A l'intersection de la combinatoire des mots et de la géométrie discrète : palindromes, symétries et pavages

Alexandre Blondin Massé

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques**

Arrêté ministériel : 7 août 2006

Présentée par

Alexandre BLONDIN MASSÉ

Thèse dirigée par **M. Laurent Vuillon** et
codirigée par **M. Srečko Brlek**

préparée au sein du **Laboratoire de mathématiques**
dans **l'École Doctorale de Mathématiques, Sciences et Technologies de l'Information, Informatique**

À l'intersection de la combinatoire des mots et de la géométrie discrète : palindromes, symétries et pavages

Thèse soutenue publiquement le **2 décembre 2011**,
devant le jury composé de :

M. Srečko BRLEK
Professeur de l'Université du Québec à Montréal, codirecteur de thèse
M. Laurent VUILLON
Professeur de l'Université de Savoie, directeur de thèse
M. Christophe REUTENAUER
Professeur de l'Université du Québec à Montréal, président
Mme Valérie BERTHÉ
Directrice de recherche à l'Université Paris Diderot, Paris 7, rapporteur
M. Xavier PROVENÇAL
Maître de conférence à l'Université de Savoie, examinateur
Mme Lila KARI
Professeure de University of Western Ontario, rapporteur

REMERCIEMENTS

J'aimerais tout d'abord remercier mon directeur québéco-franco-italiano-croate Srečko Brlek grâce auquel j'ai mené à terme mes études graduées. Tu m'as non seulement donné l'occasion d'entreprendre une carrière de recherche passionnante, mais tu as aussi toujours su me guider dans la bonne direction malgré mon entêtement, ma tendance à me disperser et ma désorganisation. Pour tout ça, je te dis merci.

Merci aussi à Laurent Vuillon, mon directeur savoyard, pour son accueil chaleureux à Chambéry, ses conseils précieux, ses intuitions géniales et son agréable compagnie. Mon année en France restera gravée à tout jamais dans ma mémoire et j'espère un jour y séjourner à nouveau pour profiter de la montagne, de la cuisine et, surtout, des idées mathématiques qui s'y trouvent.

Je tiens aussi à remercier Sébastien Labbé, mon collègue et ami, avec qui j'ai partagé encore une fois de nombreuses heures à programmer en Sage, à discuter de toutes sortes de sujets et à rédiger des articles. Il me sera très difficile de trouver un autre compère avec qui je travaillerai avec autant de passion et d'efficacité. Merci aussi à Ariane Garon, avec qui j'ai réussi à énumérer correctement les 2-carrés. Je me rappellerai toujours nos séances de travail Skype franco-allemande qui nous ont permis de surmonter toutes les épreuves. Merci finalement à Geneviève Paquin d'avoir représenté avec moi les irréductibles Québécois en Savoie et d'avoir rendu mon séjour là-bas si agréable. J'ai eu l'occasion d'y découvrir à la fois une bonne amie et une collègue de travail qui m'a beaucoup appris.

Merci aux professeurs et à mes amis du LaCIM, qui ont rendu cet endroit si chaleureux et stimulant : Lise Tourigny, qui nous a maintenant quittés pour un autre département à l'université, Christophe Reutenauer, pour les remarques judicieuses,

Franco Saliola, pour son enthousiasme contagieux, Jérôme Tremblay, pour sa disponibilité sans limite et Jean-Philippe Labbé, pour nos nombreuses discussions. Merci à Manon Gauthier d'avoir su si bien me guider dans les méandres administratifs de la cotutelle.

Merci à mes amis belges et français rencontrés ces dernières années, Thierry, Thomas, Julien, Pierre, Pierre-Étienne, Matthieu, Florian, Émilie, Florent, ainsi qu'à mes amis québécois Xavier et Annie qui ont envahi la France de concert avec Geneviève, Sébastien, Ariane et moi. Nos retrouvailles tous ensemble à Marseille en février 2010 demeureront inoubliables. Merci également à mes amis italiens Andrea et Simone et à leur sens de l'humour inégalable.

Merci à mes amis lexicologues, Étienne, Olivier, Mélanie, Guillaume, Odile et Yassine. Bien que nos travaux ne fassent pas l'objet de cette thèse, il n'y a aucun doute que nos discussions et nos avancées ont influencé mon moral au cours des cinq dernières années.

Merci aux différents membres de ma famille, pour votre soutien et vos encouragements : maman, papa, Maxime, Xavier, Korallie, Nevya, Gabrielle, Mikaël, Linda, Carole et André.

Merci à ma compagne Annie, pour son support infaillible, qui a toujours cru en moi et qui continue de le faire aujourd'hui. Merci à mes deux petits bonshommes, Ludovic et Jolan, d'égayer ma vie quotidiennement par leur curiosité, leur sens de l'humour et leur énergie. Enfin, merci à ma petite dernière, Cassandre, d'être venue compléter cette famille de plus en plus nombreuse avec sa tranquille sagesse et son sourire.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Dans cette thèse, différents problèmes de la combinatoire des mots et de géométrie discrète sont considérés. Nous étudions d'abord l'occurrence des palindromes dans les codages de rotations, une famille de mots incluant entre autres les mots sturmiens et les suites de Rote. En particulier, nous démontrons que ces mots sont pleins, c'est-à-dire qu'ils réalisent la complexité palindromique maximale. Ensuite, nous étudions une nouvelle famille de mots, appelés *mots pseudostandards généralisés*, qui sont générés à l'aide d'un opérateur appelé *clôture pseudopalindromique itérée*. Nous présentons entre autres une généralisation d'une formule décrite par Justin qui permet de générer de façon linéaire et optimale un mot pseudostandard généralisé.

L'objet central, le *f-palindrome* ou *pseudopalindrome* est un indicateur des symétries présentes dans les objets géométriques. Dans les derniers chapitres, nous nous concentrons davantage sur des problèmes de nature géométrique. Plus précisément, nous donnons la solution à deux conjectures de Provençal concernant les pavages par translation, en exploitant la présence de palindromes et de périodicité locale dans les mots de contour. À la fin de plusieurs chapitres, différents problèmes ouverts et conjectures sont brièvement présentés.

Mots-clés : Palindrome, pseudopalindrome, clôture pseudopalindromique itérée, codages de rotations, symétries, chemins discrets, pavages.

INTRODUCTION

La combinatoire des mots est une discipline récente, étudiée de façon autonome depuis quelques dizaines d'années. On la retrouve implicitement depuis plus longtemps dans les travaux de Bernoulli, Markov, Thue, Birkhoff et Morse, mais ce n'est que depuis la deuxième moitié du 20e siècle qu'une littérature lui est spécifiquement consacrée. La structure des mots finis et infinis se révèle essentielle voire centrale dans de nombreux domaines de la science, par exemple en théorie des nombres (Allouche et Shallit, 2000; Adamczewski, 2002), en bio-informatique (Marathe, Condon et Corn, 1999; Czeizler et al., 2009; Kari et Mahalingam, 2010) et en physique (Hof, Knill et Simon, 1995; Allouche, 1997), ainsi qu'en géométrie digitale, discipline dans laquelle s'inscrit cette thèse.

Dans le mémoire de maîtrise du présent auteur, le sujet central d'étude était le f-palindrome ou pseudopalindrome (Blondin Massé, 2008). Dans le même ordre d'idée, le pseudopalindrome occupe une place d'importance dans les trois premiers chapitres de cette thèse, puisqu'il révèle les symétries des objets combinatoires qu'il représente. Parmi les sujets qui ont été abordés au cours des trois dernières années, ce sont les travaux qui s'articulent autour de ces objets combinatoires riches en applications qui sont présentés. Dans les chapitres 2 et 3, nous nous concentrons plus particulièrement sur la complexité palindromique des codages de rotations et sur l'étude des mots pseudostandards généralisés, qui sont construits à partir de pseudopalindromes et d'opérateurs sur ces mots.

La deuxième partie de ce document est consacrée à l'étude de problèmes géométriques en utilisant des outils de la combinatoire des mots. Cette approche récente a mis en évidence qu'il était possible d'obtenir des algorithmes efficaces et souvent optimaux pour extraire diverses informations structurelles sur des chemins discrets et des mots

de contour (Beauquier et Nivat, 1991; Brlek, Labelle et Lacasse, 2005; Brlek, Labelle et Lacasse, 2006b; Brlek et al., 2009; Brlek, Koskas et Provençal, 2011b). Par exemple, la présence de périodes locales et de palindromes dans les mots de contour de certaines tuiles révèle des propriétés symétriques essentielles à la résolution de problèmes. Grâce à ces observations, nous sommes en mesure de répondre affirmativement à deux conjectures formulées par Brlek, Dulucq, Fédou et Provençal qu'on retrouve dans la thèse de ce dernier (Provençal, 2008).

La plupart des nouveaux résultats de cette thèse ont déjà été publiés dans des articles de conférence et de revue, sont sous presse ou en révision. Les thèmes abordés se divisent en six chapitres comme suit.

Dans le chapitre 1, on introduit les définitions et les notations habituelles de la combinatoire des mots, ainsi que certaines notions plus particulièrement liées à l'étude des palindromes et des pseudopalindromes. La plupart des concepts présentés proviennent du livre de Lothaire, une des principales références dans le domaine (Lothaire, 1983; Lothaire, 1997).

Au chapitre 2, on considère les codages de rotations, une généralisation naturelle des mots sturmiens. Ces suites symboliques sont obtenues en partitionnant l'intervalle [0,1) en deux sous-intervalles et en codant la trajectoire d'un point subissant des rotations successives d'angle α. Nous présentons des résultats plus et moins connus sur la structure de ces mots, en nous concentrant en particulier sur leurs mots de retour. Le théorème principal de ce chapitre affirme que les codages de rotations sont pleins, c'est-à-dire qu'ils contiennent le plus grand nombre possible de palindromes distincts dans chacun de leurs facteurs. Les résultats qui y sont présentés ont fait l'objet d'un article de conférence présenté à EuroComb en septembre 2009, à Bordeaux, en France (Blondin Massé et al., 2009), et d'une version étendue qui est sous presse à la revue Theoretical Computer Sciences (Blondin Massé et al., 2011).

Nous enchaînons au chapitre 3 avec l'étude de mots introduits plus récemment dans la littérature, appelés *mots pseudostandards généralisés* (de Luca et De Luca, 2006).

Cette famille de mots contient entre autres les mots sturmiens standards, les suites de Rote standards, le mot de Thue-Morse et certains mots quasi-sturmiens. Nous décrivons un algorithme optimal permettant de générer des mots pseudostandards généralisés, en nous inspirant d'une formule introduite par Justin dans un de ses articles (Justin, 2005) que nous généralisons en conséquence. Ces nouveaux résultats ont également fait l'objet d'un article et d'une présentation aux Journées montoises d'informatique théorique à Amiens, en France, en septembre 2010 (Blondin Massé, Paquin et Vuillon, 2010).

Le chapitre 4 est consacré à des généralités sur les chemins discrets, les mots de contour (qui sont des chemins discrets fermés et auto-évitants), les polyominos et les pavages. Une section en particulier porte sur la notion de polyomino premier et composé.

Ensuite, au chapitre 5, nous étudions les tuiles n-hexagonales et n-carrées, c'est-à-dire des polyominos admettant plusieurs pavages hexagonaux ou carrés. Nous démontrons entre autres qu'il n'existe aucune tuile n-carrée, pour $n \geq 3$, résolvant ainsi une conjecture de Provençal dans sa thèse (Provençal, 2008). Nous concluons ce chapitre en proposant divers problèmes ouverts soulevés par cette nouvelle approche. Les thèmes abordés dans cette partie ont fait l'objet d'un article de conférence et d'une communication à Lattice Paths 2011, à Sienne, en Italie (Blondin Massé et al., 2011) ainsi que d'une version étendue qui a été acceptée et devrait bientôt paraître dans la revue Discrete Applied Mathematics (Blondin Massé, Brlek et Labbé, 2012).

Le dernier chapitre de cette thèse, le chapitre 6, traite du problème de génération de tuiles 2-carrées. De tous les chapitres de cette thèse, c'est sans doute celui qui illustre de la façon la plus éloquente l'intérêt d'utiliser une approche basée sur la combinatoire des mots pour résoudre des problèmes de géométrie discrète. En effet, la structure induite par les mots de contour de tuiles 2-carrée révèle la présence de périodes locales, ce qui nous permet de décrire des opérateurs inversibles sur l'espace des tuiles 2-carrées. Ces opérateurs préservent également plusieurs propriétés fondamentales sur les mots de contour, dont le nombre d'enroulements et la structure palindromique. Nous démontrons une deuxième conjecture de Provençal et Vuillon à l'effet que les 2-carrés premiers sont

fixés par une rotation d'angle π (Provençal, 2008). Les résultats sur l'énumération des tuiles 2-carrées ont été présentés dans un article à la conférence GASCom en 2010, à Montréal (Blondin Massé, Garon et Labbé, 2011a) et font l'objet d'un article de journal qui a été soumis durant l'été 2011 (Blondin Massé, Garon et Labbé, 2011b).

CHAPITRE I

GÉNÉRALITÉS SUR LES MOTS

Dans ce chapitre, nous introduisons les définitions et les notations usuelles de la combinatoire des mots. Nous adoptons principalement la notation de Lothaire, une des références principales dans le domaine (Lothaire, 1983).

1.1 Mots

Un *alphabet* A est un ensemble fini dont les éléments sont appelés *lettres* ou *symboles*. Un *mot fini* w sur un alphabet A est une suite finie $(w_1,w_2,\ldots,w_n)$ d'éléments de A, où $n \in \mathbb{N}$ (souvent, pour des raisons arithmétiques, on commence l'indexation par 0). Afin d'alléger la notation, on écrit $w = w_1 w_2 \cdots w_n$. L'entier n est appelé la *longueur* de w, notée $|w|$. Il existe un unique mot w tel que $|w| = 0$. Ce mot est appelé *mot vide* et est noté ε.

On désigne par A^n l'ensemble des mots de longueur n sur A, où $n \in \mathbb{N}$. D'autre part, l'ensemble des mots de longueur quelconque sur A est noté A^* et est défini par

$$A^* = \bigcup_{n \geq 0} A^n.$$

Étant donné deux mots $u = u_1 u_2 \cdots u_m$ et $v = v_1 v_2 \cdots v_n$ sur A, où $m,n \in \mathbb{N}$, on appelle *concaténation* de u et v le mot $u \cdot v = u_1 u_2 \cdots u_m v_1 v_2 \cdots v_n$. Remarquons que la concaténation de deux mots est une opération associative sur A^*, de sorte que $(A^*, \cdot)$ est un monoïde, appelé *monoïde libre*, dont l'élément neutre est ε.

Soit $w = w_1 w_2 \cdots w_n$ un mot de longueur n sur un alphabet A. On dit d'un mot u qu'il est *facteur* de w s'il existe des mots x et y tels que $w = xuy$. En particulier, dans le cas où $x = \varepsilon$ (respectivement $y = \varepsilon$), on dit que u est un *préfixe* (respectivement *suffixe*) de w. L'*ensemble des facteurs* ou le *langage* de w est noté $\mathrm{Fact}(w)$, alors que l'ensemble des facteurs de longueur n de w est noté $\mathrm{Fact}_n(w)$. D'autre part, l'ensemble des préfixes (respectivement suffixes) de w est noté $\mathrm{Pref}(w)$ (respectivement $\mathrm{Suff}(w)$). L'unique préfixe (respectivement suffixe) de w de longueur i, où $0 \leq i \leq n$, est noté $\mathrm{Pref}_i(w)$ (respectivement $\mathrm{Suff}_i(w)$). On dit que le nombre i est une *occurrence* de u s'il existe des mots x et y tels que $w = xuy$, où $|x| = i + 1$. On désigne par $|w|_u$ le nombre d'occurrences de u dans w. En outre, u est dit *unioccurrent* dans w si $|w|_u = 1$.

La *n-ième puissance* d'un mot non vide w, notée w^n, est donnée par $w^n = ww \cdots w$ (n fois). On dit que w est *primitif* s'il n'existe aucun mot u tel que $w = u^n$, pour un certain entier n. En particulier, le *carré* de w est donné par w^2.

On dit de deux mots u et v qu'ils sont *conjugués* s'il existe des mots x et y satisfaisant $u = xy$ et $v = yx$. Par exemple, $u = aabab$ et $v = abaab$ sont conjugués (il suffit de prendre $x = aab$ et $y = ab$). On peut montrer que la relation « être conjugué de » est une relation d'équivalence. La *classe de conjugaison* d'un mot w, notée $[w]$, est l'ensemble des conjugués de w. On se convainc facilement que sa cardinalité est donnée par $|[w]| = |w|$ si w est primitif.

Supposons que $v \in \mathrm{Pref}(w)$. Alors $v^{-1}w$ est le mot satisfaisant $v(v^{-1}w) = w$, c'est-à-dire que $v^{-1}w$ est le mot obtenu de w en supprimant le préfixe v. De la même façon, si $v \in \mathrm{Suff}(w)$, alors wv^{-1} est le mot satisfaisant $(wv^{-1})v = w$.

Soit $\varphi : A^* \to B^*$ une fonction, où A et B sont deux alphabets. On dit que φ est un *morphisme* s'il préserve la concaténation, c'est-à-dire que $\varphi(uv) = \varphi(u)\varphi(v)$, pour n'importe quels u,$v \in A^*$. D'autre part, On dit que φ est un *antimorphisme* si $\varphi(uv) = \varphi(v)\varphi(u)$, pour n'importe quels u,$v \in A^*$.

Supposons que $A = \{a,b\}$. Le *complément* d'un mot $w \in A^*$, dénoté par $\overline{w}$, est le mot

obtenu par l'application du morphisme échangeant les lettres de w, c'est-à-dire que $\bar{\cdot}$ est le morphisme défini par $\overline{a} = b$ et $\overline{b} = a$. Il est clair que l'opération de complémentation est une involution.

La *complexité (factorielle)* d'un mot est la fonction indiquant, pour chaque naturel n, le nombre de mots de longueur n, c'est-à-dire la fonction

$$C_w(n) : \mathbb{N} \to \mathbb{N} : n \mapsto |\{\mathrm{Fact}_n(\mathbf{w})|.$$

1.2 Pseudopalindromes

L'*image miroir* de $w = w_1 w_2 \cdots w_n$, notée $\widetilde{w}$, est définie par $\widetilde{w} = w_n \cdots w_2 w_1$. Un *palindrome* est un mot p satisfaisant $p = \widetilde{p}$. L'ensemble des facteurs palindromiques d'un mot w est noté $\mathrm{Pal}(w)$ et l'ensemble des facteurs palindromiques de longueur n de w est noté $\mathrm{Pal}_n(w)$. De plus, afin d'alléger grandement l'écriture par la suite, on dénote par $\mathrm{PLPS}(w)$ le plus long palindrome suffixe de w.

On dénote par $\widehat{\cdot}$ l'antimorphisme sur les alphabets binaires correspondant à la composition des opérateurs $\widetilde{\ }$ et $\bar{\cdot}$, c'est-à-dire que pour tout mot binaire w sur $A = \{a,b\}$, on a

$$\widehat{w} = \widetilde{\overline{w}} = \overline{\widetilde{w}}.$$

Le fait que les opérateurs $\widetilde{\cdot}$ et $\bar{\cdot}$ commutent est clair. Un *antipalindrome* est un mot w satisfaisant $w = \widehat{w}$. On dénote par $\mathrm{Antipal}(w)$ l'ensemble des facteurs antipalindromiques de w. On peut montrer que si w est un antipalindrome et $p \in \mathrm{Pal}(w)$, alors $\overline{p} \in \mathrm{Pal}(w)$. De plus, le lecteur vérifiera que l'unique mot qui est à la fois un palindrome et un antipalindrome est le mot vide ε. De la même façon que pour les palindromes, on désigne par $\mathrm{PLAS}(w)$ le plus long antipalindrome suffixe de w.

Notons que $\widetilde{\ }$ et $\widehat{\ }$ sont des antimorphismes, c'est-à-dire que pour tous mots u et v, On a $\widetilde{uv} = \widetilde{v}\widetilde{u}$ et $\widehat{uv} = \widehat{v}\widehat{u}$.

Exemple 1. Considérons le mot $w = 00101100$ sur l'alphabet $A = \{0,1\}$. Alors

$$\begin{aligned}
\mathrm{Fact}_0(w) &= \{\varepsilon\} \\
\mathrm{Fact}_1(w) &= \{0,1\} \\
\mathrm{Fact}_2(w) &= \{00, 01, 10, 11\} \\
\mathrm{Fact}_3(w) &= \{001, 010, 011, 100, 101, 110\} \\
\mathrm{Fact}_4(w) &= \{0010, 0101, 0110, 1011, 1100\} \\
\mathrm{Fact}_5(w) &= \{00101, 01011, 10110, 01100\} \\
\mathrm{Fact}_6(w) &= \{001011, 010110, 101100\} \\
\mathrm{Fact}_7(w) &= \{0010110, 0101100\} \\
\mathrm{Fact}_8(w) &= \{00101100\} \\
\mathrm{Pal}(w) &= \{\varepsilon, 0, 1, 00, 11, 010, 101, 0110\} \\
\mathrm{Antipal}(w) &= \{\varepsilon, 01, 10, 0101, 1100, 001011\}
\end{aligned}$$

Soient u,v deux mots. Il est facile de vérifier que $\ll u$ est facteur de $v \gg$ est une relation réflexive, antisymétrique et transitive et donc une relation d'ordre (par contre, ce n'est pas une relation d'ordre total). On introduit une restriction de cette relation sur les palindromes comme suit.

Définition 1. Soient p et q deux palindromes. On écrit $p \preceq q$ s'il existe un mot x tel que $q = xp\widetilde{x}$ et on dit que p est un *facteur palindromique central* de q.

Le lecteur vérifie facilement la proposition suivante.

Proposition 1. La relation $p \preceq q$, où p et q sont des palindromes, est un ordre partiel (mais ce n'est pas un ordre total). □

Étant donné un mot w, il est possible de représenter sous forme d'arbre les palindromes apparaissant dans w à l'aide de la relation $\preceq$. Plus précisément, on a la définition suivante.

Définition 2. Soit w un mot. Alors l'*arbre des palindromes de* w est l'arborescence dont les sommets sont donnés par les éléments de $\mathrm{Pal}(w)$, ainsi qu'un sommet distingué supplémentaire qu'on appelle *racine*. De plus, il existe un arc de la racine vers ε et chacune des lettres de w. Finalement, on a un arc du sommet p vers le sommet q si $q = \alpha p \alpha$, pour une certaine lettre α.

Remarquons que chaque ordre partiel induit une arborescence sur un ensemble de mots. Cependant, dans ce mémoire, nous nous intéressons plus particulièrement aux facteurs palindromiques et antipalindromiques.

Par ailleurs, dans les représentations graphiques d'arbre des palindromes, nous ne dessinons pas l'orientation des arcs puisqu'il est facile de la déduire en consultant l'étiquette des sommets source et but.

Notons que la relation $\preceq$ peut être également étendue aux antipalindromes, c'est-à-dire que si p et q sont deux antipalindromes, alors on écrit $p \preceq q$ lorsqu'il existe un x tel que $q = xp\widehat{x}$. L'arbre des antipalindromes d'un mot w est défini de la même façon qu'à la Définition 2.

Définition 3. Soit w un mot binaire. Alors l'*arbre des antipalindromes de* w est l'arborescence dont les sommets sont donnés par les éléments de $\mathrm{Antipal}(w)$ et tel que la racine est ε. De plus, on a un arc du sommet p vers le sommet q si $q = \alpha p \overline{\alpha}$, où α et $\overline{\alpha}$ sont les deux lettres de l'alphabet.

Nous illustrons ces notions par un exemple. Il est pratique de représenter par des arbres les facteurs palindromiques et antipalindromiques d'un mot puisqu'ils donnent rapidement une intuition de leur structure combinatoire. En particulier, ils mettent en évidence les symétries et les extensions possibles pour chaque palindrome.

Exemple 2. En reprenant le mot $w = 00101100$ on obtient les arbres des palindromes et des antipalindromes de w aux figures 1.1 et 1.2.

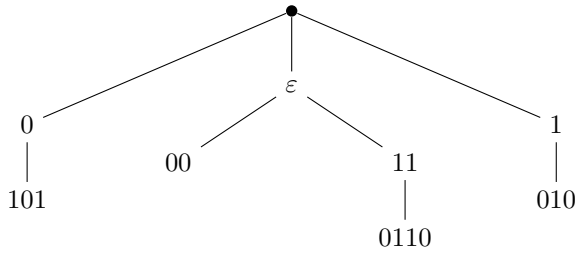


Figure 1.1: Arbre des palindromes du mot $w = 00101100$.

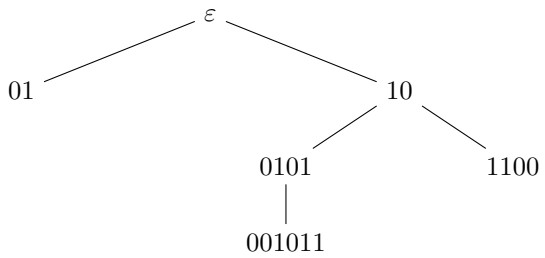


Figure 1.2: Arbre des antipalindromes du mot $w = 00101100$.

Dans un article de Brlek et al., les auteurs introduisent la notion de *défaut palindromique*, qui mesure la quantité de palindromes distincts manquants dans un même mot (Brlek et al., 2004) en se basant sur une observation d'un article de Droubay, Justin et Pirillo (Droubay, Justin et Pirillo, 2001). En effet, considérons un mot fini w. Alors w contient un nombre linéaire de palindromes distincts. Plus précisément, on a le fait suivant :

Théorème 1. (Droubay, Justin et Pirillo, 2001) Soit w un mot fini. Alors $|\text{Pal}(w)| \leq |w| + 1$.

Démonstration. Une démonstration se trouve dans (Droubay, Justin et Pirillo, 2001) ainsi que dans le mémoire du présent auteur (Blondin Massé, 2008). Par contre, afin de rendre cette thèse auto-suffisante, nous l'incluons ici.

Soit p_i le préfixe de longueur i de w, où $0 \leq i \leq n$. Soit P_i le nombre de palindromes suffixes de p_i et unioccurrents dans p_i, où $0 \leq i \leq n$. On montre que $P_i \leq 1$, pour $0 \leq i \leq n$. En procédant par contradiction, supposons qu'il existe un indice j tel que $P_j \geq 2$. En particulier, il existe deux palindromes distincts u et v qui sont suffixes de

p_j et qui sont unioccurrents dans p_j. De plus, comme u et v sont différents, sans perte de généralité, on peut supposer que $|u| > |v|$. Il existe donc un mot non vide x tel que $u = xv$. Or, $u = \widetilde{u} = \widetilde{xv} = \widetilde{v}\widetilde{x} = v\widetilde{x}$, c'est-à-dire que v apparaît au moins deux fois dans u et donc dans p_j, contredisant la supposition que v est unioccurrent dans p_j. On en conclut que $P_i \leq 1$ pour $0 \leq i \leq n$ et donc $|\mathrm{Pal}(w)| \leq |w| + 1$. □

Un mot w qui réalise la borne du théorème 1 est dit *plein*. En outre, on définit le *défaut palindromique* de w, noté $D(w)$, par

$$D(w) = |w| + 1 - |\mathrm{Pal}(w)|.$$

Il est clair que w est plein si et seulement si $D(w) = 0$.

1.3 Mots infinis

Un *mot infini* $\mathbf{w}$ sur un alphabet A est une suite dénombrable d'éléments de A. En général, les mots infinis seront dénotés en gras. La plupart des définitions de la section 1.1 s'étendent aux mots infinis. Un mot $\mathbf{w}$ est dit *périodique* s'il existe un mot non vide v tel que $v^n \in \mathrm{Pref}(\mathbf{w})$, pour tout $n \in \mathbb{N}$. On écrit alors $\mathbf{w} = v^\omega$. On dit que $\mathbf{w}$ est *récurrent* si pour tout $u \in \mathrm{Fact}(\mathbf{w})$, on a $|\mathbf{w}|_u = \infty$. Il existe une notion plus forte de récurrence : $\mathbf{w}$ est dit *uniformément récurrent* si pour tout $u \in \mathrm{Fact}(\mathbf{w})$, il existe un entier n tel que pour tout $v \in \mathrm{Fact}_n(\mathbf{w})$, $u \in \mathrm{Fact}(v)$. Autrement dit, un mot uniformément récurrent a la propriété que chaque paire d'occurrences consécutive apparaît avec une distance bornée. Soit $u \in \mathrm{Fact}(w)$. Un mot v est appelé *mot de retour complet de u dans w* si

(i) $v \in \mathrm{Fact}(w)$,

(ii) $|v|_u = 2$,

(iii) $u \in \mathrm{Pref}(v)$ et

(iv) $u \in \mathrm{Suff}(v)$.

On désigne par $\mathrm{CRet}_w(u)$ l'ensemble des mots de retour complet de u dans w. Notons que cette notion est définie pour les mots finis et infinis, mais elle est surtout considérée dans les mots infinis. En particulier, le nombre de mots de retour complet d'un mot u dans w est fini si w est uniformément récurrent.

Exemple 3. Soit $u = aababbaabbabaa$. Le mot

$$\mathbf{w} = u^\omega = (aababbaabbabaa)^\omega = aababbaabbabaaaaababbaabbabaa\cdots$$

est un exemple de mot périodique. On peut par ailleurs montrer que u^n est un palindrome, pour tout $n \geq 0$. Il est clair que $\mathbf{w}$ est récurrent et même uniformément récurrent. Finalement, on constate que

$$\mathrm{CRet}_{\mathbf{w}}(aa) = \{aaa, aababbaa, aabbabaa\}.$$

Une extension naturelle de la définition de mot de retour complet s'avère utile dans les chapitres qui suivent. Soient $u, v \in \mathrm{Fact}(w)$. On dit que w est un *mot de retour complet de u vers v* s'il existe un j tel que

(i) u est un préfixe de w ;

(ii) v est un suffixe de w ;

(iii) si $u \neq v$, alors $|w|_v = 1$.

On désigne par $\mathrm{CRet}_w(u)$ l'ensemble des mots de retour complet de u dans w. Notons que cette notion est définie pour les mots finis et infinis, mais elle est surtout considérée dans les mots infinis. En particulier, le nombre de mots de retour complet d'un mot u dans w est fini si w est uniformément récurrent.

1.4 Morphismes

Rappelons qu'un morphisme est une application $\varphi : A^* \to B^*$ telle que $\varphi(uv) = \varphi(u)\varphi(v)$, pour n'importe quels u,$v \in A^*$. Il est par conséquent suffisant de connaître

l'action de φ sur les lettres de A pour l'étendre au monoïde libre A^*. On dit d'un mot w qu'il est *point fixe* du morphisme φ si $w = \varphi(w)$.

Soit $\varphi : A^* \to A^*$ un morphisme sur un alphabet A. On peut montrer qu'un mot w dont la première lettre est α est fixé par φ si et seulement si la première lettre de $\varphi(\alpha)$ est α (Allouche et Shallit, 2003, chapitre 7). Dans ce cas, on écrit $\varphi^\omega(\alpha)$. La notation est justifiée par le fait que $\varphi^n(\alpha)$ est un préfixe de w pour tout entier $n \geq 0$. Notons que certains mots finis peuvent être point fixe d'un morphisme.

Exemple 4. Soient $A = \{a,b\}$ et $\mu : A^* \to A^*$ le morphisme donné par $\mu(a) = ab$ et $\mu(b) = ba$. Alors μ admet exactement deux points fixes :

$$\mu(\mathbf{t}) = \mathbf{t} = abbabaabbaababbabaababbaabbababaab\cdots$$
$$\mu(\overline{\mathbf{t}}) = \overline{\mathbf{t}} = baababbaabbabaababbabaabbaababba\cdots$$

On démontre facilement que $\mathrm{CRet}_{\mathbf{t}}(a) = \{aa,aba,abba\}$. Le mathématicien Axel Thue a également démontré que $\mathbf{t}$ est sans chevauchement, c'est-à-dire ne contient pas de facteur de la forme $auaua$, où u est un mot et a une lettre. Une traduction de ses travaux se trouve dans (Berstel, 1995). À noter que $\mathbf{t}$ et $\overline{\mathbf{t}}$ sont également des points fixes du morphisme $\theta : A^* \to A^*$ défini par $\theta(a) = abba$ et $\theta(b) = baab$, c'est-à-dire que les images sont des palindromes. Le mot $\mathbf{t}$ est appelé *mot de Thue-Morse*. Il possède de nombreuses propriétés palindromiques intéressantes et une grande quantité de travaux sont consacrés à son étude (Brlek, 1989; Berstel, 1995; Allouche et Shallit, 2000; Blondin Massé et al., 2007; Blondin Massé et al., 2008), dont une importante partie est résumée dans le "survey" de Allouche et Shallit (Allouche et Shallit, 1999).

1.5 Mots célèbres

Nous terminons ce chapitre en présentant quelques familles de mots infinis célèbres, qu'on retrouve dans les chapitres subséquents.

Les mots infinis les plus connus sont sans doute les mots sturmiens. Ils contiennent en

particulier le mot de Fibonacci

$$\mathbf{f} = abaababaabaababaababa\cdots$$

défini comme le point fixe du morphisme

$$\varphi : \{a,b\}^* \to \{a,b\} : a \mapsto ab, b \mapsto a.$$

Définition 4. Soit $\mathbf{w}$ un mot infini apériodique. On dit que w est *sturmien* s'il existe des nombres $\alpha, \rho \in \mathbb{R}$ tels que $0 \leq \alpha < 1$, α est irrationnel et $\mathbf{w}$ est égal à un des deux mots infinis suivants

$$s_{\alpha,\rho}[n] = \begin{cases} a & \text{si } \lfloor \alpha(n+1)+\rho \rfloor = \lfloor \alpha n + \rho \rfloor, \\ b & \text{sinon,} \end{cases}$$

$$s'_{\alpha,\rho}[n] = \begin{cases} a & \text{si } \lceil \alpha(n+1)+\rho \rceil = \lceil \alpha n + \rho \rceil, \\ b & \text{sinon,} \end{cases}$$

c'est-à-dire que $\mathbf{w}$ correspond à la discrétisation d'une demi-droite d'intercept ρ et de pente irrationnelle α.

Plusieurs caractérisations équivalentes de ces mots existent. Plus précisément, les énoncés suivants sont équivalents :

1. $\mathbf{w}$ est *sturmien* ;
2. $\mathbf{w}$ a une complexité $n+1$;
3. $\mathbf{w}$ est équilibré, c'est-à-dire que pour tout facteur de même longueur $u, v \in \mathrm{Fact}_n(\mathbf{w})$, on a $|u|_a - |v|_a \leq 1$ pour toute lettre a.

On dit d'un mot sturmien qu'il est *standard* si $\rho = \alpha$. Nous étudions les mots standards dans le chapitre 3 ainsi que certaines généralisations de cette notion. De plus, il a été démontré que les mots sturmiens sont pleins en palindromes (Droubay, Justin et Pirillo, 2001).

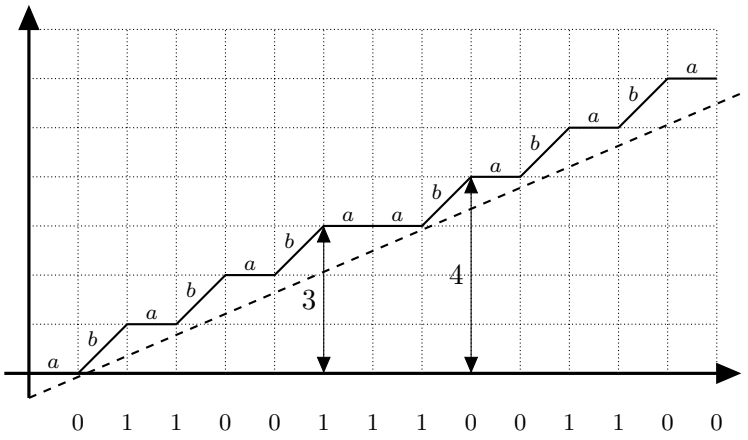


Figure 1.3: Interprétation géométrique illustrant le lien entre les mots sturmiens et les suites de Rote. Dans le dessin, le mot $01100111001100\cdots$ est une suite de Rote qui code la hauteur modulo 2 du chemin discret décrit par le mot sturmien $ababababaababa\cdots$.

Une famille de mots très proche de celle des mots sturmiens est celle des mots *quasi-sturmiens*, qui sont les mots de complexité $n+k$, pour un entier $k \geq 2$ (Alessandri, 1995). Une certaine littérature leur est également consacrée, notamment par la communauté étudiant les systèmes dynamiques (Damanik et Lenz, 2000; Lenz, 2003).

En dernier lieu, il est intéressant de mentionner les *suites de Rote*, qui sont les mots de complexité $2n$. Ils ont été principalement étudiés par Rote, qui a démontré que, si un mot $\mathbf{w}$ a une complexité $2n$ et a un langage fermé sous l'opération de complémentation, alors la suite des différences finies de $\mathbf{w}$ est sturmienne (Rote, 1994). Cette propriété s'interprète au niveau géométrique comme suit : les suites de Rote codent la hauteur modulo 2 d'une demi-droite de pente irrationnelle, comme l'illustre la figure 1.3.

À l'instar des mots sturmiens, on dit d'une suite de Rote dont le langage est fermé par complémentation qu'elle est *standard* si le mot sturmien qui lui est associé est lui-même standard.

Les codages de rotations, qui sont introduits dans le chapitre suivant, sont en quelque sorte une généralisation des mots sturmiens et de certaines suites de Rote. Ils incluent également certains mots quasi-sturmiens. Nous montrons en particulier que tous les codages de rotations sont pleins en palindromes.

CHAPITRE II

CODAGES DE ROTATIONS

Dans ce chapitre, nous nous intéressons aux codages de rotations, qui sont des suites symboliques binaires codant la trajectoire d'un point de l'intervalle unité [0,1) lorsqu'on lui fait subir des rotations successives d'angle $\alpha \in \mathbb{R}/\mathbb{Z}$. Plus précisément, nous rappelons les notions de base et les faits connus sur ces suites, notamment issus de la théorie des systèmes dynamiques. Nous démontrons également que ces suites symboliques sont pleines, c'est-à-dire qu'elles réalisent la borne supérieure du nombre de palindromes distincts qu'un mot infini peut contenir dans chacun de ses facteurs. La majorité des notions et des résultats présentés ont fait l'objet d'un article de revue publié récemment (Blondin Massé et al., 2011).

2.1 Topologie du cercle unité

L'espace de base des codages de rotations est le cercle unité $\mathbb{R}/\mathbb{Z}$ muni de sa projection naturelle

$$\pi : \mathbb{R} \to \mathbb{R}/\mathbb{Z} : x \mapsto x + \mathbb{Z}.$$

L'ensemble des représentants utilisés est donné par l'intervalle fermé à gauche et ouvert à droite $[0, 1) \subseteq \mathbb{R}$. Dans l'espace $\mathbb{R}/\mathbb{Z}$, on appelle *intervalle* tout ensemble $A \subseteq \mathbb{R}/\mathbb{Z}$ pour lequel il existe un intervalle $B \subseteq \mathbb{R}$ vérifiant $\pi(B) = A$. Ainsi, $[1/4, 11/20]$ et [0.75,0.08] sont deux intervalles de $\mathbb{R}/\mathbb{Z}$, c'est-à-dire qu'il n'est pas nécessaire que les bornes a et b d'un intervalle $[a, b]$ satisfassent $a < b$. En revanche, tout intervalle de $\mathbb{R}/\mathbb{Z}$

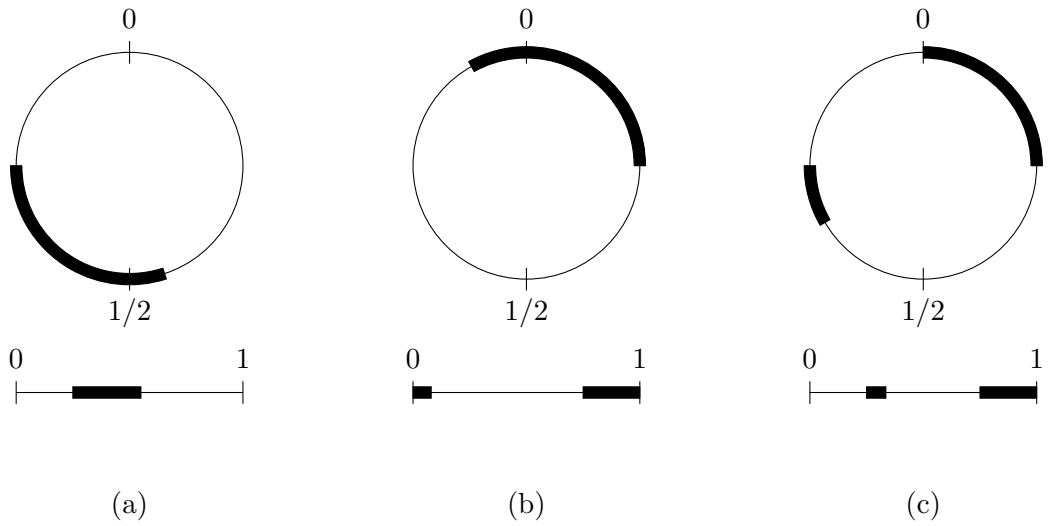


Figure 2.1: Intervalles du cercle unité. (a) L'intervalle $[0.25; 0.55]$, (b) l'intervalle $[0.75; 0.08]$ et (c) ce n'est pas un intervalle.

est complètement déterminé par le couple (ordonné) (a, b), avec $a, b \in \mathbb{R}/\mathbb{Z}$ (voir figure 2.1).

Les notions d'*intervalle fermé*, *ouvert*, *semi-ouvert*, *ouvert à gauche*, *ouvert à droite*, *fermé à gauche* et *fermé à droite* dans $\mathbb{R}$ s'étendent naturellement au cas d'intervalles de $\mathbb{R}/\mathbb{Z}$. Le *bord* d'un intervalle non vide I, dénoté par $\partial(I)$, est l'ensemble de ses deux bornes, qu'elles soient incluses ou non dans l'intervalle. La *clôture* d'un intervalle I, qu'on note par $\overline{I}$, est donnée par $\overline{I} = I \cup \partial(I)$ et son *intérieur* est l'ensemble ouvert $\mathrm{Int}(I) = \overline{I} - \partial(I)$.

La fonction de base que nous considérons sur $\mathbb{R}/\mathbb{Z}$ est la *rotation d'angle* $\alpha \in \mathbb{R}$ définie par $R_\alpha(x) = x + \alpha \in \mathbb{R}/\mathbb{Z}$, l'addition étant considérée modulo $\mathbb{Z}$. Il est évident que R_α est une bijection. La fonction R_α s'étend naturellement à tout sous-ensemble de points $X \subseteq \mathbb{R}/\mathbb{Z} : R_\alpha(X) = \{R_\alpha(x) \mid x \in X\}$ et en particulier aux intervalles. Il est pratique d'employer la notation exponentielle pour désigner la composition de la fonction R_α avec elle-même, de sorte que $R_\alpha^m(x) = x + m\alpha \in \mathbb{R}/\mathbb{Z}$, avec $m \in \mathbb{Z}$.

2.2 Échanges d'intervalles

Un échange d'intervalles est une fonction affine par morceaux qui associe deux partitions de l'espace en intervalles par rapport à une permutation donnée.

