré par les $a_{i,j}$.

Théorème : Le monoïde B_n^{+*} a une structure de Garside, où le rôle de Δ_n est joué par $\delta_n = \sigma_{n-1}...\sigma_2\sigma_1$, et où les diviseurs de δ_n sont en bijection avec les **partitions non croisées** de $\{1, ..., n\}$.

- (Birman–Ko–Lee '97) d'autres générateurs de B_n :

$$a_{i,j} = \sigma_{j-1} \ldots \sigma_{i+1} \sigma_i \sigma_{i+1}^{-1} \ldots \sigma_{j-1}^{-1} \text{ pour } 1 \leqslant i < j \leqslant n.$$

- Monoïde dual : le sous-monoïde B_n^{+*} de B_n engendré par les $a_{i,j}$.

Théorème : Le monoïde B_n^{+*} a une structure de Garside, où le rôle de Δ_n est joué par $\delta_n = \sigma_{n-1} \ldots \sigma_2 \sigma_1$, et où les diviseurs de δ_n sont en bijection avec les partitions non croisées de $\{1, \ldots, n\}$.

De là, solution du problème d'isotopie par « greedy normal form ».

- **Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;**

- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

⇝ **automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.**

- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

$\rightsquigarrow$ **automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.**

$$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$$

- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

⇝ automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.

$$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$$

- Toute tresse de B_n^{+*} admet une unique décomposition

$$\beta = \beta_1 \cdot \phi_n(\beta_2) \cdot \ldots \cdot \phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^{+*}

- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

$\rightsquigarrow$ automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.

$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$

- Toute tresse de B_n^{+*} admet une unique décomposition

$$\beta = \beta_1 \cdot \phi_n(\beta_2) \cdot \ldots \cdot \phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^{+*} t.q., pour tout $r \geqslant 2$,

aucun $a_{i,j}$ avec $j < n$ ne divise $\phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.

- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

$\rightsquigarrow$ automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.

$$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$$

- Toute tresse de B_n^{+*} admet une unique décomposition

$$\beta = \beta_1 \cdot \phi_n(\beta_2) \cdot \ldots \cdot \phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^{+*} t.q., pour tout $r \geqslant 2$,
aucun $a_{i,j}$ avec $j < n$ ne divise $\phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.

- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

$\rightsquigarrow$ automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.

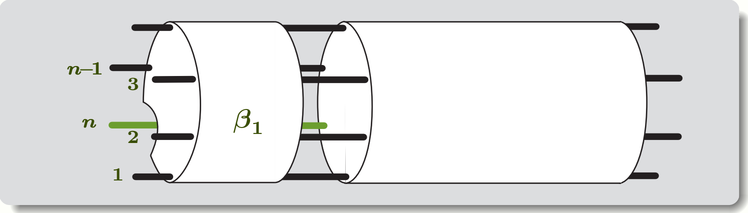
$$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$$

- Toute tresse de B_n^{+*} admet une unique décomposition

$$\beta = \beta_1 \cdot \phi_n(\beta_2) \cdot \ldots \cdot \phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^{+*} t.q., pour tout $r \geqslant 2$,

aucun $a_{i,j}$ avec $j < n$ ne divise $\phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

$\rightsquigarrow$ automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.

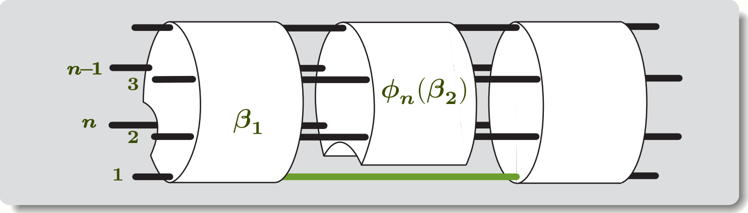
$$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$$

- Toute tresse de B_n^{+*} admet une unique décomposition

$$\beta = \beta_1 \cdot \phi_n(\beta_2) \cdot \ldots \cdot \phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^{+*} t.q., pour tout $r \geqslant 2$,

aucun $a_{i,j}$ avec $j < n$ ne divise $\phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

⇝ automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.

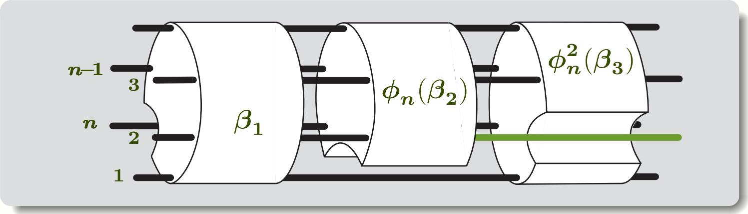
$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$

- Toute tresse de B_n^{+*} admet une unique décomposition

$$\beta = \beta_1 \cdot \phi_n(\beta_2) \cdot \ldots \cdot \phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^{+*} t.q., pour tout $r \geqslant 2$,

aucun $a_{i,j}$ avec $j < n$ ne divise $\phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

$\rightsquigarrow$ automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.