Soient J,$K \subseteq \mathbb{R}/\mathbb{Z}$ deux intervalles fermés à gauche ouverts à droite de longueur λ. Soit $q \geq 1$ un entier et $\boldsymbol{\lambda} = (\lambda_1,\lambda_2,\ldots,\lambda_q)$ un vecteur à valeurs dans $\mathbb{R}^+$ tel que $\sum_{i=1}^{q} \lambda_i = \lambda$ et soit σ une permutation de l'ensemble $\{1,2,\ldots,q\}$. On partitionne les intervalles J et K en q sous-intervalles comme suit. Pour $1 \leq i \leq q$, posons

$$J_i = \left[\sum_{j<i} \lambda_j, \sum_{j\leq i} \lambda j\right) \quad \text{et} \quad K_i = \left[\sum_{k<\sigma^{-1}(i)} \lambda_{\sigma(k)}, \sum_{k\leq\sigma^{-1}(i)} \lambda_{\sigma(k)}\right).$$

Alors l'*échange de q intervalles par rapport à la permutation σ et au vecteur $\boldsymbol{\lambda}$* est l'application F qui satisfait les deux conditions suivantes :

(i) l'image du sous-intervalle J_i est K_i, c'est-à-dire $F(J_i) = K_i$ et

(ii) $F|_{J_i}$ (la restriction de F à l'intervalle J_i) est une translation pour $i = 1, 2, \ldots, q$.

Exemple 5. Soient les intervalles $J = [0{,}25; 0{,}50)$ et $K = [0{,}70; 0{,}95)$, le vecteur $\boldsymbol{\lambda} = (0{,}10; 0{,}10; 0{,}05)$ et la permutation $\sigma = (321)$. Alors

$$J_1 = [0{,}25; 0{,}35), \quad J_2 = [0{,}35; 0{,}45) \quad \text{et} \quad J_3 = [0{,}45; 0.50).$$

D'autre part, puisque $\sigma = (321)$, les longueurs des sous-intervalles K_1, K_2 et K_3 seront inversées par rapport à J_1, J_2 et J_3, de sorte que

$$K_3 = [0{,}70; 0{,}75), \quad K_2 = [0{,}75; 0{,}85) \quad \text{et} \quad K_1 = [0{,}85; 0.95).$$

L'échange d'intervalles F résultant du vecteur $\boldsymbol{\lambda}$ et de la permutation σ est donc complètement décrit par les sous-intervalles J_i et K_i, pour $i = 1{,}2{,}3$. Par exemple, $F(0{,}25) = 0{,}85$, $F(0{,}40) = 0{,}80$ et $F(0{,}49) = 0{,}74$. La figure 2.2 représente l'échange d'intervalles en question.

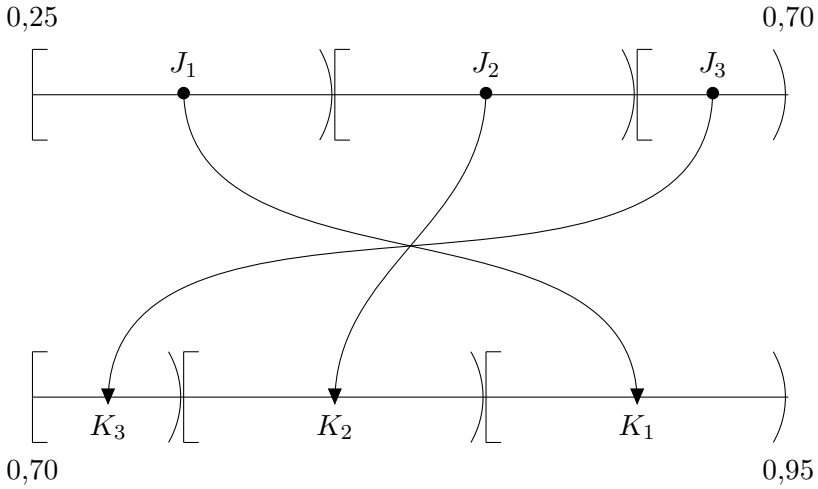


Figure 2.2: Échange d'intervalles obtenu à partir de l'intervalle $J = [0{,}25; 0{,}50)$ vers l'intervalle $K = [0{,}70; 0{,}95)$ selon le vecteur $\boldsymbol{\lambda} = (0{,}10; 0{,}10; 0{,}05)$ et la permutation $\sigma = (321)$.

2.3 La fonction de premier retour de Poincaré

Soit $\alpha \in \mathbb{R}$ et $J \subseteq \mathbb{R}/\mathbb{Z}$ un intervalle du cercle unité. On définit une fonction

$$T_\alpha : J \to \mathbb{N}^+ : x \mapsto \min\{t \in \mathbb{N}^+ \mid x + t\alpha \in J\}.$$

Moins formellement, $T_\alpha(x)$ est égal au nombre de rotations d'angle α nécessaires pour que le point $x \in J$ revienne dans l'intervalle J. On l'appelle également *temps de retour* du point $x \in J$, d'où l'utilisation de la variable t. Une autre fonction utile est celle qui code le lieu d'arrivée exact du point lors de son premier retour :

$$P_\alpha : J \to J : x \mapsto x + T_\alpha(x) \cdot \alpha.$$

Exemple 6. Soient $\alpha = 0{,}2345$, $J = [0{,}01; 0{,}16]$ et $x = 0{,}0833$. Alors $T_\alpha(x) = 4$ et $P_\alpha(x) = 0{,}2713$. Une représentation graphique de la situation se trouve à la figure 2.3.

Il n'est pas immédiat que la fonction T_α est bien définie pour tout $x \in J$. En effet, rien ne nous garantit à première vue que tout point x auquel on applique des rotations successives d'angle α ait une orbite qui intersecte l'intervalle J à nouveau. La proposition

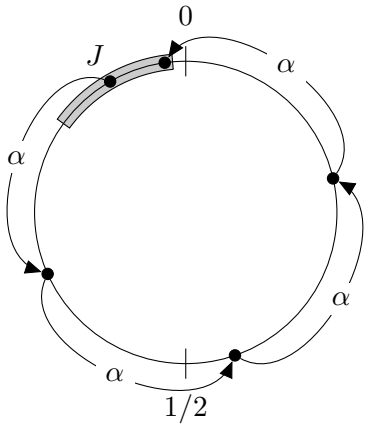


Figure 2.3: Temps de premier retour et fonction de premier retour de Poincaré avec $\alpha = 0{,}2345$ et $J = [0{,}01; 0{,}16]$. Il suffit de quatre rotations pour revenir à l'intervalle J en partant du point $x = 0{,}0833$. Par conséquent, $T_\alpha(x) = 4$ et donc $P_\alpha(x) = x + 4\alpha = 0{,}2713$.

qui suit résoud la question :

Proposition 2. Soit $\alpha \in \mathbb{R}$ un réel non nul et $J \subseteq \mathbb{R}/\mathbb{Z}$ un intervalle du cercle unité. Alors les fonctions T_α et P_α sont bien définies.

Démonstration. Il suffit de montrer que $T_\alpha(x) \in \mathbb{N}$ pour tout $x \in J$.

Supposons d'abord que α est rationnel, c'est-à-dire $\alpha = p/q$ pour certains entiers p et q. Alors $x + q\alpha = x + p = x \in J$, puisque l'addition est prise modulo $\mathbb{Z}$, de sorte que $T_\alpha(x) \leq p$ et donc $T_\alpha(x) \in \mathbb{N}$.

Il reste à considérer le cas α irrationnel. La conclusion suit du fait que l'ensemble $\{x + t\alpha \mid t \in \mathbb{R}\}$ est dense dans [0,1) et donc que $T_\alpha(x)$ est bien fini. □

Dans la suite de cette thèse, il s'avère utile de généraliser la fonction de premier retour de Poincaré en distinguant les intervalles de départ et d'arrivée.

Soient $\alpha \in \mathbb{R}$ et $J, K \subseteq \mathbb{R}/\mathbb{Z}$ deux intervalles non vides fermés à gauche et ouverts à droite. On définit l'application $T_\alpha(J, K)$ par

$$T_\alpha(J, K) : J \to \mathbb{N}^+ \cup \{+\infty\} : x \mapsto \inf\{t \in \mathbb{N}^+ \mid x + t\alpha \in K\}.$$

En d'autres termes, $T_\alpha(J,K)(x)$ indique le nombre de rotations d'angle α qui doivent être appliquées au point $x \in J$ pour qu'il se trouve pour la première fois dans l'intervalle K. Il s'agit bien d'une généralisation de la fonction de premier retour de Poincaré usuelle, qu'on retrouve en posant $J = K$. La fonction $T_\alpha(J,K)$ est appelée *temps de retour*. De façon analogue, posons

$$P_\alpha(J,K) : J' \to K : x \mapsto x + T_\alpha(J,K)(x) \cdot \alpha,$$

avec $J' = \{x \in J \mid T_\alpha(J,K)(x) < \infty\}$ l'ensemble des points de J ayant un temps de retour fini. Plus informellement, étant donné $x \in J$, $P_\alpha(J,K)(x)$ est le point correspondant au lieu d'arrivée de x la première fois qu'il atteint l'intervalle K. La fonction $P_\alpha(J,K)$ est appelée *fonction de premier retour de Poincaré généralisée*, ou plus simplement *fonction de premier retour de Poincaré* pour ne pas alourdir inutilement le texte. Il convient de noter que, bien que les fonctions $T_\alpha(J,K)$ et $P_\alpha(J,K)$ sont en général définies pour J et K des intervalles, elles sont également bien définies lorsqu'on les étend à des ensembles J et K qui ne sont pas des intervalles.

Exemple 7. Reprenons l'exemple 6, avec $\alpha = 0{,}2345$, $x = 0{,}0833$ et $J = [0{,}01; 0{,}16]$. Considérons l'intervalle d'arrivée $K = [0{,}37; 0{,}52]$. Cette fois, il faut appliquer six rotations au point x pour qu'il atteigne l'intervalle K pour la première fois (voir figure 2.4). On obtient donc $T_\alpha(J,K)(x) = 6$ et $P_\alpha(J,K)(x) = 0{,}4903$.

À l'exemple 7, on remarque que l'orbite du point x passe d'abord par l'intervalle J avant d'arriver dans l'intervalle K. En revanche, si $K = [0{,}25; 0{,}40]$, l'orbite serait arrivé directement dans l'intervalle K sans passer par J.

De la même façon que pour la fonction de premier retour de Poincaré usuelle, il faut s'assurer que la version généralisée est également bien définie. On constate que dans certains cas, il est possible que l'intervalle d'arrivée K ne soit jamais atteint par l'orbite $\{x + t\alpha \mid t \in \mathbb{N}^+\}$. En fait, c'est seulement dans le cas où α est rationnel, car si α est irrationnel, alors par densité, l'intervalle K sera toujours atteint. C'est pourquoi le codomaine de la fonction $T_\alpha(J,K)$ est $\mathbb{N}^+ \cup \{+\infty\}$.

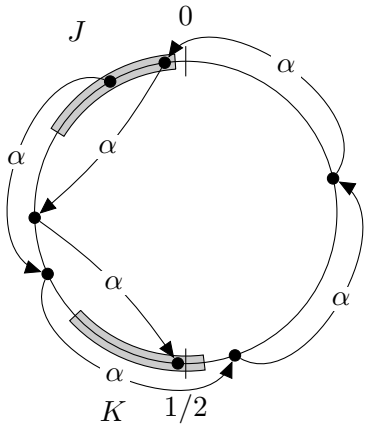


Figure 2.4: Temps de premier retour et fonction de premier retour de Poincaré généralisée avec $\alpha = 0{,}2345$, $J = [0{,}01; 0{,}16]$ et $K = [0{,}37; 0{,}52]$. Il suffit de six rotations pour arriver à l'intervalle K en partant du point $x = 0{,}0833$. Par conséquent, $T_\alpha(J, K)(x) = 6$ et $P_\alpha(x) = x + 6\alpha = 0{,}4903$.

Exemple 8. Soient $\alpha = 0{,}3$, $J = [0{,}83; 0{,}87]$, $K = [0{,}08; 0{,}12]$ et $x = 0{,}85$. Comme $\alpha = 3/10$, son orbite contient exactement 10 points :

$$X = \{0{,}05; 0{,}15; 0{,}25; 0{,}35; 0{,}45; 0{,}55; 0{,}65; 0{,}75; 0{,}85; 0.95\}.$$

Or, $X \cap K = \emptyset$, ce qui entraîne que $T_\alpha(J, K)(x) = +\infty$ et $P_\alpha(J, K)(x)$ n'est pas défini (voir figure 2.5).

Il est bien connu de la communauté étudiant les systèmes dynamiques que la fonction de premier retour de Poincaré usuelle est un échange d'intervalles. En utilisant la notation de ce chapitre, nous avons plus précisément le lemme suivant (Katok, 1980) :

Lemme 1. (Katok, 1980) Soit $\alpha \in \mathbb{R}$ et $J \subseteq \mathbb{R}/\mathbb{Z}$ un intervalle du cercle unité. Alors l'application $P_\alpha = P_\alpha(J, J)$ est un échange de $r \leq 4$ intervalles. De plus, il existe une décomposition

$$J = J_1 \cup J_2 \cup \ldots \cup J_s, \quad r \leq s \leq 4,$$

en sous-intervalles J_i deux à deux disjoints et des entiers positifs t_1, t_2, …, t_s tels que pour tout $x \in J_i$,

$$P_\alpha(x) = R_\alpha^{t_i}(x),$$

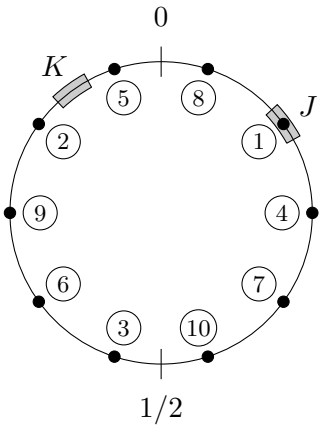


Figure 2.5: Temps de premier retour infini lorsque $\alpha = 0{,}3$, $J = [0{,}83; 0{,}87]$, $K = [0{,}08; 0{,}12]$ et $x = 0{,}85$. On remarque que l'orbite du point x contient exactement 10 points (numérotés dans l'ordre selon lequel ils sont atteints). En particulier, cet orbite et l'intervalle K sont disjoints de sorte que $T_\alpha(J,K)(x) = +\infty$ et $P_\alpha(J,K)(x)$ n'est pas défini.

où R_α est continue sur chaque sous-intervalle $R_\alpha^k(J_i)$, pour $k = 0, 1, \ldots, t_i - 1$. □

Nous n'incluons pas la démonstration ici pour deux raisons. Tout d'abord, celle-ci se trouve dans l'article de Katok (Katok, 1980). De plus, elle nécessite l'utilisation d'outils qui ne sont pas nécessaires à la compréhension de cette thèse et que nous ne souhaitons pas présenter. Finalement, la proposition qui suit généralise le lemme 1 pour le cas où les intervalles de départ J et d'arrivée K peuvent être différents, et par le fait même, en constitue une démonstration.

Proposition 3. Soient $\alpha \in \mathbb{R}/\mathbb{Z}$ et $J, K \subseteq \mathbb{R}/\mathbb{Z}$ deux intervalles fermés à gauche ouverts à droite vérifiant $|J| = |K| \leq \alpha$. Alors $P_\alpha(J,K)$ est une bijection si et seulement si $P_\alpha(J,K)$ est un échange de q intervalles de permutation $(q, q-1, \ldots, 2, 1)$, où $q \in \{1{,}2{,}3\}$.

Démonstration. Afin d'alléger le texte, posons $P = P_\alpha(J,K)$ et $T = T_\alpha(J,K)$.

($\Leftarrow$) Par définition même d'échange d'intervalles.

($\Rightarrow$) Considérons le nombre d'éléments dans l'image $T(J) \subseteq \mathbb{N}^+$. Si $|T(J)| = 1$, alors P est un échange de 1 intervalle et la proposition est vérifiée. Autrement, supposons que $|T(J)| \geq 2$.

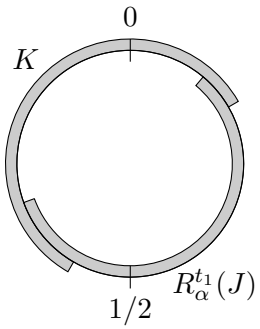


Figure 2.6: Schéma illustrant le cas où l'intersection de deux intervalles ne donne pas un intervalle. Il s'agit d'un argument utilisé pour visualiser ce qui se passe lorsque K et $R_\alpha^{t_1}(J)$ se chevauchent de part et d'autre dans la proposition 3.

Soient t_1 et t_2 les deux plus petites valeurs de $T(J)$, avec $t_1 < t_2$, $J_i = T^{-1}(t_i)$ et $K_i = P(J_i)$ pour $i = 1, 2$. Tout d'abord, remarquons que l'ensemble K_1 est un intervalle. En effet, si K_1 n'est pas un intervalle, alors les intervalles J et K doivent être de longueur au moins $1/2$ (puisqu'ils sont de même longueur et que $R_\alpha^{t_1}(J)$ et K doivent se chevaucher de part et d'autre comme à la figure 2.6) et donc $1/2 < |K| \leq \alpha$. En particulier $t_1 = 1$ et donc $\alpha < |J|$, ce qui est contradictoire. On en conclut que $K_1 = K \cap R_\alpha^{t_1}(J)$ est bien un intervalle et $K_2 = K \cap R_\alpha^{t^2}(J \setminus J_1)$. Or, P est une bijection par hypothèse, de sorte que K_1 et K_2 sont disjoints et touchent chacun une borne de l'intervalle K. Si $|T(J)| = 2$, alors $J = J_1 \cup J_2$ et $K = K_1 \cup K_2$, de sorte que P est l'échange de deux intervalles décrit par J_1 et J_2 de permutation (2,1).

Supposons maintenant que $|T(J)| \geq 3$. Soient $t_3 = \min(T(J) \setminus \{t_1,t_2\})$, $J_3 = T^{-1}(t_3)$ et $K_3 = P(J_3) = K \cap R_\alpha^{t_3}(J \setminus (J_1 \cup J_2))$. Autrement dit, t_3 est le prochain plus petit temps de retour, J_3 et K_3 sont respectivement les ensembles de départ et d'arrivée des points ayant le temps de retour t_3. Comme K_3 est non vide et P est une bijection, alors $R^{t_3}(J \setminus (J_1 \cup J_2))$ doit intersecter K dans l'intervalle $K \setminus (K_1 \cup K_2)$. Or, la longueur de cet intervalle fermé à gauche et ouvert à droite est $|K \setminus (K_1 \cup K_2)| = |J \setminus (J_1 \cup J_2)|$, ce qui signifie qu'il y a tout juste suffisamment d'espace. Ainsi, $J_3 = J \setminus (J_1 \cup J_2)$, $|T(J)| = 3$ et donc P est un échange de 3 intervalles décrit par J_1, J_2, J_3 de permutation (3,2,1), ce qui termine la démonstration. □

i	$R_\alpha^i(x) = (x + i\alpha) \bmod \mathbb{Z}$	$R_\alpha^i(x) \in [0; 0{,}30234023)$?	c_i
0	0,23435636	oui	1
1	0,45679159	non	0
2	0,67922683	non	0
3	0,90166206	non	0
4	0,12409730	oui	1
5	0,34653254	non	0

Tableau 2.1: Premières valeurs du codage de rotations $\mathbf{C} = c_0c_1c_2\cdots$ obtenu avec les paramètres $x = 0{,}23435636$, $\alpha = 0{,}222435236$ et $\beta = 0{,}30234023$.

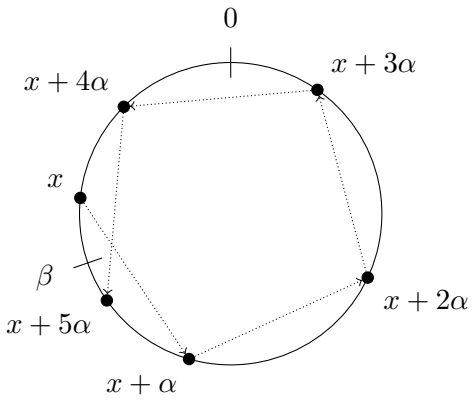


Figure 2.7: Orbite des premières valeurs obtenues à partir du codage de rotations de paramètres $x = 0{,}23435636$, $\alpha = 0{,}222435236$ et $\beta = 0{,}30234023$.

2.4 Codages de rotations

Soient $x, \alpha, \beta \in \mathbb{R}/\mathbb{Z}$. Le cercle unité $\mathbb{R}/\mathbb{Z}$ est partitionné en deux intervalles fermés à gauche et ouverts à droite non vides $I_1 = [0,\beta)$ et $I_0 = [\beta,1)$. Ensuite, on définit un mot infini (ou une suite symbolique) $\mathbf{C}(x) = c_0c_2c_2\ldots$ sur $A = \{0,1\}$ en posant

$$c_i = \begin{cases} 1 & \text{si } R_\alpha^i(x) \in [0,\beta), \\ 0 & \text{si } R_\alpha^i(x) \in [\beta,1), \end{cases}$$

Ce mot est appelé *codage de rotations* de x de paramètres (α, β).

Exemple 9. Soient $x = 0{,}23435636$, $\alpha = 0{,}222435236$ et $\beta = 0{,}30234023$. Alors

$$\mathbf{C}(x) = 10001000110001000110001000110001000110000\cdots.$$

Le tableau 2.1 indique les premières valeurs de $\mathbf{C}$ et la figure 2.7 représente l'orbite de ces premières valeurs sur le cercle unité.

Il est immédiat que $\mathbf{C}(x)$ est périodique si et seulement si α est rationnel. Lorsque α est irrationnel et que $\beta \in \{\alpha, 1-\alpha\}$, alors le codage obtenu est un mot sturmien. Autrement, si $\beta \notin \mathbb{Z} + \alpha\mathbb{Z}$, on retrouve les mots de complexité $2n$, ou mots de Rote (Rote, 1994). En dernier lieu, le cas $\beta \in \mathbb{Z} + \alpha\mathbb{Z}$ nous donne les mots quasi-sturmiens, c'est-à-dire les mots de complexité $n + k$, où k est un entier (Rote, 1994).

2.5 Partition du cercle unité

Soient $x, \alpha, \beta \in \mathbb{R}/\mathbb{Z}$. Étant donné un mot $w \in \Sigma^*$, on définit I_w comme l'ensemble des points du cercle unité à partir desquels le mot w est lu sous des rotations d'angle α :

$$I_w = \{\gamma \in \mathbb{R}/\mathbb{Z} \mid \mathrm{Pref}_{|w|}(\mathbf{C}(\gamma)) = w\}.$$

Il est facile de calculer les ensembles I_w :

$$I_w = \bigcap_{i=0}^{n-1} R_\alpha^{-i}(I_{w_i}) \tag{2.1}$$

avec $I_1 = [0,\beta)$ et $I_0 = [\beta,1)$. Il suit de la formule (2.1) que I_w n'est pas nécessairement un intervalle puisque rien ne garantit que l'intersection de deux intervalles du cercle unité est également un intervalle (voir Figure 2.6). Nous étudions cette situation en détail un peu plus loin dans ce chapitre.

Exemple 10. Reprenons l'exemple 9 où nous avons $x = 0{,}23435636$, $\alpha = 0{,}222435236$ et $\beta = 0{,}30234023$ et donc

$$\mathbf{C}(x) = 1000100011000100011000100011000100011000\cdots.$$

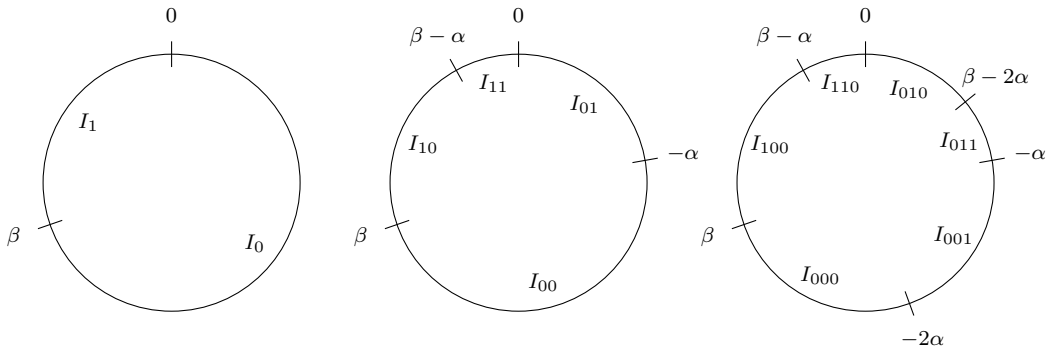


Figure 2.8: Représentation des intervalles I_w, où $|w| \in \{1,2,3\}$ pour le codage de rotations de paramètres $x = 0{,}23435636$, $\alpha = 0{,}222435236$ et $\beta = 0{,}30234023$. On remarque que les intervalles correspondant à des facteurs de même longueur forment une partition du cercle unité.

Calculons les ensembles I_w pour tout facteur w de $\mathbf{C}$ de longueur au plus 3. On obtient :

$$
\begin{aligned}
I_0 &= [\beta, 0) \\
I_1 &= [0, \beta) \\
I_{00} &= I_0 \cap R_\alpha^{-1}(I_0) = [\beta, 0) \cap [\beta - \alpha, -\alpha) = [\beta, -\alpha) \\
I_{11} &= I_1 \cap R_\alpha^{-1}(I_1) = [0, \beta) \cap [-\alpha, \beta - \alpha) = [0, \beta - \alpha) \\
I_{01} &= I_0 \cap R_\alpha^{-1}(I_1) = [\beta, 0) \cap [-\alpha, \beta - \alpha) = [-\alpha, 0) \\
I_{10} &= I_1 \cap R_\alpha^{-1}(I_0) = [0, \beta) \cap [\beta - \alpha, -\alpha) = [\beta - \alpha, \beta) \\
I_{000} &= I_0 \cap R_\alpha^{-1}(I_0) \cap R_\alpha^{-2}(I_0) = [\beta, 0) \cap [\beta - \alpha, -\alpha) \cap [\beta - 2\alpha, -2\alpha) = [\beta, -2\alpha) \\
I_{001} &= I_0 \cap R_\alpha^{-1}(I_0) \cap R_\alpha^{-2}(I_1) = [\beta, 0) \cap [\beta - \alpha, -\alpha) \cap [-2\alpha, \beta - 2\alpha) = [-2\alpha, -\alpha) \\
I_{010} &= I_0 \cap R_\alpha^{-1}(I_1) \cap R_\alpha^{-2}(I_0) = [\beta, 0) \cap [-\alpha, \beta - \alpha) \cap [\beta - 2\alpha, -2\alpha) = [\beta - 2\alpha, 0) \\
I_{011} &= I_0 \cap R_\alpha^{-1}(I_1) \cap R_\alpha^{-2}(I_1) = [\beta, 0) \cap [-\alpha, \beta - \alpha) \cap [-2\alpha, \beta - 2\alpha) = [-\alpha, \beta - 2\alpha) \\
I_{100} &= I_1 \cap R_\alpha^{-1}(I_0) \cap R_\alpha^{-2}(I_0) = [0, \beta) \cap [\beta - \alpha, -\alpha) \cap [\beta - 2\alpha, -2\alpha) = [\beta - \alpha, \beta) \\
I_{110} &= I_1 \cap R_\alpha^{-1}(I_1) \cap R_\alpha^{-2}(I_0) = [0, \beta) \cap [-\alpha, \beta - \alpha) \cap [\beta - 2\alpha, -2\alpha) = [0, \beta - \alpha)
\end{aligned}
$$

Remarquons ici que pour $n = 1, 2, 3$, on a que

$$\{I_w \mid w \in \mathrm{Fact}_n(\mathbf{C})\}$$

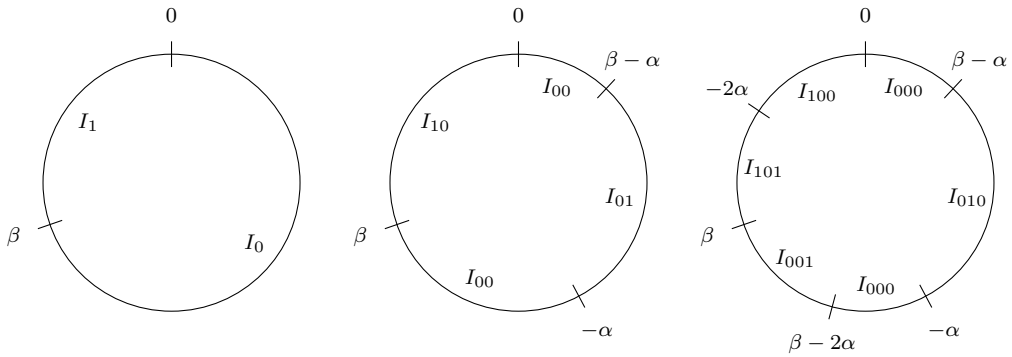


Figure 2.9: Partitions induites par le codage de rotations de paramètres $x = 0{,}23435636$, $\alpha = 0{,}422435236$ et $\beta = 0{,}30234023$ pour les longueurs 1, 2 et 3. Les ensembles I_{00} et I_{000} ne sont pas des intervalles.

est une partition du cercle unité [0,1). De plus, chacun des I_w pour w un facteur de longueur au plus 3 est bien un intervalle. Une représentation graphique de la situation se trouve à la figure 2.8.

Proposition 4. Soit n un entier positif non nul. L'ensemble

$$P_n = \{I_w \mid w \in \mathrm{Fact}_n(\mathbf{C}(\gamma)\}$$

est une partition du cercle unité. De plus, l'ensemble des bornes de la partition P_n est donné par

$$\mathcal{P} = \{-i\alpha \mid i = 0, 1, \ldots, n-1\} \cup \{\beta - i\alpha \mid i = 0, 1, \ldots, n-1\}.$$

Démonstration. Par récurrence sur n et à l'aide de la formule (2.1). □

Exemple 11. Considérons le codage de rotations obtenu avec les paramètres $x = 0{,}23435636$, $\alpha = 0{,}422435236$ et $\beta = 0{,}30234023$. Alors

$$\mathbf{C}(x) = 10100001010000100001010000101000001010010\cdots.$$

En calculant les ensembles I_w où $w \in \mathrm{Fact}_n(\mathbf{C})$ et $n = 1, 2, 3$, on constate que I_{00} et I_{000} ne sont pas des intervalles (voir figure 2.9).

Nous décrivons un peu plus bas certaines conditions sous lesquelles l'ensemble I_w est un intervalle.

La fonction de Poincaré présentée plus haut est particulièrement adaptée pour étudier les mots de retour des codages de rotations. Plus précisément, nous avons le lemme suivant :

Lemme 2. Soient u,$v \in \text{Fact}(\mathbf{C})$ et $T = T_\alpha(I_u,I_v)$ la fonction de premier retour de Poincaré. Alors l'ensemble des mots de retour complet de u à v dans $\mathbf{C}$ est exactement décrit par

$$\{\mathbf{C}_{T(\gamma)+|v|}(\gamma) \mid \gamma \in I_u\}.$$

Démonstration. Le mot w est un mot de retour complet de u vers v dans $\mathbf{C}$ si et seulement si

(1) j est une occurrence de u,

(2) k est la première occurrence de v dans w strictement plus grande que j et

(3) $w = \mathbf{C}_{[j,k+|v|-1]}$.

Ceci est équivalent à ce qu'il existe $\gamma \in I_w$ tel que $\gamma \in I_u$, $R_\alpha^{|w|-|v|} \in I_v$ et $R_\alpha^i \notin I_v$ pour $0 < i < |w|-|v|$, c'est-à-dire qu'il existe $\gamma \in I_w$ tel que $\gamma \in I_u$ et $T_\alpha(I_u, I_v)(\gamma) = |w|-|v|$ et donc il existe $\gamma \in I_u$ tel que $w = \mathbf{C}_{|w|}(\gamma) = \mathbf{C}_{T_\alpha(I_u,I_v)(\gamma)+|v|}(\gamma)$. □

Un autre fait connu dans la communauté des systèmes dynamiques est le suivant, qui se démontre en modifiant légèrement la démonstration du lemme 1 de Katok (Katok, 1980).

Lemme 3. Soit $w \in \Sigma^*$ tel que I_w est un intervalle. Alors

(i) $P_\alpha(I_w, I_w)$ est un échange de q intervalles, où $q \in \{1,2,3\}$.

(ii) w a au plus 3 mots de retour complets. □

Le cas où I_w n'est pas un intervalle est décrit par les deux lemmes qui suivent. Nous commençons d'abord par un lemme un peu plus technique.

Lemme 4. Soit E un ensemble fini d'indices, $(A_i)_{i\in E}$ une famille d'intervalles fermés à gauche ouverts à droite de $\mathbb{R}/\mathbb{Z}$. Soit $\ell = \min\{|A_i| : i \in E\}$ et $L = \max\{|A_i| : i \in E\}$. Si $\ell + L \leq 1$, alors $\bigcap_{i\in E} A_i$ est un intervalle.

Démonstration. La démonstration se fait par récurrence sur $n = |E|$. Pour $n = 1$, Il n'y a rien à démontrer puisqu'on calcule l'intersection d'un seul intervalle, qui est clairement un intervalle.

Autrement, soit $k \in E$ un indice tel que $|A_k| = L = \max\{|A_i| : i \in E\}$ et $\ell = \min\{|A_i| : i \in E\}$. Alors

$$\bigcap_{i\in E} A_i = A_k \cap \left(\bigcap_{i\in E\setminus\{k\}} A_i \right).$$

Par l'hypothèse de récurrence, $\bigcap_{i\in E\setminus\{k\}} A_i$ est un intervalle et sa longueur est au plus ℓ. Or, $\ell + L \leq 1$, de sorte que A_k et $\bigcap_{i\in E\setminus\{k\}} A_i$ ne peuvent s'intersecter en leur deux extrémités. □

Nous poursuivons avec un lemme garantissant, sous des conditions raisonnables, que l'ensemble I_w est bien un intervalle.

Lemme 5. Soit $w \in \Sigma^*$ et $\alpha, \beta \in \mathbb{R}/\mathbb{Z}$. Alors les énoncés suivants sont vérifiés :

(i) Si les lettres 0 et 1 apparaissent toutes deux dans w, alors I_w est un intervalle et $|I_w| \leq \alpha$;

(ii) Si $\alpha < \beta$ et $\alpha < 1 - \beta$, alors I_w est un intervalle.

Démonstration. (i) Soit $L = \{|R_\alpha^{-i}(I_{w_i})| : 0 \leq i \leq n-1\}$. Si les lettres 0 et 1 apparaissent dans w, alors $L = \{\beta, 1-\beta\}$ de sorte que $\min(L)+\max(L) = 1$. Il s'en suit que l'intersection $I_w = \bigcap_{0\leq i\leq n-1} R_\alpha^{-i}(I_{w_i})$ vérifie les conditions du lemme 4 de sorte que I_w est un intervalle.

Au moins un des deux facteurs 01 et 10 doit apparaître dans w. Dans le premier cas, la longueur de I_w est bornée comme suit : $|I_w| \leq |I_{01}| = |R_\alpha(I_0) \cap R_\alpha^{-1}(I_1)| \leq \alpha$. Une inégalité semblable peut être obtenue à partir du facteur 10.

(ii) Nous démontrons la contraposée. Supposons qu'il existe un entier positif n et un mot $w \in \mathrm{Fact}_n(\mathbf{C})$ tel que I_w est un intervalle mais I_{wa} n'est pas un intervalle, où a est une lettre. Il suit de la partie (i) de cette démonstration que $w = a^n$ et $|I_a| > |I_b|$, où $b \neq a$ est l'autre lettre. Or, $I_w = \bigcap_{i=0}^{n-1} R_\alpha^{-i}(I_a)$. En particulier, $I_w \subseteq R_\alpha^{-n+1}(I_a)$ de sorte que

$$R_\alpha^{-n+1}(I_b) \subseteq [0,1) \setminus I_w.$$

De plus, $I_{wa} = I_w \cap R_\alpha^{-n}(I_a)$ et ainsi $R_\alpha^{-n}(I_b) \subseteq I_w$. Ceci entraîne que $R_\alpha^{-n+1}(I_b) \cap R_\alpha^{-n}(I_b) = \emptyset$, $R_\alpha(I_b) \cap I_b = \emptyset$ et $\alpha \geq |I_b| = \min\{\beta, 1-\beta\}$. □

2.6 Symétries de la partition

Soit $n \in \mathbb{N}$. L'ensemble

$$\mathcal{P}_n = \{-i\alpha \mid i = 0, 1, \ldots, n-1\} \cup \{\beta - i\alpha \mid i = 0, 1, \ldots, n-1\}$$

des points correspondant aux frontières de la partition est invariant sous la réflexion déterminée par l'axe passant par les points y_n et y_n' vérifiant les équations

$$2y_n = 2y_n' = \beta - (n-1)\alpha.$$

Plus précisément, il s'agit de la réflexion

$$S_n : \mathbb{R}/\mathbb{Z} \to \mathbb{R}/\mathbb{Z} : x \mapsto 2y_n - x.$$

Lorsqu'on se ramène à un codage de rotations $\mathbf{C}$, elle permet en particulier de démontrer que le langage $\mathrm{Fact}(\mathbf{C})$ est invariant sous l'opérateur image miroir.

Exemple 12. Considérons les facteurs de longueur 3 d'un codage de rotations de paramètre $\alpha = 0{,}135$ et $\beta = 0{,}578$. Une représentation graphique se trouve à la figure 2.10. On vérifie que

$$S_3(0) \quad = \quad \beta - 2\alpha - 0 = \beta - 2\alpha$$

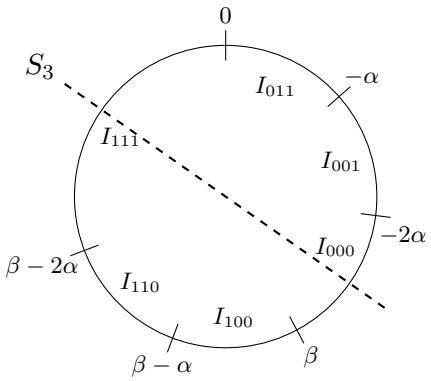


Figure 2.10: Représentation graphique de la réflexion S_3 appliquée à l'ensemble $\mathcal{P}_3$ d'un codage de rotations de paramètres $\alpha = 0{,}135$, $\beta = 0{,}578$. Les points -2α, $-\alpha$ et 0 sont respectivement envoyés sur les points β, $\beta - \alpha$ et $\beta - 2\alpha$. En outre, les intervalles délimités par ces points vérifient bien l'identité $S_3(\mathrm{Int}(I_w)) = \mathrm{Int}(I_{\widetilde{w}})$.

$$\begin{aligned} S_3(-\alpha) &= \beta - 2\alpha + \alpha = \beta - \alpha \\ S_3(-2\alpha) &= \beta - 2\alpha + 2\alpha = \beta \end{aligned}$$

De plus, l'identité $S_3(\mathrm{Int}(I_w)) = \mathrm{Int}(I_{\widetilde{w}})$ est vérifiée dans cet exemple puisque

$$\begin{aligned} S_3(\mathrm{Int}(I_{111})) &= \mathrm{Int}(I_{111}) \\ S_3(\mathrm{Int}(I_{011})) &= \mathrm{Int}(I_{110}) \\ S_3(\mathrm{Int}(I_{100})) &= \mathrm{Int}(I_{001}) \\ S_3(\mathrm{Int}(I_{000})) &= \mathrm{Int}(I_{000}) \end{aligned}$$

Le lemme qui suit décrit les propriétés qui nous intéressent par rapport à la réflexion S_n :

Lemme 6. Soit $m \in \mathbb{N}$. Alors les énoncés suivants sont vérifiés :

(i) Si $S_n(x) = R_\alpha^m(x)$, alors $S_n(x + \alpha) = R_\alpha^{m-1}(x)$.

(ii) Si $x \in \mathrm{Int}(I_w)$, alors $S_n(x) \in I_{\widetilde{w}}$, où $n = |w|$.

(iii) Si $S_n(x) = R_\alpha^m(x)$, alors $\mathbf{C}_{n+m}(x)$ est un palindrome.

Démonstration. (i) Nous avons

$$\begin{aligned} S_n(x+\alpha) &= 2y_n - x - \alpha = S_n(x) - \alpha \\ &= x + m\alpha - \alpha = x + (m-1)\alpha. \end{aligned}$$

(ii) Soit w un facteur de $\mathbf{C}$ de longueur n. Rappelons que $I_w = \bigcap_{0\leq i\leq n-1} R_\alpha^{-i}(I_{w_i})$. De plus, on remarque que

$$R_\alpha^{-i}(I_{w_i}) = \begin{cases} [-i\alpha, \beta - i\alpha) & \text{si } w_i = 1, \\ [\beta - i\alpha, -i\alpha) & \text{si } w_i = 0. \end{cases}$$

Aussi,

$$\begin{aligned} S_n(\mathrm{Int}(R_\alpha^{-i}(I_{w_i}))) &= \begin{cases} (-(n-i-1)\alpha, \beta - (n-i-1)\alpha) & \text{si } w_i = 1, \\ (\beta - (n-i-1)\alpha, -(n-i-1)\alpha) & \text{si } w_i = 0. \end{cases} \\ &= \mathrm{Int}(R_\alpha^{-(n-i-1)}(I_{w_i})) \end{aligned}$$

Il vient donc

$$\begin{aligned} S_n(\mathrm{Int}(I_w)) &= S_n\left(\bigcap_{0\leq i\leq n-1} \mathrm{Int}(R_\alpha^{-i}(I_{w_i}))\right) \\ &= \bigcap_{0\leq i\leq n-1} S_n(\mathrm{Int}(R_\alpha^{-i}(I_{w_i}))) \\ &= \bigcap_{0\leq i\leq n-1} \mathrm{Int}(R_\alpha^{-(n-i-1)}(I_{w_i})) \\ &= \bigcap_{0\leq i\leq n-1} \mathrm{Int}(R_\alpha^{-i}(I_{\widetilde{w}_i})) \\ &= \mathrm{Int}(I_{\widetilde{w}}), \end{aligned}$$

tel que voulu.

(iii) Soit $n \in \mathbb{N}$ et w le mot de longueur n tel que $x \in I_w$. La démonstration est faite par récurrence sur m. Si $m = 0$, alors $x \in I_{\widetilde{w}}$ par (ii) de sorte que $w = \widetilde{w} = \mathbf{C}_{n+0}(x)$. Si

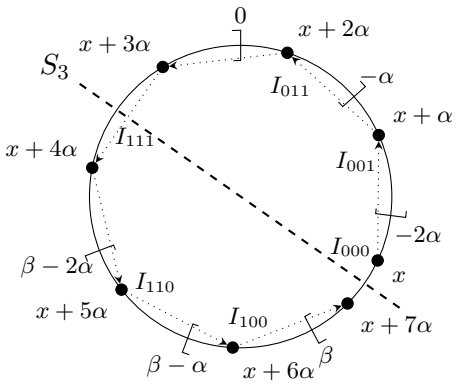


Figure 2.11: Trajectoire du point $x = (\beta - 9\alpha)/2 = 0{,}6815$ pour la partition P_3 avec les paramètres $\alpha = 0{,}135$ et $\beta = 0{,}578$ sous des rotations d'angle α. La trajectoire est symétrique par rapport à la réflexion S_3. De plus, $\mathbf{C}_{10}(x) = 0001111000$ est un palindrome.