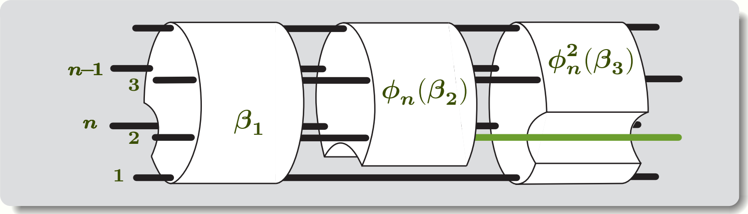
$$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$$

- Toute tresse de B_n^{+*} admet une unique décomposition

$$\beta = \beta_1 \cdot \phi_n(\beta_2) \cdot ... \cdot \phi_n^{p-1}(\beta_p)$$

avec $\beta_1, ..., \beta_p$ dans B_{n-1}^{+*} t.q., pour tout $r \geqslant 2$,

aucun $a_{i,j}$ avec $j < n$ ne divise $\phi_n(\beta_r) \cdot ... \cdot \Phi_n^{p-r+1}(\beta_p)$.



- Automorphisme Φ_n de B_n^+ («flip») = conjugaison par Δ_n ;

$\rightsquigarrow$ automorphisme ϕ_n de B_n^{+*} («cyclage») = conjugaison par δ_n.

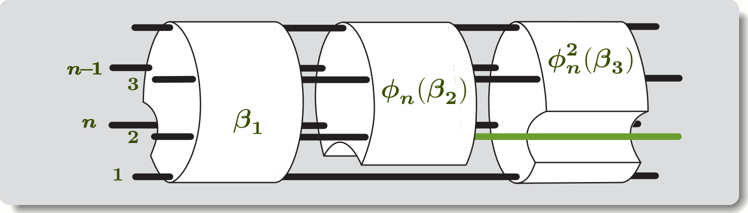
$$\phi_n(a_{i,j}) = a_{i+1 \bmod n, j+1 \bmod n}$$

- Toute tresse de B_n^{+*} admet une unique décomposition

$$\beta = \beta_1 \cdot \phi_n(\beta_2) \cdot \ldots \cdot \phi_n^{p-1}(\beta_p)$$

avec $\beta_1, \ldots, \beta_p$ dans B_{n-1}^{+*} t.q., pour tout $r \geqslant 2$,

aucun $a_{i,j}$ avec $j < n$ ne divise $\phi_n(\beta_r) \cdot \ldots \cdot \Phi_n^{p-r+1}(\beta_p)$.



- En itérant, forme normale — et solution au problème d'isotopie.

Solution : Algorithme(s) de relaxation

Solution : Algorithme(s) de relaxation

- **Domaine :** topologie

Solution : Algorithme(s) de relaxation

- Domaine : topologie
- Point de vue : tresse = homéomorphisme

Solution : Algorithme(s) de relaxation

- Domaine : topologie
- Point de vue : tresse = homéomorphisme
- Méthode : forme normale

Solution : Algorithme(s) de relaxation

- Domaine : topologie
- Point de vue : tresse = homéomorphisme
- Méthode : forme normale
- Auteur : X.Bressaud '05

Solution : Algorithme(s) de relaxation

- Domaine : topologie
- Point de vue : tresse = homéomorphisme
- Méthode : forme normale
- Auteur : X.Bressaud '05
- Mots-clés : homémorphisme, lacet, stratégie de relaxation

Solution : Algorithme(s) de relaxation

- Domaine : topologie
- Point de vue : tresse = homéomorphisme
- Méthode : forme normale
- Auteur : X.Bressaud ’05
- Mots-clés : homémorphisme, lacet, stratégie de relaxation
- Arrière-plan : systèmes dynamiques

Solution : Algorithme(s) de relaxation

- **Domaine :** topologie
- **Point de vue :** tresse = homéomorphisme
- **Méthode :** forme normale
- **Auteur :** X.Bressaud '05
- Mots-clés : homémorphisme, lacet, stratégie de relaxation
- Arrière-plan : systèmes dynamiques
- Extensions : frontière de Poisson, problèmes de stabilisation

- **Tresse = homéomorphisme.**

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C :

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une stratégie pour relaxer (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de complexité.

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) :

- Tresse $=$ homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une stratégie pour relaxer ($=$ demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de complexité.

- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) :
  C $=$ diamètre principal de D_n,

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

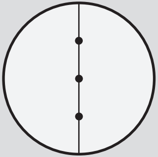
- Exemple 1 (Fenn et al. ’97, Dynnikov–Wiest ’06) : C = diamètre principal de D_n, stratégie = « arc utile ».

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) : C = diamètre principal de D_n, stratégie = « arc utile ».

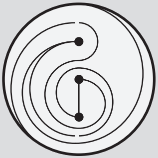


- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une stratégie pour relaxer (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de complexité.

- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) : C = diamètre principal de D_n, stratégie = « arc utile ».



- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

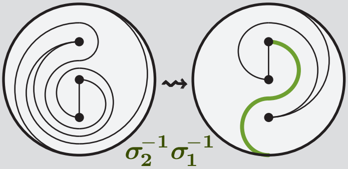
- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) : C = diamètre principal de D_n, stratégie = « arc utile ».

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) : C = diamètre principal de D_n, stratégie = « arc utile ».

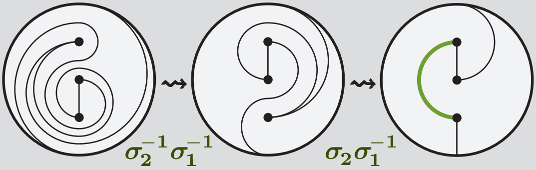


- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) : C = diamètre principal de D_n, stratégie = « arc utile ».

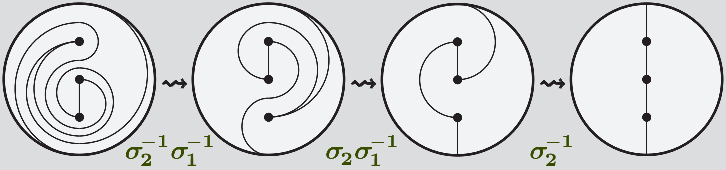


- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) :
  C = diamètre principal de D_n, stratégie = « arc utile ».

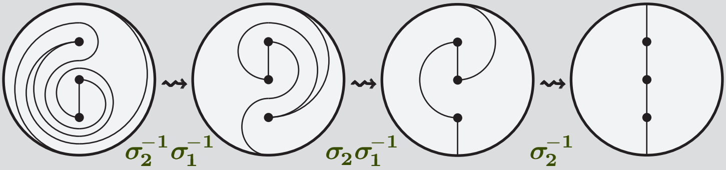


- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir
une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C :
la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) :
C = diamètre principal de D_n, stratégie = « arc utile ».



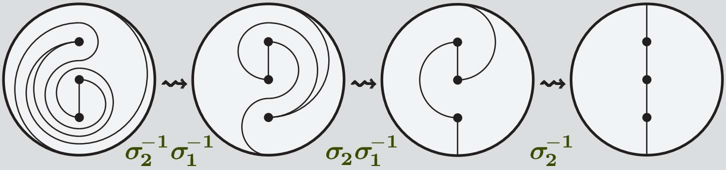
d'où $\beta = \sigma_2\sigma_1\sigma_2^{-1}\sigma_1\sigma_2$

- Tresse = homéomorphisme.

Principe : Fixer une (ou des) courbe de base C, et définir une **stratégie** pour **relaxer** (= demêler) $\beta(C)$ et revenir à C : la suite des $\sigma_i^{\pm 1}$ utilisés donne une expression de β^{-1}.

— suppose de définir une notion de **complexité**.

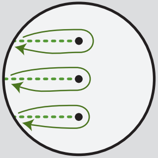
- Exemple 1 (Fenn et al. '97, Dynnikov–Wiest '06) :
  C = diamètre principal de D_n, stratégie = « arc utile ».



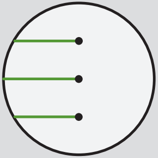
d'où $\beta = \sigma_2\sigma_1\sigma_2^{-1}\sigma_1\sigma_2$

- **Exemple 2 (Bressaud '05) :**
  C = axes des **lacets standards**

- **Exemple 2 (Bressaud '05) :**
  C = axes des **lacets standards** (cf. représentation d'Artin) ;
  stratégie : relaxer $\beta(x_1)$, puis $\beta(x_2)$, etc.
  en diminuant les intersections avec les demi-axes.



- **Exemple 2 (Bressaud '05) :**
  C = axes des **lacets standards** (cf. représentation d'Artin) ;
  stratégie : relaxer $\beta(x_1)$, puis $\beta(x_2)$, etc.
  en diminuant les intersections avec les demi-axes.

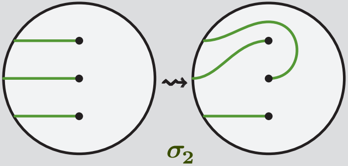


- **Exemple 2** (**Bressaud** '05) :

  C = axes des **lacets standards** (cf. représentation d'Artin) ;

  stratégie : relaxer $\beta(x_1)$, puis $\beta(x_2)$, etc.

  en diminuant les intersections avec les demi-axes.