$m = 1$, alors $x + \alpha \in I_{\widetilde{w}}$ par la partie (ii). Ainsi, $\mathbf{C}_{n+1}(x) = a\widetilde{w} = wb$, où $a, b \in \{0{,}1\}$. On voit clairement alors que $a = w_0 = b$ et donc $\mathbf{C}_{n+1}(x)$ est un palindrome. Plus généralement, pour $m \geq 2$, $\mathbf{C}_{m+n}(x) = a\mathbf{C}_{m+n-2}(x+\alpha)b$, où $a = w_0 = b$ puisque a est la première lettre de w et b est la dernière lettre de $\widetilde{w}$. Il suit de la partie (i) que $S_n(x+\alpha) = (x+\alpha) + (m-2)\alpha$, ce qui entraîne que $\mathbf{C}_{m+n-2}(x+\alpha)$ est un palindrome, en vertu de l'hypothèse de récurrence. □

Exemple 13. En continuité avec l'exemple 12, considérons le point $x = (\beta - 9\alpha)/2 = 0{,}6815$. Nous avons $S_3(x) = x + 7\alpha$ et l'orbite du point x sous des rotations d'angle α jusqu'au premier retour dans l'intervalle $[\beta, -2\alpha)$ est symétrique par rapport à la réflexion S_3 (voir figure 2.11. La réflexion S_3 et plus généralement les réflexions S_n, où $n \in \mathbb{N}$, jouent un rôle clé dans la démonstration que tout mot de retour complet d'un palindrome est lui-même un palindrome. Nous démontrons plus rigoureusement ce fait dans la section suivante.

2.7 Tout codage de rotations est plein

Cette section est consacrée à la démonstration du fait que les codages de rotations sont pleins en palindromes, c'est-à-dire qu'ils réalisent la borne supérieure décrite par Droubay, Justin et Pirillo (Droubay, Justin et Pirillo, 2001). L'idée centrale du théorème

repose sur le fait suivant.

Proposition 5. (Bucci et al., 2009) Un mot w est plein si et seulement si pour tout facteur palindrome u de w, les mots de retour complets de u sont égalements des palindromes.

Dans cette perspective, soit $u \in \mathrm{Pal}(\mathbf{C})$, où $\mathbf{C} = \mathbf{C}(x)$ est un codage de rotations de paramètres (α, β). Notre objectif est de montrer que tout mot de retour complet de u dans $\mathbf{C}$ est effectivement un palindrome. Nous distinguons deux cas selon que I_u est un intervalle ou non.

2.7.1 Cas 1 : I_u est un intervalle

Rappelons que, en vertu du lemme 3, la fonction $P_\alpha(I_u, I_u)$ est un échange de q intervalles, où $q \in \{1, 2, 3\}$. Soient $(J_i)_{1 \leq i \leq q}$ les q sous-intervalles de I_u et $t_i \geq 1$ les entiers vérifiant $P_\alpha(I_u, I_u)(J_i) = R_\alpha^{t_i}(J_i)$, où $i < j$ impliques $t_i < t_j$. Tout point du sous-intervalle J_i requiert le même nombre de rotations t_i d'angle α pour atteindre à nouveau l'intervalle I_u. En revanche, dans le cas général, il est possible que deux points de J_i codent chacun un mot différent de longueur t_i. Par exemple, en reprenant l'exemple 9, on constate que le facteur 100 admet exactement trois mots de retour complets dans

$$\mathbf{C}(x) = 1000100011000100011000100011000100011000100011000100011000100\cdots,$$

en l'occurrence, 1000100, 10001100 et 10000100 (ce dernier apparaît plus loin dans $\mathbf{C}$).

Cette observation étant soulevée, il est possible de démontrer que, à la condition que I_u soit un intervalle, alors pour tout temps de retour t_i associé à un intervalle J_i, il existe un unique mot de retour complet de longueur t_i :

Lemme 7. Supposons que u est un palindrome et I_u est un intervalle. Soient x,$y \in J_i$ tels que $1 \leq i \leq q$. Alors $\mathbf{C}_{t_i}(x) = \mathbf{C}_{t_i}(y)$.

Démonstration. Le raisonnement se fait par l'absurde. Supposons donc qu'il existe un entier k, $0 \leq k < t_i$, tel que $R_\alpha^k(x) \in I_0 = [0, \beta)$ et $R_\alpha^k(y) \in I_1 = [\beta, 0)$ (sans perte

de généralité, on intervertit x et y dans l'autre cas). Alors

$$\beta \in (R_\alpha^k(x), R_\alpha^k(y)] \subset R_\alpha^k(\mathrm{Int}(J_i)).$$

Soit $n = |u|$. Il y a deux cas à considérer :

(1) Supposons que $k < n$. Alors $\beta - k\alpha \in \mathrm{Int}(J_i)$, ce qui est contradictoire puisque $\beta - k\alpha$ est un point de $\mathcal{P}_n$ et ne peut donc pas être contenu dans l'intérieur d'un intervalle J_i ;

(2) Supposons maintenant que $k \geq n$. Alors $\beta - \ell\alpha \in R_\alpha^{k-\ell}(\mathrm{Int}(J_i))$ pour $0 \leq \ell < n$. Or, le fait que u est un palindrome entraîne qu'au moins une des deux extrémités de I_u est de la forme $\beta - \ell\alpha$ (car elles sont échangées par la réflexion S_n). Par conséquent le point x ou le point y retourne dans l'intervalle I_u après au plus $n \leq k < t_i$ rotations, ce qui contredit la minimalité de t_i. □

Le lemme 7 nous permet de conclure que les mots de retour complets d'un palindrome u sont complètement déterminés par l'échange d'intervalles induit par la fonction de premier retour de Poincaré. Il suffit donc pour la suite de choisir convenablement un représentant pour calculer le mot de retour complet de tous les points d'un sous-intervalle J_i donné, pour $i \in \{1, 2, \ldots, q\}$. Il s'avère que le point milieu m_i de l'intervalle J_i est particulièrement adapté pour ses propriétés symétriques : en effet, les lemmes 2 et 7 entraînent que si $u \in \mathrm{Fact}_n(\mathbf{C})$ est un palindrome tel que I_u est un intervalle, alors

$$\mathrm{CRet}_{\mathbf{C}}(u, u) = \{C_{t_i+n}(m_i) \mid 1 \leq i \leq q\}.$$

Nous sommes maintenant en mesure de traiter le cas où I_u est un intervalle, avec u un palindrome.

Proposition 6. Si u est un palindrome et I_u est un intervalle, alors tout mot de retour complet de u est un palindrome.

Démonstration. Soit $T = T_\alpha(I_u, I_u)$, $P = P_\alpha(I_u, I_u)$, $n = |u|$ et $w \in \mathrm{CRet}_{\mathbf{C}}(u)$. Puisque I_u est un intervalle, le lemme 7 s'applique, de sorte qu'on peut choisir un des

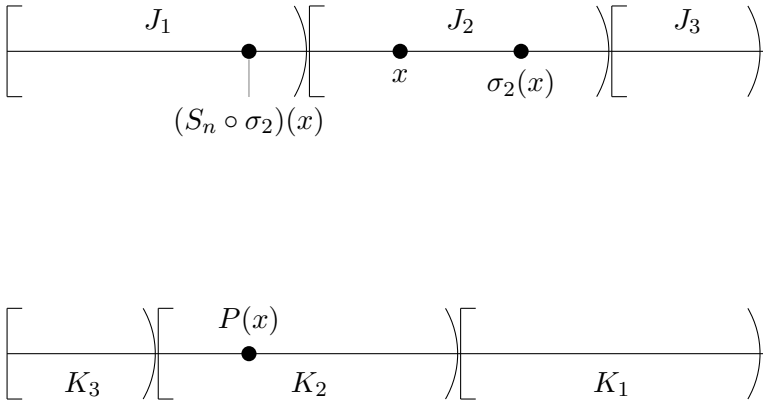


Figure 2.12: Illustration de l'identité $P(x) = (S_n \circ \sigma_i)(x)$, où σ_i est la réflexion par rapport au centre du sous-intervalle J_i et S_n est la réflexion par rapport à l'intervalle entier $J_1 \cup J_2 \cup J_3$. Notons ici que les intervalles de départ et d'arrivée sont les mêmes.

points m_i $(i = 1, 2, 3)$ comme représentant. Plus précisément, il existe donc un entier $i \in \{1, 2, 3\}$ tel que $w = \mathbf{C}_{T(m_i)+n}(m_i)$, où m_i est le point milieu de l'intervalle J_i. Soit

$$\sigma_i : J_i \to J_i : \gamma \mapsto 2m_i - \gamma$$

la réflexion déterminée par le point m_i. On remarque que pour tout point $x \in I_u$, on a l'identité $P(x) = (S_n \circ \sigma_i)(x)$ (voir figure 2.12). Comme P est un échange d'intervalles de permutation (321), (21) ou (1), on en déduit les égalités suivantes :

$$m_i + T(m_i)\alpha = P(m_i) = (S_n \circ \sigma_i)(m_i) = S_n(m_i).$$

Or, la démonstration du lemme 7 nous garantit qu'aucun des points $m_i + \ell\alpha$ n'est dans $\mathcal{P}_n$, de sorte que le lemme 6 (iii) s'applique et on en conclut que w est un palindrome. □

2.7.2 Cas 2 : I_u n'est pas un intervalle

Dans le cas où I_u n'est pas un intervalle, il suit du lemme 5 que $u = a^n$ pour une certaine lettre $a \in \{0, 1\}$. En conséquence, tout mot de retour complet w de u est de l'une des deux formes suivantes :

(i) $w = a^{n+1}$;

(ii) $w \in a^n b\Sigma^* \cap \Sigma^* b a^n$, avec $a \neq b$;

La première forme est facile à traiter, car, clairement, a^{n+1} est un palindrome. Nous considérons donc en détail la forme (ii).

Proposition 7. Si $u' = a^n b$ et $v' = ba^n$, alors $P_\alpha(I_{u'}, I_{v'})$ est une bijection.

Remarque 1. La proposition 7 peut sembler évidente, mais il n'est pas clair que l'orbite d'un point $x \in I_{u'}$ atteigne l'intervalle $I_{v'}$ avant de repasser par $I_{u'}$. Par ailleurs, il existe des cas où la fonction $P_\alpha(I_u, I_v)$ n'est pas une bijection, lorsque u et v sont des facteurs quelconques d'un même codage de rotations.

Démonstration. Il suffit de démontrer que $P_{-\alpha}(I_{v'}, I_{u'})$ est l'inverse de la fonction $P_\alpha(I_{u'}, I_{v'})$. Le raisonnement se fait par contradiction. Supposons donc qu'il existe $x \in I_{u'}$ et $y \in I_{v'}$ tels que $y = P_\alpha(I_{u'}, I_{v'})(x)$ et $T_{-\alpha}(I_{v'}, I_{u'})(y) < T_\alpha(I_{u'}, I_{v'})(x)$, c'est-à-dire que l'orbite de y passe d'abord par $I_{u'}$ avant d'atteindre x lorsqu'on applique des rotations d'angle $-\alpha$. En vertu du lemme 2, cette condition se traduit par le fait que $\mathbf{C}$ possède un mot de retour complet w de $u' = a^n b$ vers $v' = ba^n$ contenant u' au moins deux fois. Or, si $u' = a^n b$ apparaît au moins deux fois, ce doit être également le cas de $v' = ba^n$: il suffit de regarder la deuxième occurrence de u' et de revenir en arrière jusqu'au dernier b rencontré. Ceci est une contradiction, car $|w|_{v'} > 1$ contredit la définition de mot de retour complet. □

Le fait suivant est immédiat :

Corollaire 1. Si $u' = a^n b$ et $v' = ba^n$, alors $P_\alpha(I_{u'}, I_{v'})$ est un échange de q intervalles, où $q \in \{1, 2, 3\}$.

Le corollaire 1 entraîne que deux points $x, y \in I_{u'}$ d'un même sous-intervalle J_i, où $1 \leq i \leq q \leq 3$ requièrent le même nombre de rotations d'angle α pour atteindre l'intervalle $I_{v'}$. Posons donc

$$t_i = T_\alpha(I_{u'}, I_{v'})(J_i), \quad \text{pour } 1 \leq i \leq q.$$

Comme il a été mentionné dans la sous-section précédente, nous devons tout de même nous assurer que deux points d'un même sous-intervalle J_i codent également le même mot de retour complet. Cet aspect est considéré dans le lemme suivant :

Lemme 8. Soient $u' = a^n b$ et $v' = ba^n$ et $x, y \in J_i$, où $1 \leq i \leq q$. Alors $\mathbf{C}_{t_i}(x) = \mathbf{C}_{t_i}(y)$.

Démonstration. Posons $n' = n + 1$. La démonstration se fait encore une fois par l'absurde. Supposons qu'il existe un entier k, $0 \leq k < t_i$, tel que $R_\alpha^k(x) \in I_0 = [0, \beta)$ et $R_\alpha^k(y) \in I_1 = [\beta, 0)$ (sans perte de généralité, on intervertit x et y dans l'autre cas). Alors

$$\beta \in (R_\alpha^k(x), R_\alpha^k(y)] \subset R_\alpha^k(\mathrm{Int}(J_i)).$$

D'une part, supposons que $k < n'$. Alors $\beta - k\alpha \in \mathrm{Int}(J_i)$, ce qui est absurde puisque $\beta - k\alpha \in \mathcal{P}_{n'}$ et aucun point de $\mathcal{P}_{n'}$ n'est contenu dans un intervalle.

D'autre part, supposons que $k \geq n'$. Alors $\beta - \ell\alpha \in R_\alpha^{k-\ell}(\mathrm{Int}(J_i))$ pour $0 \leq \ell \leq n$. Or, au moins une des extrémités des intervalles $I_{u'}$ et $I_{v'}$ est de la forme $\beta - \ell\alpha$ (puisque $u' = a^n b$ et $v' = ba^n$). Par conséquent le point x ou le point y retourne dans l'intervalle I_u après au plus $n' \leq k < t_i$ rotations, ce qui contredit la minimalité de t_i. □

Nous choisissons encore une fois le point milieu m_i comme représentant du sous-intervalle J_i, pour $1 \leq i \leq q$. Les lemmes 2 et 8 nous garantissent que si $u' = a^n b$ et $v' = ba^n$, alors

$$\mathrm{CRet}_{\mathbf{C}}(u', v') = \{\mathbf{C}_{t_i+n+1}(m_i) \mid 1 \leq i \leq q\}. \tag{2.2}$$

Proposition 8. Soit $T = T_\alpha(I_{a^n b}, I_{ba^n})$. Alors

$$\mathrm{CRet}(a^n) \subseteq \{a^{n+1}\} \cup \{\mathbf{C}_{T(m_i)+n+1}(m_i) \mid 1 \leq i \leq q\}.$$

Démonstration. Soit $us \in \mathrm{CRet}(u)$, où s est un mot non vide, et b la première lettre de s. Si $b = a$, alors $us = ua = a^{n+1}$. Si $b \neq a$, alors $us = ubtu = a^n bta^n$, pour un certain mot t. En particulier, b doit être la dernière lettre de bt. Ainsi, us est exactement un des mots de retour complet de $a^n b$ vers ba^n décrit par l'équation (2.2). □

Il ne reste plus qu'à démontrer le résultat principal pour le cas où I_u n'est pas un intervalle.

Proposition 9. Si u est un palindrome et que I_u n'est pas un intervalle, alors tout mot de retour complet de u est un palindrome.

Démonstration. La démonstration est très semblable à celle de la proposition 6. Soit $n = |u|$ et $w \in \mathrm{CRet}_{\mathbf{C}}(u)$. Puisque I_u n'est pas un intervalle, nous savons que $u = a^n$, où $a \in \{0, 1\}$. La proposition 8 implique que $w = a^{n+1}$, qui est bien un palindrome, où $w = \mathbf{C}_{T(m_i)+n+1}(m_i)$ pour $1 \leq i \leq q$ et $T = T_\alpha(I_{a^n b}, I_{ba^n})$. Posons $P = P_\alpha(I_{a^n b}, I_{ba^n})$ et soit

$$\sigma_i : J_i \to J_i : \gamma \mapsto 2m_i - \gamma$$

la réflexion par rapport au point milieu m_i. Alors

$$m_i + T(m_i)\alpha = P_\alpha(I_{a^n b}, I_{ba^n})(m_i) = (S_{n+1} \circ \sigma_i)(m_i) = S_{n+1}(m_i),$$

de sorte que w est un palindrome, par le lemme 6 (iii). □

2.7.3 Théorème principal

Les propositions 6 et 9 donnent le théorème suivant :

Théorème 2. (Blondin Massé et al., 2011) Tout codage de rotations sur deux intervalles est plein.

Démonstration. Découle des propositions 6 et 9 ainsi que du fait qu'un mot est plein si et seulement si chacun de ses mots de retour complet d'un palindrome est un palindrome.

□

De plus, la proposition suivante, qui peut être déduite du lemme 7 et de la proposition 9, constitue un raffinement des résultats de Katok (Katok, 1980) :

Corollaire 2. L'ensemble des mots $w \in \mathrm{Fact}(\mathbf{C})$ ayant quatre mots de retour complets

est fini. De plus, dans ce cas, I_w n'est pas un intervalle et w est la puissance d'une lettre. □

Exemple 14. Considérons le codage de rotations de paramètres $x = 0{,}23435636$, $\alpha = 0{,}422435236$ et $\beta = 0{,}30234023$:

$$\mathbf{C}(x) = 10100001010000100001010000101000001010010\cdots .$$

On constate que les mots de retour complets de 000 sont tous des palindromes :

```
sage: x = 0.23435636
sage: alpha = 0.422435236
sage: beta = 0.30234023
sage: w = words.CodingOfRotationWord(alpha, beta, x)
sage: w
10100001010000100001010000101000001010010...
sage: u = w[4:7]
sage: u
000
sage: w[:1000].complete_return_words(u)
set([word: 00010100101000, word: 0000, word: 0001000, word:
    000101000])
```

2.7.4 Démonstration alternative

Récemment et de façon indépendante, une formule liant complexités factorielle et palindromique a été publiée (Bucci et al., 2009). Plus précisément, les auteurs démontrent qu'un mot infini w dont le langage est préservé sous l'opérateur image miroir est plein si et seulement si

$$\mathrm{Pal}_n(w) + \mathrm{Pal}_{n+1}(w) = \mathrm{Fact}_{n+1}(w) - \mathrm{Fact}_n(w) + 2, \quad \text{pour tout } n \in \mathbb{N}. \tag{2.3}$$

Puisque l'équation (2.3) est vérifiée dans le cas où le nombre de mots de retour complet est au plus 3, il semble probable qu'une démonstration plus directe du théorème 2 puisse en être déduite. En revanche, ce n'est pas aussi clair pour les cas où le nombre de mots

de retour complet est 2 ou 4. Dans ce chapitre, tous les cas ont été traités, incluant le cas périodique et les cas dégénérés.

Avant de conclure, remarquons que le fait que le nombre de mots de retour complets est borné par 3 lorsque $\alpha < \min\{\beta, 1-\beta\}$ peut également être trouvé dans les travaux de Keane, Rauzy et Adamczewski dans le cas où α est irrationnel (Keane, 1975; Rauzy, 1979; Adamczewski, 2002). Aussi, il était déjà connu que $|\mathrm{CRet}(w)| = k$ lorsqu'on considère un échange de k intervalles non dégénéré (Vuillon, 2007) et que $|\mathrm{CRet}(w)| = 2$ pour un codage de rotations avec $\alpha = \beta$ (qui se trouve être le cas sturmien) (Vuillon, 2001). Néanmoins, les démonstrations qui se trouvent dans ce chapitre sont vérifiées pour n'importe quelles valeurs de α et β, qu'elles soient rationnelles ou irrationnelles.

2.8 Suites de Rote complémentaires

Une sous-famille de mots inclus dans les codages de rotations sont les *suites (ou mots) de Rote complémentaires*, qu'on définit comme les mots infinis ayant une complexité factorielle de $2n$ (Rote, 1994) et dont le langage est stable par complémentation (le morphisme échangeant les lettres). Il s'agit en réalité des codages de rotations de paramètre $\beta = 1/2$. Dans cette section, nous nous consacrons à la démonstration du fait que les suites de Rote complémentaires sont pleines, en nous basant sur l'interaction entre les palindromes et les antipalindromes s'y retrouvant. À noter qu'une section similaire se trouve dans le mémoire de l'auteur de cette thèse, mais qu'elle y est incluse par souci de complétude (Blondin Massé, 2008).

Rappelons qu'un *antipalindrome* q est un mot vérifiant $\overline{q} = \widetilde{q}$, où $\bar{\cdot}$ est le morphisme involutif non triviale sur $\Sigma = \{0, 1\}$ qui échange les lettres $0 \mapsto 1, 1 \mapsto 0$.

Une relation d'ordre partiel sur les palindromes est définie comme suit. Soient p et q deux palindromes. On écrit $p \prec q$ s'il existe un mot non vide x tel que $q = xp\widetilde{x}$. Autrement dit, p est un *palindrome centré* dans le palindrome q.

Une autre fonction utile est celle de différence. Plus précisément, soit $w = w_1 w_2 \cdots w_n$ un

mot de longueur $n \geq 2$. Alors la *différence de* w, notée $\Delta(w)$, est le mot $v = v_1 v_2 \cdots v_{n-1}$ défini par

$$v_i = (w_{i+1} - w_i) \bmod 2, \quad \text{pour } i = 1, 2, \ldots, |w| - 2.$$

Exemple 15. Soit $w = 001101110$. Alors

$$\begin{aligned} \Delta(w) &= (0-0)(1-0)(1-1)(0-1)(1-0)(1-1)(1-1)(0-1) \\ &= 01011001 \end{aligned}$$

On constate que $|\Delta(w)| = |w| - 1$.

Les suites de Rote complémentaires sont liées aux mots sturmiens par un théorème structurel (Rote, 1994) :

Théorème 3. (Rote, 1994) Un mot infini $\mathbf{w}$ est une suite de Rote complémentaire si et seulement si le mot infini $\Delta(\mathbf{w})$ est un mot sturmien.

Par exemple, considérons la suite de Rote complémentaire

$$x = 1100011000110011100111001110011000\cdots$$

et son mot sturmien associé

$$\begin{aligned} y &= \Delta(1100011000110011100111001110011000\cdots) \\ &= 0100101001010100101001010010100\cdots. \end{aligned}$$

L'idée de base consiste à utiliser le lien entre les palindromes et les antipalindromes des suites de Rote et des mots sturmiens. Dans un premier temps, nous énonçons quelques propriétés de l'opérateur Δ sans les démontrer.

Lemme 9. (Blondin Massé, 2008; Blondin Massé et al., 2009) Soient $u, v \in \Sigma^*$ tels que $|u|, |v| \geq 2$. Alors

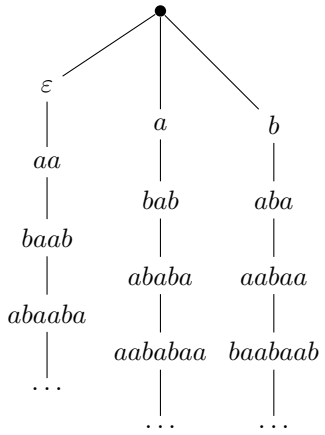


Figure 2.13: Arbre des palindromes facteurs du mot de Fibonacci. On voit qu'il contient exactement trois branches infinies, c'est-à-dire que chaque palindrome peut être étendu de façon unique.

(i) $\Delta(u) = \Delta(v)$ si et seulement si $v = u$ ou $v = \overline{u}$;

(ii) u est un palindrome ou un antipalindrome si et seulement si $\Delta(u)$ est un palindrome ;

(iii) u est un antipalindrome si et seulement si $\Delta(u)$ est un palindrome impair de lettre centrale 1.

Le nombre de mots de retour complets des mots sturmiens est connu depuis quelque temps déjà :

Théorème 4. (Justin et Vuillon, 2000; Vuillon, 2001) Un mot binaire $\mathbf{w}$ est sturmien si et seulement si tout facteur non vide u de $\mathbf{w}$ satisfait $|\mathrm{CRet}_{\mathbf{w}}(u)| = 2$.

Un avant-dernier lemme donne une idée plus précise de la structure des palindromes présents dans les mots sturmiens. Informellement, il indique que les palindromes présents dans un mot sturmien donné forme un arbre contenant exactement trois branches infinies (voir figure 2.13).

Lemme 10. (Blondin Massé, 2008; Blondin Massé et al., 2009) Soit $\mathbf{s}$ un mot sturmien et $p, q \in \mathrm{Pal}(\mathbf{s})$, où $|p| \geq |q|$. Supposons qu'il existe un mot non vide r tel que $r \prec p$ et $r \prec q$. Alors $q \prec p$. □

Finalement, nous énonçons et démontrons un dernier lemme utile pour le théorème 5.

Lemme 11. (Blondin Massé, 2008; Blondin Massé et al., 2009) Soit $\mathbf{r}$ un mot de Rote complémentaire et $u \in \mathrm{Pal}(\mathbf{r})$. Alors il existe un palindrome p et un antipalindrome q dont u est préfixe et tel que

$$\mathrm{CRet}_{\mathbf{r}}(\Delta(u)) = \{\Delta(p), \Delta(q)\}.$$

Démonstration. Nous savons du théorème 3 que $\Delta(\mathbf{r})$ est sturmien et du lemme 9 que $\Delta(u)$ est un palindrome. Par conséquent, $\Delta(u)$ possède deux mots de retour complet, par le théorème 4. De plus, comme $\Delta(\mathbf{r})$ est plein (tout mot sturmien étant plein), ces deux mots de retour sont également des palindromes.

Soient p et q les deux mots dont u est préfixe et tels que

$$\mathrm{CRet}_{\mathbf{r}}(\Delta(u)) = \{\Delta(p), \Delta(q)\}.$$

En vertu du lemme 9 (i), ces mots existent et sont uniques. Aussi, par le lemme 9 (ii), on a que p et q sont soient des palindromes ou des antipalindromes.

Nous montrons dans un premier temps que p et q ne peuvent pas être deux antipalindromes. Supposons le contraire, c'est-à-dire que $\Delta(p)$ et $\Delta(q)$ sont tous deux des palindromes de longueur impaire de lettre centrale 1. Par le lemme 10, on en déduit que $\Delta(p) \prec \Delta(q)$ ou $\Delta(q) \prec \Delta(p)$. Dans le premier cas, on obtient alors $|\Delta(q)|_{\Delta(u)} \geq 4$ alors que dans le deuxième, on trouve $|\Delta(p)|_{\Delta(u)} \geq 4$, contredisant le fait que $\Delta(p)$ et $\Delta(q)$ sont des mots de retour complets.

Il reste à vérifier que p et q ne sont pas tous deux des palindromes. Puisque $\mathbf{r}$ est récurrent (étant un cas particulier de codage de rotations), il existe $v \in \mathrm{Fact}(\mathbf{r})$ tel que

(1) u est préfixe de v ;

(2) $\overline{u}$ est un suffixe de v ;

(3) $|v|_u = |v|_{\overline{u}} = 1$,

c'est-à-dire que $\Delta(v)$ est un mot de retour complet de $\Delta(u)$ dans $\Delta(\mathbf{r})$. Or, v n'est pas un palindrome puisque u est préfixe et u est suffixe, de sorte que ce soit être un antipalindrome. □

Il ne reste plus qu'à conclure cette section.

Théorème 5. (Blondin Massé, 2008; Blondin Massé et al., 2009) Les mots de Rote complémentaires (c'est-à-dire tels que $\beta = 1/2$) sont pleins.

Démonstration. Soit $\mathbf{r}$ un mot de Rote, $u \in \mathrm{Pal}(\mathbf{r})$ et v un mot de retour complet de u dans $\mathbf{r}$. Il suffit de montrer que v est un palindrome.

Dans un premier temps, remarquons que $|v|_u = 2$, mais il est possible d'avoir $|v|_{\overline{u}} > 0$. Soit $n = |v|_{\overline{u}}$. Par le lemme 11, il existe un palindrome p et un antipalindrome q tels que $\Delta(p)$ et $\Delta(q)$ sont les deux mots de retour complet de $\Delta(u)$ dans $\mathbf{r}$, où u est préfixe de p et de q. Si $n = 0$, alors $v = p$ est un palindrome, tel que voulu. Sinon,

$$v = (q\overline{u}^{-1})(\overline{pu}^{-1})^n\overline{q}$$

et donc

$$\begin{aligned}\widetilde{v} &= \widetilde{\overline{q}}(\widetilde{\overline{u}^{-1}}\widetilde{\overline{p}})^n(\widetilde{\overline{u}^{-1}}\widetilde{q}) \\ &= q(\overline{u}^{-1}\overline{p})^n(\overline{u}^{-1}\overline{q}) \\ &= v,\end{aligned}$$

ce qui démontre que v est un palindrome. Ainsi, $\mathbf{r}$ est plein. □

2.9 Autres problèmes

Dans ce chapitre, nous nous sommes concentrés sur les codages de rotations obtenus par partition de l'intervalle $[0, 1)$ en deux sous-intervalles I_0 et I_1. Clairement, cette situation peut être généralisée en partitionnant l'intervalle $[0, 1)$ en k sous-intervalles I_0, I_1, …, I_{k-1} pour obtenir une suite symbolique sur un alphabet à k lettres.

En revanche, il existe de nombreux exemples de codages de rotations sur des alphabets de plus grande taille qui ne sont pas pleins. Un problème intéressant consisterait à déterminer quel jeu de paramètres α et $|I_i|$ (pour $i = 0, 1, 2, \ldots, k-1$) donnent effectivement des mots pleins.

Dans un même ordre d'idée, des travaux récents ont porté sur la notion de mots pleins en f-palindromes (Pelantová et Starosta, 2011). Il serait pertinent d'étudier l'occurrence de facteurs f-palindromiques dans les codages de rotations.

CHAPITRE III

MOTS PSEUDOSTANDARDS GÉNÉRALISÉS

De toutes les familles de mots infinis, les mots sturmiens constituent probablement la plus célèbre. Ces mots sur un alphabet binaire sont présents dans des domaines aussi variés que l'astronomie, les systèmes dynamiques, la théorie des nombres, la géométrie discrète et la cristallographie. Ils présentent plusieurs caractérisations remarquables.

Une d'entre elles concernent les mots sturmiens standards, c'est-à-dire ceux qui correspondent aux demi-droites discrètes commençant à l'origine (0,0) du plan $\mathbb{R}^2$. Par ailleurs, tout mot sturmien standard peut être obtenu à l'aide d'un mot directeur infini binaire quelconque et en appliquant la clôture palindromique successivement sur chaque mot obtenu.

L'objectif de ce chapitre est d'étudier une famille de mots introduits dans (de Luca et De Luca, 2006) appelés *mots pseudostandards généralisés*. Nous décrivons en particulier un algorithme permettant de les générer de façon efficace pour les alphabets binaires. Fait intéressant, la démonstration menant à cette algorithme repose sur la notion de *pseudopériode* : nous énonçons et démontrons une généralisation du célèbre théorème de Fine et Wilf qui s'avère très utile par la suite.

3.1 Clôture palindromique itérée

Soit $w = w_1 w_2 \cdots w_{n-1} w_n$ un mot sur un alphabet binaire. Rappelons que l'image miroir de w, notée $\widetilde{w}$, est donnée par $\widetilde{w} = w_n w_{n-1} \cdots w_2 w_1$ et qu'un palindrome est un mot

vérifiant l'équation $\widetilde{w} = w$. Dans ce chapitre, afin d'alléger la notation, nous dénotons également l'antimorphisme $\widetilde{\cdot}$ par R. Les palindromes correspondent donc exactement aux points fixes de R.

La *clôture palindromique à droite* (ou simplement *clôture palindromique*) de w, notée $w^{(+)}$ est donnée par

$$w^{(+)} = wR(p),$$

où $w = ps$ et $s = \mathrm{PLPS}(w)$ est le plus long palindrome suffixe de w. En d'autres termes, $w^{(+)}$ est le plus court palindrome ayant w comme préfixe.

Soit w un mot fini et a la dernière lettre de w. Alors la *clôture palindromique itérée* $\psi(w)$ *de* w est définie comme suit

$$\psi(w) = \begin{cases} \varepsilon & \text{si } |w| = 0, \\ (\psi(w')a)^{(+)} & \text{si } w = w'a \text{ pour } w' \text{ un mot et } a \text{ une lettre.} \end{cases}$$

Il est intéressant de remarquer que, par la définition même de clôture palindromique itérée, pour tout mot w et toute lettre a, $\psi(w)$ est un préfixe de $\psi(wa)$. Par conséquent, on étend naturellement la notion de clôture palindromique itérée à n'importe quel mot infini $\mathbf{w} = (\mathbf{w}[n])_{n\geq 1}$ comme suit :

$$\psi(\mathbf{w}) = \lim_{n\to\infty} \psi(\mathrm{Pref}_n(\mathbf{w})) = \lim_{n\to\infty} \psi(\mathbf{w}[1]\mathbf{w}[2]\cdots\mathbf{w}[n]).$$

On dit alors de $\mathbf{w}$ qu'il est le *mot directeur* (ou *suite directrice*) de $\psi(\mathbf{w})$. En outre, on sait de (De Luca, 1997) que ψ décrit une bijection entre l'ensemble des mots infinis sur l'alphabet $\{a, b\}$ qui ne sont pas ultimement périodiques et l'ensemble des mots sturmiens standards sur l'alphabet $\{a, b\}$. Remarquons aussi que si $\mathbf{w}$ est ultimement périodique, alors le mot $\psi(\mathbf{w})$ est périodique (et donc ne peut pas être sturmien) (Droubay, Justin et Pirillo, 2001).

Clairement, l'opérateur ψ est aussi bien défini pour les mots sur un alphabet à k lettres,

avec $k \geq 3$.

Exemple 16. Le mot de Fibonacci

$$\mathbf{f} = \psi((01)^\omega) = \underline{0}\,\underline{1}\,0\,\underline{0}\,1\,0\,\underline{1}\,0\,0\,1\,0\,\underline{0}\,1\,0\,1\,0\,0\,1\,0\,\underline{1}\cdots$$

est un mot sturmien standard de mot directeur $(01)^\omega$. Nous avons utilisé la notation usuelle dans l'exemple précédent en soulignant les lettres du mot directeur. Autrement dit, tout préfixe terminant juste avant une lettre soulignée est un palindrome.

3.2 Clôture pseudopalindromique itérée

Il semble naturel d'étendre les notions de clôture palindromique et de clôture palindromique itérée aux pseudopalindromes (de Luca et De Luca, 2006). Rappelons qu'un antimorphisme sur un alphabet $\mathcal{A}$ est une application $\vartheta : \mathcal{A}^* \to \mathcal{A}^*$ vérifiant $\vartheta(uv) = \vartheta(v)\vartheta(u)$ pour tout $u, v \in \mathcal{A}^*$. On dit d'un antimorphisme qu'il est involutif si $\vartheta^2 = \mathrm{Id}$. Il est facile de vérifier que tout antimorphisme involutif ϑ peut être décomposé en $\vartheta = \sigma \circ R = R \circ \sigma$, où σ est une involution sur l'alphabet $\mathcal{A}$.

Sur un alphabet binaire, il n'existe que deux involutions, soit Id et $\bar{\cdot}$ (l'échange de lettres). Il existe donc deux antimorphismes : R, l'opérateur image miroir, et $E = R \circ \bar{\cdot}$. L'antimorphisme est habituellement appelé *antimorphisme d'échange*. Dans la suite, nous définissons l'opérateur $\bar{\cdot}$ sur $\{R, E\}$ par $\overline{R} = E$ et $\overline{E} = R$.

Un mot w est appelé ϑ-palindrome s'il est fixé par ϑ, c'est-à-dire si $\vartheta(w) = w$. Clairement, les R-palindromes sont exactement les palindromes usuels. Lorsque l'alphabet est $\{A, C, G, T\}$ et que Θ est l'antimorphisme défini par $A \leftrightarrow T$ et $C \leftrightarrow G$, on retrouve l'involution préférée des biologistes et bio-informaticiens. La notion de clôture palindromique est naturellement étendue comme suit. Soit w un mot fini et ϑ un antimorphisme involutif sur un alphabet quelconque. Alors la *clôture ϑ-palindromique de w*, notée $w^{\oplus_\vartheta} = w\Theta(p)$, où $w = ps$ avec s le plus long ϑ-palindrome suffixe de w. Bref, la clôture ϑ-palindromique du mot w est le plus court ϑ-palindrome possédant w comme

préfixe.

Exemple 17. Sur l'alphabet $\{0,1\}$, puisque le plus long E-palindrome suffixe de $w = 0010$ est 10, on obtient $w^{\oplus_E} = 0010 \cdot E(00) = 001011$.

Remarquons que 001011 est un E-palindrome, aussi appelé antipalindrome, puisque $E(001011) = R(\overline{001011}) = R(110100) = 001011$.

On étend l'opérateur ψ à un opérateur ψ_ϑ comme suit. Soit w un mot fini et a une lettre. Alors

$$\psi_\vartheta(w) = \begin{cases} \varepsilon & \text{si } |w| = 0, \\ (\psi_\vartheta(w')a)^{\oplus_\vartheta} & \text{si } w = w'a \text{ pour un mot } w' \text{ et une lettre } a. \end{cases}$$

Lorsque $\mathbf{w}$ est un mot infini, comme $\psi_\vartheta(w)$ est préfixe de $\psi_\vartheta(wa)$, l'opérateur ψ_ϑ est aussi défini pour les mots infinis :

$$\psi_\vartheta(\mathbf{w}) = \lim_{n\to\infty} \psi_\vartheta(\mathrm{Pref}_n(\mathbf{w})) = \lim_{n\to\infty} \psi_\vartheta(\mathbf{w}[1]\,\mathbf{w}[2]\cdots\mathbf{w}[n]).$$

La famille de mots ainsi obtenus à l'aide de l'opérateur φ_ϑ est appelée *famille des mots ϑ-standards* ou simplement *famille de mots pseudostandards* quand l'antimorphisme ϑ n'est pas précisé. Cette famille de mots inclut entre autres les mots sturmiens et épisturmiens standards (de Luca et De Luca, 2006).

Exemple 18. Considérons l'alphabet $\mathcal{A} = \{0,1\}$. Alors

$$\begin{aligned} \psi_E(001) &= ((\psi_E(0)0)^{\oplus_E}1)^{\oplus_E} \\ &= (01\cdot 0)^{\oplus_E}1)^{\oplus_E} \\ &= (0101\cdot 1)^{\oplus_E} \\ &= 0101100101. \end{aligned}$$

3.3 Mots pseudostandards généralisés

Une famille de mots encore plus générale peut être obtenue en appliquant, chaque fois qu'une nouvelle lettre du mot directeur est ajoutée, une clôture pseudopalindromique différente.

Plus formellement, soit $\mathcal{I}$ l'ensemble des antimorphismes involutifs sur un alphabet $\mathcal{A}^*$ et $\mathcal{I}^\omega$ l'ensemble des suites infinies sur $\mathcal{I}$. Soit $\Theta = \vartheta_1\vartheta_2\vartheta_3\cdots \in \mathcal{I}^\omega$ et $^{\oplus_i}$ l'opérateur de clôture ϑ_i-palindromique pour tout entier $i \geq 1$. On définit récursivement l'opérateur ψ_Θ par

$$\psi_\Theta(w) = \begin{cases} \varepsilon & \text{si } |w| = 0, \\ (\psi_\Theta(w')a)^{\oplus_{|w|}} & \text{si } w = w'a \text{ pour un mot } w' \text{ et une lettre } a, \end{cases}$$

où a est une lettre et w un mot. L'opérateur ψ_Θ s'étend également aux mots infinis. Un mot $\psi_\Theta(w)$ est appelé *mot pseudostandard généralisé* et le couple (Θ, w) est appelé *bi-suite directrice de* $\psi_\Theta(w)$.

L'ensemble des mots pseudostandards généralisés contient plusieurs mots ou familles de mots célèbres :

1. Les mots sturmiens standards sont ceux donnés par $\Theta = R^\omega$ avec w sur un alphabet binaire ;
2. Les mots épisturmiens standards sont exactement ceux pour lesquels $\Theta = R^\omega$;
3. Les mots ϑ-standards sont obtenus en prenant $\Theta = \vartheta^\omega$;
4. Le mot de Thue-Morse est décrit par $\psi_{(ER)^\omega}(01^\omega)$ (de Luca et De Luca, 2006) ;

Par abus de notation, nous considérons bien définie l'expression $\psi_\Theta(w)$, que Θ et w soient finis ou infinis ou encore de longueurs différentes. Dans ce dernier cas, $\psi_\Theta(w)$ est le mot obtenu en prenant les préfixes maximums de même longueur de Θ et w.

Nous terminons cette section avec l'exemple du mot de Thue-Morse.

Exemple 19. Calculons les premiers préfixes du mot $\psi_{(ER)^\omega}(01^\omega)$:

$$\begin{aligned}
\psi_E(0) &= \breve{0}1 \\
\psi_{ER}(01) &= \breve{0}1\underline{1}0 \\
\psi_{ERE}(011) &= \breve{0}1\underline{1}0\breve{1}001 \\
\psi_{ERER}(0111) &= \breve{0}1\underline{1}0\breve{1}001\underline{1}0010110 \\
&\vdots
\end{aligned}$$

Nous conservons la notation de soulignement pour la clôture palindromique habituelle et nous indiquons que la clôture E-palindromique est identifiée à l'aide du symbole $\breve{}$.

3.4 Formule de Justin

D'un point de vue algorithmique, on est amené à se demander quelle est la façon la plus efficace de générer un mot pseudostandard généralisé. L'algorithme 1 décrit le pseudocode d'un algorithme permettant de générer le mot pseudostandard généralisé de bi-suite directrice (Θ, w).

Algorithme 1 Génération d'un mot pseudostandard généralisé

```
1: fonction GeneralizedPseudostandardWord(Θ, w)
2:     Entrée : Une suite d'antimorphismes involutifs Θ et un mot fini w
3:     Sortie : Le mot pseudostandard généralisé de bi-suite directrice finie (Θ, w)
4:     n ← min{|w|, |Θ|}
5:     u ← ε
6:     pour i ∈ {1, 2, ..., n} faire
7:         u ← u · w[i]
8:         Soit m la longueur du plus long Θ[i]-palindrome suffixe de u
9:         Soit p le préfixe de u de longueur |u| − m
10:        u ← u · Θ[i](p)
11:    fin pour
12:    retourner u
13: fin fonction
```

L'essentiel du temps d'exécution de l'algorithme se déroule à la ligne 8 où on calcule la longueur d'un plus long pseudopalindrome suffixe. Si on calcule, à chaque tour de boucle,

ce suffixe en question, le temps d'exécution peut être quadratique, voire cubique. Or, il est possible d'obtenir la longueur m directement sans avoir à inspecter à nouveau le mot entier.

L'idée consiste à étendre une formule récursive donnée par Justin pour la clôture palindromique itérée (Justin, 2005).

Lemme 12. (Justin, 2005) Soit $w \in \mathcal{A}^*$ et $a \in \mathcal{A}$. Si w contient la lettre a, alors nous pouvons écrire $w = v_1 a v_2$ où v_2 ne contient pas la lettre a. Alors

$$\psi(wa) = \begin{cases} \psi(w) \cdot a \cdot \psi(w) & \text{si } a \text{ n'apparaît pas dans } w, \\ \psi(w) \cdot \psi(v_1)^{-1} \cdot \psi(w) & \text{si } a \text{ apparaît dans } w. \end{cases}$$

Exemple 20. Illustrons le lemme 12 sur le mot directeur $w = 01001$. On obtient

$$\begin{aligned} \psi(0) &= 0^{(+)} = 0 = \varepsilon \cdot 0 \cdot \varepsilon = \psi(\varepsilon) \cdot 0 \cdot \psi(\varepsilon) \\ \psi(01) &= (01)^{(+)} = 010 = 0 \cdot 1 \cdot 0 = \psi(0) \cdot 1 \cdot \psi(0) \\ \psi(010) &= (0100)^{(+)} = 010010 = 010 \cdot \varepsilon \cdot 010 = \psi(01) \cdot \psi(\varepsilon)^{-1} \cdot \psi(01) \\ \psi(0100) &= (0100100)^{(+)} = 010010010 = 010010 \cdot (010)^{-1} \cdot 010010 \\ &= \psi(010) \cdot \psi(01)^{-1} \cdot \psi(010) \\ \psi(01001) &= (0100100101)^{(+)} = 01001001010010010 \\ &= 010010010 \cdot (0)^{-1} \cdot 010010010 \\ &= \psi(0100) \cdot \psi(0)^{-1} \cdot \psi(0100) \end{aligned}$$

La formule récursive du lemme 12 révèle ainsi un algorithme efficace permettant de calculer la clôture palindromique itérée d'un mot w sans avoir à calculer à chaque étape le plus long palindrome suffixe (voir algorithme 2).

Cet algorithme est linéaire et optimal. En outre, il n'utilise comme espace supplémentaire qu'un tableau associatif LastLength de taille proportionnelle à celle de l'alphabet $\mathcal{A}$. À noter que le pseudocode présenté dans l'algorithme 2 a été inclus dans Sage en 2010

Algorithme 2 Calcul efficace de la clôture palindromique itérée d'un mot

```
1: fonction ClôturePalindromiqueItérée(w)
2:     Entrée : Un mot w sur un alphabet quelconque
3:     Sortie : La clôture palindromique itérée ψ(w) de w
4:     n ← |w|
5:     u ← ε
6:     pour i ∈ {1, 2, ..., n} faire
7:         LastLength(w[i]) = |u|
8:         si w[i] n'apparaît pas dans Pref_{i−1}(w) alors
9:             u ← u · w[i] · u
10:        sinon
11:            s ← |u| − LastLength(w[i])
12:            u ← u · Suff_s(u)
13:        fin si
14:    fin pour
15:    retourner u
16: fin fonction
```

(voir l'annexe A.1, à partir page 154, où les versions non récursive et récursive sont incluses).

Il n'est pas trivial d'étendre le lemme 12 au cas des mots pseudostandards généralisés, car différents antimorphismes involutifs peuvent être appliqués. Par contre, l'énumération exhaustive suggère que les longueurs des pseudopalindromes obtenus par clôture itérée suivent des règles bien précises et qu'il suffit d'inspecter le mot directeur localement afin de deviner le mot obtenu pour l'itération suivante. Nous présentons ces observations un peu plus bas. En revanche, il est nécessaire d'introduire d'abord le concept de pseudopériodicité.

3.5 Pseudopériodicité

L'étude de la clôture palindromique itérée est intimement reliée à la notion de périodicité (De Luca, 1997). Plus précisément, on constate que la superposition de palindromes induit des périodes locales dans les mots. Pour expliquer la structure des mots pseudostandards généralisés, il semble donc naturel de considérer la pseudopériodicité induite par la superposition de pseudopalindromes.

Un des théorèmes les plus rencontrés en combinatoire des mots finis est celui de Fine et Wilf, qui décrit exactement sous quelles conditions l'existence de deux périodes dans un même mot en donne une troisième plus petite. Rappelons qu'un entier positif p est une période d'un mot w si $w[i] = w[i+p]$ pour tout entier i tel que $1 \leq i \leq |w| - p$.

Théorème 6. (Lothaire, 1983) Soit w un mot et p, q deux périodes de w. Si $|w| \geq p + q - \text{pgcd}(p, q)$, alors $\text{pgcd}(p, q)$ est aussi une période de w.

Il existe plusieurs généralisations au théorème 6. Nous présentons ici une extension pour les pseudopériodes d'un mot binaire.

Définition 5. Soit w un mot fini et σ une permutation involutive d'un alphabet quelconque. Un entier positive p est appelé *σ-période* de w si, pour tout entier i tel que $1 \leq i \leq |w| - p$, alors $w[i] = \sigma(w[i+p])$.

Exemple 21. Le mot 011011 a la R-période 3 et la E-période 5, puisque $0111011 = 011 \cdot R(0)R(1)R(1)$ et $01101 \cdot E(0)$.

Comme l'illustre l'exemple 21, la restriction d'un antimorphisme involutif aux lettres de l'alphabet est une permutation involutive. Notons également que les R-périodes sont exactement les périodes usuelles pour les mots. Le lecteur vérifie également que si p est une σ-période d'un mot w, où σ est une permutation involutive, alors $2p$ est une R-période de w.

Une définition de *pseudopériodicité* a déjà été introduite dans la littérature (Czeizler et al., 2009). En revanche, les deux notions ne sont pas équivalentes. Plus précisément, les auteurs disent d'un entier positif p qu'il est une σ-période d'un mot w si w peut être écrit comme un produit d'éléments de l'ensemble $\{u,\sigma(u)\}$, où u est un mot de longueur p. Par exemple, le nombre 2 est une σ-période du mot $w = 12323212$ pour $\sigma : 1 \mapsto 3, 2 \mapsto 2, 3 \mapsto 1$ puisque $w = u\sigma(u)\sigma(u)u$, pour $u = 12$. Par opposition, la définition 5 impose une alternance du mot u avec son complément et ne pose pas de problème même si p ne divise pas $|w|$.

Il a déjà été observé que la superposition de palindromes implique une périodicité locale (De Luca, 1997). Ces propriétés se généralisent naturellement à la superposition de pseudopalindromes. Le lemme suivant est pratique lorsqu'on manipule des pseudopalindromes pseudopériodiques.

Lemme 13. Soit w un ϑ-palindrome, p une E-période de w, où $p < |w|$. Soit u le suffixe de w de longueur p et $i = \lfloor |w|/p \rfloor$. Alors

(i) $w(\overline{u}u)^{-j}$ est un ϑ-palindrome pour tout entier j tel que $0 \leq j \leq (i-1)/2$;

(ii) $wu^{-1}(\overline{u}u)^{-j}$ est un $\overline{\vartheta}$-palindrome pour tout entier j tel que $0 \leq j \leq (i-2)/2$.

(iii) $w(\overline{u}u)^{j}$ est un ϑ-palindrome pour tout entier $j \geq 0$;

(iv) $w(\overline{u}u)^{j}\overline{u}$ est un $\overline{\vartheta}$-palindrome pour tout entier $j \geq 0$;

Démonstration. Puisque p est une E-période de w, il existe un mot x, un mot non vide y et un entier positif k tel que $|xy| = p$ et $w \in \{(xy\overline{xy})^k x, (xy\overline{xy})^{k-1}xy\overline{x}\}$.

Nous considérons d'abord le cas $w = (xy\overline{xy})^k x$. Alors $u = \overline{y}x$ et $k = i/2$. De plus, puisque w est un ϑ-palindrome, on obtient

$$(xy\overline{xy})^k x = w = \vartheta(w) = (\vartheta(x)\vartheta(\overline{y})\vartheta(\overline{x})\vartheta(y))^k \vartheta(x).$$

On en déduit que $x = \vartheta(x) = \overline{\vartheta}(\overline{x})$ et $y = \vartheta(\overline{y}) = \overline{\vartheta}(y)$. Supposons maintenant que j est un entier satisfaisant $0 \leq j \leq (i-1)/2$. Alors $w(\overline{u}u)^{-j} = (xy\overline{xy})^{k-j}x$, qui est bien un ϑ-palindrome puisque $x = \vartheta(x)$, $\overline{x} = \vartheta(\overline{x})$, $y = \vartheta(\overline{y})$ et $\overline{y} = \vartheta(y)$. De la même façon, si j satisfait $0 \leq j \leq (i-2)/2$, alors $wu^{-1}(\overline{u}u)^{-j} = (xy\overline{xy})^{k-j-1}xy\overline{x}$, qui est dans ce cas un $\overline{\vartheta}$-palindrome, puisque $x = \overline{\vartheta}(\overline{x})$, $\overline{x} = \overline{\vartheta}(x)$, $y = \overline{\vartheta}(y)$ et $\overline{y} = \overline{\vartheta}(\overline{y})$.

Il reste à vérifier le cas $w = (xy\overline{xy})^{k-1}xy\overline{x}$. Alors $u = y\overline{x}$ et $k = (i+1)/2$. Le fait que w soit un ϑ-palindrome entraîne

$$(xy\overline{xy})^{k-1}xy\overline{x} = w = \vartheta(w) = (\vartheta(\overline{x})\vartheta(y)\vartheta(x)\vartheta(\overline{y}))^{k-1}\vartheta(\overline{x})\vartheta(y)\vartheta(x).$$

Par conséquent, $x = \vartheta(\overline{x}) = \overline{\vartheta}(x)$ et $y = \vartheta(y) = \overline{\vartheta}(\overline{y})$. Soit j un entier qui satisfait les inégalités $0 \leq j \leq (i-1)/2$. Alors $w(\overline{u}u)^{-j} = (xy\overline{xy})^{k-j-1}xy\overline{x}$, qui est un ϑ-palindrome

puisque $x = \vartheta(\overline{x})$, $\overline{x} = \vartheta(x)$, $y = \vartheta(\overline{y})$ et $\overline{y} = \vartheta(y)$. Finalement, supposons que j soit un entier tel que $0 \leq j \leq (i-2)/2$. On trouve alors $wu^{-1}(\overline{u}u)^{-j} = (xy\overline{xy})^{k-j-1}x$, qui est un $\overline{\vartheta}$-palindrome en vertu des égalités $x = \overline{\vartheta}(x)$, $\overline{x} = \overline{\vartheta}(\overline{x})$, $y = \overline{\vartheta}(\overline{y})$ et $\overline{y} = \overline{\vartheta}(y)$.

Les idées pour (iii) et (iv) sont semblables, mais le ϑ-palindrome est étendu plutôt que coupé. □

Exemple 22. Le mot $w = 0100110110010$ est un R-palindrome et possède la E-période 5 puisque

$$w = 01001 \cdot 10110 \cdot 010 = 01001 \cdot \overline{01001} \cdot 010.$$

Le lemme 13 indique qu'on obtient des R-palindromes et des E-palindromes alternés en supprimant des suffixes dont la longueur est un multiple de la période. En effet, les mots

$$w(10010)^{-1} = 01001 \cdot 101 \quad \text{et} \quad w(01101 \cdot 10010)^{-1} = 010$$

sont respectivement des E-palindromes et des R-palindromes.

Avant de démontrer le théorème principal de cette section, nous présentons maintenant trois autres lemmes qui établissent des relations entre pseudopalindromes et pseudopériodicité.

Lemme 14. Soit u un mot fini, p un ϑ_1-palindrome et q un ϑ_2-palindrome, où ϑ_1 et ϑ_2 sont des antimorphismes involutifs, tels que $pu = q$. Alors $|u|$ est une $(\vartheta_1 \circ \vartheta_2)$-période de q.

Démonstration. La situation est illustrée à la figure 3.1. Soit i un entier satisfaisant $1 \leq i \leq |p|$. Puisque p est un ϑ_1-palindrome préfixe de q, on obtient $q[i] = \vartheta_1(q[|p| + 1 - i])$. Or q est un ϑ_2-palindrome, ce qui entraîne

$$\begin{aligned} q[i] &= \vartheta_1(q[|p| + 1 - i]) \\ &= (\vartheta_2 \circ \vartheta_1)(q[|q| + 1 - |p| - 1 + i]) \\ &= (\vartheta_1 \circ \vartheta_2)(q[|q| - |p| + i]) \end{aligned}$$

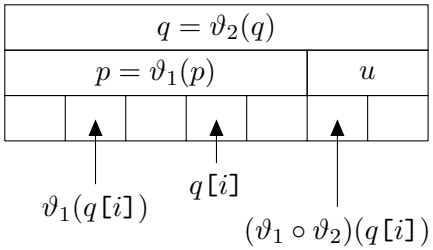


Figure 3.1: Pseudopériode induite dans deux pseudopalindromes superposés.

$$= \quad (\vartheta_1 \circ \vartheta_2)(q[i + |u|]),$$

tel que voulu.

Ce ne sont pas toutes les pseudopériodes imaginables qui peuvent coexister dans un même mot :

Lemme 15. Soit w un mot fini, p une ϑ_1-période de w et q une ϑ_2-période de w. Supposons que $|w| > p > q$, q divise p et posons $p = mq$ pour un certain entier $m \geq 2$. Alors une des deux conditions suivantes est vérifiée :

(i) $\vartheta_1 = R$ et m est pair ;

(ii) $\vartheta_1 = \vartheta_2$ et m est impair.

Démonstration. Soit i un entier tel que $1 \leq i \leq |w| - p$. Alors

$$R(w[i]) = w[i] = \vartheta_1(w[i + p]) = \vartheta_1(w[i + mq]) = (\vartheta_2^m \circ \vartheta_1)(w[i]),$$

de sorte que $R = \vartheta_2^m \circ \vartheta_1$. Par conséquent, si m est pair, alors $\vartheta_1 = R$ et si m est impair, alors $R = \vartheta_1 \circ \vartheta_2$ et donc $\vartheta_1 = \vartheta_2$. □

Le lemme suivant est une extension simple du lemme 8.1.3 de Lothaire pour les pseudopériodes (Lothaire, 1983).

Lemme 16. Soit w un mot fini et v un facteur de w. Supposons que p est une ϑ_1-période de w telle que $|v| > p$ et q est une ϑ_2-période de v telle que q divise p. Alors q est une ϑ_2-période de w.

Démonstration. Nous considérons d'abord le cas $q = p$. Alors $q = p$ est à la fois une ϑ_1-période et une ϑ_2-période de v, avec $|v| > p$. Ceci entraîne que $\vartheta_1 = \vartheta_2$ et le lemme est démontré.

Il reste à vérifier le cas $q < p$. Alors il existe un entier $m \geq 2$ tel que $p = qm$. Soit k un entier satisfaisant $1 \leq k \leq |w|$, tel que $v = w[k]w[k+1]\cdots w[k+|v|-1]$. Soit $K = \{k,k+1,\ldots,k+|v|-1\}$ l'ensemble des incides d'une occurrence de v dans w. De plus, soit i un entier vérifiant $1 \leq i \leq |w| - q$. Puisque $|v| > p$, alors il existe un entier $i' \in K$ tel que $i' \equiv i \bmod p$. Par conséquent, comme p est une ϑ_1-période de w, on a $w[i'] = \vartheta_1^{|\ell|}(w[i])$, où ℓ est l'entier défini par $i' - i = p\ell$. Puisque ϑ_1 est involutif, on en déduit que $w[i] = \vartheta_1^{|\ell|}(w[i'])$.

Soit $j = i+q$. Un raisonnement semblable à celui du paragraphe précédent nous mène à la conclusion qu'il existe un entier $j' \in K$ vérifiant $j' \equiv j \bmod p$. En particulier, on peut choisir j' de sorte que $j' \in \{i'+q, i'+q-p\}$. En effet, on a $i'+q \equiv i'+q-p \equiv i+q \bmod p$ et au moins une valeur parmi $i'+q$ et $i'+q-p$ doit être contenue dans K (puisque $|v| > p$ et $i' \in K$). Ainsi, on peut écrire $j' - j = p\ell'$ avec $\ell' \in \{\ell-1, \ell\}$ et donc $w[j'] = \vartheta_1^{|\ell'|}(w[j])$.

Pour terminer la démonstration, il reste à considérer deux cas : $\ell' = \ell$ ou $\ell' = \ell - 1$. Supposons d'abord que $\ell' = \ell$, ce qui entraîne que $j' = i'+q$. Puisque q est une ϑ_2-période de v, on trouve alors $w[i'] = \vartheta_2(w[j'])$. Il s'en suit que $w[j] = (\vartheta_1^{|2\ell|} \circ \vartheta_2)(w[i]) = \vartheta_2(w[i])$. Supposons maintenant que $\ell' = \ell - 1$ et donc $j' = i' + q - p$. Rappelons que $p = qm$. Alors $j' = i' + (1-m)q$ de sorte que $w[i'] = \vartheta_2^{|1-m|}(w[j'])$. Ceci entraîne que $w[j] = (\vartheta_1^{|2\ell-1|} \circ \vartheta_2^{|1-m|})(w[i]) = (\vartheta_1 \circ \vartheta_2^{|1-m|})(w[i])$. On sait du lemme 15 que soit $\vartheta_1 = R$ et m est pair, soit $\vartheta_1 = \vartheta_2$ et m est impair. Dans les deux cas, on obtient que $w[j] = \vartheta_2(w[i])$. En conclusion, on a démontré que $w[i+q] = \vartheta_2(w[i])$ pour n'importe quel entier i tel que $1 \leq i \leq |v| - q$, ce qui revient à dire que q est une ϑ_2-période de w. □

Nous sommes maintenant en mesure de démontrer une généralisation du théorème de

Fine et Wilf :

Théorème 7. Soit w un mot fini. Soit p une ϑ_1-période de w et q une ϑ_2-période de w, avec $(\vartheta_1,\vartheta_2) \neq (R,R)$. Si $|w| \geq p+q$, alors pgcd(p,q) est une E-période de w.

Démonstration. Soit $g = \text{pgcd}(p,q)$ et $I = \{1,2,\ldots,|w|\}$. En vertu de l'identité de Bezout, on sait que le plus grand commun diviseur de deux entiers s'écrit comme combinaison linéaire des deux entiers en question, c'est-à-dire qu'il existe deux entiers x et y tels que

$$g = xp - yq. \tag{3.1}$$

Comme il existe une infinité de valeurs possibles de x et y vérifiant l'équation (3.1) et que ces valeurs forment un espace vectoriel, on peut supposer sans perte de généralité que $x,y \geq 1$. Soit i un entier satisfaisant $1 \leq i \leq |w| - g$. On doit montrer que $w[i] = E(w[i+g])$. Posons $k = x+y$, $1 \leq a \leq i$ et soit $J = \{a,a+1,\ldots,a+p+q-1\} \subseteq I$ un sous-ensemble de I qui contient à la fois i et $i+g$ et tel que $\text{Card}(J) = p+q$, où $a \in I$. Nous construisons deux suites $d_1,d_2,\ldots,d_k$ et $i_0,i_1,\ldots,i_k$ comme suit. Posons $i_0 = i$ et, pour $j = 0,1,\ldots,k-1$, on définit :

$$d_{j+1} = \begin{cases} p & \text{si } i_j + p \in J, \\ -q & \text{si } i_j - q \in J \end{cases} \quad \text{et} \quad i_{j+1} = i_j + d_{j+1}.$$

Nous montrons dans l'ordre les propriétés suivantes :

(i) $i_j \in J$ pour $j = 0,1,\ldots,k$,

(ii) La suite $(d_j)_{j\in\{1,2,\ldots,k\}}$ est bien définie,

(iii) $i_k = i+g$ et

(iv) g est une E-période de w.

(i) Par définition de l'ensemble J, on a que $i_0 = i \in J$. En utilisant un argument par contradiction, supposons qu'il existe un entier $j \in \{0,1,\ldots,k-1\}$ tel que $i_j \in J$, mais $i_{j+1} \notin J$, c'est-à-dire que $i_j + p \notin J$ et $i_j - q \notin J$. Puisque J est un ensemble connexe par rapport à la topologie discrète des nombres naturels, ceci entraînerait que

$|(i_j+p)-(i_j-q)| = p+q > \mathrm{Card}(J)$, ce qui contredirait le fait que J contient exactement $p+q$ éléments.

(ii) Il suffit de montrer que les conditions $i_j + p \in J$ et $i_j - q \in J$ sont mutuellement exclusives pour $j = 0,1,\ldots,k$, c'est-à-dire qu'elles ne peuvent être vérifiées simultanément. En procédant encore une fois par contradiction, supposons qu'il existe un tel j. Alors $|(i_j + p) - (i_j - q)| = p + q < \mathrm{Card}(J)$, ce qui est absurde.

(iii) Jusqu'à maintenant, nous avons démontré que la suite $i_0, i_1, \ldots, i_k$ est contenue dans J et donc dans I. Pour montrer que g est une E-période de w, nous devons nous assurer que $i_k = i + g$ ou, de façon équivalente, que le mot $d = d_1 d_2 \cdots d_k$ a exactement x occurrences de p et y occurrences de $-q$. Nous utilisons encore une fois un argument par contradiction. Sans perte de généralité, supposons que p apparaît plus de x fois dans le préfixe $d_1 d_2 \cdots d_j d_{j+1}$ de d. Alors $i_{j+1} = i_j + d_{j+1} = i_j + p$. De plus, l'équation (3.1) implique que $i_j = i + xp - y'q$ pour un certain entier non négatif $y' < y$. En revanche, on a $i + g = i + xp - yq$. Or, $i_j - q = i + xp - (y' + 1)q$ et, puisque $y' < y' + 1 \leq y$, on en déduit que $i + g \leq i_j - q \leq i_j$. Par conséquent, $i_j - q \in J$, ce qui contredit le fait que la suite $d_1 d_2 \cdots d_k$ est définie de façon unique.

(iv) Pour conclure, posons $\vartheta = \vartheta_1^x \circ \vartheta_2^y \in \{R,E\}$. Puisque la suite $i_0 = i$, i_1, …, $i_k = i + g$ est unique, est contenue dans I, commence par i et termine par $i + g$, il suit du raisonnement ci-haut que $w[i] = \vartheta(w[i + g])$ pour tout $i \in I$, c'est-à-dire que g est une ϑ-période de w. Supposons par contradiction que $\vartheta = R$. Comme g divise à la fois p et q, ceci entraîne que p et q sont toutes deux des R-périodes de w. Mais par hypothèse, au moins p ou q est une E-période de w, ce qui est absurde. Ainsi, g est une E-période de w, tel que voulu. □

Exemple 23. Vérifions s'il est possible de construire un mot w de longueur 9 sur l'alphabet {0,1} commençant par la lettre 0, admettant 4 comme R-période et 3 comme E-période. La figure 3.2 montre qu'on a nécessairement $w = (01)^4 0$. En effet, on a que $w[1] = 0$. De plus, puisque w admet 4 comme R-période, on trouve $w[1] = w[5] = w[9] = 0$ et, puisque w admet 3 comme E-période, on déduit $w[4] = E(w[1]) =$

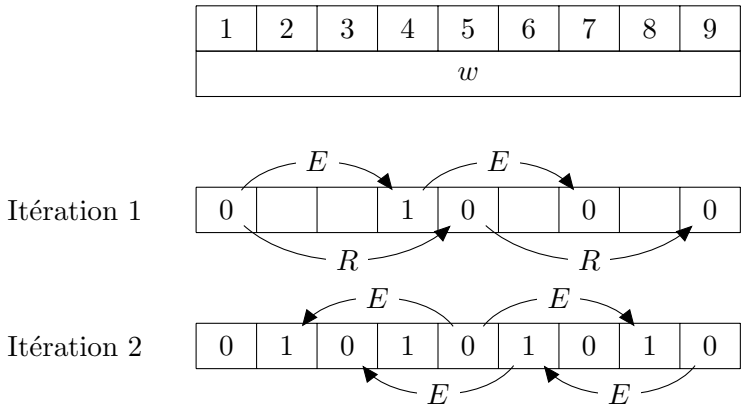


Figure 3.2: Illustration du théorème 7 sur un mot de longueur 9 commençant par 0 et admettant la R-période 4 et la E-période 3. Après deux itérations, on arrive à remplir toutes les cases avec des lettres pour obtenir le mot $w = (01)^4 0$.

$E(0) = 1$ et $w[7] = E(w[4]) = E(1) = 0$. Les nouvelles lettres connues et le fait que w admette 3 comme E-période nous permettent de calculer les quatre dernières lettres manquantes : $w[2] = w[8] = E(w[5]) = E(0) = 1$, $w[6] = E(w[9]) = E(0) = 1$ et $w[3] = E(w[6]) = E(1) = 0$. Effectivement, le mot $w = (01)^4 0$ admet 4 comme R-période et 3 comme E-période. En outre, en vertu du théorème 7, w admet 1 comme E-période, ce qui est vérifié ici.

Remarque 2. La borne $|w| \geq p + q$ du théorème 7 est optimale. En effet, considérons le mot $w = 000111$. Le lecteur constate facilement que les nombres 3 et 4 sont tous deux des E-périodes de w. En revanche, pgcd(3,4) = 1 n'est pas une E-période de w et $|w| = 6 < 7 = 3 + 4$.

Remarque 3. Notons que le théorème 7 suppose l'existence d'un mot w admettant simultanément une ϑ_1-période et une ϑ_2-période, mais rien ne garantit qu'un tel mot existe. Par exemple, on peut démontrer qu'il n'existe aucun mot de longueur 7 qui admet les E-périodes 3 et 4. Si c'était le cas, alors le théorème 7 s'appliquerait et donc w admetterait 1 comme E-période et donc devrait être de la forme $w = (\alpha\overline{\alpha})^3\alpha$ pour une certaine lettre α. Or, w admet la E-période 4, ce qui entraîne que $\alpha = w[1] = E(w[5]) = E(\alpha) = \overline{\alpha}$, ce qui est absurde.

3.6 Forme normalisée

Lorsqu'on considère la clôture palindromique itérée usuelle, on a le fait suivant :

Proposition 10. Soit $u = \psi(w)$. Alors u' est un préfixe palindrome de u si et seulement s'il existe un préfixe w' de w tel que $u' = \psi(w')$.

Démonstration. ($\Leftarrow$) Découle directement de la définition de clôture palindromique et de clôture palindromique itérée.

($\Rightarrow$) On procède par l'absurde, c'est-à-dire qu'on suppose qu'il existe deux préfixes consécutifs w' et w'' de w tels qu'il existe un palindrome préfixe p entre les deux palindromes préfixes $u' = \psi(w')$ et $u'' = \psi(w'')$. Ceci contredit le fait que u'' est le plus court palindrome ayant $u\alpha$ comme préfixe, où α est la lettre qu'on ajoute avant d'appliquer la clôture palindromique. □

La proposition 10 nous indique que les palindromes préfixes de $u = \psi(w)$ sont exactement ceux obtenus par clôture palindromique. En d'autres termes, aucun palindrome préfixe n'est omis par la clôture palindromique itérée. Or, cette propriété n'est pas vérifiée lorsqu'on permet d'utiliser les clôtures R-palindromique et E-palindromique en même temps.

Exemple 24. Calculons les pseudopalindromes préfixes du mot $w = \psi_{RERE}(0011)$. On trouve

$$\begin{aligned} \psi_R(0) &= 0 \\ \psi_{RE}(00) &= 0011 \\ \psi_{RER}(001) &= 0011100 \\ \psi_{RERE}(0011) &= 00111001100011 \end{aligned}$$

et on voit que le palindrome 00 n'apparaît pas dans les itérations successives. De la

même façon, calculons les préfixes de $u = \psi_{RRE}(011)$:

$$\begin{aligned} \psi_R(0) &= 0 \\ \psi_{RR}(01) &= 010 \\ \psi_{RRE}(011) &= 0101. \end{aligned}$$

On remarque dans ce cas que le E-palindrome 01 ne se trouve pas non plus dans les pseudopalindromes préfixes obtenus.

Bien qu'il existe une infinité d'exemples de bi-suite directrice ne décrivant pas tous les pseudopalindromes préfixes, il est toujours possible de réécrire la bi-suite de sorte qu'ils y soient tous décrits. Plus précisément, nous montrons dans cette section qu'il est toujours possible de réécrire une bi-suite directrice sous une forme « normale ».

Définition 6. Une bi-suite directrice (finie ou infinie) (Θ, w) est dite *normalisée* ou *sous forme normale* si pour tout pseudopalindrome préfixe u' de $\psi(w)$, il existe un préfixe w' de w tel que $u' = \psi(w')$. Tout pseudopalindrome qui ne vérifie pas cette condition est dit *manqué* par $\psi(w)$.

Le lemme suivant découle directement de la définition de pseudopalindrome manqué :

Lemme 17. Soit (Θ, w) une bi-suite directrice finie. Si (Θ, w) est sous forme normale et que $(\Theta\tau, wa)$ ne l'est pas, où $a \in \mathcal{A}$ et $\tau \in \{R, E\}$, alors tout ϑ-palindrome préfixe manqué p est tel que $|\psi_\Theta(w)| < |p| < |\psi_{\Theta\tau}(wa)|$.

Démonstration. Trivial. □

Lemme 18. Les plus courts préfixes d'une bi-suite normalisée contenant les deux lettres distinctes de $\mathcal{A}$ sont de la forme $(R^{i+1}, a^i\overline{a})$ pour $i \geq 2$ ou $(R^iE, a^i\overline{a})$ pour $i \geq 1$, où $a \in \{0, 1\}$.

Démonstration. Par inspection simple. On observe en particulier qu'une bi-suite normalisée ne peut commencer avec l'antimorphisme E. Les autres cas sont faciles à vérifier. □

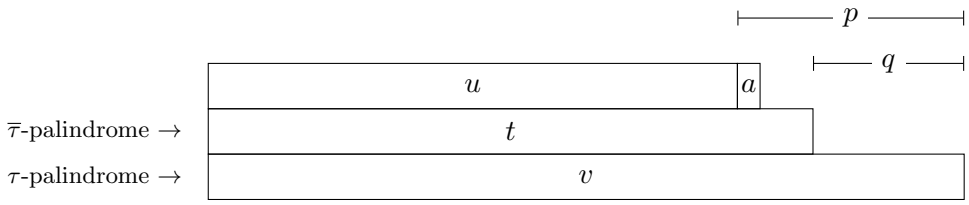


Figure 3.3: Illustration de la démonstration du lemme 19.

Les mots pseudostandards généralisés admettent des périodes à différentes échelles. Le lemme qui suit décrit les pseudopériodes induites par des bi-suites qui ne sont pas sous forme normale.

Lemme 19. Soit (Θ, w) une bi-suite finie normalisée. Supposons que $(\Theta\tau, wa)$ ne soit pas normalisée, où $\tau \in \{R, E\}$ et $a \in \mathcal{A}$. Soit $u = \psi_\Theta(w)$, $v = \psi_{\Theta\tau}(wa)$ et t un pseudopalindrome préfixe manqué par $(\Theta\tau, wa)$. Finalement, soit $p = |v|-|u|$, $q = |v|-|t|$ et $g = \text{pgcd}(p, q)$.

(i) Si $|v| \geq p + q$, alors g est une E-période de v, $p = 2g$, $q = g$ et u est un τ-palindrome ;

(ii) Sinon, $\tau = E$ et q est une ϑ-période de t, où ϑ est le dernier antimorphisme de la suite Θ.

Démonstration. La situation est illustrée à la figure 3.3. Remarquons que t est un $\overline{\tau}$-palindrome, sinon, v ne serait pas le plus court τ-palindrome admettant ua comme préfixe. De plus, il suit du lemme 14 que p est une $(\vartheta \circ \tau)$-période de v, puisque v est un τ-palindrome et u est un ϑ-palindrome. De façon semblable, q est une $(\tau \circ \overline{\tau})$-période, c'est-à-dire une E-période de v.

(i) Premièrement, supposons que $|v| \geq p + q$. Alors le théorème 7 s'applique de sorte que $g = \text{pgcd}(p, q)$ est une E-période de v. Par le lemme 13, on conclut que le préfixe y de v de longueur $|v| - 2g$ est un τ-palindrome. Puisque aucun τ-palindrome n'apparaît entre u et v (autrement v ne serait pas le plus court τ-palindrome ayant ua comme préfixe), on a nécessairement $|y| \leq |u|$, c'est-à-dire que $p = |v| - |u| \leq 2g$. En combinant le fait que $0 < q < p \leq 2g$, que $|u| < |t| < |v|$ et que g divise à la fois $q = |v| - |t|$ et

$p - q = |t| - |u|$, on en conclut que $p = 2g$ et donc $q = g$. En particulier, $u = y$ est un τ-palindrome.

(ii) Il reste à considérer le cas $|v| < p + q$. Notons que, par définition de clôture pseudopalindromique, on a $|v| \leq 2|u| + 2$, avec $|v| = 2|u| + 2$ si et seulement si v est un E-palindrome. Nous montrons d'abord que $|v| = 2|u| + 2$ en procédant par l'absurde. Supposons donc, au contraire, que $|v| \leq 2|u| + 1$. Alors

$$\begin{aligned} |v| &< p + q \\ &= |v| - |u| + |v| - |t| \\ &\leq 2|u| + 1 - |t| + |v| - |u| \\ &= |u| + 1 - |t| + |v| \\ &\leq |u| + 1 - |u| - 1 + |v| \\ &\leq |v|, \end{aligned}$$

ce qui est absurde (la dernière inégalité suit de l'inégalité $|t| \geq |u| + 1$). Ainsi, $|v| = 2|u| + 2$. Ceci entraîne que $\tau = E$ et t est un R-palindrome (puisque $\overline{\tau} = R$). En nous basant sur des inégalités semblables, on montre que $|t| = |u| + 1$. Il ne reste qu'à appliquer le lemme 14 pour conclure que q est une ϑ_n-période de t. □

Lorsqu'une suite n'est pas normalisée, on peut également déduire certaines informations sur les antimorphismes impliqués.

Lemme 20. Soit (Θ, w) une bi-suite normalisée de longueur n, $\vartheta_n = \Theta[n]$ et supposons que $(\Theta\tau, wa)$ ne soit pas normalisée, où $\tau \in \{R, E\}$ et $a \in \mathcal{A}$. Alors $\vartheta_n = \tau$ ou $(\Theta\tau, wa) = (R^{n-1}E, a^n)$.

Démonstration. Soit $u = \psi_\Theta(w)$, $v = \psi_{\Theta\tau}(wa)$ et t un pseudopalindrome préfixe manqué par $(\Theta\tau, wa)$. La situation est illustrée à la figure 3.4. Aussi, soit $p = |v| - |u|$, $q = |v| - |t|$ et $g = \text{pgcd}(p, q)$. Si $|v| \geq p + q$, alors par le lemme 19, On a $\vartheta_n = \tau$, tel que voulu. Sinon, supposons par contradiction que $\vartheta_n \neq \tau$. Encore une fois par le lemme 19,

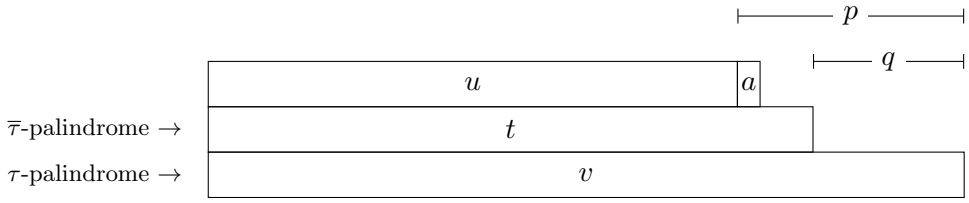


Figure 3.4: Illustration de la démonstration du lemme 20.

on en déduit que $\tau = E$, $\vartheta_n = R$ et 1 est une R-période de T. En conséquence, $u = a^n$ et on est exactement dans le cas $(\Theta\tau, wa) = (R^{n-1}E, a^n)$. □

Certains motifs mènent nécessairement à une bi-suite non normalisée. Le lemme qui suit, dont la démonstration est plus technique, explique plus en détail la situation.

Lemme 21. Soit Θ une suite d'antimorphismes involutifs, $\vartheta \in \{R, E\}$, $w \in \mathcal{A}^*$ et $a, b \in \mathcal{A}$. Supposons que $(\Theta\vartheta\overline{\vartheta}, wab)$ est normalisée. Finalement, soit $u = \psi_{\Theta\vartheta}(wa)$, $v = \psi_{\Theta\vartheta\overline{\vartheta}}(wab)$, $p = |v| - |u|$ et s le suffixe de longueur p de v. Alors

(i) p est la plus petite E-période de v ;

(ii) $\psi_{\Theta\vartheta\overline{\vartheta}\vartheta}(ua\overline{bb}) = v\overline{s}$;

(iii) $\psi_{\Theta\vartheta\overline{\vartheta}\overline{\vartheta}}(ua\overline{bb}) = v\overline{s}s$;

(iv) $(\Theta\vartheta\overline{\vartheta}\overline{\vartheta\vartheta}, ua\overline{bb})$ n'est pas normalisée, mais $(\Theta\vartheta\overline{\vartheta}\vartheta\overline{\vartheta}, ua\overline{bb}b)$ est normalisée et les deux bi-suites engendrent le même mot.

Démonstration. (i) Il suit directement du lemme 14 que p est une E-période de v. Il reste à montrer que cette période est minimale. Supposons au contraire qu'il existe une E-période $p' < p$. Soit s' le suffixe de longueur p' de v. Alors le mot $u\overline{s'}$ est un $\overline{\vartheta}$-palindrome et b est bien la première lettre de s', contredisant le fait que v le plus court $\overline{\vartheta}$-palindrome dont ub est préfixe.

(ii) Comme v est un $\overline{\vartheta}$-palindrome admettant p comme E-période, il suit du lemme 13 que $v\overline{s}$ est bien un ϑ-palindrome. Supposons maintenant que $v\overline{s}$ n'est pas le plus court $\overline{\vartheta}$-palindrome admettant $v\overline{b}$ comme préfixe. Soit x le plus long $\overline{\vartheta}$-palindrome suffixe de v. Alors $|x| > |v| - p$ et, par le lemme 14, v possède la E-période $|v| - |x| < p$, contredisant (i).

(iii) Ici aussi, par le lemme 13, on sait que $v\overline{s}s$ est un $\overline{\vartheta}$-palindrome. Un argument similaire à celui présenté en (ii) nous permet de conclure qu'il s'agit bien du plus court palindrome admettant $v\overline{b}$ comme préfixe.

(iv) Découle des parties (ii) et (iii). □

Nous sommes maintenant en mesure de décrire les formes exactes des bi-suites directrices non normalisées :

Lemme 22. Une bi-suite s est normalisée si et seulement si elle ne contient aucun préfixe d'une des formes suivantes :

(i) $(RR, a\overline{a})$;

(ii) $(R^{i-1}E, a^i)$;

(iii) $(R^iEE, a^i\overline{aa})$;

(iv) $(\Theta REE, wab\overline{b})$ ou $(\Theta ERR, wab\overline{b})$,

où $a, b \in \{0, 1\}$, $i \geq 1$ est un entier et (Θ, w) est une bi-suite directrice finie.

Démonstration. ($\Rightarrow$) Nous démontrons la contraposée, c'est-à-dire que nous supposons que la bi-suite directrice admet une de ces quatre formes comme préfixe et nous démontrons qu'elle n'est alors pas normalisée.

Clairement, (i) $\psi_{RR}(a\overline{a}) = a\overline{a}a$ manque le E-palindrome préfixe $a\overline{a}$, (ii) $\psi_{R^{i-1}E}(a^i) = a^i\overline{a}^i$ manque le palindrome préfixe a^i et (iii) $\psi_{R^iEE}(a^i\overline{aa}) = a^i\overline{a}^{i+1}a^{i+1}\overline{a}^i$ manque le palindrome préfixe $a^i\overline{a}^{i+1}a^i$. Le cas (iv) suit directement du lemme 21.

($\Leftarrow$) Nous démontrons encore une fois la contraposée, c'est-à-dire que nous supposons que s n'est pas normalisée. Si $|s| \leq 2$, alors par inspection, on constate que les seules bi-suites directrices non normalisées sont celles des cas (i) et (ii). Supposons maintenant que $|s| \geq 3$. Soit (Θ, w) le plus court préfixe non normalisé de $\mathbf{s}$ et $n = |w|$. Soit

$$u = \psi_{\vartheta_1\vartheta_2\cdots\vartheta_{n-1}}(w\texttt{[1]}w\texttt{[2]}\cdots w\texttt{[}n-1\texttt{]}), v = \psi_\Theta(w)$$

et t un $\overline{\vartheta_n}$-palindrome manqué par (Θ, w). En outre, soit $p = |v| - |u|$, $q = |v| - |t|$ et

$g = \text{pgcd}(p, q)$. Par le lemme 20, il y a deux possibilités : soit nous nous retrouvons dans le cas (ii), soit (Θ, w) admet comme suffixe un des éléments de l'ensemble

$$\{(RRR, abc), (EEE, abc), (ERR, abc), (REE, abc)\},$$

où $a, b, c \in \{0, 1\}$. D'autre part, le lemme 19 implique que $|v| \geq p+q$, g est une E-période de v, $p = 2g$, $q = g$ et u est un ϑ_n-palindrome, c'est-à-dire que $\vartheta_{n-1} = \vartheta_n$. Posons

$$y = \psi_{\vartheta_1\vartheta_2\cdots\vartheta_{n-2}}(w\,[1]\,w\,[2]\cdots w\,[n-2]\,).$$

Dans un premier temps, remarquons que $|u| - |y| \leq g$. Autrement, il existerait un préfixe $\overline{\vartheta_n}$-palindromique entre y et u, à savoir le suffixe de v de longueur $|v| - 3g$ par le lemme 13, contredisant le fait que $(\vartheta_1\vartheta_2\cdots\vartheta_{n-1}, w\,[1]\,w\,[2]\cdots w\,[n-1]\,)$ est normalisée. Soit $g' = |u| - |y|$. Si $g' = g$, alors $\vartheta_{n-2} = \overline{\vartheta_n}$ et $c = \bar{b}$, c'est-à-dire que s termine avec $(ERR, ab\bar{b})$ ou $(REE, ab\bar{b})$. Par le lemme 21, ce qui correspond au cas (iv). Il reste à considérer le cas $g' < g$. Nous montrons dans les paragraphes qui suivent que cela nous mène au cas (iii) ou (iv).

Premièrement, montrons que $|y| < g$. Pour cela, supposons au contraire que $|y| \geq g$. Ceci implique que $|u| \geq g + g'$. Puisque g est une E-période de v (en particulier une E-période de u) et que, par le lemme 14, g' est une $(\vartheta_{n-2} \circ \vartheta_n)$-période de u, on déduit du théorème 7 que $g'' = \text{pgcd}(g, g')$ est une E-période de u. De plus, cette E-période g'' se propage dans tout le mot v (puisque g'' divise g, g est une E-période de v et $|u| > g'$, par le lemme 16, ce qui présente une contradiction : ceci impliquerait qu'il existe un ϑ_n-palindrome entre les ϑ_n-palindromes u et v qui correspondrait au préfixe de v de longueur $|v| - 2g''$ (par le lemme 13). Or, $g'' < g$ (puisque $g' < g$). Par conséquent, l'inégalité $|y| < g$ est démontrée.

Maintenant, nous montrons que $|v| = 4g - 2$. Rappelons de la définition de clôture pseudopalindromique que $|v| \leq 2|u| + 2$. De plus, $|v| = |u| + 2g$, $|y| \leq g - 1$ et $g' =$

$|u| - |y| \leq g - 1$. D'une part, on a

$$|v| \leq 2|u| + 2 = 2|v| - 4g + 2,$$

ce qui implique $|v| \geq 4g - 2$. D'autre part,

$$|v| = |u| + 2g \leq |y| + g - 1 + 2g \leq g - 1 + g - 1 + 2g = 4g - 2,$$

de sorte que $|v| = 4g - 2$. En particulier, en vertu des égalités et inégalités $|v| = 4g - 2$, $|v| - |u| = 2g$, $|u| - |y| \leq g - 1$ et $|y| \leq g - 1$, on a $|y| = g - 1$, $|u| = 2g - 2$ et $|v| = 2|u| + 2$. Ainsi, $\vartheta_n = E$ et $g' = |u| - |y| = g - 1$.