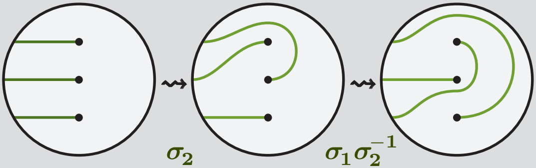


- **Exemple 2** (**Bressaud** '05) :

  C = axes des **lacets standards** (cf. représentation d'Artin) ;

  stratégie : relaxer $\beta(x_1)$, puis $\beta(x_2)$, etc.

  en diminuant les intersections avec les demi-axes.

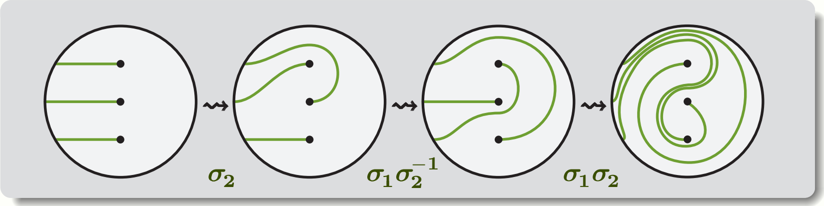


- **Exemple 2** (**Bressaud** '05) :

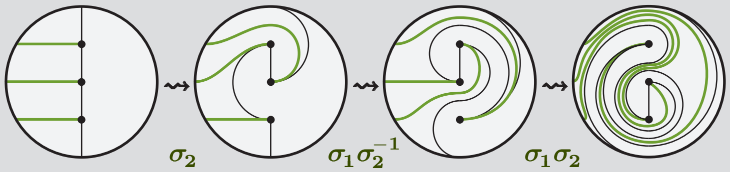
  C = axes des **lacets standards** (cf. représentation d'Artin) ;

  stratégie : relaxer $\beta(x_1)$, puis $\beta(x_2)$, etc.

  en diminuant les intersections avec les demi-axes.



- **Exemple 2 (Bressaud '05) :**
  C = axes des **lacets standards** (cf. représentation d'Artin) ;
  stratégie : relaxer $\beta(x_1)$, **puis** $\beta(x_2)$, **etc.**
  en diminuant les intersections avec les demi-axes.

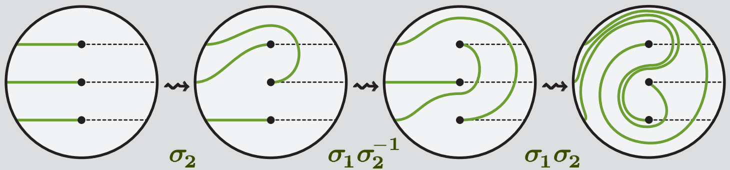


- **Exemple 2** (**Bressaud** '05) :

  C = axes des **lacets standards** (cf. représentation d'Artin) ;

  stratégie : relaxer $\beta(x_1)$, puis $\beta(x_2)$, etc.

  en diminuant les intersections avec les demi-axes.

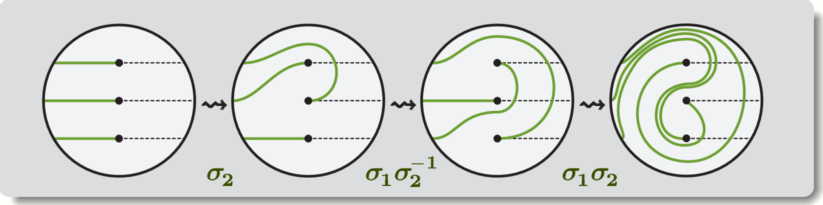


- **Exemple 2** (**Bressaud** '05) :

  C = axes des **lacets standards** (cf. représentation d'Artin) ;

  stratégie : relaxer $\beta(x_1)$, puis $\beta(x_2)$, etc.

  en diminuant les intersections avec les demi-axes.



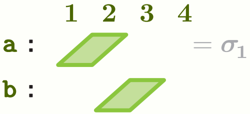
- Une forme normale – donc une solution au problème d'isotopie –

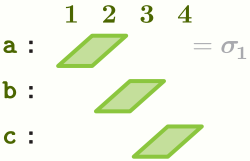
… mais surtout un **algorithme** calculant la forme normale de $w\sigma_i^{\pm 1}$

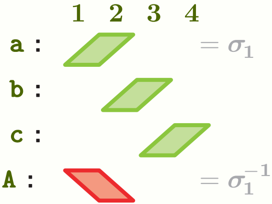
à partir de celle de w et de i.