Finalement, notons que, puisque $|v| \geq p + q = 3g$, le préfixe z de v de longueur $|v| - 3g$ est un $\overline{\vartheta_n}$-palindrome, c'est-à-dire un R-palindrome, par le lemme 13. Or, le lemme 14 implique que $|y| - |z| = g - g' = 1$ est un $(\vartheta_{n-2} \circ R)$-période de y, c'est-à-dire une ϑ_{n-2}-période de y. Si $\vartheta_2 = R$ et puisque y est de longueur $g-1$ et contient la lettre $\overline{b}$, on trouve $y = \overline{b}^{g-1}$, de sorte que $u = \overline{b}^{g-1} b^{g-1}$ et $v = \overline{b}^{g-1} b^g \overline{b}^g b^{g-1}$, ce qui correspond exactement au cas (iii). Autrement, si $\vartheta_2 = E$, alors en inspectant les pseudopalindromes préfixes z, y et u, on trouve que $(REE, a\overline{b}b)$, où $a \in \mathcal{A}$, est un facteur de $\mathbf{s}$ et on retourne dans le cas (iv), ce qui conclut la démonstration. □

Nous concluons cette section sur la forme normale avec un théorème décrivant exactement la procédure à suivre pour transformer une bi-suite directrice non normalisée en une bi-suite normalisée.

Théorème 8. Soit (Θ, w) une bi-suite directrice, avec Θ une suite finie ou infinie d'antimorphismes involutifs et w un mot de même longueur que Θ. Alors il existe une bi-suite normalisée (Θ', w') telle que $\psi_\Theta(w) = \psi_{\Theta'}(w')$. Plus précisément, il est possible de construire la bi-suite (Θ', w') en utilisant les règles de réécriture suivantes lors de la lecture de la bi-suite de gauche à droite ;

(i) remplacer le préfixe $(RR, a\overline{a})$ par le préfixe $(RER, a\overline{a}a)$;

(ii) remplacer le préfixe $(R^{i-1}E, a^i)$ par le préfixe $(R^iE, a^i\overline{a})$;

(iii) remplacer le préfixe $(R^iEE, a^i\overline{a}\overline{a})$ par le préfixe $(R^iERE, a^i\overline{a}\overline{a}a)$;

(iv) remplacer tout facteur $(\vartheta\overline{\vartheta}\overline{\vartheta}, ab\overline{b})$ par le facteur $(\vartheta\overline{\vartheta}\vartheta\overline{\vartheta}, ab\overline{b}b)$, où $\vartheta \in \{R, E\}$ et $a, b \in \{0, 1\}$.

Démonstration. Le lemme 22 nous indique la forme des préfixes et des facteurs interdits dans les bi-suites directrices normalisées alors que le lemme 21 nous décrit la procédure à suivre pour corriger tout facteur de la forme $(\vartheta\overline{\vartheta}\overline{\vartheta}, ab\overline{b})$ afin de rendre la bi-suite sous forme normale. Il reste à expliquer comment normaliser les préfixes de la forme (i), (ii) ou (iii). Par le lemme 22, on sait que les préfixes $(RER, a\overline{a}a)$, $(R^iE, a^i\overline{a})$ et $(R^iERE, a^i\overline{a}\overline{a}a)$ sont normalisés, puisqu'ils ne sont pas dans l'ensemble des préfixes et facteurs interdits. Maintenant, on vérifie facilement que

$$\begin{aligned}
\psi_{RR}(a\overline{a}) &= \psi_{RER}(a\overline{a}a), \\
\psi_{R^{i-1}E}(a^i) &= \psi_{R^iE}(a^i\overline{a}), \\
\psi_{R^iEE}(a^i\overline{a}\overline{a}) &= \psi_{R^iERE}(a^i\overline{a}\overline{a}a),
\end{aligned}$$

tel que voulu. □

Nous illustrons le théorème avec un exemple.

Exemple 25. Considérons la bi-suite non normalisée $d = (RRR, 011)$. Puisqu'elle a un préfixe de la forme (i) du théorème 8, on peut réécrire d en $d' = (RERR, 0101)$, où $(RER, 010)$ est normalisée.

L'étape suivante consiste à remplacer tous les facteurs de la forme $(\vartheta\overline{\vartheta}\overline{\vartheta}, ab\overline{b})$ par $(\vartheta\overline{\vartheta}\vartheta\overline{\vartheta}, ab\overline{v}b)$. Il n'y a qu'un facteur de cette forme, à savoir $(ERR, 101)$. On obtient alors la nouvelle bi-suite directrice $d'' = (RERER, 01010)$, qui est normalisée. En effet, on vérifie que d'' ne contient aucun préfixe ou facteur interdit. Finalement, vérifions que d et d'' génèrent le même mot :

$$\begin{aligned}
\psi_{RRR}(011) &= \underline{0}\,\underline{1}\,0\,\underline{1}\,0, \\
\psi_{RERER}(01010) &= \underline{0}\,\check{1}\,\underline{0}\,\check{1}\,\underline{0}.
\end{aligned}$$

3.7 Formule de Justin généralisée

En nous inspirant de la formule donnée dans le lemme 12 et en transformant une bi-suite directrice sous sa forme normale, il est possible d'exprimer de façon récursive les préfixes successifs d'un mot pseudostandard généralisé.

On peut maintenant énoncer et démontrer le théorème central de ce chapitre :

Théorème 9. Soit (Θ, w) une bi-suite finie et normalisée de longueur n et, pour $i = 1, 2, \ldots, n$, soit $\psi_i = \psi_{\vartheta_1\vartheta_2\cdots\vartheta_i}(\mathrm{Pref}_i(w))$, $\psi_0 = \varepsilon$ et α_i la dernière lettre de ψ_i.

(i) Si $|\mathrm{Pref}_{n-1}(w)|_{w[n]} = 0$ ou $|\mathrm{Pref}_{n-1}(\Theta)|_{\Theta[n]} = 0$, alors

$$\psi_n = \begin{cases} \psi_{n-1} w[n] \psi_{n-1} & \text{si } \vartheta_n = R, \\ \psi_{n-1} \overline{\psi_{n-1}} & \text{si } \vartheta_n = E \text{ et } \alpha_{n-1} \neq w[n], \\ \psi_{n-1} w[n] \overline{w[n]\psi_{n-1}} & \text{si } \vartheta_n = E \text{ et } \alpha_{n-1} = w[n]. \end{cases}$$

(ii) S'il existe un indice j tel que $\Theta[j] = \vartheta_n$ et $(\vartheta_{n-1} \circ \vartheta_n)(w[j+1]) = w[n]$, alors soit i le plus grand entier satisfaisant cette condition. On a alors

$$\psi_n = \psi_{n-1}(\vartheta_{n-1} \circ \vartheta_n)(\psi_i^{-1}\psi_{n-1}).$$

(iii) Autrement,

$$\psi_n = \begin{cases} \psi_{n-1}\vartheta_n(\psi_{n-1}) & \text{si } \vartheta_n = R \text{ ou } \alpha_{n-1} \neq w[n], \\ \psi_{n-1} w[n] (\vartheta_{n-1} \circ \vartheta_n)(w[n]\psi_{n-1}) & \text{si } \vartheta_n = E \text{ et } \alpha_{n-1} = w[n]. \end{cases}$$

Démonstration. (i) Supposons d'abord que $w = a^{n-1}\overline{a}$ où $a \in \mathcal{A}$. Alors forcément $\Theta \in \{R^n, R^{n-1}E\}$, autrement (Θ, w) ne serait pas sous forme normale par le lemme 22. Alors la formule donnée est vérifiée. Supposons maintenant que $|\vartheta_1\vartheta_2\cdots\vartheta_{n-1}|_{\vartheta_n} = 0$. Alors on a nécessairement $\vartheta_1 = R$ et donc $V_n = E$. Or, aucun E-palindrome n'est préfixe ni suffixe de ψ_{n-1}, on en déduit que le plus long E-palindrome suffixe de $\psi_{\mathrm{Pref}_{n-1}(\Theta)}(\mathrm{Pref}_{n-1}(w))w[n]$ est soit ε ou $\alpha_{n-1}w[n]$ si $w[n] = \overline{\alpha_{n-1}}$, d'où le résultat.

(ii) Par hypothèse, il existe un ϑ_n-palindrome préfixe ψ_i de ψ_{n-1} suivi de la lettre $(\vartheta_{n-1} \circ \vartheta_n)(w\,[n])$. De plus, puisque (Θ, w) est normalisée, $|\Theta'|$ est maximal et, par construction, ψ_i est exactement le plus long ϑ_n-palindrome préfixe de ψ_{n-1} suivi de la lettre $(\vartheta_{n-1} \circ \vartheta_n)(w\,[n])$. Or, ψ_{n-1} est un ϑ_{n-1}-palindrome, de telle sorte que $\vartheta_{n-1}(\psi_i)$ est le plus long $(\vartheta_{n-1} \circ \vartheta_n)$-palindrome suffixe de ψ_{n-1} précédé par $\psi_n(w\,[n])$. Alors $s = \vartheta_n(w\,[n])\vartheta_{n-1}(\psi_i)w\,[n]$ est le plus long ϑ_n-palindrome suffixe de $\psi_{n-1}w\,[n]$. Par conséquent,

$$\begin{aligned}
\psi_n &= \psi_{n-1}w\,[n](s^{-1})\vartheta_n(w\,[n])\vartheta_n(\psi_{n-1}) \\
&= \psi_{n-1}w\,[n](\vartheta_n(w\,[n])\vartheta_{n-1}(\psi_i)w\,[n])^{-1}\vartheta_n(w\,[n])\vartheta_n(\psi_{n-1}) \\
&= \psi_{n-1}w\,[n]\,w\,[n]^{-1}\vartheta_{n-1}(\psi_i)^{-1}\vartheta_n(w\,[n])^{-1}\vartheta_n(w\,[n])\vartheta_n(\psi_{n-1}) \\
&= \psi_{n-1}\vartheta_{n-1}(\psi_i)^{-1}\vartheta_n(\psi_{n-1}) \\
&= \psi_{n-1}(\vartheta_{n-1} \circ \vartheta_n)(\psi_i^{-1}\psi_{n-1}),
\end{aligned}$$

ce qui conclut cette partie. À noter que la troisième égalité est obtenue du fait que pour toute paire de mots $u, v \in \mathcal{A}^*$, on a $(uv)^{-1} = v^{-1}u^{-1}$, et que la dernière égalité découle du fait que ψ_i est un ϑ_n-palindrome et ψ_{n-1} est un ϑ_{n-1}-palindrome.

(iii) Puisque l'hypothèse du cas (ii) n'est pas satisfaite, on peut supposer que ψ_{n-1} ne contient pas de ϑ_n-palindrome suffixe précédé de la lettre $\vartheta_n(w\,[n])$ et que $|\mathrm{Pref}_{n-1}(w)|_{w\,[n]} \geq 1$. Supposons tout d'abord que $\vartheta_n = R$. Alors $w\,[n] = \alpha_{n-1}$, autrement on a une contradiction avec l'hypothèse, puisque cela implique que ψ_{n-1} a nécessairement un palindrome suffixe précédé de la lettre $\vartheta_n(w\,[n]) = w\,[n]$, à savoir un suffixe de la forme $w\,[n]\overline{w\,[n]}^i$. Donc $\vartheta_n = R$ implique $w\,[n] = \alpha_{n-1}$ et conséquemment, $\psi_n = \psi_{n-1}R(\psi_{n-1})$. Supposons maintenant que $\vartheta_n = E$. Si $w\,[n] = \alpha_{n-1}$, on déduit que $\psi_n = \psi_{n-1}w\,[n]\overline{w\,[n]\,\psi_{n-1}}$, alors que si $w\,[n] \neq \alpha_{n-1}$, on trouve $\psi_n = \psi_{n-1}\overline{\psi_{n-1}}$. En combinant tous ces cas, on obtient le résultat voulu. □

Il est intéressant de noter que la formule de Justin (lemme 12) est un cas particulier du théorème 9. En effet, si $\Theta = R^n$, alors (Θ, w) est la bi-suite directrice d'un mot

sturmien standard et on retrouve le cas (i) si $w = a^{n-1}\overline{a}$, pour $a \in \{0,1\}$ et alors $\psi_n = \psi_{n-1} w\,\text{⟦}n\text{⟧}\,\psi_{n-1}$. Autrement, le cas (ii) s'applique et on obtient $\psi_n = \psi_{n-1}\psi_i^{-1}\psi_{n-1}$.

D'autre part, il est possible de dériver une seconde formule pour le cas précis des mots E-standards (ceux obtenus par clôture E-palindromique itérée seulement). Il s'agit donc du cas où $\Theta = E^n$, qui correspond à un cas non normalisé, et en utilisant des idées semblables à celles présentées dans la démonstration du théorème 9, on obtient le fait suivant :

Proposition 11. Soit (E^{n+1}, wa) une bi-suite directrice d'un mot E-standard, avec $w \in \{0,1\}^n$. Si w contient la lettre a, écrivons $w = v_1 a v_2$ où v_2 ne contient pas la lettre a. Alors

$$\psi_E(wa) = \begin{cases} \psi_E(w)aE(a)E(\psi_E(w)) & \text{si } a \text{ n'apparaît pas dans } w\,; \\ \psi_E(w)\psi_E(v_1)^{-1}E(\psi_E(w)) & \text{sinon.} \end{cases}$$

où ψ_E dénote la clôture E-palindromique itérée. □

Illustrons le théorème 9 sur le mot de Thue-Morse.

Exemple 26. On sait que le mot de Thue-Morse est un mot pseudostandard généralisé donné par $\mathbf{t} = \psi_{(ER)^\omega}(01^\omega)$. Avant d'appliquer le théorème 9, il faut transformer la bi-suite directrice $d = ((ER)^\omega, 01^\omega)$ sous forme normale. Il suffit de considérer la nouvelle bi-suite $d' = ((RE)^\omega, 01^\omega)$ qui est normalisée. On obtient alors les préfixes successifs suivants :

$$\begin{aligned} t_0 &= \varepsilon; \\ t_1 &= 0; \\ t_2 &= t_1\overline{t_1} = 01, \text{par le théorème 9(i), deuxième cas;} \\ t_3 &= t_2 E(t_0^{-1}t_2) = 01E(01) = 0110, \text{par le théorème 9(ii),;} \\ t_4 &= t_3 E(t_3) = 01101001, \text{par le théorème 9(iii), premier cas,;} \\ t_5 &= t_4 E(t_4) = 0110100110010110, \text{par le théorème 9(iii), premier cas,} \end{aligned}$$

et ainsi de suite, ce qui correspond à la construction habituelle du mot de Thue-Morse.

En guise de conclusion, on présente la traduction des formules données au théorème 9 en pseudocode. À noter que l'algorithme calcule le mot pseudostandard généralisé généré à partir de n'importe quelle bi-suite directrice, qu'elle soit sous forme normale ou non.

Algorithme 3 Calcul d'un mot pseudostandard généralisé fini

fonction MotPseudostandardGeneralise(Θ, w)
▷ On normalise d'abord selon les cas préfixes
  si (RR,$a\overline{a}$) est préfixe de (Θ, w) **alors**
    $\Theta \leftarrow RER(RR)^{-1}\Theta, w \leftarrow a\overline{a}a(a\overline{a})^{-1}w$
  sinon si $(R^{i-1}E, a^i)$ est préfixe de (Θ, w) pour un entier $i \geq 1$ **alors**
    $\Theta \leftarrow R\Theta, w \leftarrow a^i\overline{a}a^{-i}w$
  sinon si $(R^iEE, a^i\overline{a}\,\overline{a})$ est préfixe de (Θ, w) pour un entier $i \geq 1$ **alors**
    $\Theta \leftarrow R^iERE(R^iEE)^{-1}\Theta, w \leftarrow a^i\overline{a}\,\overline{a}a(a^i\overline{a}\,\overline{a})^{-1}w$
  fin si

▷ On traite les autres cas
  $\psi_0 \leftarrow \varepsilon$
  pour $i \in \{1,2,\ldots,n\}$ **faire**
    si $i \leq n-2$ et $(\Theta, w)[i:i+2] \in \{(ERR,ab\overline{b}), (REE,ab\overline{b})\}$ **alors**
▷ On normalise "en ligne"
      $\Theta \leftarrow \vartheta_1 \cdots \vartheta_{i+1}\overline{\vartheta}_{i+1}\vartheta_{i+2}\vartheta_{i+3}\cdots\vartheta_n$
      $w \leftarrow w[1\ldots i+2]\overline{w[i+2]}\,w[i+3\ldots n]$
    fin si
    si $|w[1:i-1]|_{w[i]} = 0$ ou $|\mathrm{Pref}_{i-1}(\Theta)|_{\vartheta_i} = 0$ **alors**
      si $\vartheta_i = R$ **alors** $\psi_i \leftarrow \psi_{i-1}w[i]\psi_{i-1}$
      sinon si $\vartheta_i = E$ et $\vartheta_{i-1}(w[1]) \neq w[i]$ **alors** $\psi_i \leftarrow \psi_{i-1}\overline{\psi}_{i-1}$
      sinon $\psi_i \leftarrow \psi_{i-1}w[i]\overline{w[i]}\,\overline{\psi_{i-1}}$
      fin si
    sinon si il existe j tel que $\mathrm{Pref}_i(\Theta, w) = (\vartheta_1\cdots\vartheta_j\vartheta_i\vartheta_{j+2}\vartheta_{j+3}\cdots\vartheta_i, w[1:j+1]\vartheta_{i-1} \circ \vartheta_i(w[i])w[j+3:i])$ **alors**
      $\psi_i \leftarrow \psi_{i-1}(\vartheta_{i-1} \circ \vartheta_i)(\psi_j^{-1}\psi_{i-1})$
    sinon
      si $\vartheta_i = R$ ou $\vartheta_{i-1}(w[1]) \neq w[i]$ **alors** $\psi_i \leftarrow \psi_{i-1}\vartheta_i(\psi_{i-1})$
      sinon $\psi_i \leftarrow \psi_{i-1}w[i](\vartheta_{i-1} \circ \vartheta_i)(w[i]\psi_{i-1})$
      fin si
    fin si
  fin pour
  retourner ψ_n
fin fonction

La complexité de l'algorithme est optimale, car il parcourt au plus deux fois les suites Θ et w. Celui-ci a été implémenté en Python et se trouve en annexe à la section A.2, page 161. Il sera éventuellement inclus dans Sage.

3.8 Problèmes ouverts

Dans ce chapitre, nous avons étudié une extension naturelle du théorème de Fine et Wilf ainsi qu'un théorème décrivant une formule récursive pour calculer la clôture pseudo-palindromique itérée d'une bi-suite directrice sur un alphabet binaire lorsque les deux antimorphismes R et E sont permis, dans n'importe quel ordre.

Il nous apparaît raisonnable de croire que les idées proposées sont généralisables à des alphabets de k lettres, où $k \geq 3$. Par ailleurs, on connaît très peu d'information sur les mots pseudostandards généralisés. Il serait intéressant de proposer une caractérisation de cette grande famille de mots, qui contient les mots sturmiens standards, les suites de Rote standards, certains mots quasi-sturmiens et le mot de Thue-Morse.

Par exemple, nous savons que les mots sturmiens sont de complexité $n + 1$, alors que les suites de Rote ont une complexité de $2n$. Celle du mot de Thue-Morse, quant à elle, oscille autour de $3n$. Quelle est la complexité maximale pouvant être atteinte par les mots pseudostandards généralisés ? Des évidences calculatoires nous suggèrent la conjecture suivante :

Conjecture 1. Soit $\mathbf{w}$ un mot pseudostandard généralisé. Alors il existe un entier n_0 tel que $F_{\mathbf{w}}(n) \leq kn$ pour tout entier $n \geq n_0$ et pour toute constante $k > 4$.

Autrement dit, il semble que le mieux qu'on puisse faire, c'est d'osciller autour de $4n$.

En dernier lieu, l'étude de la pseudopériodicité telle que définie dans ce chapitre est pertinente en soi et il semble très probable que le théorème de Fine et Wilf généralisé (théorème 7) puisse être étendu à des alphabets quelconques également.

CHAPITRE IV

POLYOMINOS ET PAVAGES

4.1 Polyominos

L'espace que nous étudions dans ce chapitre et les suivants est le *plan discret* aussi appelé *grille discrète* $\mathbb{Z}^2 = \mathbb{Z} \times \mathbb{Z}$. On appelle *cellule unité* (ou *pixel*) tout sous-ensemble c de $\mathbb{R}^2$ défini par

$$c = \{(x,y) \in \mathbb{R}^2 \mid a \leq x \leq a+1, b \leq y \leq b+1\}, \quad \text{où } a,b \in \mathbb{Z}^2$$

Dans la suite, on dénote par $\mathcal{C}$ l'ensemble des cellules unités de $\mathbb{R}^2$. Un ensemble de deux cellules unités distinctes $c, d \in \mathcal{C}$ est dit 4-*connexe* si c et d ont une arête commune, c'est-à-dire que $c \cap d$ est un segment de droite de longueur 1. Plus généralement, un ensemble de cellules $C \subseteq \mathcal{C}$ est dit 4-*connexe* si, pour toute paire de cellules unités distinctes $c, d \in C$, il existe une suite finie $e_1, e_2, \ldots, e_n$ de cellules unités dans C telles que

(i) $c = e_1$ et $d = e_n$;

(ii) Pour tout indice $i = 1, 2, \ldots, n-1$, l'ensemble $\{e_i, e_{i+1}\}$ est un ensemble 4-connexe.

Autrement dit, il existe un chemin reliant toute paire de cellules unités n'utilisant que des déplacements verticaux ou horizontaux. Remarquons qu'un ensemble 4-connexe peut être fini ou infini.

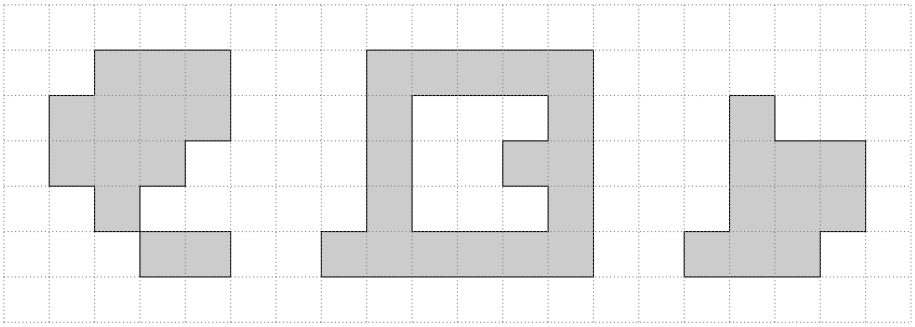


Figure 4.1: Ensemble de cellules unités (a) pas 4-connexe (b) avec trou. (c) Un polyomino.

Soit $C \subseteq \mathcal{C}$ un sous-ensemble de cellules unités. On dit que C est *sans trou* si son complément $\overline{C} = \mathcal{C} - C$ est lui aussi 4-connexe. Un *polyomino* est un ensemble de cellules unités $C \subseteq \mathcal{C}$ qui est à la fois 4-connexe et sans trou.

4.2 Mots de contour

Il existe plusieurs façons de représenter les polyominos. On peut simplement décrire l'ensemble des cellules qui les constituent, les représenter par leur projection selon différents angles, les représenter par leur bord, etc. Dans cette thèse, nous codons les polyominos par leur *mot de contour* : il s'agit en fait d'un mot sur un alphabet à quatre lettres décrivant les quatre déplacements élémentaires *haut*, *bas*, *gauche* et *droite*. Cette représentation présente plusieurs caractéristiques intéressantes :

1. Elle est simple et unique, à conjugaison et orientation près ;
2. Dans de nombreux cas, l'aire d'un polyomino est quadratique par rapport à son périmètre, de sorte que le mot de contour, qui est de longueur égale au périmètre, occupe un espace raisonnable ;
3. Toute la théorie issue de la combinatoire des mots peut être appliquée ;
4. De nombreuses propriétés de symétrie et de convexité d'un polyomino donné se détectent facilement en étudiant simplement son mot de contour ;

En revanche, elle se généralise difficilement aux dimensions 3 et plus.

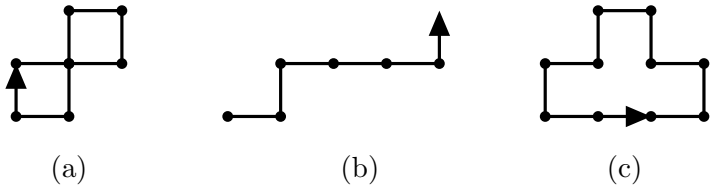


Figure 4.2: Exemples de chemins. (a) Un chemin fermé qui n'est pas auto-évitant, (b) un chemin auto-évitant qui n'est pas fermé et (c) un mot de contour, qui est fermé et auto-évitant.

Dans les paragraphes qui suivent, nous introduisons plus formellement la terminologie nécessaire à l'étude des mots de contour.

L'alphabet permettant de représenter les mots de contour est $\mathcal{F} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$, souvent appelé *code de Freeman*. Il code les déplacements élémentaires sur la grille discrète selon la correspondance suivante :

$$\mathbf{0} : \rightarrow, \quad \mathbf{1} : \uparrow, \quad \mathbf{2} : \leftarrow, \quad \mathbf{3} : \downarrow .$$

Un *chemin* p est un mot sur $\mathcal{F}$, c'est-à-dire un élément de $\mathcal{F}^*$. On dit de p qu'il est *fermé* si $|p|_{\mathbf{0}} = |p|_{\mathbf{2}}$ et $|p|_{\mathbf{1}} = |p|_{\mathbf{3}}$. Un *sous-chemin* q de p est simplement un facteur de p. De plus, p est dit *auto-évitant* s'il ne possède aucun sous-chemin fermé, à l'exception du chemin lui-même et du chemin vide. Un *mot de contour* est un chemin auto-évitant et fermé. La figure 4.2 illustre ces concepts.

Certaines isométries sur la grille discrète se traduisent par des morphismes sur l'alphabet de Freeman $\mathcal{F}$. Tout d'abord, les morphismes

$$\rho^i : \mathcal{F}^* \to \mathcal{F}^* : x \mapsto x + i, \quad (i = 0, 1, 2, 3),$$

où l'addition est considérée modulo 4, correspondent aux rotations d'angle 0, $\pi/2$, π et $3\pi/2$ respectivement (voir figure 4.3). Dans le même ordre d'idée, les morphismes

$$\sigma_i : \mathcal{F}^* \to \mathcal{F}^* : x \mapsto i - x, \quad (i = 0, 1, 2, 3),$$

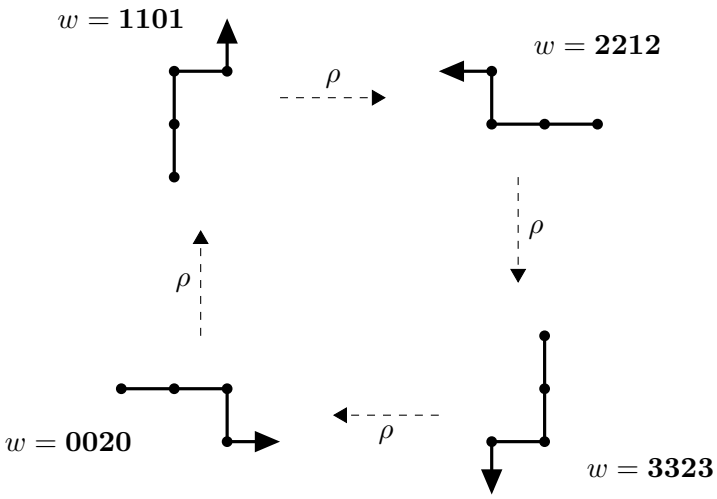


Figure 4.3: Effet des morphismes ρ^i pour $i = 0, 1, 2, 3$ sur le chemin $w = \mathbf{1101}$. En particulier, $\rho^{-1} = \rho^3$.

correspondent aux réflexions par rapport à des axes d'angle 0, $\pi/4$, $\pi/2$ et $3\pi/4$ (voir figure 4.4). Il existe une autre opération sur les mots présentant d'intéressantes propriétés géométriques, qui est donnée par

$$\widehat{p} = \rho^2(\widetilde{p}).$$

qui se traduit par le parcours du chemin p en sens inverse (voir figure 4.5). On dit alors que w et $\widehat{w}$ sont des chemins homologues.

Remarque 4. Tout mot de contour w est primitif, c'est-à-dire qu'il ne peut s'écrire comme puissance entière d'un autre mot. Cette propriété découle directement du fait que w est à la fois fermé et auto-évitant.

Proposition 12. Soit P un polyomino de périmètre n. Alors il existe exactement $2n$ mots de contour décrivant P.

Démonstration. Soit w un mot de contour de P. Alors tout mot $u \in [\![w]\!]$ est également un mot de contour de P. De plus, $|[\![w]\!]| = n$, car w est primitif. Maintenant, considérons le mot $\widehat{w}$. Clairement, $\widehat{w}$ est un mot de contour de P. De plus, tout mot $u \in [\![\widehat{w}]\!]$ est un mot de contour de P et $|[\![\widehat{w}]\!]| = n$. Finalement, on a $[\![w]\!] \cap [\![\widehat{w}]\!] = \emptyset$ (puisque les mots de ces deux classes ont des nombres d'enroulements distincts, voir section 4.4, page 85),

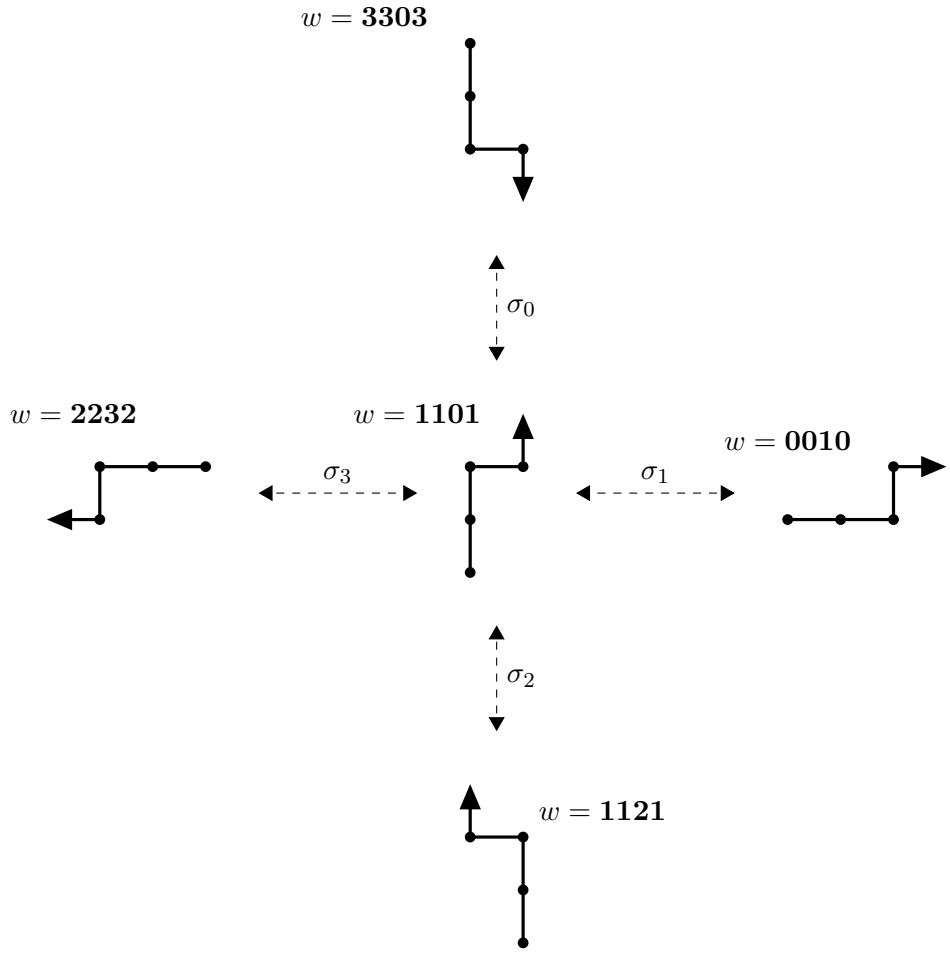


Figure 4.4: Effet des morphismes σ_i pour $i = 0, 1, 2, 3$ sur le chemin $w = \mathbf{1101}$. Notons que $\sigma_i^{-1} = \sigma_i$.

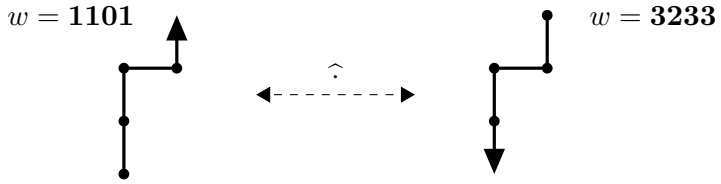


Figure 4.5: Effet de l'antimorphisme $\widehat{\cdot}$ sur le chemin $w = \mathbf{1101}$. Clairement, $\widehat{\widehat{w}} = w$.

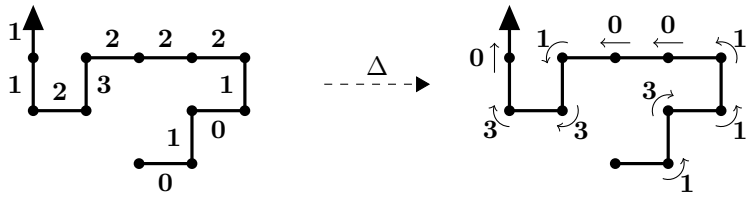


Figure 4.6: Interprétation géométrique de l'opérateur Δ sur le chemin $w = \mathbf{01012223211}$. Le mot $\Delta(w) = \mathbf{1311001330}$ décrit les virages effectués pour parcourir le chemin w selon la correspondance suivante : **0** : *aller en avant*, **1** : *tourner à gauche* et **3** : *tourner à droite*. Comme le chemin est auto-évitant, il n'y a pas de lettre **2** : *reculer*.

ce qui termine la démonstration. □

Comme le mot de contour d'un polyomino n'est pas unique, il est pratique de représenter un chemin fermé w par sa classe de conjugaison $[w]$, qu'on appelle aussi *mot circulaire*.

4.3 Virages

Le code de Freeman permet de décrire des chemins selon les quatre directions élémentaires *haut*, *bas*, *gauche*, *droit*. Il existe une autre représentation pratique des chemins discrets basée sur la notion de virages.

À cette fin, nous introduisons l'opérateur Δ sur $\mathcal{F}$, qui est en fait une extension à un alphabet de 4 lettres de celui présenté au chapitre 2. Plus précisément, soit $w = w_1 w_2 \cdots w_n \in \mathcal{F}^*$ un chemin de longueur $n \geq 2$. Le *mot des différences finies* $\Delta(w) \in \mathcal{F}^*$ *de* w est défini par

$$\Delta(w) = (w_2 - w_1) \cdot (w_3 - w_2) \cdots (w_n - w_{n-1}).$$

D'un point de vue géométrique, les lettres **0**, **1**, **2** et **3** correspondent respectivement au mouvement *aller en avant*, *tourner à gauche*, *reculer* et *tourner à droite*. La figure 4.6 illustre l'effet de l'opérateur Δ sur le chemin $w = \mathbf{01012223211}$.

Il est utile de définir une opération qui inverse en quelque sorte l'effet de l'opérateur Δ. Soit $w = w_1 w_2 \cdots w_n$ un mot de longueur $n \geq 1$ et $\alpha \in \mathcal{F}$ une lettre. Le *mot des*

sommes partielles $\Sigma_\alpha(w)$ *de* w *à partir de* α est donné par

$$\Sigma_\alpha(w) = (\alpha) \cdot (\alpha + w_1) \cdot (\alpha + w_1 + w_2) \cdots (\alpha + w_1 + w_2 + \cdots + w_n).$$

L'opérateur Δ vérifie la propriété suivante, facile à démontrer :

Proposition 13. Soient $u = u_1u_2\cdots u_m, v = v_1v_2\cdots v_n \in \mathcal{F}^*$ des mots de longueur $m, n \geq 2$ respectivement. Alors $\Delta(uv) = \Delta(u)\Delta(u_nv_1)\Delta(v)$.

Lorsqu'on considère des chemins qui ne sont pas auto-évitants, il est possible qu'ils contiennent certains facteurs de l'ensemble $\mathcal{R} = \{\mathbf{02}, \mathbf{13}, \mathbf{20}, \mathbf{31}\}$, c'est-à-dire l'ensemble des chemins de longueur 2 dont les pas sont de directions opposées. Néanmoins, il est toujours possible de réduire un mot $w \in \mathcal{F}^*$ en un unique mot w' pour qu'il ne contienne pas de tels facteurs : il suffit de supprimer récursivement toutes les occurrences de facteurs de $\mathcal{R}$ dans w. Remarquons par contre que le mot réduit n'est pas forcément auto-évitant.

Exemple 27. Le mot $w = \mathbf{10021}$ se réduit au mot $w = \mathbf{101}$ après suppression du facteur **02**. Le mot $u = \mathbf{111333}$ se réduit au mot vide ε après avoir supprimé récursivement trois fois le facteur **13** :

$$\mathbf{111333} \to \mathbf{1133} \to \mathbf{13} \to \varepsilon.$$

4.4 Nombre d'enroulements

La notion de nombre d'enroulements est fondamentale en topologie algébrique et joue un rôle important en calcul vectoriel, en géométrie et en analyse complexe. En général, on considère le nombre d'enroulements d'une courbe fermée, mais il est également possible de généraliser la notion aux courbes non fermées. La figure 4.7 illustre le nombre d'enroulements autour d'un point x. Plus précisément, il s'agit du nombre de tours qu'une courbe décrit autour d'un point donné. En pratique, il est plutôt utilisé pour caractériser les courbes fermées selon leur *indice* (en analyse complexe par exemple).

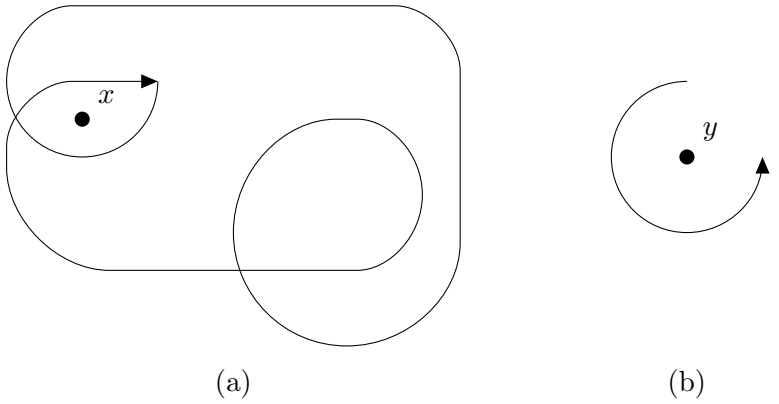


Figure 4.7: Illustration de la notion de nombre d'enroulements. (a) Autour du point x, il est égal à -1 (par convention, le sens anti-horaire est positif). Par ailleurs, si on imagine un observateur qui se déplace le long de la courbe il aura fait exactement 1 tour sur lui-même dans le sens horaire. (b) Autour du point y, le nombre d'enroulements est de 3/4. L'angle de départ est de $\pi/2$ et l'angle de fin est de 2π de sorte que trois quarts de tours ont été effectués dans le sens anti-horaire.

Lorsqu'on traduit la notion de nombre d'enroulements sur la grille discrète, sa définition est très simple. Il s'agit en fait de compter le nombre de virages à gauche et à droite effectués, alors que les mouvements *en avant* ne modifient pas le nombre d'enroulements. Plus formellement, soit $w \in \mathcal{F}^*$ un chemin et $w' \in \mathcal{F}^*$ son chemin réduit après suppression récursive des facteurs dans $\{\mathbf{02}, \mathbf{13}, \mathbf{20}, \mathbf{31}\}$ (voir section précédente). Alors le *nombre d'enroulements de* w est défini par

$$\mathcal{T}(w) = \frac{|\Delta(w')|_{\mathbf{1}} - |\Delta(w')|_{\mathbf{3}}}{4}.$$

Si w est un chemin fermé, alors un ajustement est nécessaire. Le *mot des premières différences d'un mot circulaire* $[\![w]\!]$ est défini par

$$\Delta([\![w]\!]) = [\![\Delta(w) \cdot (w_1 - w_n)]\!],$$

c'est-à-dire qu'on doit prendre en compte le virage entre la dernière et la première lettre du mot w. Il est également possible de réduire un mot circulaire $[\![w]\!]$ en un mot circulaire $[\![w']\!]$ en supprimant récursivement les pas de directions opposés de l'ensemble

$\{\mathbf{02}, \mathbf{13}, \mathbf{20}, \mathbf{31}\}$. Il est alors possible d'étendre naturellement la définition de *nombre d'enroulements d'un mot circulaire* par

$$\mathcal{T}(\llbracket w \rrbracket) = \frac{|\Delta(\llbracket w' \rrbracket)|_{\mathbf{1}} - |\Delta(\llbracket w' \rrbracket)|_{\mathbf{3}}}{4}.$$

Il est bien connu que le nombre d'enroulements d'un chemin fermé w appartient à $\mathbb{Z}$: en analyse complexe, par exemple, on parle d'*indice* d'une courbe. Une démonstration pour le cas des chemins sur la grille discrète se trouve dans (Brlek, Labelle et Lacasse, 2005; Brlek, Labelle et Lacasse, 2006a) où les auteurs utilisent le terme *winding number*.

Soit w un mot de contour décrivant le contour d'un polyomino en sens anti-horaire. Toute occurrence d'un facteur de l'ensemble $\{\mathbf{01}, \mathbf{12}, \mathbf{23}, \mathbf{30}\}$ est appelé *point saillant* et celle d'un facteur de l'ensemble $\{\mathbf{03}, \mathbf{32}, \mathbf{21}, \mathbf{10}\}$ est appelé *point rentrant* (Daurat et Nivat, 2003). En adaptant la notation de Daurat et Nivat à celle de cette thèse, nous avons le résultat suivant :

Théorème 10. (Daurat et Nivat, 2003) Soit $\llbracket w \rrbracket$ un mot de contour circulaire. Alors le nombre d'enroulements de $\llbracket w \rrbracket$ est $\mathcal{T}(\llbracket w \rrbracket) = 1$ (respectivement $\mathcal{T}(\llbracket w \rrbracket) = -1$) si le parcours est orienté dans le sens anti-horaire (respectivement horaire).

La fonction $\mathcal{T}$ ainsi que les isométries ρ^i et σ_i $(i = 0, 1, 2, 3)$ vérifient les propriétés suivantes :

Proposition 14. Soit $w \in \mathcal{F}^*$ un chemin.

(i) $\mathcal{T}(\rho^i(w)) = \mathcal{T}(w)$;

(ii) $\mathcal{T}(\sigma_i(w)) = -\mathcal{T}(w)$;

(iii) $\mathcal{T}(\widehat{w}) = -\mathcal{T}(w)$.

Démonstration. Soit w un chemin et w' son chemin réduit associé.

(i) Le résultat suit du fait que $\Delta(\rho^i(w)) = \Delta(w)$.

(ii) Il suffit d'observer que la réflexion σ_i inverse les virages à gauche avec les virages à droite et laisse inchangés les pas vers l'avant et les pas vers l'arrière.

(iii) L'antimorphisme $\widehat{\cdot}$ inverse les lettres **0** et **2** ainsi que les lettres **1** et **3**. Par conséquent

$$\begin{aligned}\mathcal{T}(\widehat{w}) &= \frac{|\Delta(\widehat{w})|_{\mathbf{3}} - |\Delta(\widehat{w})|_{\mathbf{1}}}{4} \\ &= \frac{|\Delta(w)|_{\mathbf{1}} - |\Delta(w)|_{\mathbf{3}}}{4} \\ &= -\mathcal{T}(w).\end{aligned}$$

□

4.5 Pavages

Lorsque les polyominos ont été introduits dans la littérature, c'est surtout pour leur aspect ludique qu'on s'y intéressait. Le mot « polyomino » a été proposé pour la première fois par Golomb en 1953 dans une présentation et ils ont été popularisés par M. Gardner dans la colonne « Mathematical Games » de la revue « Scientific American » dont il était responsable.

Le plus souvent, les problèmes impliquant les polyominos sont des problèmes de pavages, c'est-à-dire consistant à couvrir certaines surfaces (carré, rectangle, demi-plan, forme quelconque, etc.) de telle sorte que chaque case est couverte par un et un seul polyomino.

La difficulté du pavage dépend fortement des contraintes qu'on impose sur les polyominos dont on dispose :

1. De façon générale, on considère le problème de paver une région donnée à l'aide d'un ensemble de polyominos auxquels on peut faire subir des rotations et des réflexions. Un exemple classique consiste à paver un rectangle 6×10 avec tous les pentaminos à isométrie près, dont les 2339 solutions ont été énumérées depuis longtemps (Haselgrove et Haselgrove, 1960).
2. Les tuiles de Wang (Wang, 1961), qui ne sont pas des polyominos ont été abondamment étudiées dans la littérature et elles sont souvent utilisées pour démontrer que le problème de déterminer si on peut paver le plan à partir d'un ensemble de polyominos quelconque est indécidable (Golomb, 1970).

Figure 4.8: Des pavages hexagonaux réguliers dans l'environnement (à gauche) un mur de briques et (au centre) les alvéoles d'une ruche. On trouve également des pavages carrés dans (à droite) les carreaux d'une salle de bain.

3. Le problème de paver le plan avec une copie d'un seul polyomino a aussi été beaucoup étudié. On connaît exactement toutes les tuiles d'aire au plus 14 qui pavent le plan (Rhoads, 2003).

Dans cette thèse, nous nous intéressons à une classe très spécifique de polyominos : ceux qui pavent le plan par translation seulement, c'est-à-dire qu'il est interdit d'appliquer des rotations ou des réflexions à l'unique tuile utilisée dans le pavage. Lorsqu'on impose autant de contraintes sur le polyomino, on obtient une caractérisation très pratique :

Théorème 11. (Beauquier et Nivat, 1991) Soit P un polyomino admettant un pavage régulier du plan $\mathbb{R}^2$. Alors il existe un mot de contour w de P tel que

$$w = X \cdot Y \cdot Z \cdot \widehat{X} \cdot \widehat{Y} \cdot \widehat{Z}, \tag{4.1}$$

où $X, Y, Z \in \mathcal{F}^*$, avec au plus un mot parmi X, Y et Z qui est vide. □

La factorisation $(X, Y, Z, \widehat{X}, \widehat{Y}, \widehat{Z})$ est souvent appelée une *BN-factorisation* de P et, par abus de notation, on écrit simplement $XYZ\widehat{X}\widehat{Y}\widehat{Z}$. Informellement, le théorème 11 affirme que, pour paver le plan à l'aide d'une seule tuile, on a nécessairement un pavage carré (comme ceux qu'on observe souvent sur les planchers) ou un pavage hexagonal (comme ceux qu'on trouve dans les ruches d'abeille ou qu'on observe sur murs de briques), tels qu'illustrés à la figure 4.8.

La démonstration du théorème de Beauquier-Nivat est complexe et un article com-

plet lui est consacré (Beauquier et Nivat, 1991). Un polyomino P admettant une BN-factorisation avec X, Y et Z non vide est appelé *tuile hexagonale*, alors que si un des mots X, Y et Z est vide, on l'appelle plutôt *tuile carrée* (dans la littérature, on trouve aussi *pseudo-hexagone* et *pseudo-carré*).

Proposition 15. Soit $w = XY\widehat{X}\widehat{Y}$ un mot de contour d'une tuile carrée. Alors

$$\Delta(\texttt{[}w\texttt{]}) = \texttt{[}\Delta(X)\cdot\alpha\cdot\Delta(Y)\cdot\alpha\cdot\Delta(\widehat{X})\cdot\alpha\cdot\Delta(\widehat{Y})\cdot\alpha\texttt{]},$$

où $\alpha = \mathbf{1}$ si w a une orientation positive et $\alpha = \mathbf{3}$ si w a une orientation négative.

Démonstration. Découle directement des propriétés de l'opérateur Δ et de la définition de mot de contour. □

En conséquence, les premières et dernières lettres de la BN-factorisation d'une tuile carrée sont contraintes. Dénotons par $\textsc{f}(w)$ la première lettre d'un mot w et par $\textsc{l}(w)$ la dernière lettre.

Proposition 16. Soit $w = XY\widehat{X}\widehat{Y}$ un mot de contour d'orientation positive d'une tuile carrée. Alors $\textsc{f}(X) = \textsc{l}(X)$ et $\textsc{f}(Y) = \textsc{l}(Y)$.

Démonstration. On sait de la proposition 15 que

$$\textsc{f}(X) - \textsc{l}(\widehat{Y}) = \textsc{f}(Y) - \textsc{l}(X) = \textsc{f}(\widehat{X}) - \textsc{l}(Y) \in \{\mathbf{1}, \mathbf{3}\}.$$

Puisque $\textsc{l}(\widehat{Y}) = \overline{\textsc{f}(Y)}$ et $\textsc{f}(\widehat{X}) = \overline{\textsc{l}(X)}$, On en déduit

$$\textsc{f}(X) - \overline{\textsc{f}(Y)} \overset{(1)}{=} \textsc{f}(Y) - \textsc{l}(X) \overset{(2)}{=} \overline{\textsc{l}(X)} - \textsc{l}(Y) \in \{\mathbf{1}, \mathbf{3}\}.$$

En prenant la somme des égalités (1) et (2), on obtient

$$\textsc{f}(X) - \textsc{l}(X) + \textsc{f}(Y) - \overline{\textsc{f}(Y)} = \textsc{f}(Y) - \textsc{l}(Y) + \overline{\textsc{l}(X)} - \textsc{l}(X) \in \{\mathbf{1}+\mathbf{1}, \mathbf{3}+\mathbf{3}\}$$

Or, $\alpha - \overline{\alpha} = \mathbf{2}$ peu importe $\alpha \in \mathcal{F}$ et $\mathbf{1} + \mathbf{1} = \mathbf{3} + \mathbf{3} = \mathbf{2}$, de sorte que

$$\textsc{f}(X) - \textsc{l}(X) + \mathbf{2} = \textsc{f}(Y) - \textsc{l}(Y) + \mathbf{2} = \mathbf{2}$$

On en conclut que

$$\textsc{f}(X) = \textsc{l}(X) \quad \text{et} \quad \textsc{f}(Y) = \textsc{l}(Y),$$

tel que voulu. □

4.6 Arithmétique des polyominos

La définition de tuile carrée mène naturellement à la définition de *polyomino premier* et *polyomino composé*. Plus précisément, soit C une tuile carrée admettant un mot de contour $AB\widehat{A}\widehat{B}$, où $A, B \in \mathcal{F}^*$. À partir de C, nous pouvons construire un morphisme $\mu_{A,B}$ défini par

$$\mu_{A,B} : \mathcal{F}^* \to \mathcal{F}^* : \mathbf{0} \mapsto A, \mathbf{1} \mapsto B, \mathbf{2} \mapsto \widehat{A}, \mathbf{3} \mapsto \widehat{B}.$$

Soit P un polyomino quelconque de mot de contour w. Alors $\mu_{A,B}(w)$ est également un polyomino. Lorsque le contexte est clair, nous omettons l'indice et écrivons seulement μ. Plus généralement, nous avons la définition suivante :

Définition 7. Un morphisme $\mu : \mathcal{F}^* \to \mathcal{F}^*$ est dit *homologue* si $\mu(\mathbf{0}) = \widehat{\mu(\mathbf{2})}$ et $\mu(\mathbf{1}) = \widehat{\mu(\mathbf{3})}$.

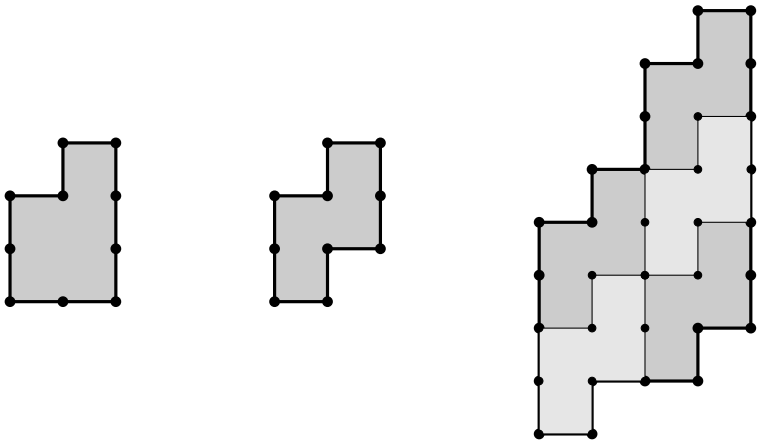
Un polyomino P est dit *composé* s'il existe un morphisme μ, un mot de contour w de P et un mot de contour $u \in \mathcal{F}^*$ tels que

(i) μ est un morphisme homologue ;

(ii) $\mu(\mathbf{0123})$ est une tuile carrée ;

(iii) $\mu(u) = w$;

(iv) Le polyomino Q dont le mot de contour est u vérifie $Q \neq P$ et $Q \neq C$, où C est le carré unité.



(a) (b) (c)

Figure 4.9: Exemple de polyominos composé et premier. (a) Un polyomino premier admettant $u = \mathbf{0011123233}$ comme mot de contour. (b) Une tuile carrée première admettant $AB\widehat{A}\widehat{B}$ comme mot de contour, où $A = \mathbf{010}$ et $B = \mathbf{11}$. (c) Un polyomino composé admettant $\mu(u) = AABBB\widehat{A}\widehat{B}\widehat{A}\widehat{B}\widehat{B}$ comme mot de contour, où $\mu(\mathbf{0}) = A$ et $\mu(\mathbf{1}) = B$: il est pavable à l'aide de la tuile carrée dessinée en (b).

Autrement, P est dit *premier*. Moins formellement, un polyomino est dit premier s'il est impossible de le paver de façon carrée à l'aide d'un plus petit polyomino autre que le carré unité.

Exemple 28. Considérons le polyomino décrit par le mot de contour $u = \mathbf{0011123233}$ et une tuile carrée décrite par le mot de contour $\mathbf{010 \cdot 11 \cdot 232 \cdot 33}$ (voir figure 4.9 (a) et (b)). Il est possible de construire un nouveau polyomino en appliquant le morphisme homologue μ satisfaisant $\mu(\mathbf{0}) = \mathbf{010}$ et $\mu(\mathbf{1}) = \mathbf{11}$, dont un mot de contour est

$$\mathbf{0100101111112323323233333}.$$

Il est également possible de vérifier que le polyomino décrit par u est premier.

Soient P et Q deux polyominos. On dit que P divise Q, noté $P \mid Q$, s'il existe un mot de contour u de P, un mot de contour w de Q et un morphisme homologue μ vérifiant $\mu(u) = w$. Bien que le sujet ne soit pas abordé dans cette thèse, il serait intéressant d'étudier l'arithmétique des polyominos par rapport à cette relation de divisibilité. En

particulier, il semble que le problème de décider si un polyomino est premier ou composé dans un temps raisonnable est ouvert (Provençal, 2008).

CHAPITRE V

TUILES N-CARRÉES ET N-HEXAGONALES

Au chapitre 4, nous avons introduit les définitions de tuiles carrées et de tuiles hexagonales. Ce chapitre est consacré à l'étude des tuiles admettant des pavages multiples. En particulier, dans la section 5.4, nous démontrons qu'il n'existe aucun polyomino admettant plus de deux pavages carrés distincts. Le lecteur constatera que ce chapitre contient de nombreux éléments en commun avec le mémoire de maîtrise de Ariane Garon, malgré le fait qu'ils sont présentés différemment, principalement au niveau de la notation (Garon, 2010). Les résultats présentés plus loin ont fait l'objet d'un article commun (Blondin Massé et al., 2011).

5.1 Pavages multiples

La caractérisation de Beauquier et Nivat nous garantit que si un polyomino admet un pavage régulier, alors il s'agit d'une tuile carrée ou d'une tuile hexagonale. Combien de pavages carrés distincts un polyomino admet-il ? Combien de pavages hexagonaux ? Y a-t-il un maximum possible ? Nous abordons ces différentes questions dans les paragraphes qui suivent.

Définition 8. Soit P un polyomino, w un mot de contour de P et $n \geq 1$ un entier. Alors P est appelé *tuile n-carrée* s'il existe n mots $u_1, u_2, \ldots, u_n \in [w]$ tels que

(i) $u_i = X_i Y_i \widehat{X_i} \widehat{Y_i}$, pour certains mots $X_i, Y_i \in \mathcal{F}^*$, c'est-à-dire que u_i décrit une BN-factorisation carrée ;

(ii) $\mathcal{T}(u_i) = \mathcal{T}(u_j)$ pour $i, j \in \{1, 2, \ldots, n\}$, c'est-à-dire que les différentes factorisa-

tions correspondent à des mots de contour ayant la même orientation ;

(iii) Pour tous $i, j \in \{1, 2, \ldots, n\}$ tels que $i \neq j$, on a

$$X_i Y_i \widehat{X_i} \widehat{Y_i} \quad \notin \quad \{X_j Y_j \widehat{X_j} \widehat{Y_j}, \quad Y_j \widehat{X_j} \widehat{Y_j} X_j, \quad \widehat{X_j} \widehat{Y_j} X_j Y_j, \quad \widehat{Y_j} X_j Y_j \widehat{X_j}\},$$

c'est-à-dire que u_i et u_j correspondent à des BN-factorisations distinctes, même à conjugaison près.

Une définition analogue concerne les tuiles hexagonales :

Définition 9. Soit P un polyomino, w un mot de contour de P et $n \geq 1$ un entier. Alors P est appelé *tuile n-hexagonale* s'il existe n mots $u_1, u_2, \ldots, u_n \in [w]$ tels que

(i) $u_i = X_i Y_i Z_i \widehat{X_i} \widehat{Y_i} \widehat{Z_i}$, pour certains mots $X_i, Y_i \in \mathcal{F}^*$, c'est-à-dire que u_i décrit une BN-factorisation hexagonale ;

(ii) $\mathcal{T}(u_i) = \mathcal{T}(u_j)$ pour $i, j \in \{1, 2, \ldots, n\}$, c'est-à-dire que les différentes factorisations correspondent à des mots de contour ayant la même orientation ;

(iii) Pour tous $i, j \in \{1, 2, \ldots, n\}$ tels que $i \neq j$, on a

$$\begin{aligned} X_i Y_i Z_i \widehat{X_i} \widehat{Y_i} \widehat{Z_i} \quad \notin \quad & \{X_j Y_j Z_j \widehat{X_j} \widehat{Y_j} \widehat{Z_j}, \quad Y_j Z_j \widehat{X_j} \widehat{Y_j} \widehat{Z_j} X_j, \quad Z_j \widehat{X_j} \widehat{Y_j} \widehat{Z_j} X_j Y_j \\ & \widehat{X_j} \widehat{Y_j} \widehat{Z_j} X_j Y_j Z_j, \quad \widehat{Y_j} \widehat{Z_j} X_j Y_j Z_j \widehat{X_j}, \quad \widehat{Z_j} X_j Y_j Z_j \widehat{X_j} \widehat{Y_j}\} \end{aligned}$$

c'est-à-dire que u_i et u_j correspondent à des BN-factorisations distinctes, même à conjugaison près.

Exemple 29. Considérons le mot de contour circulaire

$$[w] = [\mathbf{0100100101211232232232303}].$$

La classe de conjugaison $[w]$ de w contient deux BN-factorisations hexagonales

$$\begin{aligned} u_1 &= \mathbf{010010 \cdot 010 \cdot 1211 \cdot 232232 \cdot 232 \cdot 3303} \\ u_2 &= \mathbf{010 \cdot 010010 \cdot 1211 \cdot 232 \cdot 232232 \cdot 3303} \end{aligned}$$

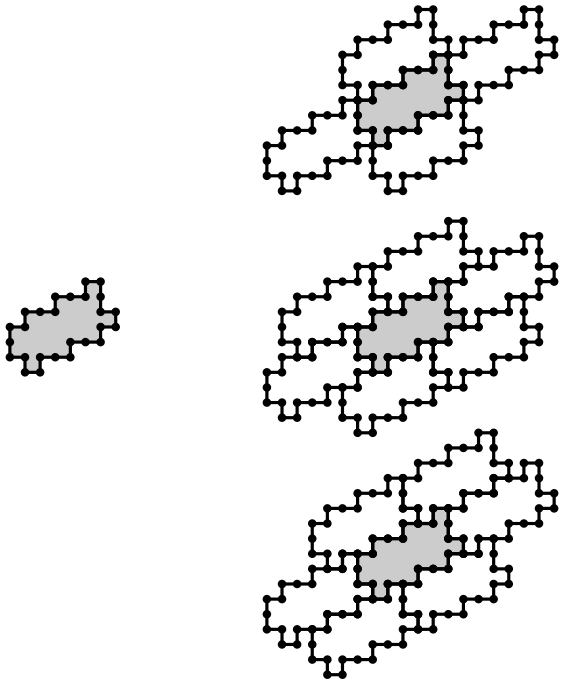


Figure 5.1: Une tuile 2-hexagonale et 1-carrée ainsi que ses trois pavages distincts.

et une BN-factorisation carrée

$$u_3 \quad = \quad \mathbf{010010010} \cdot \mathbf{1211} \cdot \mathbf{232232232} \cdot \mathbf{3303}.$$

La figure 5.1 illustre le polyomino ayant w comme mot de contour, ses deux pavages hexagonaux et son pavage carré.

Étant donné un entier $n \geq 1$, il est facile de construire un exemple de tuile n-hexagonale. Par exemple, considérons le polyomino rectangulaire $(n+1) \times 1$ admettant $\mathbf{0}^{n+1}\mathbf{12}^{n+1}\mathbf{3}$ comme mot de contour. Alors la famille de mots $\{u_i\}_{1 \leq i \leq n}$ définie par

$$u_i = \mathbf{0}^i \cdot \mathbf{1} \cdot \mathbf{2}^{n+1-i} \cdot \mathbf{2}^i \cdot \mathbf{3} \cdot \mathbf{0}^{n+1-i}$$

correspond à n pavages hexagonaux distincts.

En revanche, lorsqu'on tente de construire une tuile n-carrée pour n arbitraire, une

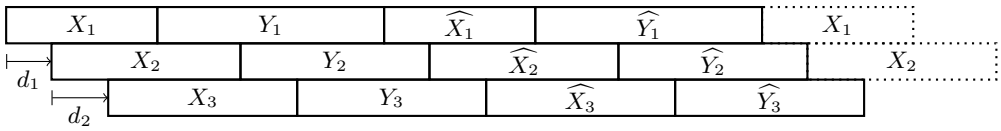


Figure 5.2: Représentation schématique d'un mot de contour admettant trois BN-factorisations carrées.

énumération exhaustive suggère que c'est impossible si $n \geq 3$. La suite de ce chapitre est dédiée à la démonstration de ce fait.

5.2 Équations sur les mots

Une approche intéressante pour démontrer qu'il n'existe aucune tuile n-carrée pour $n \geq 3$ consiste à étudier les équations circulaires induites par des BN-factorisations multiples. Ces contraintes induisent une périodicité locale qui interdit la superposition de trois BN-factorisations carrées.

Dans la suite, nous supposons qu'il existe un polyomino P admettant trois pavages carrés distincts et nous montrons que cela mène à une contradiction. Soient w un mot de contour de P et $X_1, Y_1, X_2, Y_2, X_3, Y_3$ des mots tels que

$$w \equiv X_1 Y_1 \widehat{X_1}\widehat{Y_1} \equiv_{d_1} X_2 Y_2 \widehat{X_2}\widehat{Y_2} \equiv_{d_2} X_3 Y_3 \widehat{X_3}\widehat{Y_3}.$$

Sans perte de généralité, on peut supposer que w décrit le contour de P dans le sens anti-horaire. Par ailleurs, il s'avère utile de représenter schématiquement ces équations circulaires comme à la figure 5.2.

Le premier fait que nous pouvons démontrer à propos de factorisations carrées multiples, c'est qu'elles doivent être alternées. Plus formellement, nous avons le lemme suivant (Provençal, 2008) :

Lemme 23. (Provençal, 2008; Brlek, Provençal et Fédou, 2009) Soit P une tuile et w un de ses mots de contour. Supposons que P soit une tuile 2-carrée, c'est-à-dire qu'il

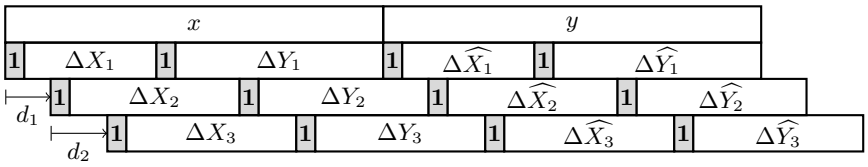


Figure 5.3: Représentation schématique des virages d'un mot de contour admettant trois BN-factorisations carrées.

existe des mots X_1, Y_1, X_2, Y_2 correspondant à deux factorisations distinctes tels que

$$X_1 Y_1 \widehat{X_1} \widehat{Y_1} \equiv_d X_2 Y_2 \widehat{X_2} \widehat{Y_2}.$$

Alors $0 < d < |X_1| < d + |X_2|$. □

Autrement dit, le lemme 23 nous indique que les schémas des figures 5.2 et 5.3 sont bien adaptés à la réalité — en supposant bien entendu qu'il existe des tuiles 3-carrées — et qu'il n'y a aucune occurrence des facteurs X_i ou Y_i qui est incluse dans une autre.

Remarquons également qu'il est pratique de traduire ces équations sur l'alphabet des virages (voir figure 5.3).

Dans la section suivante, nous étudions plus en détail l'espace des positions d'une telle configuration.

5.3 Espace des positions

Considérons la superposition de trois BN-factorisations carrées sur l'alphabet des virages représentées à la figure 5.3. Soit

$$I = \{0, d_1, d_1 + d_2, |X_1|, d_1 + |X_2|, d_1 + d_2 + |X_3|\}$$

l'ensemble des six coins du mot de contour induits par les trois BN-factorisations. Puisque le mot de contour est parcouru en sens anti-horaire, on sait de la proposition 15 que ces six coins doivent être des virages à gauche **1**. De plus, en vertu du

lemme 23, ils sont tous distincts, c'est-à-dire $|I| = 6$.

Dans l'optique d'étudier certaines propriétés symétriques du mot de contour circulaire de cette hypothétique tuile 3-carrée, étant donné une factorisation $XY\widehat{X}\widehat{Y}$, il convient de diviser le mot de contour des différences premières en deux morceaux de longueur égale comme suit :

$$\begin{aligned} x &= x_0x_1x_2\cdots x_{n-1} = \mathbf{1}\Delta X_1 \cdot \mathbf{1} \cdot \Delta Y_1, \\ y &= y_0y_1y_2\cdots y_{n-1} = \mathbf{1}\Delta\widehat{X_1} \cdot \mathbf{1} \cdot \Delta\widehat{Y_1}, \end{aligned}$$

où $n = |x| = |y|$ est le demi-périmètre de la tuile. Remarquons que $\mathbf{1}$ apparaît autant dans x que dans y pour chaque position $i \in I$, tel qu'illustré à la figure 5.3. Nous introduisons trois réflexions sur $\mathbb{Z}_n$:

$$\begin{aligned} s_1 &: \quad i \mapsto (|X_1| - i) \bmod n, \\ s_2 &: \quad i \mapsto (|X_2| + 2d_1 - i) \bmod n, \\ s_3 &: \quad i \mapsto (|X_3| + 2(d_1 + d_2) - i) \bmod n. \end{aligned}$$

Comme toute réflexion est involutive, elles vérifient $s_1^2 = s_2^2 = s_3^2 = 1$. En particulier, comme le produit de trois réflexions est aussi une réflexion, nous avons $(s_js_ks_\ell)^2 = 1$ pour tous $j, k, \ell \in \{1, 2, 3\}$, ce qui est équivalent à l'identité

$$s_ks_\ell s_js_ks_\ell = s_j. \tag{5.1}$$

Pour $j \neq k$, les réflexions s_j et s_k sont dites *perpendiculaires* si $(s_js_k)^2 = 1$ ou, de façon équivalente, $s_js_k = s_ks_j$. On démontre facilement le fait suivant :

Proposition 17. Soit s_j une réflexion perpendiculaire aux deux réflexions s_k, s_ℓ, pour $j \neq k \neq \ell \neq j$. Alors $s_k = s_\ell$.

Démonstration. Comme s_j est perpendiculaire à s_k et s_ℓ, on a $(s_js_k)^2 = 1 = (s_js_\ell)^2$, de sorte que $(s_js_k)(s_js_\ell) = 1$ (puisque $(s_js_k)^{-1} = s_js_k$ et $(s_js_\ell)^{-1} = s_js_\ell$). Alors

$s_k = s_j s_\ell s_j = s_\ell s_j^2 = s_\ell$, tel que voulu. □

Il suit du lemme 23 que s_1, s_2 et s_3 sont distinctes deux à deux. Dans la suite, il est important de remarquer que nous ne souhaitons pas appliquer les réflexions s_1, s_2 et s_3 à n'importe quelle position. En conséquence, on dit que s_1 est *valide* en i si $i \notin \{0, |X_1|\}$, que s_2 est *valide* en i si $i \notin \{d_1, |X_2| + d_1\}$ et que s_3 est *valide* en i si $i \notin \{d_1 + d_2, |X_3| + d_1 + d_2\}$.

Dans la suite de ce chapitre, on pose $\overline{\alpha} = \sigma_0(\alpha)$ pour toute lettre $\alpha \in \mathcal{F}$, c'est-à-dire que $\overline{\mathbf{0}} = \mathbf{0}, \overline{\mathbf{1}} = \mathbf{3}, \overline{\mathbf{2}} = \mathbf{2}, \overline{\mathbf{3}} = \mathbf{1}$. Aussi observons que, pour $w \in \{X_1, X_2, X_3, Y_1, Y_2, Y_3\}$ et pour toute position i dans w, $1 \leq i \leq |w| - 1$, on a

$$(\Delta w)\,[i] = \overline{(\Delta \widehat{w})\,[|w| - i]}. \tag{5.2}$$

L'équation (5.2) se traduit alors simplement en fonction des mots x et y ainsi que des réflexions s_1, s_2 et s_3 :

Lemme 24. Soit $i \in \mathbb{Z}_n$ et $j \in \{1, 2, 3\}$ tel que s_j est valide sur i. Alors une des deux conditions suivantes est satisfaite :

(i) $y_i = \overline{x_{s_j(i)}}$ et $x_i = \overline{y_{s_j(i)}}$;

(ii) $x_i = \overline{x_{s_j(i)}}$ et $y_i = \overline{y_{s_j(i)}}$;

Démonstration. Il y a trois cas à considérer selon la valeur de j. Supposons d'abord que $j = 1$ et supposons que $0 < i < |X_1|$. Alors on trouve

$$\begin{aligned} x_i &= (\Delta X_1)_i = \overline{(\Delta \widehat{X_1})_{|X_1|-i}} = \overline{y_{s_j(i)}}, \\ y_i &= (\Delta \widehat{X_1})_i = \overline{(\Delta X_1)_{|X_1|-i}} = \overline{x_{s_j(i)}}. \end{aligned}$$

D'autre part, si $|X_1| < i < n$, alors $s_j(i) = |X_1| - i + n$ et

$$\begin{aligned} x_i &= (\Delta Y_1)_{i-|Y_1|} = \overline{(\Delta \widehat{Y_1})_{n-i}} = \overline{y_{s_j(i)}}, \\ y_i &= (\Delta \widehat{Y_1})_{i-|Y_1|} = \overline{(\Delta Y_1)_{n-i}} = \overline{x_{s_j(i)}}. \end{aligned}$$

Maintenant, supposons que $j = 2$ et $0 < i < d_1$. On a donc

$$\begin{aligned} x_i &= (\Delta\widehat{Y_2})_{|Y_2|+i-d_1} = \overline{(\Delta Y_2)_{d_1-i}} = \overline{x_{2d_1+|X|-i}} = \overline{x_{s_j(i)}}, \\ y_i &= (\Delta Y_2)_{|Y_2|+i-d_1} = \overline{(\Delta\widehat{Y_2})_{d_1-i}} = \overline{x_{2d_1+|X|-i}} = \overline{y_{s_j(i)}}. \end{aligned}$$

Les autres cas se démontrent de façon similaire. □

Une conséquence immédiate du lemme 24 est la suivante :

Corollaire 3. Soit $i \in \mathbb{Z}_n$ et $j \in \{1, 2, 3\}$ tel que s_j est valide sur i. Si $x_i = y_i$ alors $x_{s_j(i)} = y_{s_j(i)}$. □

Autrement dit, si on démarre avec un virage à gauche **1** dans un mot, alors celui-ci se propage à l'aide des réflexions s_1, s_2 et s_3 dans le mot de contour circulaire en alternant entre virage à droite **3** et virage à gauche **1**.

Soit $k \neq j$. Il est clair que si s_j n'est pas valide sur un indice i, alors nécessairement s_k est valide sur i, étant donné que I contient six éléments distincts. Plus généralement, on dit d'une suite $(s_{j_m}, \ldots, s_{j_2}, s_{j_1})$ qu'elle est *valide* sur i si chaque s_{j_k} est valide sur $s_{j_{k-1}} \cdots s_{j_2} s_{j_1}(i)$, pour $k = 1, 2, \ldots, m$. Par abus de notation, on dit que l'expression $s_{j_m} \cdots s_{j_2} s_{j_1}$ est *valide* sur l'indice i.

Lemme 25. Soit $i \in I$ et $S = s_{j_m} s_{j_{m-1}} \cdots s_{j_2} s_{j_1}$ une expression valide sur i, où s_{j_k} est une réflexion dans $\{s_1, s_2, s_3\}$. Alors $x_{S(i)} = y_{S(i)}$ et

$$x_{S(i)} = \begin{cases} x_i & \text{si } m \text{ est pair,} \\ \overline{x_i} & \text{si } m \text{ est impair.} \end{cases}$$

Démonstration. Par récurrence sur m et en vertu du lemme 24. □

Le lemme 25 stipule que si on observe la propagation d'une lettre α à partir d'une position donnée et que le chemin suivi est de longueur paire, alors la lettre finale est

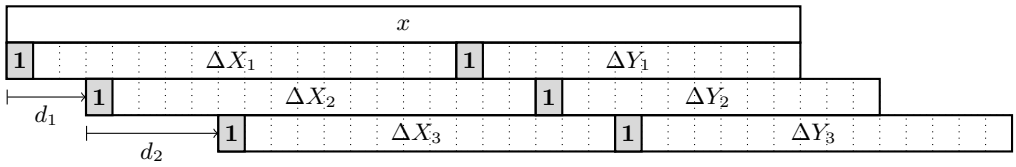


Figure 5.4: Représentation schématique des trois BN-factorisations d'une tuile 3-carrée de demi-périmètre $n = 30$ et de paramètres $d_1 = 3$, $d_2 = 5$, $|X_1| = 17$, $|X_2| = 17$ et $|X_3| = 15$.

également α, alors que si le chemin est de longueur impaire, la lettre finale est son complément $\overline{\alpha}$.

On est maintenant prêt à démontrer le résultat principal de ce chapitre, à savoir qu'il n'existe aucune tuile 3-carrée.

5.4 Tout polyomino admet au plus deux pavages carrés

Intuitivement, l'idée de la démonstration est la suivante. Nous supposons qu'il existe une tuile 3-carrée et qu'elle admet donc trois BN-factorisations alternées. Il est alors possible d'utiliser la propagation des six coins **1** dans le mot à l'aide des réflexions s_1, s_2 et s_3. La contradiction vient alors du fait que nous trouvons toujours un chemin de longueur 5 valide qui lie deux coins **1**, ce qui est impossible étant donné que les chemins de longueur impaire doivent donner des lettres alternées.

Exemple 30. Illustrons la démonstration du théorème 12 sur une tuile de longueur 30. Plus précisément, supposons qu'il existe une tuile 3-carrée de demi-périmètre $n = 30$ et donc les trois BN-factorisations sont définies selon les paramètres $d_1 = 3$, $d_2 = 5$, $|X_1| = 17$, $|X_2| = 17$ et $|X_3| = 15$ (voir figure 5.4). Nous montrons qu'une telle tuile ne peut exister.

En effet, dans ce cas, on a que les réflexions s_1, s_2 et s_3 sont définies sur $\mathbb{Z}_{30}$ par

$$\begin{aligned} s_1 : i &\mapsto 17 - i, \\ s_2 : i &\mapsto 23 - i, \\ s_3 : i &\mapsto 1 - i. \end{aligned}$$

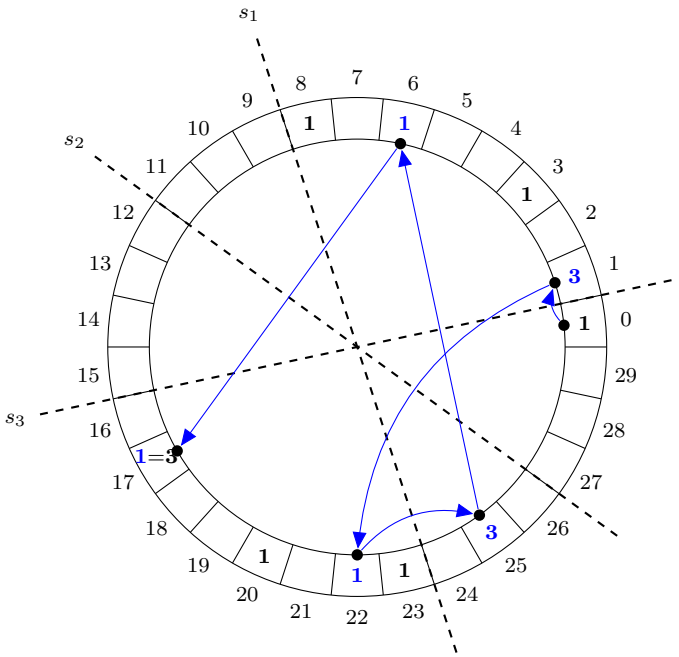


Figure 5.5: Représentation circulaire des réflexions s_1, s_2 et s_3 sur l'espace des positions $\mathbb{Z}_n$ selon les paramètres $n = 30$, $d_1 = 3$, $d_2 = 5$, $|X_1| = 17$, $|X_2| = 17$ et $|X_3| = 15$. Le produit $s_2s_3s_1s_3s_3$ est valide sur la position 0 et donc $\mathbf{1} = x_0 = \overline{x_{s_2s_3s_1s_2s_3(0)}} = \overline{x_{17}} = \overline{\mathbf{1}} = \mathbf{3}$, ce qui est absurde.

Appliquons successivement s_3, s_2, s_1 et à nouveau s_3, s_2. Alors par le lemme 25, on obtient

$$\mathbf{1} = x_0 = \overline{x_{s_2s_3s_1s_2s_3(0)}} = \overline{x_{17}} = \overline{\mathbf{1}} = \mathbf{3},$$

ce qui est une contradiction (voir figure 5.5).

Théorème 12. Il n'existe aucune tuile n-carrée telle que $n \geq 3$.

Démonstration. Il suffit de généraliser les idées utilisées dans l'exemple 30.

La démonstration se fait par l'absurde. Supposons donc qu'il existe une tuile n-carrée P, où $n \geq 3$. En particulier P doit être une tuile 3-carrée et la notation introduite plus tôt dans le chapitre s'appliquent (voir figures 5.2 et 5.3).

Dans un premier temps, on montre que

$$s_2s_3(0) = |X_1| = s_3s_2(0). \tag{5.3}$$

On raisonne en étudiant l'identité $s_1 = s_2s_3s_1s_2s_3$. Les différents cas sont illustrés à la figure 5.6.

Il y a au moins une situation parmi les six situation suivantes qui s'applique :

(a) $s_2s_3s_1s_2s_3$ est valide sur 0 ;

(b) s_3 n'est pas valide sur 0 ;

(c) s_2 n'est pas valide sur $s_3(0)$;

(d) s_2 n'est pas valide sur $s_3s_1s_2s_3(0)$;

(e) s_3 n'est pas valide sur $s_1s_2s_3(0)$;

(f) s_1 n'est pas valide sur $s_2s_3(0)$.

On vérifie maintenant que chaque cas mène à une contradiction.

(a) Supposons que $s_2s_3s_1s_2s_3$ est valide sur 0. Alors

$$\mathbf{3} = \overline{1} = \overline{x_0} = x_{s_2s_3s_1s_2s_3(0)} = x_{s_1(0)} = x_{|X_1|} = \mathbf{1},$$

ce qui est absurde (voir figure 5.6(a)).

(b) Il est impossible que s_3 ne soit pas valide sur 0 puisque s_3 est valide sur toute position i sauf $i = d_1 + d_2 \neq 0$ et $i = |X_3| \neq 0$ (voir figure 5.6(b)).

(c) Supposons maintenant que s_2 n'est pas valide sur $s_3(0)$. Alors $s_3(0) \in I$ et donc

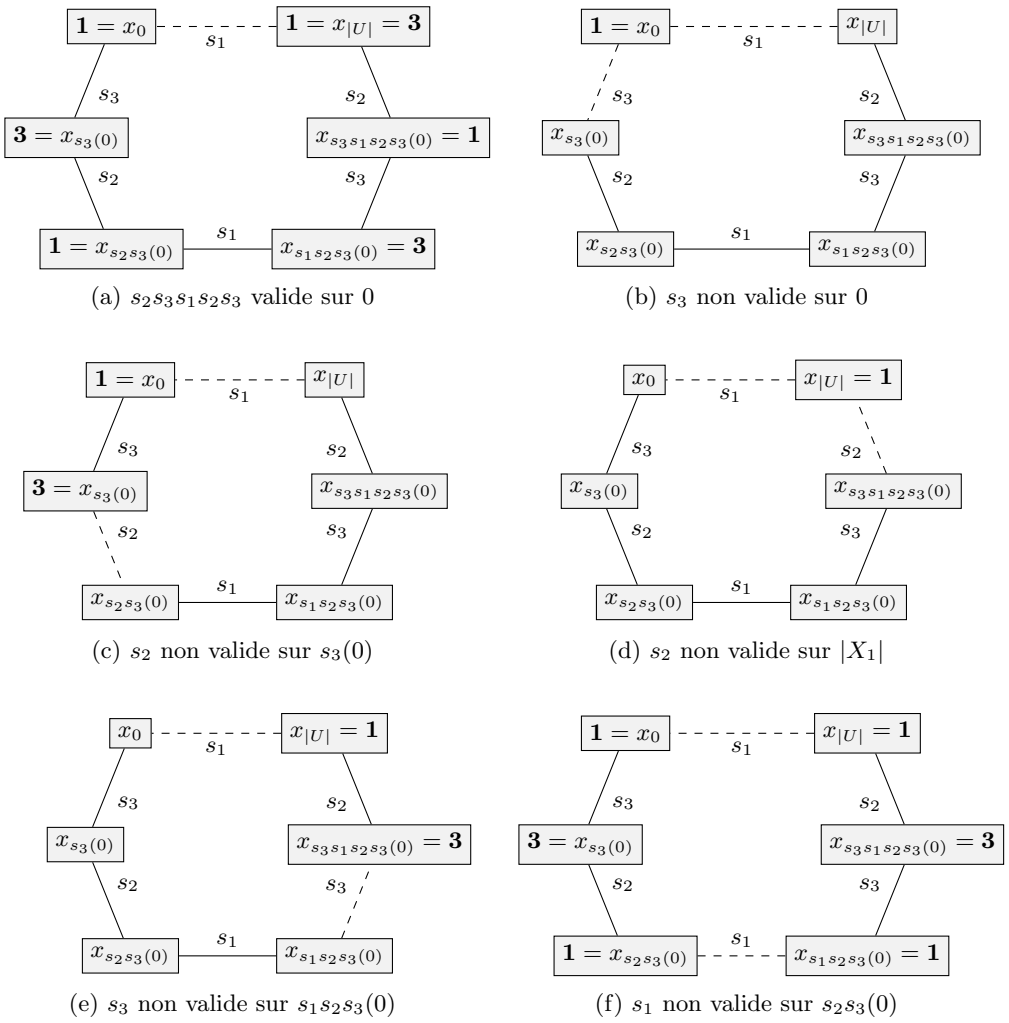
$$\mathbf{3} = \overline{\mathbf{1}} = \overline{x_0} = x_{s_3(0)} = \mathbf{1},$$

ce qui constitue à nouveau une contradiction (voir figure 5.6(c)).

(d) Supposons que s_2 n'est pas valide sur $s_3s_1s_2s_3(0)$. Comme

$$|X_1| = s_1(0) = s_2s_3s_1s_2s_3(0) = s_2(s_3s_1s_2s_3(0)),$$

ceci signifie que s_2 n'est pas valide sur $|X_1|$ non plus, contredisant le fait que s_2 est valide sur toute position autre que $d_1 \neq |X_1|$ et $|X_2| + d_1 \neq |X_1|$ (voir figure 5.6(d)).



(a) $s_2s_3s_1s_2s_3$ valide sur 0

(b) s_3 non valide sur 0

(c) s_2 non valide sur $s_3(0)$

(d) s_2 non valide sur $|X_1|$

(e) s_3 non valide sur $s_1s_2s_3(0)$

(f) s_1 non valide sur $s_2s_3(0)$

Figure 5.6: Illustration des différents cas de la démonstration du théorème 12. Deux positions $i_1, i_2 \in \mathbb{Z}_n$ sont liées par une arête pleine si la réflexion est valide sur i_1 et i_2 et par une arête pointillée si la réflexion n'est pas valide sur i_1 ou i_2.

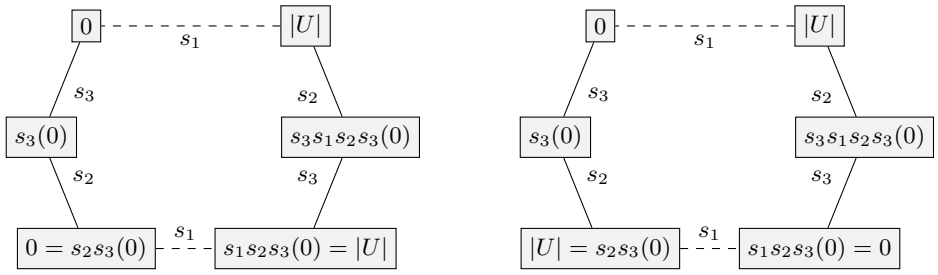


Figure 5.7: Illustration des deux sous-cas de la démonstration du théorème 12 dans le cas où s_1 n'est pas valide en $s_2s_3(0)$. Les sommets correspondent aux six coins **1** induits par les trois BN-factorisations carrées. Une arête pleine relie deux sommets i_1 et i_2 si la réflexion est valide sur i_1 et i_2, alors qu'une arête pointillée relie deux sommets i_1 et i_2 si la réflexion n'est pas valide sur i_1 et i_2.