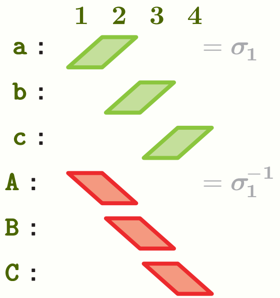
$\mathbf{1} \quad \mathbf{2} \quad \mathbf{3} \quad \mathbf{4}$

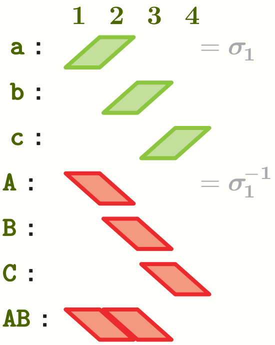
a :  $= \sigma_1$


1 2 3 4
a :
b :
$= \sigma_1$

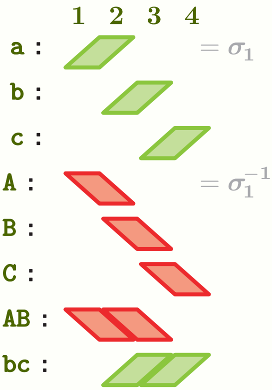

1 2 3 4
a :
b :
c :
$= \sigma_1$

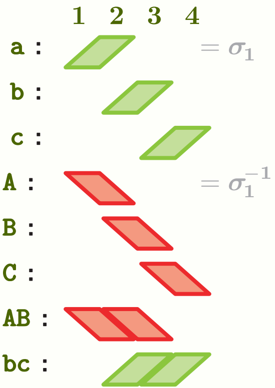

1 2 3 4
a :
$= \sigma_1$
b :
c :
A :
$= \sigma_1^{-1}$


1 2 3 4
a :
$= \sigma_1$
b :
c :
A :
$= \sigma_1^{-1}$
B :
C :

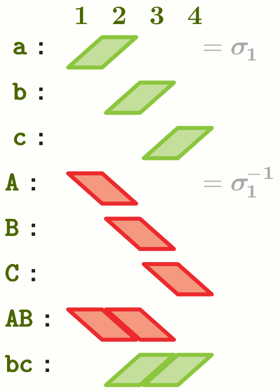

1 2 3 4
a :
$= \sigma_1$
b :
c :
A :
$= \sigma_1^{-1}$
B :
C :
AB :

Algorithme de Bressaud

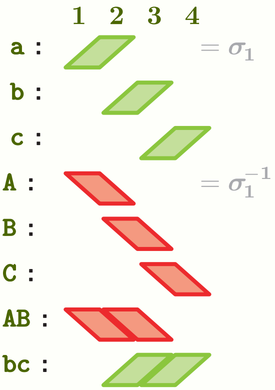




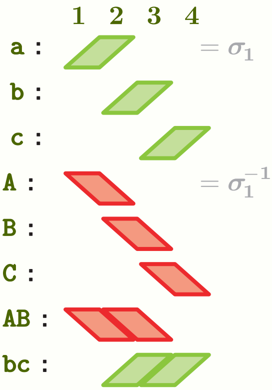
- Forme normale de Bressaud de ε $= \varepsilon$.



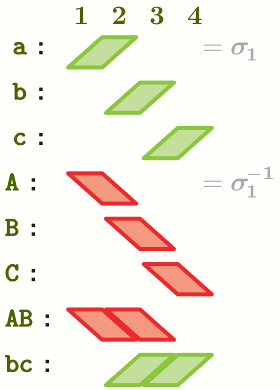
- Forme normale de Bressaud de **A**



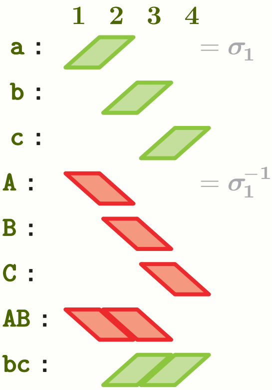
- Forme normale de Bressaud de A $=$ A.



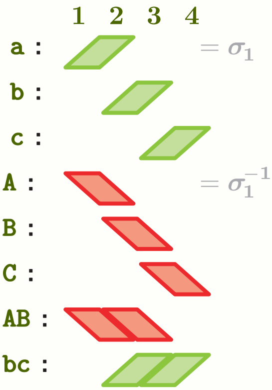
- Forme normale de Bressaud de AB



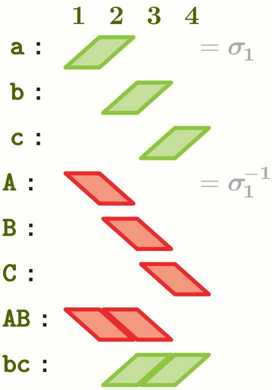
- **Forme normale de Bressaud de AB $=$ AB.**