(e) on doit maintenant étudier le cas où s_3 n'est pas valide sur $s_1s_2s_3(0)$. Alors s_3 n'est pas valide non plus sur $s_3s_1s_2s_3(0)$. Or,

$$\mathbf{1} = x_{s_3s_1s_2s_3(0)} = s_2s_1(0) = s_2(|X_1|) = \overline{\mathbf{1}} = \mathbf{3},$$

puisque s_2 est valide sur $|X_1|$, résultant encore une fois en une contradiction (voir figure 5.6(e)).

(f) Il ne reste plus qu'à considérer le cas où s_1 n'est pas valide sur $s_2s_3(0)$ (voir figure 5.6(f)). Dans ce cas, on en déduit que

$$\{s_2s_3(0), s_1s_2s_3(0)\} = \{0, |X_1|\},$$

puisque s_1 est valide en toute position autre que 0 et $|X_1|$. Il y a deux sous-cas possibles :

(1) $s_2s_3(0) = 0$. Alors $s_2s_3 = 1$ et donc $s_2 = s_3$, contredisant le fait que les trois réflexions s_1, s_2 et s_3 sont distinctes.

(2) $s_2s_3(0) = |X_1|$, tel que voulu.

Le raisonnement que nous avons présenté basé sur l'identité $s_1 = s_2s_3s_1s_2s_3$ s'applique également à l'identité $s_1 = s_3s_2s_1s_3s_2$ et nous permet de montrer que $s_3s_2(0) = |X_1|$. En

outre, les identités $s_2 = s_1 s_3 s_2 s_1 s_3$ et $s_2 = s_3 s_1 s_2 s_3 s_1$ entraînent les égalités suivantes :

$$s_1 s_3(d_1) = d_1 + |X_2| = s_3 s_1(d_1).$$

On en conclut donc que $s_3 s_2 = s_2 s_3$ et $s_1 s_3 = s_3 s_1$, c'est-à-dire que s_3 est perpendiculaire à s_1 et à s_2. Ceci entraîne que $s_1 = s_2$, une contradiction.

Ainsi, aucune tuile 3-carrée ne peut exister et la démonstration est complète. □

5.5 Autres problèmes

Les idées de la section précédente semblent s'étendre naturellement pour démontrer certaines propriétés des tuiles hexagonales. Par exemple, il nous apparaît raisonnable de croire qu'il n'existe aucune tuile 2-carrée qui est aussi 1-hexagonale :

Conjecture 2. Soit P une tuile 2-carrée. Alors P n'est pas n-hexagonale pour tout $n \geq 1$.

Dans le chapitre qui suit, nous proposons une énumération exhaustive des tuiles 2-carrées. Il serait aussi pertinent d'énumérer de façon efficace les tuiles n-hexagonales, pour $n \geq 2$. Finalement, l'étude des tuiles admettant plusieurs pavages de même type se généralise naturellement en dimensions supérieures ainsi que sur la grille hexagonale. Il serait intéressant de vérifier si une borne constante comme celle trouvée dans ce chapitre existe dans d'autres situations.

CHAPITRE VI

TUILES 2-CARRÉES

Au chapitre précédent, nous avons introduit la notion de tuile n-hexagonale et de tuile n-carrée, où $n \geq 1$ est un entier. Nous avons en particulier montré qu'il existe des tuiles n-hexagonales pour tout $n \geq 1$, mais qu'il n'existe aucune tuile n-carrée pour $n \geq 3$. Dans ce chapitre, nous concentrons notre attention sur l'énumération des tuiles 2-carrées, que nous appelons aussi *tuiles* 2*-carrées*. Il est important de préciser ici que ce chapitre contient plusieurs passages communs avec le mémoire de Ariane Garon, notamment en ce qui concerne les démonstrations (Garon, 2010). En revanche, de nombreux aspects, dont l'exploration informatique du problème, ainsi que la démonstration de la conjecture 3, sont nouveaux.

Rappelons qu'un polyomino est une tuile 2-carrée s'il admet un mot de contour $w = AB\widehat{AB} \equiv XY\widehat{XY}$ où A, B, X, Y sont des mots non vides et les deux factorisations sont non trivialement distinctes. Dans sa thèse (Provençal, 2008), X. Provençal produit un tableau contenant les premiers 2-carrés premiers de périmètre 32 ou moins obtenus par programmation (voir tableau 6.1).

D'autre part, l'auteur y conjecture que toutes les tuiles 2-carrées premières possèdent des propriétés symétriques particulières :

Conjecture 3. (Provençal, 2008) Soit P une tuile 2-carrée première et $w = AB\widehat{AB}$ un de ses mots de contour admettant une BN-factorisation carrée. Alors A et B sont des palindromes. De façon équivalente, P est invariante sous rotation d'angle π.

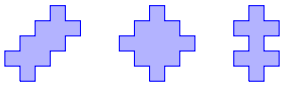
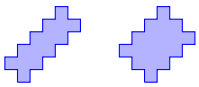
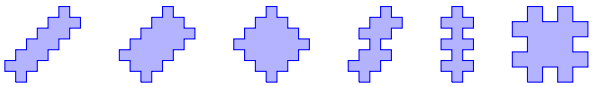
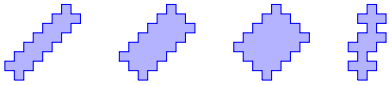
Périmètre	Tuiles
12	
16	
20	
24	
28	
32	

Tableau 6.1: Tableau des tuiles 2-carrées premières de périmètre au plus 32 obtenues par exploration informatique dans la thèse de X. Provençal.

Nous démontrons cette conjecture un peu plus loin dans ce chapitre. Un intérêt des tuiles 2-carrées invariantes sous rotation d'angle π est qu'elles admettent des mots de contour d'une forme bien précise :

Lemme 26. Soit W un mot de contour d'une tuile carrée et A, B deux mots tels que $W \equiv AB\widehat{A}\widehat{B}$. Alors A et B sont des palindromes si et seulement si $W = w\rho^2(w)$ pour un certain mot w.

Démonstration. ($\Rightarrow$) Si A et B sont des palindromes, alors

$$W \equiv AB\widehat{A}\widehat{B} = AB\widetilde{\widehat{A}\widehat{B}} = AB\overline{AB} = u\rho^2(u),$$

où $u = AB$. Clairement, puisque $W \equiv u\rho^2(u)$, on en déduit que $W = w\rho^2(w)$ pour un certain mot w.

($\Leftarrow$) Si $w\rho^2(w) = W \equiv AB\widehat{A}\widehat{B}$, on en déduit que $AB\widehat{A}\widehat{B} = AB\overline{AB}$, de sorte que A et B sont des palindromes. □

En nous inspirant du tableau 6.1 et de la conjecture 3, nous nous intéressons à caractériser la forme générale des 2-carrés ainsi qu'à leur génération. Dans un premier temps, les premières tuiles du tableau 6.1 nous ont naturellement amenés à définir deux familles de 2-carré : les tuiles de Christoffel et les tuiles de Fibonacci.

6.1 Tuiles de Christoffel

La première famille de tuile 2-carrée que nous présentons dans cette thèse et qui présente un intérêt d'un point de vue de la combinatoire des mots est la famille des tuiles de Christoffel. L'idée de définir une telle famille provient de la quatrième tuile de périmètre 28 et de la quatrième tuile de périmètre 32 dans le tableau 6.1.

Rappelons que les mots de Christoffel sont des versions finies des mots sturmiens, c'est-à-dire qu'ils sont obtenus par discrétisation d'un segment de droite dans le plan dont les extrémités ont des coordonnées entières. Soit $(p,q) \in \mathbb{N}^2$ tels que $\text{pgcd}(p,q) = 1$ et soit S le segment de droite liant les points (0,0) et (p,q). Le mot w est appelé *mot de Christoffel inférieur* si le chemin induit par w sur la grille discrète se trouve sous le segment S et que leur combinaison délimite un polygone ne contenant aucun point à coordonnées entières. Un *mot de Christoffel supérieur* est défini de façon analogue. Un *mot de Christoffel* est simplement un mot de Christoffel inférieur ou supérieur. La figure 6.1(a) illustre un mot de Christoffel inférieur.

Soient w et w' les mots de Christoffel inférieur et supérieur associés au couple (p,q). Il est bien connu qu'ils vérifient l'égalité $w' = \widetilde{w}$. De plus, nous avons $w = \mathbf{0}m\mathbf{1}$ et $w' = \mathbf{1}m\mathbf{0}$, où m est un palindrome. Le mot m est appelé *mot de coupe*. Ces mots ont beaucoup été étudiés dans la littérature (voir par exemple (Glen, Lauve et Saliola, 2008), où ils sont appelés *mots centraux*).

Le théorème qui suit donne une caractérisation très pratique des mots de Christoffel :

Théorème 13. (Pirillo, 1999) Un mot m sur l'alphabet $\{\mathbf{0},\mathbf{1}\} \subseteq \mathcal{F}$ est un mot central si et seulement si $\mathbf{0}m\mathbf{1}$ et $\mathbf{1}m\mathbf{0}$ sont conjugués.

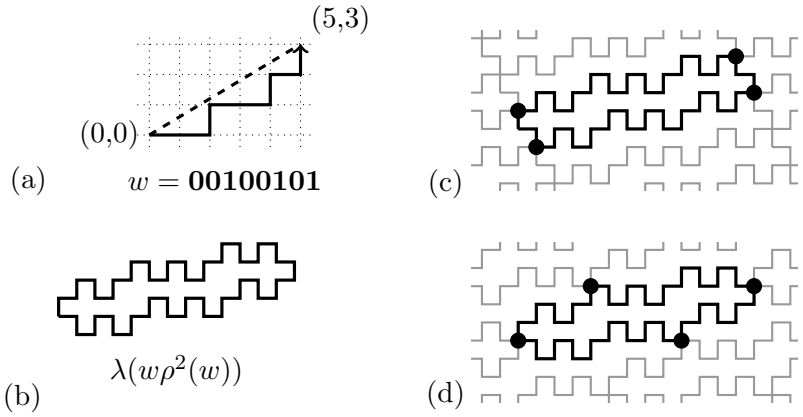


Figure 6.1: (a) Le mot de Christoffel inférieur $w = \mathbf{00100101}$. (b) La tuile de Christoffel $\lambda(w)\rho^2(\lambda(w))$ qui est 2-carrée, dont les pavages sont représentés en (c) et en (d).

Une autre proposition s'avère utile pour la suite de cette section :

Proposition 18. (Borel et Reutenauer, 2006) Soient w et w' les mots de Christoffel inférieur et supérieur associés à un couple (p, q). Alors w et w' sont conjugués par palindromes, c'est-à-dire que $w = uv$ et $w' = vu$ pour certains palindromes u et v.

Considérons le morphisme $\lambda : \mathcal{F}^* \to \mathcal{F}^*$ défini par

$$\mathbf{0} \mapsto \mathbf{0301}, \quad \mathbf{1} \mapsto \mathbf{01}, \quad \mathbf{2} \mapsto \mathbf{2123} \quad \text{et} \quad \mathbf{3} \mapsto \mathbf{23},$$

qui peut être interprété comme un morphisme qui crénèle les quatre pas de base. Deux propriétés du morphisme λ sont utilisées pour la démonstration du théorème principal de cette section :

Lemme 27. Soit $v, v' \in \{\mathbf{0},\mathbf{1}\}^*$. Alors

(i) $\lambda(v) \equiv \lambda(v')$ si et seulement si $v \equiv v'$;

(ii) $\mathbf{1}\lambda(v)$ est un palindrome si et seulement si v est un palindrome.

Démonstration. (i) ($\Leftarrow$) Évident. ($\Rightarrow$) Supposons que $\lambda(v) \equiv \lambda(v')$. Alors il existe un mot u tel que $\lambda(v)u = u\lambda(v')$. En particulier, u est préfixe de $\lambda(v)$ et donc il existe un préfixe propre (mais possiblement vide) p d'un mot dans $\{\mathbf{01}, \mathbf{0301}\}$ et un mot w tel

que $u = \lambda(w)p$. En conséquence, $\lambda(vw)p = \lambda(w)p\lambda(v')$, ce qui montre que p est suffixe de $\lambda(v')$ et donc suffixe d'un mot dans l'ensemble $\{\mathbf{01}, \mathbf{0301}\}$. Comme p est à la fois préfixe et suffixe propre de mots dans $\{\mathbf{01}, \mathbf{0301}\}$, la seule possibilité est $p = \varepsilon$ et donc $\lambda(vw) = \lambda(wv')$. Or, λ est injectif et donc $vw = wv'$ tel que voulu.

(ii) On vérifie directement que $1\lambda(\mathbf{0})$ et $1\lambda(\mathbf{1})$ sont des palindromes. Soit $v = v_1v_2\cdots v_n$. Alors

$$\begin{aligned}
\widetilde{\mathbf{1}\lambda(v)} &= \widetilde{\lambda(v_n)\lambda(v_{n-1})}\cdots\widetilde{\lambda(v_2)\mathbf{1}\lambda(v_1)} \\
&= \widetilde{\lambda(v_n)}\widetilde{\lambda(v_{n-1})}\cdots\widetilde{\mathbf{1}\lambda(v_2)}\lambda(v_1) \\
&= \cdots \\
&= \widetilde{\lambda(v_n)}\widetilde{\mathbf{1}\lambda(v_{n-1})}\cdots\lambda(v_2)\lambda(v_1) \\
&= \mathbf{1}\lambda(\widetilde{v}).
\end{aligned}$$

Par conséquent, si v est un palindrome, alors $\mathbf{1}\lambda(v)$ aussi. Réciproquement, si $\mathbf{1}\lambda(v)$ est un palindrome, comme λ est injectif, $v = \widetilde{v}$ est donc un palindrome. □

Nous sommes maintenant en mesure de démontrer le théorème principal de cette section.

Théorème 14. Soit $w = \mathbf{0}v\mathbf{1} \in \{\mathbf{0}, \mathbf{1}\}^* \subseteq \mathcal{F}^*$.

(i) Si v est un palindrome, alors $\lambda(w\rho^2(w))$ est une tuile carrée ;

(ii) Le mot $\lambda(w\rho^2(w))$ décrit une tuile 2-carrée si et seulement si w est un mot de Christoffel.

Démonstration. (i) Dans un premier temps, remarquons qu'une factorisation carrée est donnée comme suit :

$$\begin{aligned}
\lambda(w\rho^2(w)) &= \lambda(\mathbf{0}v\mathbf{1}\rho^2(\mathbf{0}v\mathbf{1})) \\
&= \mathbf{0301}\lambda(v)\mathbf{01212}\rho^2(\mathbf{1}\lambda(v))\mathbf{23} \\
&\equiv \mathbf{303}\cdot\mathbf{01}\lambda(v)\mathbf{0}\cdot\mathbf{121}\cdot\mathbf{2}\rho^2(\mathbf{1}\lambda(v))\mathbf{2} \\
&= \mathbf{303}\cdot\mathbf{01}\lambda(v)\mathbf{0}\cdot\widehat{\mathbf{303}}\cdot\widehat{\mathbf{01}\lambda(v)\mathbf{0}}.
\end{aligned}$$

Il reste à montrer que $\lambda(w\rho^2(w))$ est simple. Pour simplifier l'argument, nous donnons une idée de la démonstration dans le cas où v est un mot central. Clairement, $\lambda(w)$ et $\lambda(\rho^2(w))$ sont simples puisqu'ils contiennent trois lettres et aucun facteur de la forme $\alpha\overline{\alpha}$. De plus, si P et Q dénotent respectivement le point de départ et le point d'arrivée de $\lambda(w)$, alors le chemin $\lambda(w)$ est sous la droite PQ alors que $\lambda(\rho^2(w))$ se trouve au-dessus de la droite PQ.

(ii) ($\Rightarrow$) Supposons que la tuile obtenue est 2-carré. Soit $W = \lambda(w\rho^2(w))$ un mot de contour de cette tuile tel que $w = \mathbf{0}v\mathbf{1} \in \mathbf{0}\mathrm{Pal}(\mathcal{F}^*)\mathbf{1}$ (voir lemme 26). Puisque W se factorise comme

$$W = \mathbf{303} \cdot \mathbf{01}\lambda(v)\mathbf{0} \cdot \widehat{\mathbf{303}} \cdot \widehat{\mathbf{01}\lambda(v)\mathbf{0}}, \tag{6.1}$$

et puisque les factorisations doivent alterner (voir le lemme 23, page 98), la seconde factorisation doit commencer avec la deuxième ou la troisième lettre de W. Soient W' et W'' tels que $W \equiv_1 W'$ et $W \equiv_2 W''$ et soient V' et V'' les premières moitiés respectives de W' et W''. Alors, par le lemme 26, V' ou V'' est un produit de deux palindromes. Dans un premier temps, supposons qu'il existe deux palindromes x et y tels que $V' = xy$. Alors $V' = \lambda(\mathbf{0}v\mathbf{1}) = \mathbf{0301}\lambda(v)\mathbf{01} = xy$. En appliquant l'opérateur miroir à chaque membre, on trouve $\widetilde{\lambda(\mathbf{0}v\mathbf{1})} = yx$, ce qui entraîne que $\lambda(\mathbf{0}v\mathbf{1}) \equiv \widetilde{\lambda(\mathbf{0}v\mathbf{1})}$ sont conjugués. Or,

$$\begin{aligned}
\widetilde{\lambda(\mathbf{0}v\mathbf{1})} &= \mathbf{10}\widetilde{\lambda(v)}\mathbf{1030} \\
&= \mathbf{101}\widetilde{\lambda(v)}\mathbf{030} \\
&= \mathbf{101}\lambda(v)\mathbf{030} \\
&\equiv \mathbf{01}\lambda(v)\mathbf{0301} \\
&= \lambda(\mathbf{1}v\mathbf{0}),
\end{aligned}$$

ce qui signifie que $\lambda(\mathbf{0}v\mathbf{1}) \equiv \lambda(\mathbf{1}v\mathbf{0})$. Par conséquent, en vertu du lemme 27, on conclut que $\mathbf{0}v\mathbf{1} \equiv \mathbf{1}v\mathbf{0}$. Ainsi, par le théorème 13, v est un mot central et donc $w = \mathbf{0}v\mathbf{1}$ est un

mot de Christoffel inférieur. Il reste à considérer le cas où la seconde factorisation est obtenue du mot V''. Nous aurions alors $V'' = \mathbf{301}\lambda(v)\mathbf{012} = xy$. Or, de tels palindromes x et y ne peuvent exister puisque $\mathbf{2}$ n'apparaît qu'à la fin de V''.

($\Leftarrow$) Supposons que $w = \mathbf{0}v\mathbf{1}$ est un mot de Christoffel inférieur. Il est bien connu que v est un palindrome. De plus, il suit de (i) que $\lambda(w\rho^2(w))$ est le mot de contour d'une tuile carrée. Par le lemme 23 sur les factorisations alternées, nous avons $w = \mathbf{0}m\mathbf{01}m'\mathbf{1}$ pour certains palindromes m et m'. Donc

$$\begin{aligned}\lambda(w\rho^2(w)) &= \lambda(\mathbf{0}m\mathbf{01}m'\mathbf{1})\rho^2(\lambda(\mathbf{0}m\mathbf{01}m'\mathbf{1})) \\ &= \mathbf{0301}\lambda(m)\mathbf{030}\cdot\mathbf{101}\lambda(m')\mathbf{01}\cdot\mathbf{2123}\rho^2(\lambda(m))\mathbf{212}\cdot\mathbf{323}\rho^2(\lambda(m'))\mathbf{23},\end{aligned}$$

ce qui montre que la tuile P admet une deuxième factorisation carrée. □

Une tuile 2-carrée isométrique à une tuile de la forme $\lambda(w\rho^2(w))$ où w un mot de Christoffel sur $\{\mathbf{0}, \mathbf{1}\}$ est appelée *tuile de Christoffel*. Remarquons que la conjecture 3 est vérifiée par les tuiles de Christoffel. Nous concluons cette section en exhibant deux statistiques intéressantes sur cette famille de tuiles.

Proposition 19. Soit T une tuile de Christoffel obtenue à partir du mot de Christoffel de paramètres (p, q), avec p et q premiers entre eux. Alors le périmètre et l'aire de T sont donnés respectivement par $\mathbf{P}(T) = 8p + 4q$ et $\mathbf{A}(T) = 4p + 3q - 2$.

Démonstration. Soit $w = \mathbf{0}v\mathbf{1}$ le mot de Christoffel de paramètres (p, q). D'une part, nous avons

$$\begin{aligned}\mathbf{P}(T) &= |\lambda(w)\rho^2(\lambda(w))| \\ &= 2|\lambda(w)| \\ &= 2(4|w|_0 + 2|w|_1) \\ &= 8p + 4q.\end{aligned}$$

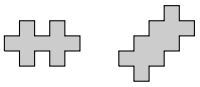
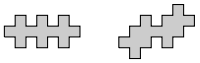
Périmètre	Tuiles
12	
16	
20	
24	
28	
32	
36	
40	
44	

Tableau 6.2: Premières tuiles de Christoffel.

D'autre part, il suit de l'équation (6.1) de la démonstration du théorème 14 que l'aire de T est exactement celle du parallélograme déterminé par les vecteurs $\vec{A} = \overrightarrow{\mathbf{303}} = (1, -2)$ et

$$\vec{B} = \overrightarrow{\mathbf{01}\lambda(v)\mathbf{0}} = (2, 1) + (2|v|_0 + |v|_1, |v|_1) = (2p + q - 1, q).$$

Par conséquent, $\mathbf{A}(T) = |\vec{A} \times \vec{B}| = 4p + 3q - 2$.

Les premières tuiles de Christoffel se trouvent dans le tableau 6.2

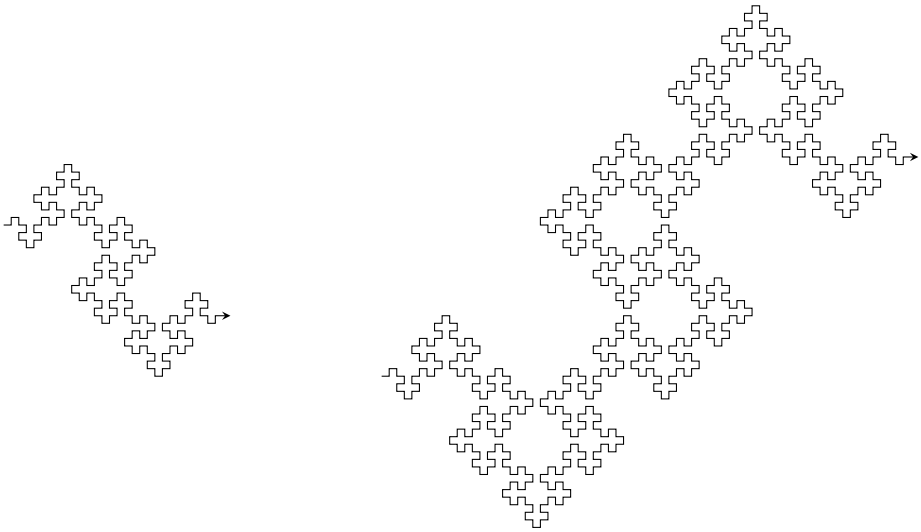


Figure 6.2: Préfixes de longueur 233 (à gauche) et 987 (à droite) du chemin de Fibonacci **p**.

6.2 Tuiles de Fibonacci

Parmi l'ensemble des mots sturmiens, le mot de Fibonacci

$$\mathbf{f} = abaababaabaababaababa\cdots$$

est sans doute le plus connu. Rappelons qu'il est défini comme la limite de la suite $f_{-1} = b$, $f_0 = a$ et, pour $n \geq 1$, $f_n = f_{n-1}f_{n-2}$. De façon équivalente, on peut montrer que **f** est le point fixe du morphisme $\varphi : \{a, b\} \to \{a, b\}$ défini par $\varphi(a) = ab$ et $\varphi(b) = a$.

Il est possible de dériver du mot de Fibonacci **f** un chemin sur l'alphabet $\mathcal{F}$ possédant des propriétés géométriques remarquables. La construction est obtenue comme suit. Tout d'abord, écrivons le mot de Fibonacci sur l'alphabet $\{\mathbf{2}, \mathbf{0}\} \subset \mathcal{F}$ plutôt que $\{a, b\}$. Alors

$$\mathbf{f} = \mathbf{2022020222022020220202}\cdots$$

Ensuite, appliquons l'opérateur Σ_1 suivi de l'opérateur Σ_0. Rappelons que l'opérateur Σ_α est l'opérateur des sommes partielles qui correspond à inverser l'opérateur des différences

finies (voir page 85). Nous obtenons alors le mot

$$\mathbf{p} = \Sigma_0\Sigma_1\mathbf{f} = \mathbf{0103032303010121010301012 1}\cdots$$

qui est un chemin infini sur la grille discrète (voir la figure 6.2).

Ce chemin a été découvert indépendamment dans (Monnerot Dumaine, 2009), où la construction est équivalente mais légèrement différente.

Il est pratique de décrire le chemin $\mathbf{p}$ au moyen des virages à droite et à gauche (encodé respectivement par **3** et **1**) plutôt que les quatre pas élémentaires **0**, **1**, **2** et **3**. La suite des virages du chemin $\mathbf{p}$ est donnée par $\Sigma_1\mathbf{f}$ ou, de façon équivalente, par $\Delta\mathbf{p}$. Nous posons alors $\mathbf{q} = \Delta\mathbf{p}$.

Dans la suite, dans le but de simplifier la notation, nous utilisons la notation $\overline{\cdot}$ sur l'alphabet $\mathcal{F}$ en remplacement de la réflexion σ_0 : $\overline{\mathbf{0}} = \mathbf{0}$, $\overline{\mathbf{1}} = \mathbf{3}$, $\overline{\mathbf{2}} = \mathbf{2}$ et $\overline{\mathbf{3}} = \mathbf{1}$ et les mots $w \in \mathcal{F}^*$ vérifiant $\widetilde{w} = \overline{w}$ sont appelés σ_0-palindromes.

Considérons la suite $(q_n)_{n\in\mathbb{N}} \in \mathcal{F}^*$ définie par $q_0 = \varepsilon$, $q_1 = \mathbf{3}$ et

$$q_n = \begin{cases} q_{n-1}q_{n-2} & \text{si } n \equiv 2 \bmod 3, \\ q_{n-1}\overline{q_{n-2}} & \text{si } n \equiv 0, 1 \bmod 3. \end{cases}$$

pour $n \geq 2$. Les premiers termes de $(q_n)_{n\in\mathbb{N}}$ sont

$$\begin{aligned} q_0 &= \varepsilon \\ q_1 &= \mathbf{3} \\ q_2 &= \mathbf{3} \\ q_3 &= \mathbf{31} \\ q_4 &= \mathbf{311} \\ q_5 &= \mathbf{31131} \\ q_6 &= \mathbf{31131133} \end{aligned}$$

$$
\begin{aligned}
q_7 &= \mathbf{3113113313313} \\
q_8 &= \mathbf{311311331331331131133}
\end{aligned}
$$

Il suit clairement de la définition que $|q_n|$ est le n-ième nombre de Fibonacci.

Proposition 20. (Blondin Massé et Paquin, 2009) Le mot infini $\mathbf{q}$ est la limite de la suite $(q_n)_{n\in\mathbb{N}}$.

Démonstration. Puisque $\Delta(\mathbf{q}) = \mathbf{f}$, il suffit de montrer que $\Delta(q_n)\alpha_n = f_{n-1}$ pour tout entier $n \geq 3$, avec $\alpha_n = \sigma_2^n(\mathbf{2})$. La démonstration se fait par induction sur n. Tout d'abord, nous avons

$$
\begin{aligned}
\Delta(q_3)\sigma_2^3(\mathbf{2}) &= \Delta(\mathbf{31})\mathbf{0} = \mathbf{20} = f_2, \\
\Delta(q_4)\sigma_2^4(\mathbf{2}) &= \Delta(\mathbf{311})\mathbf{2} = \mathbf{202} = f_3, \\
\Delta(q_5)\sigma_2^5(\mathbf{2}) &= \Delta(\mathbf{31131})\mathbf{0} = \mathbf{20220} = f_4.
\end{aligned}
$$

Maintenant, supposons que le résultat est vrai pour tout entier m tel que $3 \leq m < n$ et montrons qu'il est également vrai pour n. Nous présentons seulement le cas $n \equiv 2 \bmod 3$ puisque les arguments sont semblables pour les cas $n \equiv i \bmod 3$, $i \in \{0,1\}$. Soit $n = 3k+2$ où k est un entier. Alors

$$
\begin{aligned}
\Delta(q_{3k+2})\alpha_{3k+2} &= \Delta(q_{3k+1}q_{3k})\alpha_{3k+2} \\
&= \Delta(q_{3k+1})\Delta(\sigma_0^k(\mathbf{3})\mathbf{3})\Delta(q_{3k})\alpha_{3k+2} \\
&= \Delta(q_{3k+1})\sigma_2^k(\mathbf{0})\Delta(q_{3k})\alpha_{3k+2} \\
&= \Delta(q_{3k+1})\alpha_{3k+1}\Delta(q_{3k})\alpha_{3k} \\
&= f_{3k+1}f_{3k} \\
&= f_{3k+2},
\end{aligned}
$$

ce qui démontre le résultat. □

Étant donné une lettre $\alpha \in \mathcal{F}$, le chemin $\Sigma_\alpha q_n$ présente certaines propriétés de symétrie.

Lemme 28. Soit $n \in \mathbb{N}$ et $\alpha = \sigma_0^n(\mathbf{3})$. Alors $q_{3n+1} = p\alpha$, $q_{3n+2} = r\alpha$ et $q_{3n+3} = s\overline{\alpha}$ pour un certain σ_0-palindrome p et certains palindromes r et s.

Démonstration. La démonstration se fait par induction sur n. Pour $n = 0$, nous avons

$$\begin{aligned} q_1 &= \varepsilon \cdot \mathbf{3}, \\ q_2 &= \varepsilon \cdot \mathbf{3}, \\ q_3 &= \mathbf{3} \cdot \mathbf{1}. \end{aligned}$$

Maintenant, supposons que $q_{3n+1} = p\alpha$, $q_{3n+2} = r\alpha$ et $q_{3n+3} = s\overline{\alpha}$, où p est un σ_0-palindrome, r, s sont des palindromes et $\alpha = \sigma_0^n(\mathbf{3})$. Alors

$$\begin{aligned} q_{3n+4} &= q_{3n+3}\overline{q_{3n+2}} = q_{3n+2}\overline{q_{3n+1}q_{3n+2}} = r\alpha\overline{p\alpha r} \cdot \sigma_0^{n+1}(\mathbf{3}), \\ q_{3n+5} &= q_{3n+4}q_{3n+3} = q_{3n+3}\overline{q_{3n+2}}q_{3n+3} = s\overline{\alpha r\alpha}s \cdot \sigma_0^{n+1}(\mathbf{3}), \\ q_{3n+6} &= q_{3n+5}\overline{q_{3n+4}} = q_{3n+4}q_{3n+3}\overline{q_{3n+4}} = r\alpha\overline{p\alpha r\alpha}s\overline{\alpha r\alpha}p\alpha r \cdot \sigma^{n+2}(\mathbf{3}). \end{aligned}$$

On constate que $r\alpha\overline{p\alpha r}$ est un σ_0-palindrome et $s\overline{\alpha r\alpha}s$, $r\alpha\overline{p\alpha r\alpha}s\overline{\alpha r\alpha}p\alpha r$ sont des palindromes, tel que voulu. □

La démonstration du fait que le chemin infini $\mathbf{q}$ est auto-évitant est légèrement technique. En outre, il est possible de construire des polyominos à partir de certains préfixes de $\mathbf{q}$:

Lemme 29. Soit $n \in \mathbb{N}$ et $\alpha \in \mathcal{F}$.

(i) Le chemin $\Sigma_\alpha q_n$ est simple.

(ii) Le chemin $\Sigma_\alpha(q_{3n+1})^4$ est le mot de contour d'un polyomino.

Démonstration. (i) La démonstration se fait par récurrence sur n. Clairement, les chemins $\Sigma_\alpha q_1 = \Sigma_\alpha \mathbf{3}$, $\Sigma_\alpha q_2 = \Sigma_\alpha \mathbf{3}$ et $\Sigma_\alpha q_3 = \Sigma_\alpha \mathbf{31}$ sont simples. Supposons maintenant que c'est le cas de tous les chemins $\Sigma_\alpha q_m$ tels que $1 \leq m < n$ et montrons que le résultat

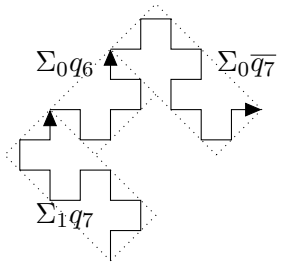


Figure 6.3: Décomposition du chemin $\Sigma_{\mathbf{1}} q_9$ selon les chemins $\Sigma_{\mathbf{1}} q_7$, $\Sigma_{\mathbf{0}} q_6$ et $\Sigma_{\mathbf{1}} \overline{q_7}$.

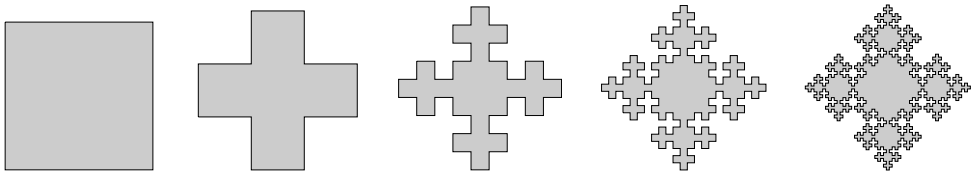


Figure 6.4: Tuiles de Fibonacci d'ordre $n = 0, 1, 2, 3, 4$.

s'applique aussi au chemin $\Sigma_\alpha q_n$. L'idée est de diviser le chemin $\Sigma_\alpha q_n$ en trois morceaux tel qu'illustré à la figure 6.3. En vertu de l'hypothèse d'induction, nous savons que les chemins $\Sigma_\beta q_{n-3}$ et $\Sigma_\gamma q_{n-2}$ sont également simples. Il ne reste alors qu'à constater que les trois chemins plus petits sont chacun contenus dans des boîtes disjointes.

(ii) Il est suffisant de montrer que $\Sigma_\alpha (q_{3n+1})^3$ est simple. Dans un premier temps, remarquons que

$$\begin{aligned}
\overline{q_{3n+5}} &= \overline{q_{3n+4} q_{3n+3}} \\
&= \overline{q_{3n+3}} q_{3n+2} \overline{q_{3n+2}} q_{3n+1} \\
&= \overline{q_{3n+2}} q_{3n+1} q_{3n+1} q_{3n} \overline{q_{3n+2}} q_{3n+1}.
\end{aligned}$$

Or, q_{3n+1} est un préfixe de $q_{3n}\overline{q_{3n+2}}$, de sorte que q_{3n+1}^3 est un facteur de $\overline{q_{3n+5}}$. Il suit de (i) que $\Sigma_\alpha (q_{3n+1})^3$ est également simple. □

La *tuile de Fibonacci d'ordre n* est le polyomino admettant $\Sigma_\alpha (q_{3n+1})^4$ comme mot de contour, où $n \in \mathbb{N}$. Les cinq premières tuiles de Fibonacci sont illustrées à la figure 6.4.

Les résultats précédents nous permettent de conclure ce qui suit :

Théorème 15. Les tuiles de Fibonacci d'ordre $n > 0$ sont des tuiles 2-carrées.

Démonstration. Nous savons du lemme 28 que $q_{3n+1} = px$ pour un certain σ_0-palindrome p et une lettre $x \in \{\mathbf{1}, \mathbf{3}\}$. Si $x = \mathbf{3}$, alors nous considérons l'image miroir du chemin, c'est-à-dire $\sigma_0(\widetilde{(q_{3n+1})}^4)$, qui est conjugué à $(p\overline{x})^4$, de sorte que nous pouvons supposer, sans perte de généralité, que $x = \mathbf{1}$. Alors, d'une part, nous obtenons

$$\Sigma_\alpha(q_{3n+1})^4 = \Sigma_\alpha(p\mathbf{1} \cdot p\mathbf{1} \cdot \sigma_0(\widetilde{p})\mathbf{1} \cdot \sigma_0(\widetilde{p})) = \Sigma_\alpha p \cdot \Sigma_{\rho(\alpha)} p \cdot \widehat{\Sigma_\alpha p} \cdot \widehat{\Sigma_{\rho(\alpha)} p},$$

où ρ est la rotation de $\pi/2$ et puisque $\mathcal{T}(p) = 0$. D'autre part, le conjugué $q'_{3n+1} = \overline{q_{3n-1}} q_{3n}$ de q_{3n+1} correspond à un autre mot de contour de la même tuile. En utilisant encore une fois le lemme 28, nous pouvons écrire $q_{3n} = r\mathbf{1}$ et $q_{3n-1} = q\mathbf{3}$ pour certains palindromes q et r. Par conséquent, $p\mathbf{1} = q_{3n+1} = q_{3n}\overline{q_{3n-1}} = r\mathbf{1}\overline{q}\mathbf{1}$ de sorte que $p = r\mathbf{1}\overline{q}$. Mais p est un σ_0-palindrome, ce qui signifie que $q'_{3n+1} = \overline{q_{3n-1}} q_{3n} = \overline{q}\mathbf{1}r\mathbf{1} = \widetilde{p}\mathbf{1} = \overline{p}\mathbf{1}$. Ainsi, comme $\overline{p}$ est un σ_0-palindrome également, nous trouvons

$$\Sigma_\alpha(q'_{3n+1})^4 = \Sigma_\alpha(\overline{p}\mathbf{1} \cdot \overline{p}\mathbf{1} \cdot \widetilde{p}\mathbf{1} \cdot \widetilde{p}) = \Sigma_\alpha \overline{p} \cdot \Sigma_{\rho(\alpha)} \overline{p} \cdot \widehat{\Sigma_\alpha \overline{p}} \cdot \widehat{\Sigma_{\rho(\alpha)} \overline{p}}.$$

Clairement, les deux factorisations carrées exhibées sont non trivialement distinctes, de sorte que le résultat est démontré. □

Tout comme pour les tuiles de Christoffel, les tuiles de Fibonacci vérifient la conjecture 3.

Corollaire 4. Soit $AB\widehat{A}\widehat{B}$ une BN-factorisation carrée d'une tuile de Fibonacci. Alors A et B sont des palindromes.

Démonstration. La conclusion suit du théorème 15. En effet, puisque p est un σ_0-palindrome, alors $\Sigma_\alpha p$ est un palindrome. Le même argument s'applique pour la seconde factorisation. □

Une autre propriété remarquable des tuiles de Fibonacci est que la suite des aires est

donnée par

$$\mathbf{A}(n) = 1, 5, 29, 169, 985, 5741, 33461, \ldots$$

qui correspond précisément à la sous-suite des nombres de Pell d'indices impairs

$$P(n) = 0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, 33461, \ldots$$

qui satisfait la récurrence $P_n = 2P_{n-1} + P_{n-2}$. Les tuiles de Fibonacci constituent donc un nouvel exemple d'objet combinatoire mettant en relation à la fois le nombre d'or et le nombre d'argent (voir suite A000329 dans (Sloane, 2007)).

6.3 Exploration informatique

On est tenté de conjecturer que toute tuile 2-carrée est soit une tuile de Christoffel, soit une tuile de Fibonacci, mais la situation n'est pas aussi simple. Nous avons donc repris les idées de X. Provençal pour énumérer les tuiles 2-carrées de façon exhaustive en considérant tous les mots de contour admettant deux BN-factorisations distinctes mais en considérant l'alphabet des virages plutôt que l'alphabet de Freeman.

Le tableau 6.3 illustre les premières tuiles 2-carrées obtenues en exécutant le code se trouvant dans l'annexe, à la section A.3. On voit par exemple que la sixième tuile de périmètre 36 n'est ni de Christoffel ni de Fibonacci. C'est aussi le cas de la dernière de périmètre 40 et de la dernière de périmètre 48.

Nous observons que ces tuiles présentent plusieurs similarités et il semble que les idées concernant les tuiles de Christoffel et de Fibonacci sont représentatives des 2-carrés en général. Plus précisément, remarquons par exemple que la sixième tuile de périmètre 36 du tableau 6.3 ressemble à une tuile de Christoffel « plus épaisse », alors que les dernières tuiles de périmètre 40 et 44 apparaissant dans le même tableau sont des versions obliques de la tuile de la croix de périmètre 12. Il semble aussi possible de construire une tuile 2-carrée à partir d'un autre en répétant certains motifs, comme l'illustrent les nombreux « biscuits » obliques qu'on observe pour chaque périmètre et

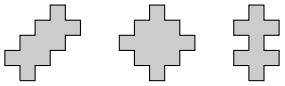
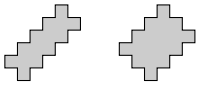
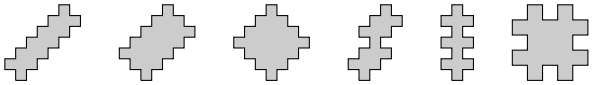
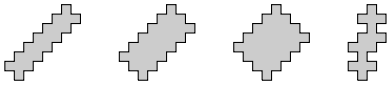
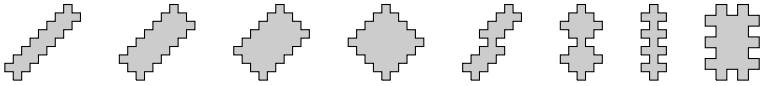
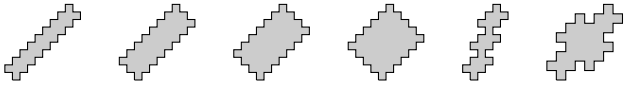
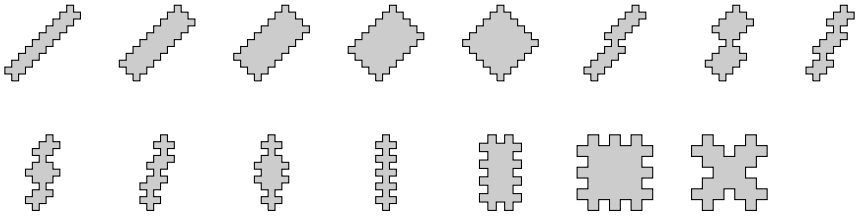
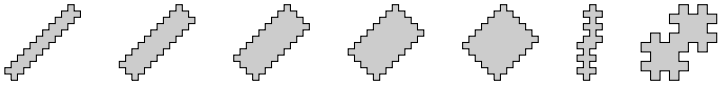
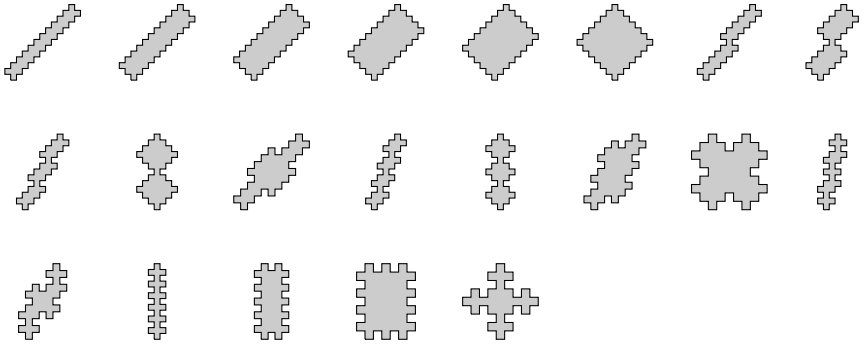
Périmètre	Tuiles
12	
16	
20	
24	
28	
32	
36	
40	
44	
48	
52	

Tableau 6.3: Tableau des premières tuiles 2-carrées obtenues par exploration informatique.