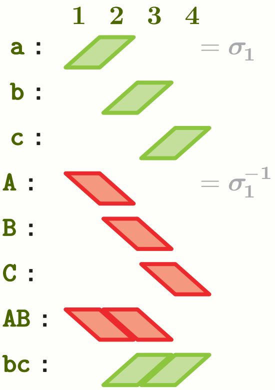
- Forme normale de Bressaud de ABa



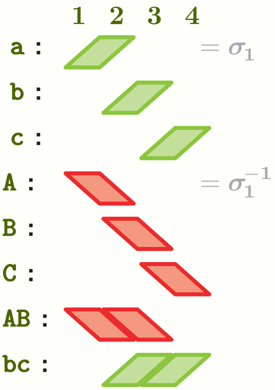
- Forme normale de Bressaud de ABa



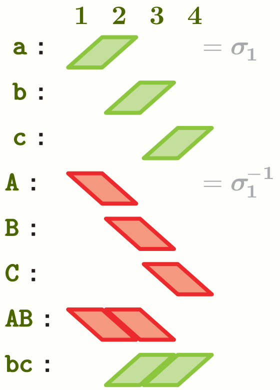
- Forme normale de Bressaud de ABa $=$ b.AB.



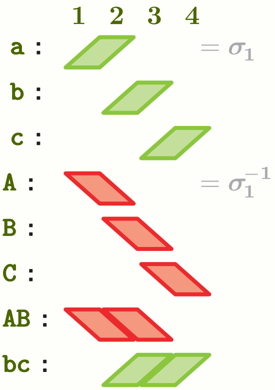
- Forme normale de Bressaud de ABaa



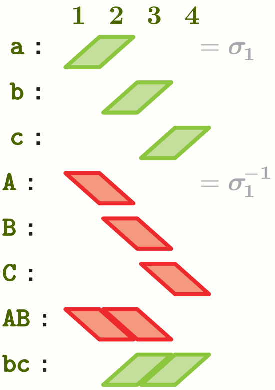
- Forme normale de Bressaud de ABaa



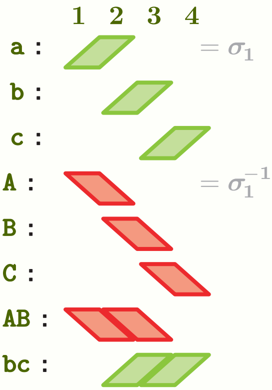
- Forme normale de Bressaud de ABaa $=$ b.b.AB.



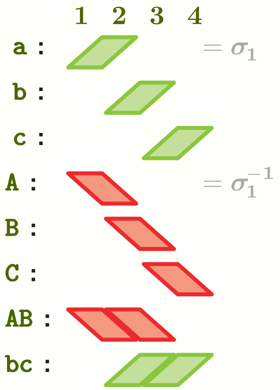
- Forme normale de Bressaud de ABaaC

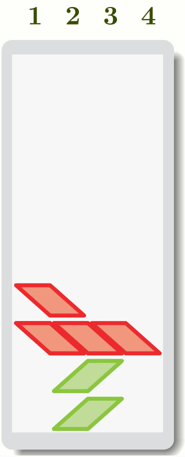


- Forme normale de Bressaud de ABaaC $=$ b.b.ABC.

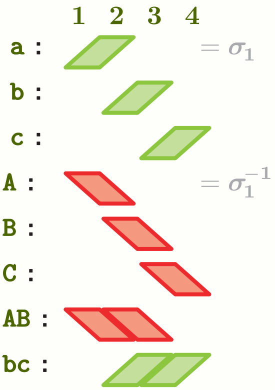


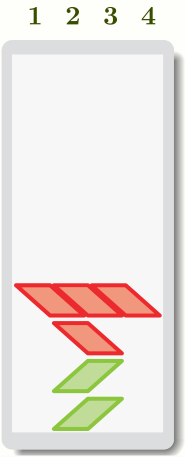
- Forme normale de Bressaud de ABaaCA



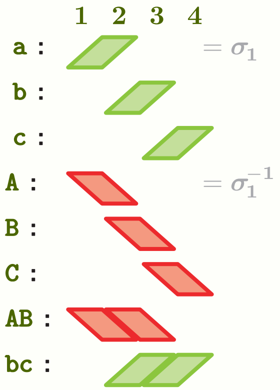


- Forme normale de Bressaud de `ABaaCA`

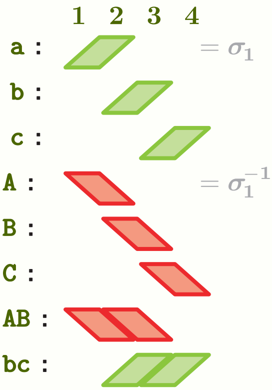




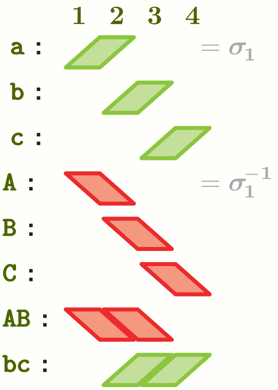
- Forme normale de Bressaud de ABaaCA



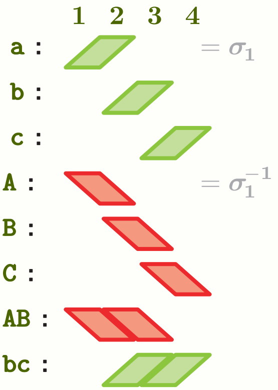
- Forme normale de Bressaud de ABaaCA $=$ b.ABC.



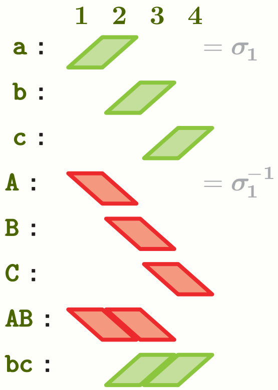
- Forme normale de Bressaud de ABaaCAb



- Forme normale de Bressaud de ABaaCAb

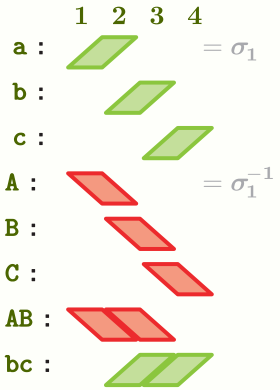


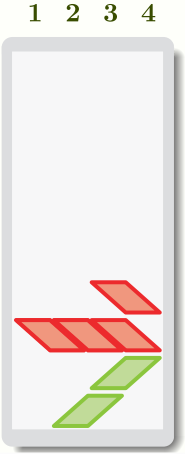
- **Forme normale de Bressaud de ABaaCAb $=$ b.c.ABC.**



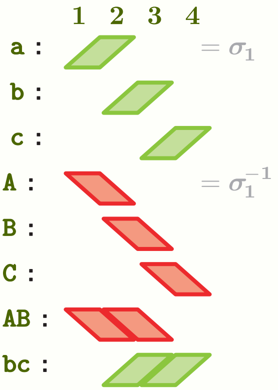
- Forme normale de Bressaud de ABaaCAbC

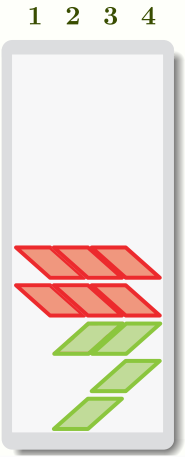
Algorithme de Bressaud





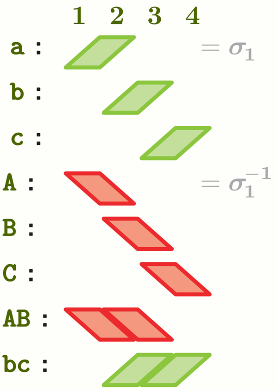
- Forme normale de Bressaud de ABaaCAbC

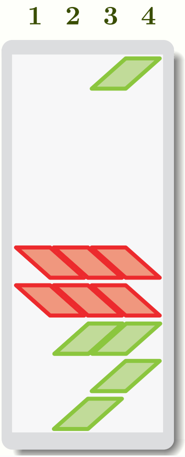




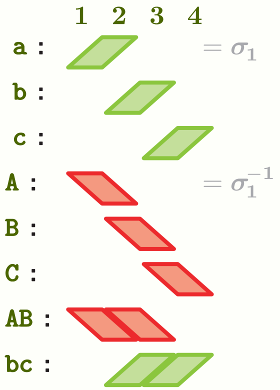
- Forme normale de Bressaud de ABaaCAbC = b.c.cb.ABC.ABC.

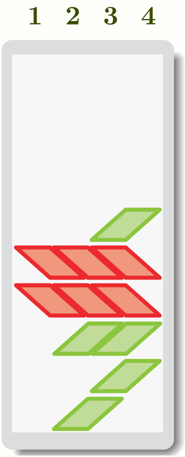
Algorithme de Bressaud



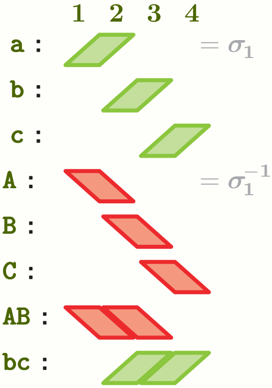


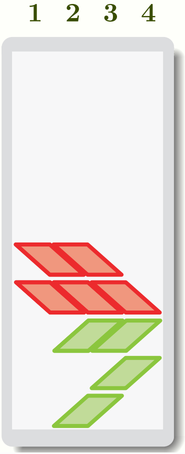
- Forme normale de Bressaud de ABaaCAbCc



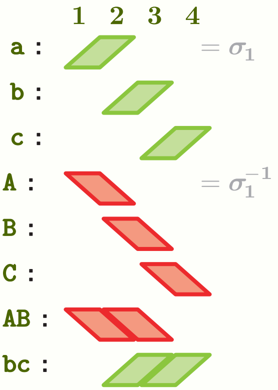


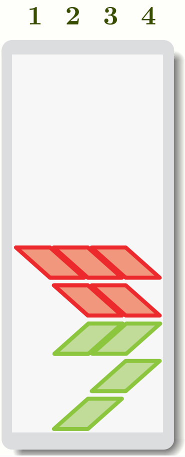
- Forme normale de Bressaud de ABaaCAbCc



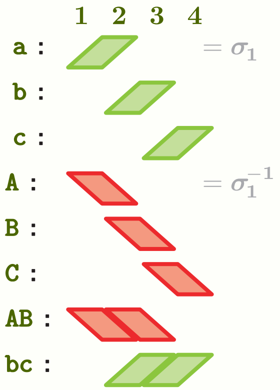


- Forme normale de Bressaud de ABaaCAbCc

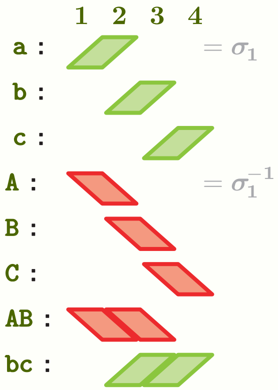


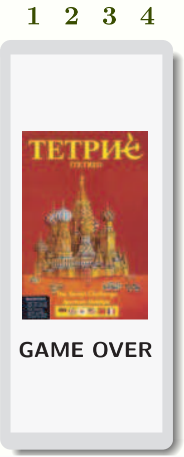


- Forme normale de Bressaud de ABaaCAbCc

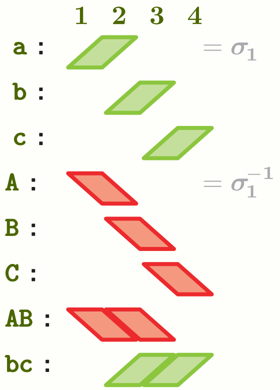


- Forme normale de Bressaud de ABaaCAbCc $=$ b.c.ABC.





- Forme normale de Bressaud de ABaaCAbCc $=$ b.c.ABC.



- Forme normale de Bressaud de ABaaCAbCc $=$ b.c.ABC.
- Question (J. Chamboredon) : Approche purement syntaxique ?

- On a survolé des solutions au problème d'isotopie des tresses,

- On a survolé des solutions au problème d'isotopie des tresses,
  variées,

- **On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...**

- On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...

Y en a-t-il encore d'autres ?

- On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...

Y en a-t-il encore d'autres ?

Quid du problème de conjugaison ?

- On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...

Y en a-t-il encore d'autres ?

Quid du problème de conjugaison ?

Quid des groupes d'Artin–Tits ?

- On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...

Y en a-t-il encore d'autres ?

Quid du problème de conjugaison ?

Quid des groupes d'Artin–Tits ?

Quid des groupes de tresses de surface ?

- On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...

Y en a-t-il encore d'autres ?

Quid du problème de conjugaison ?

Quid des groupes d'Artin–Tits ?

Quid des groupes de tresses de surface ?

Quid des mapping class groups généraux ?

- On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...

Y en a-t-il encore d'autres ?

Quid du problème de conjugaison ?

Quid des groupes d'Artin–Tits ?

Quid des groupes de tresses de surface ?

Quid des mapping class groups généraux ?

Quid de l'isotopie des nœuds ?

- On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...

Y en a-t-il encore d'autres ?

Quid du problème de conjugaison ?

Quid des groupes d'Artin–Tits ?

Quid des groupes de tresses de surface ?

Quid des mapping class groups généraux ?

Quid de l'isotopie des nœuds ?

...

- On a survolé des solutions au problème d'isotopie des tresses, variées, mettant en jeu beaucoup de jolies mathématiques...

Y en a-t-il encore d'autres ?

Quid du problème de conjugaison ?

Quid des groupes d'Artin–Tits ?

Quid des groupes de tresses de surface ?

Quid des mapping class groups généraux ?

Quid de l'isotopie des nœuds ?

...

www.math.unicaen.fr/∼dehornoy