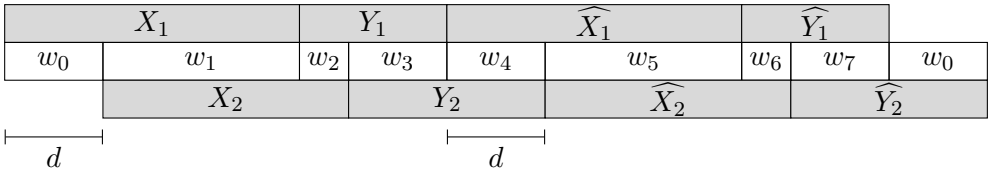


Figure 6.5: Représentation schématique des BN-factorisations d'une tuile 2-carrée.

la dernière tuile de périmètre 48.

De façon informelle, il semble possible de construire des une tuile 2-carrée à partir d'une autre en répétant certains motifs de son mot de contour ou en augmentant son niveau de fractalité de la même façon qu'on observe dans la famille des tuiles de Fibonacci. De plus, il semble possible d'inverser ces opérations de répétition et de « fractalisation ». Nous montrons dans les sections suivantes que ces idées traduisent assez bien le phénomène et que ces opérateurs inversibles permettent de générer toutes les tuiles 2-carrées. Plus particulièrement, il suffit de factoriser adéquatement les mots de contour de ces tuiles et d'étudier la structure combinatoire de ces factorisations.

6.4 Équations sur les mots

Soit P une tuile 2-carrée et w un mot de contour de P. Alors il existe des mots X_1, Y_1, X_2 et Y_2 tels que

$$w \equiv X_1Y_1\widehat{X_1}\widehat{Y_1} \equiv_d X_2Y_2\widehat{X_2}\widehat{Y_2},$$

où d est entier tel que $0 < d < |X_1|$. Nous savons du lemme 23 que les BN-factorisations doivent alterner, c'est-à-dire que $0 < d < |X_1| < d + |X_2|$. Nous avons la représentation graphique de la figure 6.5, où les w_i sont définis plus formellement dans les lignes qui suivent.

Définition 10. Une *DS-factorisation* est un 8-tuplet $(w_i)_{0\leq i\leq 7}$, où $w_i \in \mathcal{F}^+$ tel que

(i) $|w_i| = |w_{i+4}|$ pour $i = 0, 1, 2, 3$ et

(ii) $\widehat{w_iw_{i+1}} = w_{i+4}w_{i+5}$, pour $i = 0, 1, 2, 3$ et où l'addition des indices est considérée modulo 8.

Autrement dit, une DS-factorisation décrit exactement les deux BN-factorisations d'une tuile 2-carrée, en supposant qu'il n'y a aucun croisement dans le chemin. En effet, les quatre facteurs des BN-factorisations sont donnés par

$$X_1 = w_0w_1, \quad Y_1 = w_2w_3, \quad X_2 = w_1w_2 \quad \text{et} \quad Y_2 = w_3w_4.$$

En revanche, elle ne garantit en aucun cas que le chemin résultant soit auto-évitant.

Toute DS-factorisation est uniquement déterminée par les quatre mots w_0, w_1, w_2 et w_3. La *longueur* d'une DS-factorisation $S = (w_i)_{0\leq i\leq 7}$ est naturellement définie par $|S| = |w_0w_1 \cdots w_7|$.

Exemple 31. Considérons le polyomino décrit par le mot de contour

$$w = \mathbf{303} \cdot \mathbf{0103010} \cdot \mathbf{121} \cdot \mathbf{2321232} \equiv \mathbf{323} \cdot \mathbf{0301030} \cdot \mathbf{101} \cdot \mathbf{2123212}.$$

Nous pouvons encoder w par la DS-factorisation

$$(\mathbf{3}, \mathbf{03}, \mathbf{01030}, \mathbf{10}, \mathbf{1}, \mathbf{21}, \mathbf{23212}, \mathbf{32}).$$

Nous donnons maintenant une série de lemmes qui décrivent des propriétés satisfaites par les DS-factorisations. Dans cette section, $S = (w_i)_{0\leq i\leq 7}$ est une DS-factorisations et les indices i sont considérés modulo 8.

Avant de présenter le premier résultat, nous rappelons une proposition bien pratique lorsqu'on traite la périodicité.

Proposition 21. (Lothaire, 1983) Soient x, y et z des mots non vides vérifiant l'équation $xy = yz$. Alors il existe des mots u et v et un entier $i \geq 0$ tels que $x = uv$, $y = (uv)^iu$ et $z = vu$.

La structure de tuile 2-carrée induit une périodicité locale.

Lemme 30. Soit $i \in \mathbb{Z}_8$ et $d_i = |w_{i+1}| + |w_{i+3}|$. Alors

(i) Il existe des mots u_i, v_i et un entier n_i tel que

$$\widehat{w_{i-3}}w_{i-1} = u_iv_i, \tag{6.2}$$

$$w_{i+1}\widehat{w_{i+3}} = v_iu_i, \tag{6.3}$$

$$w_i = (u_iv_i)^{n_i}u_i, \tag{6.4}$$

où $0 \leq |u_i| < d_i$;

(ii) d_i est une période de w_i ;

(iii) $d_i = d_{i+2}$.

Démonstration. (i) Par définition de DS-factorisation, nous avons

$$\widehat{w_{i-3}}w_{i-1}w_i = \widehat{w_{i-3}}\widehat{w_{i+4}}\widehat{w_{i+3}} = \widehat{w_{i+5}}\widehat{w_{i+4}}\widehat{w_{i+3}} = w_iw_{i+1}\widehat{w_{i+3}}.$$

Les deux extrémités de ces égalités vérifient une équation de la forme $xy = yz$, avec $x = \widehat{w_{i-3}}w_{i-1}$, $y = w_i$ et $z = w_{i+1}\widehat{w_{i+3}}$. Par la proposition 21, les égalités en découlent.

(ii) Comme $d_i = |w_{i+1}| + |w_{i+3}| = |u_i| + |v_i|$ et puisque $w_i = (u_iv_i)^{n_i}u_i$, on en déduit que d_i est bien une période de w_i.

(iii) Provient du fait que $|w_i| = |w_{i+4}|$ pour tout $i \in \mathbb{Z}_8$. □

Les notations u_i, v_i et n_i sont utilisées dans le reste de ce chapitre, car elles permettent de définir des opérateurs sur les DS-factorisations et sont capitales dans la plupart des démonstrations. Aussi, une conséquence directe du lemme 30 est que les périodes peuvent être étendues :

Corollaire 5. Pour tout $i \in \mathbb{Z}_8$, le nombre d_i est une période de $w_{i-1}w_iw_{i+1}$.

Démonstration. Nous avons

$$\widehat{w_{i-3}}w_{i-1} \cdot w_i \cdot w_{i+1}\widehat{w_{i+3}} = (u_iv_i)^{n_i+2},$$

donc $d_i = |u_iv_i|$ en est bien une période.

Le nombre n_i correspond au nombre de répétitions des motifs selon la période d_i dans les facteurs w_i. Il y a une certaine restriction quant aux valeurs que peuvent prendre les n_i :

Lemme 31. Supposons que $n_i \neq 0$ pour un certain $i \in \mathbb{Z}_8$. Alors $n_{i+1} = n_{i+3} = n_{i+5} = n_{i+7} = 0$.

Démonstration. On procède par l'absurde, c'est-à-dire qu'on suppose qu'il existe $i \in \mathbb{Z}_8$ tel que $n_i, n_{i+1} \neq 0$. Alors $|w_i| \geq |w_{i-1}| + |w_{i+1}|$ et $|w_{i+1}| \geq |w_i| + |w_{i+2}|$. On obtient alors

$$|w_i| \geq |w_{i-1}| + |w_{i+1}| \geq |w_{i-1}| + |w_i| + |w_{i+2}| > |w_i|,$$

ce qui est absurde. De la même façon, on montre que $n_{i-1} = 0$ et en appliquant l'identité $n_i = n_{i+4}$, on obtient le résultat. □

En utilisant la notation du lemme 30, nous énonçons différentes propriétés de commutativité utiles pour la suite :

Lemme 32. Pour tout $i \in \mathbb{Z}_8$, les égalités suivantes sont vérifiées :

$$u_i v_i \cdot w_i = w_i \cdot v_i u_i, \tag{6.5}$$

$$w_i \cdot u_{i+1} v_{i+1} = \widehat{u_{i+5}}\widehat{v_{i+5}} \cdot w_i, \tag{6.6}$$

$$v_{i-1} u_{i-1} \cdot w_i = w_i \cdot \widehat{v_{i+3}}\widehat{u_{i+3}}. \tag{6.7}$$

Démonstration. L'équation (6.5) est une conséquence immédiate de l'équation (6.3). On démontre les équations (6.6) et (6.7) également à l'aide du lemme 30 :

$$\begin{aligned} w_i u_{i+1} v_{i+1} &= w_i \widehat{w_{i-2}} w_i = \widehat{u_{i+5}}\widehat{v_{i+5}} w_i, \\ v_{i-1} u_{i-1} w_i &= w_i \widehat{w_{i+2}} w_i = w_i \widehat{v_{i+3}}\widehat{u_{i+3}}. \end{aligned}$$ □

D'autres égalités sur les facteurs u_i, v_i et w_i, où $i \in \mathbb{Z}_8 8$, sont très utiles pour les démontrer que certaines opérations sur les 2-carrés sont bien définies.

Lemme 33. Pour tout $i \in \mathbb{Z}_8$, les égalités suivantes sont vérifiées :

$$w_i u_{i+1} = \widehat{u_{i+5}}\widehat{w_{i+4}} \tag{6.8}$$

$$u_i w_{i+1} = \widehat{w_{i+5}}\widehat{u_{i+4}} \tag{6.9}$$

$$w_i \widehat{v_{i+3}} = v_{i+7}\widehat{w_{i+4}} \tag{6.10}$$

$$\widehat{v_i} w_{i+3} = \widehat{w_{i+7}} v_{i+4} \tag{6.11}$$

Démonstration. Ces égalités découlent encore une fois du lemme 30. Nous obtenons les équations (6.8) et (6.10) en comparant les préfixes et les suffixes des deux extrémités de

$$w_{i+1}\widehat{v_{i+4}} \cdot \widehat{u_{i+4}}\widehat{w_{i+3}} = w_{i+1}\widehat{w_{i+3}}w_{i+1}\widehat{w_{i+3}} = v_i u_i v_i u_i = v_i\widehat{w_{i-3}} \cdot w_{i-1}u_i,$$

et en corrigeant les indices en conséquence. D'autre part, les équations (6.9) et (6.11) proviennent de la série d'égalités

$$\widehat{w_{i-3}}\widehat{u_{i+4}} \cdot \widehat{v_{i+4}}w_{i-1} = \widehat{w_{i-3}}w_{i-1}\widehat{w_{i-3}}w_{i-1} = u_i v_i u_i v_i = u_i w_{i+1} \cdot \widehat{w_{i+3}}v_i. \quad \square$$

Un dernier lemme technique nous permet de traiter un cas dégénéré un peu plus loin dans ce chapitre.

Lemme 34. Supposons que $d_i = |w_{i+1}| + |w_{i+3}|$ divises $|w_i|$, c'est-à-dire que $u_i = \varepsilon$. Soit $g = \mathrm{pgcd}(|w_{i+2}|, d_{i+2})$. Alors

(i) $w_{i+1} = \widehat{w_{i+5}}$ et $w_{i+3} = \widehat{w_{i+7}}$;

(ii) Il existe deux mots $p, q \in \mathcal{F}^+$ et deux entiers $k, \ell > 0$ tels que

$$\begin{aligned} w_{i+1}w_{i+2}w_{i+3} &= p^k, \\ w_{i+5}w_{i+6}w_{i+7} &= q^k, \\ w_{i+6} &= \widehat{p}^{\ell}, \\ w_{i+2} &= \widehat{q}^{\ell}, \end{aligned}$$

où $|p| = |q| = g$ et $\ell = |w_{i+2}|/g$;

(iii) $pw_{i+1} = w_{i+1}\widehat{q}$ et $\widehat{q}w_{i+3} = w_{i+3}p$.

Démonstration. (i) Par le lemme 30, on obtient

$$\widehat{w_{i-3}}w_{i-1} = u_iv_i = v_i = v_iu_i = w_{i+1}\widehat{w_{i+3}}.$$

Puisque $|w_{i+1}| = |w_{i-3}|$ et $|w_{i-1}| = |w_{i+3}|$, on en déduit que $w_{i+1} = \widehat{w_{i-3}} = \widehat{w_{i+5}}$ et $w_{i+3} = \widehat{w_{i-1}} = \widehat{w_{i+7}}$.

(ii) Il suit de (i) que

$$\begin{aligned} w_{i+1}w_{i+3}\widehat{w_{i+6}} &= w_{i+1}\widehat{w_{i+7}}\widehat{w_{i+6}} \\ &= w_{i+1}w_{i+2}w_{i+3} \\ &= \widehat{w_{i+6}}\widehat{w_{i+5}}w_{i+3} \\ &= \widehat{w_{i+6}}w_{i+1}w_{i+3}. \end{aligned}$$

Il s'agit d'une équation de la forme $xy = yx$ et donc il existe un $p \in \mathcal{F}^*$ tel que

$$w_{i+1}w_{i+2}w_{i+3} = \widehat{w_{i+6}}\widehat{w_{i+5}}w_{i+3} = p^k$$

avec $|p| = \mathrm{pgcd}(|x|, |y|) = g$. En particulier, $w_{i+6} = \widehat{p}^{\ell}$. L'argument pour démontrer la formule avec q est exactement le même, en décalant tous les indices de quatre.

(iii) Par la partie (ii), nous savons que $w_{i+2} = \widehat{q}^{\ell}$ et que g est une période de $w_{i+1}w_{i+2}$. En outre, p est un préfixe de $w_{i+1}w_{i+2}$. Comme g est aussi une période de $w_{i+1}\widehat{q}$, nous avons que w_{i+1} est suffixe de $w_{i+1}\widehat{q}$ et donc que $w_{i+1}\widehat{q} = pw_{i+1}$. La preuve est similaire pour la seconde égalité. □

Il semble difficile de donner un critère simple qui vérifie si une DS-factorisation décrit bien une tuile 2-carrée. En effet, soit $w = w_0w_1 \cdots w_7$ le chemin obtenu à partir de la DS-factorisation $S = (w_i)_{0 \le i \le 7}$. Il est facile de vérifier que w est un chemin fermé, puisque

$\widehat{w_i w_{i+1}} = w_{i+4}w_{i+5}$ pour tout $i \in \mathbb{Z}_8$. En revanche, il n'existe aucun critère simple permettant de déterminer rapidement si w est auto-évitant : il faut plutôt recourir à un algorithme astucieux basé sur les arbres suffixes qui détecte en temps linéaire (et donc optimal) si w est auto-évitant (Brlek, Koskas et Provençal, 2011a).

Ceci dit, il est possible d'éliminer certains chemins fermés comme candidat à être auto-évitant. Tout d'abord, remarquons que la proposition 16 (voir page 90) se traduit directement comme suit pour les DS-factorisations :

Lemme 35. $\mathcal{T}(\llbracket w \rrbracket) = \pm 1$ si et seulement si $\textsc{f}(w_i) = \textsc{l}(w_{i+1})$ pour tout $i \in \mathbb{Z}_8$.

Par conséquent, le critère qui suit nous permet de déterminer avec certitude qu'un chemin fermé donné ne décrit par le contour d'une tuile. Cette condition peut sembler artificielle, mais elle apparaît naturellement plus bas dans l'énumération des 2-carrés :

Lemme 36. Supposons qu'il existe un indice $i \in \mathbb{Z}_8$ tel que $|w_i| + |w_{i+2}| = |w_{i+1}| + |w_{i+3}|$. Alors $\mathcal{T}(S) \notin \{-1, 1\}$, de sorte que S ne décrit pas le contour d'une tuile 2-carrée.

Démonstration. Soit $d = |w_i| + |w_{i+2}| = |w_{i-1}| + |w_{i+1}|$. Nous démontrons d'abord qu'il existe un indice $j \in \mathbb{Z}_8$ tel que $|w_{j-1}w_j| \geq d$ et $|w_j w_{j+1}| \geq d$. Supposons au contraire que ce n'est pas le cas. Alors il existe au moins un indice $k \in \mathbb{Z}_8$ tel que $|w_k| + |w_{k+1}| < d$ et $|w_{k+2}| + |w_{k+3}| < d$, de sorte que

$$2d = |w_k| + |w_{k+1}| + |w_{k+2}| + |w_{k+3}| < 2d,$$

ce qui est absurde.

Maintenant, nous savons du lemme 34 que les mots $x = w_{j-2}w_{j-1}w_j$, $y = w_{j-1}w_j w_{j+1}$ et $z = w_j w_{j+1}w_{j+2}$ ont tous la période d. De plus, x possède un suffixe de longueur au moins d qui est préfixe de y, et y possède un suffixe de longueur au moins d qui est un préfixe de z, de sorte que la période d se propage dans tout le mot $w_{j-2}w_{j-1}w_j w_{j+1}w_{j+2}$. Dans un premier temps, puisque $|w_{j-2}w_{j-1}w_j w_{j+1}| = 2d$, nous avons $\textsc{f}(w_{j-2}) = \textsc{f}(w_{j+2})$. D'autre part, $w_{j+2}w_{j+3} = \widehat{w_{j-1}}\widehat{w_{j-2}}$ implique que $\textsc{f}(w_{j+2}) =$

$\overline{\text{L}(w_{j-1})}$. Pour conclure, nous procédons encore une fois par contradiction. Supposons que $\mathcal{T}(S) \in \{-1, 1\}$. Alors le lemme 35 s'applique. En particulier, $\text{L}(w_{j-1}) = \text{F}(w_{j-2})$. En combinant ces trois égalités, nous obtenons

$$\text{F}(w_{j-2}) = \text{F}(w_{j+2}) = \overline{\text{L}(w_{j-1})} = \overline{\text{F}(w_{j-2})},$$

ce qui est impossible. Ainsi, $\mathcal{T}(S) \notin \{-1, 1\}$, tel que voulu. □

6.5 Réduction de 2-carrés

Soit $\mathcal{S}$ l'ensemble des DS-factorisations. Pour décrire la structure des 2-carrés, nous considérons certaines fonctions inversibles particulières agissant sur $\mathcal{S}$. Soit $\mathcal{S} = (w_i)_{i \in \mathbb{Z}_8}$ une DS-factorisation telle que $g = \text{pgcd}(|w_2|, d_2)$, $p = \text{Pref}_g(w_1 w_2 w_3)$ et $q = \text{Pref}_g(w_5 w_6 w_7)$. Nous définissons les opérateurs suivants :

$$\begin{aligned}
\text{SHRINK}(S) &= (w_0(v_0u_0)^{-1}, w_1, w_2, w_3, w_4(v_4u_4)^{-1}, w_5, w_6, w_7),\\
\text{L-SHRINK}(S) &= (p^{-1}w_0, p^{-1}w_1, w_2, w_3, q^{-1}w_4, q^{-1}w_5, w_6, w_7),\\
\text{L-SHRINK}(S) &= (w_0q^{-1}, w_1, w_2, w_3p^{-1}, w_4p^{-1}, w_5, w_6, w_7q^{-1}),\\
\text{SWAP}(S) &= (\widehat{w_4}, (v_1u_1)^{n_1}v_1, \widehat{w_6}, (v_3u_3)^{n_3}v_3,\\
&\quad \widehat{w_0}, (v_5u_5)^{n_5}v_5, \widehat{w_2}, (v_7u_7)^{n_7}v_7).
\end{aligned}$$

Les opérateurs de base SHRINK, L-SHRINK, R-SHRINK et SWAP sont généralisés dans le but d'agir sur n'importe quel w_i ($i \in \mathbb{Z}_8$) à l'aide d'un opérateur de décalage (ou *shift*). Nous notons donc par SHIFT l'opérateur défini par

$$\text{SHIFT}(w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_0).$$

Il est évident que SHIFT(S) est une DS-factorisation. En conséquence, on définit pour

chaque $i \in \mathbb{Z}_8$ et $\Theta \in \{\text{SWAP}, \text{SHRINK}, \text{L-SHRINK}, \text{R-SHRINK}\}$ les opérateurs $\Theta_i(S)$ par

$$\Theta_i(S) = \text{SHIFT}^{-i} \circ \Theta \circ \text{SHIFT}_i(S).$$

On applique donc le décalage inverse afin de conserver la position initiale des w_i. Clairement, $\Theta_0(S) = \Theta(S)$.

La proposition suivante montre que ces opérateurs sont bien définis sur $\mathcal{S}$ sous certaines conditions simples à vérifier, c'est-à-dire que le résultat est bien une DS-factorisation.

Proposition 22. Les énoncés suivants sont vérifiés :

(i) Si $|w_i| > d_i$, alors $\text{SHRINK}_i(S)$ est une DS-factorisation ;

(ii) Si $|w_i| = d_i$ et $|w_{i+1}| > g$, alors $\text{L-SHRINK}_i(S)$ est une DS-factorisation ;

(iii) Si $|w_i| = d_i$ et $|w_{i+7}| > g$, alors $\text{R-SHRINK}_i(S)$ est une DS-factorisation ;

(iv) Si u_{i+1}, u_{i+3}, u_{i+5} et u_{i+7} sont non vides, alors $\text{SWAP}_i(S)$ est une DS-factorisation.

Démonstration. Il suffit de montrer que les équations de la définition 10 sont vérifiées. Sans perte de généralité, il est suffisant de démontrer le cas $i = 0$ seulement.

(i) Remarquons que $n_0 = n_4 \geq 1$ puisque $|w_0| > d_0$. Soit $w_0' = (u_0v_0)^{n_0-1}u_0$ et $w_4' = (u_4v_4)^{n_4-1}u_4$. Nous souhaitons montrer que

$$\text{SHRINK}(S) = (w_0', w_1, w_2, w_3, w_4', w_5, w_6, w_7)$$

est une DS-factorisation. Nous savons de l'équation (6.6) que $w_7u_0v_0 = \widehat{u_4}\widehat{v_4}w_7$. Alors nous pouvons écrire

$$\widehat{u_4}\widehat{v_4}w_7w_0' = w_7u_0v_0w_0' = w_7w_0 = \widehat{w_4}\widehat{w_3} = \widehat{u_4}\widehat{v_4}\widehat{w_4'}\widehat{w_3}$$

et donc $w_7w_0' = \widehat{w_3w_4'}$. L'argument est semblable pour montrer que $w_0'w_1 = \widehat{w_4'w_5}$ (à l'aide de l'équation (6.10) $v_0u_0w_1 = w_1\widehat{v_4}\widehat{u_4}$. Ainsi, $\text{SHRINK}(S)$ est une DS-factorisation.

(ii) Tout d'abord, rappelons que la condition $|w_0| = d_0$ implique que $w_0 = v_0$, $u_0 = \varepsilon$ et

$n_0 = 1$. Soit $w_0 = pw'_0$, $w_1 = pw'_1$, $w_4 = qw'_4$ et $w_5 = qw'_5$. Nous voulons montrer que

$$\text{L-SHRINK}(S) = (w'_0, w'_1, w_2, w_3, w'_4, w'_5, w_6, w_7)$$

est une DS-factorisation. Il faut donc montrer que $w_7 w'_0 = \widehat{w_3 w'_4}$, $w'_0 w'_1 = \widehat{w'_4 w'_5}$ et $w'_1 w_2 = \widehat{w'_5 w_6}$. Puisque $n_0 = 1$ et par le lemme 30, nous avons $w_0 = v_0 = w_1 \widehat{w_3}$. Nous déduisons de cette égalité que $w'_0 = w'_1 \widehat{w_3}$. Par le même argument, nous pouvons écrire $w'_4 = w'_5 \widehat{w_7}$. Rappelons aussi du lemme 34 que $w_1 = \widehat{w_5}$ et $w_3 = \widehat{w_7}$. En particulier, puisque $pw'_1 = w_1 = \widehat{w_5} = \widehat{w'_5 \widehat{q}}$ a la période g, ceci entraîne que $w'_1 = \widehat{w'_5}$. Un raisonnement similaire nous permet d'écrire $w'_3 = \widehat{w'_7}$. Alors

$$w_7 w'_0 = w_7 w'_1 \cdot \widehat{w_3} = w_7 \widehat{w'_5} \cdot \widehat{w_3} = \widehat{w'_4 w'_3}$$

et

$$w'_0 w'_1 = w'_1 \cdot \widehat{w_3} w'_1 = \widehat{w'_5} \cdot w_7 \widehat{w'_5} = \widehat{w'_5 w'_4}.$$

Finalement, puisque $w_1 w_2 = \widehat{w_6 w_5}$ a la période g, nous avons $pw'_1 w_2 = \widehat{w_6} \widehat{w'_5 \widehat{q}} = \widehat{\widehat{q} w_6 w_5}$ ce qui démontre que $w'_1 w_2 = \widehat{w'_5 w_6}$ et L-SHRINK(S) est donc une DS-factorisation.

(iii) L'argument est semblable à celui donné en (ii).

(iv) Soit $(w'_i) = \text{SWAP}(S)$. Premièrement, nous montrons que $w'_0 w'_1 = \widehat{w'_4 w'_5}$. Nous savons de l'équation (6.6) que $\widehat{w_4} v_1 u_1 = \widehat{v_5 u_5 w_4}$ et de l'équation (6.11) que $\widehat{w_4} v_1 = \widehat{v_5} w_0$. Ainsi

$$w'_0 w'_1 = \widehat{w_4}(v_1 u_1)^{n_1} v_1 = (\widehat{v_5 u_5})^{n_1} \widehat{w_4} v_1 = (\widehat{v_5 u_5})^{n_1} \widehat{v_5} w_0 = \widehat{w'_4 w'_5}.$$

On démontre de façon similaire que $w'_2 w'_3 = \widehat{w'_6 w'_7}$ et $w'_3 w'_4 = \widehat{w'_7 w'_0}$. On en conclut que SWAP$(S)$ est bien une DS-factorisation. □

Lorsque les conditions (ii), (iii) et (iv) du lemme précédent sont vérifiées, alors il existe une périodicité locale dans le voisinage de w_i. Intuitivement, l'action des opérateurs SHRINK, L-SHRINK et R-SHRINK correspondent à la suppression d'une occurrence d'un motif répété comme l'illustrent les figures 6.6 et 6.7.

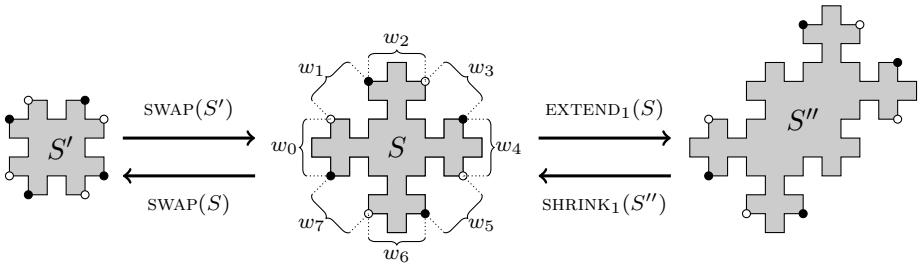


Figure 6.6: Effet des opérateurs EXTEND et SWAP sur la tuile de Fibonacci d'ordre $n = 2$. La DS-factorisation $S'' = \text{EXTEND}_1(S)$ est obtenue de S par extension des facteurs w_1 et w_5. D'autre part, la DS-factorisation $S' = \text{SWAP}(S)$ est obtenue par les échanges $w'_0 = \widehat{w_4}, w'_2 = \widehat{w_6}, w'_4 = \widehat{w_0}$ et $w'_6 = \widehat{w_2}$.

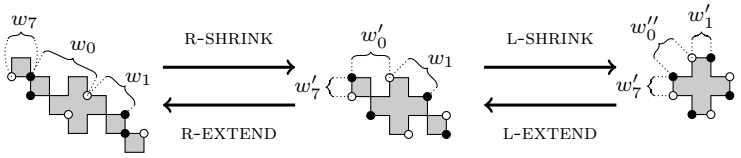


Figure 6.7: Effet des opérateurs R-SHRINK et L-SHRINK. L'opérateur R-SHRINK modifie les facteurs w_7 et w_0 alors que L-SHRINK affecte les facteurs w_0 et w_1.

Quant à l'opérateur SWAP, il est défini à l'aide des relations entre les mots w_i et les périodes $u_j v_j$. Tous ces opérateurs sont inversibles sous des conditions simples : les opérateurs inverses EXTEND, R-EXTEND et L-EXTEND sont définis à la section suivante.

La taille des DS-factorisations est réduite par ces opérateurs comme l'atteste la proposition suivante :

Proposition 23. Les énoncés suivants sont vérifiés :

(i) Si $|w_i| > d_i$, alors $|\text{SHRINK}_i(S)| < |S|$;

(ii) Si $|w_i| = d_i$ et $|w_{i+1}| > g$, alors $|\text{L-SHRINK}_i(S)| < |S|$;

(iii) Si $|w_i| = d_i$ et $|w_{i+7}| > g$, alors $|\text{R-SHRINK}_i(S)| < |S|$;

(iv) Si u_{i+1}, u_{i+3}, u_{i+5} et u_{i+7} sont non vides, alors

$$|v_{i+1}| + |v_{i+3}| < |u_{i+1}| + |u_{i+3}| \iff |\text{SWAP}_i(S)| < |S|.$$

Démonstration. Ces propriétés découlent directement de la définition des opérateurs.

□

Une autre propriété remarquable de ces opérateurs est qu'ils préservent le nombre d'enroulements des DS-factorisations manipulées :

Lemme 37. Le nombre d'enroulements $\mathcal{T}$ est invariant sous les opérateurs SHIFT, SHRINK, L-SHRINK, R-SHRINK et SWAP.

Démonstration. Il suffit de vérifier que les premières et dernières lettres des w_i sont préservées par les opérateurs. □

Il est intéressant de noter ici que le plus petit 2-carré est le pentamino en forme de croix ✚. À conjugaison et image miroir près, sa DS-factorisation est

$$b(✚) \equiv (\mathbf{0}, \mathbf{10}, \mathbf{1}, \mathbf{21}, \mathbf{2}, \mathbf{32}, \mathbf{3}, \mathbf{03}).$$

Nous disons d'une DS-factorisation S qu'elle se *réduit* à une autre DS-factorisation S' s'il existe une suite d'opérateurs $(\Theta_i)_{i\in I}$ telle que

(i) $\Theta_i \in \{\text{SHRINK}, \text{SWAP}, \text{L-SHRINK}, \text{R-SHRINK}\}$ pour $i \in I$;

(ii) $S' = (\Theta_n \circ \Theta_{n-1} \circ \cdots \circ \Theta_2 \circ \Theta_1)(S)$;

(iii) $|S_k| < |S_{k-1}|$ pour $k = 1, 2, \ldots, n$, où $S_k = (\Theta_k \circ \Theta_{k-1} \circ \cdots \circ \Theta_2 \circ \Theta_1)(S)$.

Autrement dit, à chaque étape, la taille de la solution doit diminuer. Nous sommes maintenant en mesure d'énoncer et de démontrer le théorème principal de cette section :

Théorème 16. Toute DS-factorisation se réduit à le polyomino en croix ✚ (premier ou composé).

Démonstration. Soit S un DS-factorisation telle que $\mathcal{T}(S) = \pm 1$. Il suffit de montrer que soit S est un pentamino premier ou composé, soit S peut être réduit à l'aide d'un des opérateurs SHRINK, SWAP, L-SHRINK ou R-SHRINK.

S'il existe un $i \in \{0, 1, 2, 3\}$ tel que $|w_i| > d_i$, alors S peut être réduite en $\text{SHRINK}_i(S)$. Aussi, s'il existe un indice i tel que $|u_{i+1}| + |u_{i+3}| > |v_{i+1}| + |v_{i+3}|$, alors S peut être réduite à $\text{SWAP}_i(S)$. Autrement, si ni SHRINK ni SWAP ne peut être appliqué à S, alors on en conclut que les n_i sont nécessairement égaux à 0 ou à 1.

Le cas $(n_0, n_1, n_2, n_3) = (0, 0, 0, 0)$ est impossible. En effet, supposons que nous soyons dans cette situation, c'est-à-dire que $w_i = u_i$ pour tout $i \in \mathbb{Z}_8$. Comme S ne se réduit à aucun $\text{SWAP}_i(S)$ peu importe la valeur de i, nous en concluons que $|u_{i+1}| + |u_{i+3}| \leq |v_{i+1}| + |v_{i+3}|$ pour tout i. En utilisant l'égalité $|v_i| = |w_{i-1}| + |w_{i+1}| - |u_i|$, ceci entraîne que $|u_i| + |u_{i+2}| \leq |u_{i+1}| + |u_{i+3}|$ pour tout $i \in \mathbb{Z}_8$. On en déduit que $|u_0| + |u_2| = |u_1| + |u_3|$. Or, par le lemme 36, on en tire que $\mathcal{T}(S) \neq \pm 1$, une contradiction avec le fait que le nombre d'enroulements est préservé par les opérateurs.

Supposons maintenant que $n_i = 1$ pour un certain indice i. Alors $u_i = \varepsilon$ et, par le lemme 31, nous avons $n_{i+1} = n_{i+3} = 0$. Soit $g = \gcd(|w_{i+2}|, |w_{i+1}| + |w_{i+3}|)$. Nous savons du corollaire 5 que $w_{i+1}w_{i+2}w_{i+3}$ a la période g. Si $n_{i+2} = 0$, alors nous avons en particulier que $|w_{i+2}| < |w_{i+1}| + |w_{i+3}|$. Ceci entraîne que $g < |w_{i+1}|$ ou $g < |w_{i+3}|$. Nous avons donc que S se réduit à $\text{L-SHRINK}_i(S)$ dans le premier cas et à $\text{R-SHRINK}_i(S)$ dans le second.

Considérons maintenant le cas $(n_0, n_1, n_2, n_3) = (1, 0, 1, 0)$. À ce moment-là, la DS-factorisation a la forme

$$(u_1\widehat{u_3}, u_1, u_3u_1, u_3, \widehat{u_1}u_3, \widehat{u_1}, \widehat{u_3}\widehat{u_1}, \widehat{u_3},$$

qui correspond exactement à celle d'un pentamino premier ou composé obtenu du morphisme homologue

$$\mathbf{0} \mapsto u_1, \mathbf{1} \mapsto u_3, \mathbf{2} \mapsto \widehat{u_1}, \mathbf{3} \mapsto \widehat{u_3}.$$

Le cas $(n_0, n_1, n_2, n_3) = (0, 1, 0, 1)$ mène à la même conclusion.

Remarquons que la réduction peut être appliquée jusqu'à l'obtention d'un pentamino premier ou composé. Le principe de descente infinie de Fermat s'appliquant, on a la garantie que le processus doit s'arrêter après un nombre fini d'itérations. □

Le théorème décrit un algorithme permettant de calculer la suite d'opérateurs à appliquer à un 2-carré donné pour le réduire à un pentamino composé. L'algorithme 4 donne

le pseudocode correspondant.

Algorithme 4 Réduction d'une tuile 2-carrée

fonction REDUCE(S)
  Entrée : Une DS-factorisation $S = (w_0,w_1,w_2,w_3,w_4,w_5,w_6,w_7)$
  Sortie : Une liste ordonnée d'opérateurs
  $L \leftarrow ()$
  tant que il n'existe aucun i tel que $|w_i| = d_i$ et $|w_{i+2}| = d_{i+2}$ **faire**
    si il existe i tel que $|w_i| > |w_{i-1}| + |w_{i+1}|$ **alors**
      $S \leftarrow \text{SHRINK}_i(S)$, $L \leftarrow L + (\text{SHRINK}_i)$
    sinon si il existe i tel que $|u_{i+1}| + |u_{i+3}| > |v_{i+1}| + |v_{i+3}|$ **alors**
      $S \leftarrow \text{SWAP}_i(S)$, $L \leftarrow L + (\text{SWAP}_i)$
    sinon ▷ Il existe i tel que $n_i = 1$ et $n_{i+1} = n_{i+2} = n_{i+3} = 0$
      $g \leftarrow \gcd(|w_{i-1}| + |w_{i+1}|, |w_{i+2}|)$
      si $g < |w_{i+1}|$ **alors**
        $S \leftarrow \text{L-SHRINK}_i(S)$, $L \leftarrow L + (\text{L-SHRINK}_i)$
      sinon ▷ $g < |w_{i+3}|$
        $S \leftarrow \text{R-SHRINK}_i(S)$, $L \leftarrow L + (\text{R-SHRINK}_i)$
      fin si
    fin si
  fin tant que
  retourner L ▷ S décrit le contour d'un pentamino premier ou composé
fin fonction

L'objectif de la section suivante est d'inverser les opérateurs de réduction dans le but de générer toutes les DS-factorisations, et par conséquent, toutes les tuiles 2-carrées.

6.6 Génération des 2-carrés

Les idées de la section précédente mènent naturellement à nous demander s'il est possible de produire un algorithme efficace de génération des 2-carrés en inversant les opérateurs de réduction. Dans cette section, nous présentons ces opérateurs inverses, nous décrivons certaines propriétés de commutativité qu'ils possèdent et nous fournissons un algorithme permettant de générer tous les 2-carrés ayant un périmètre donné.

Soit $S = (w_i)_{i\in\mathbb{Z}_8}$ une DS-factorisation. Soit $g = \text{pgcd}(|w_2|, d_2)$, $p = \text{Pref}_g(w_1w_2w_3)$ et $q = \text{Pref}_g(w_5w_6w_7)$. Nous définissons les opérateurs suivants :

$$\text{EXTEND}(S) \quad = \quad (w_0(v_0u_0), w_1, w_2, w_3, w_4(v_4u_4), w_5, w_6, w_7),$$

$$\begin{aligned}\text{L-EXTEND}(S) &= (pw_0, pw_1, w_2, w_3, qw_4, qw_5, w_6, w_7),\\ \text{R-EXTEND}(S) &= (w_0q, w_1, w_2, w_3p, w_4p, w_5, w_6, w_7q).\end{aligned}$$

Nous montrons plus bas que, sous certaines conditions simples, EXTEND, L-EXTEND et R-EXTEND sont les inverses de SHRINK, L-SHRINK et R-SHRINK respectivement. Aussi, il est à noter que SWAP est une involution et donc son propre inverse.

Tout d'abord, nous devons nous assurer que ces opérateurs sont bien définis sur $\mathcal{S}$ et sous quelles hypothèses :

Proposition 24. Soit $S = (w_i)_{i\in\mathbb{Z}_8}$ une DS-factorisation et $p, q \in \mathcal{F}^*$ défini plus haut. Alors

(i) EXTEND(S) est une DS-factorisation ;

(ii) si $|w_0| = d_0$, alors L-EXTEND(S) est une DS-factorisation ;

(iii) si $|w_0| = d_0$, alors R-EXTEND(S) est une DS-factorisation.

Démonstration. Il suffit de montrer que les équations de la définition 10 sont vérifiées.

(i) Soit $w'_0 = (u_0v_0)^{n_0+1}$ et $w'_4 = (u_4v_4)^{n_4+1}u_4$. Nous devons montrer que

$$\text{EXTEND}(S) = (w'_0, w_1, w_2, w_3, w'_4, w_5, w_6, w_7)$$

est une DS-factorisation. Tout d'abord, nous montrons que $\widehat{w_3w'_4} = w_7w'_0$. Puisque $w_7u_0v_0 = \widehat{u_4}\widehat{v_4}w_7$ (par l'équation 6.6), nous pouvons écrire

$$\widehat{w_3w'_4} = \widehat{w'_4}\widehat{w_3} = \widehat{u_4}\widehat{v_4}\widehat{w_4}\cdot\widehat{w_3} = \widehat{u_4}\widehat{v_4}w_7w_0 = w_7\cdot u_0v_0w_0 = w_7w'_0,$$

ce qui démontre l'égalité. On démontre de façon semblable que $\widehat{w'_4w_5} = w'_0w_1$ à l'aide des égalités $v_0u_0w_1 = w_1\widehat{v_4}\widehat{u_4}$, par l'équation (6.10), et $w_0w_1 = \widehat{w_5}\widehat{w_4}$. On en conclut que EXTEND(S) est une DS-factorisation. À noter que la démonstration est très semblable à celle utilisée pour montrer que SHRINK(S) est une DS-factorisation lorsque $n_0 \neq 0$, où un motif répété est supprimé plutôt que ajouté.

(ii) Rappelons dans un premier temps que la condition $|w_0| = d_0$ implique que $w_0 = v_0$, $u_0 = \varepsilon$ et $n_0 = 1$. Soit $w'_0 = pw_0$, $w'_1 = pw_1$, $w'_4 = qw_4$ et $w'_5 = qw_5$. Nous voulons montrer que

$$\text{L-EXTEND}(S) = (w'_0, w'_1, w_2, w_3, w'_4, w'_5, w_6, w_7)$$

est une DS-factorisation. Il suffit donc de montrer que $w_7 w'_0 = \widehat{w_3 w'_4}$, $w'_0 w'_1 = \widehat{w'_4 w'_5}$ et $w'_1 w_2 = \widehat{w'_5 w_6}$. Remarquons que $pw_1 = w_1\widehat{q}$. En effet, puisque $w_2 = \widehat{q}^{\ell}$ et $w_1 w_2$ a la période $|p| = |q|$ avec $\text{Pref}_{|p|}(w_1 w_2) = p$ et $\text{Suff}_{|p|}(w_1 w_2) = \widehat{p}$, nous avons $pw_1 w_2 = w_1 w_2 \widehat{q} = w_1 \widehat{q}^{\ell+1} = w_1 \widehat{q} w_2$. Remarquons aussi que, en vertu du lemme 30, $w_1 = \widehat{w_5}$ et $w_3 = \widehat{w_7}$ puisque $u_0 = \varepsilon$. Finalement, en combinant ces deux égalités avec le fait que $w_0 = v_0$, $w_0 = w_1\widehat{w_3}$ et $w_4 = \widehat{w_1} w_3$, nous concluons que

$$\begin{aligned}
w_7 w'_0 &= w_7 p w_0 = w_7 p w_1 \widehat{w_3} = w_7 w_1 \widehat{q w_3} = \widehat{w_3} w_1 \widehat{q w_3} = \widehat{w_3 w'_4}, \\
w'_0 w'_1 &= p w_0 p w_1 = p w_1 \widehat{w_3} p w_1 = w_1 \widehat{q w_3} w_1 \widehat{q} = \widehat{w_5 \widehat{q}} \cdot \widehat{w_4 \widehat{q}} = \widehat{w'_4 w'_5}, \\
w'_1 w_2 &= p w_1 w_2 = w_1 w_2 \widehat{q} = \widehat{w_6 \widehat{w_5 \widehat{q}}} = \widehat{w'_5 w_6},
\end{aligned}$$

de sorte que L-EXTEND(S) est une DS-factorisation. Encore une fois, les arguments sont semblables à ceux utilisés pour montrer que L-SHRINK(S) est une DS-factorisation.

(iii) La démonstration est semblable à celle fournie en (ii), les motifs répétés étant ajoutés dans w_7 et w_3 plutôt que w_1 et w_5. □

Nous avons tous les éléments pour décrire les opérateurs inverses :

Proposition 25. Soit $S = (w_i)_{i \in \mathbb{Z}_8}$ une DS-factorisation.

(i) $(\text{SHIFT}^7 \circ \text{SHIFT})(S) = \text{SHIFT}^8(S) = S$.

(ii) $(\text{SHRINK} \circ \text{EXTEND})(S) = S$ et si $|w_0| > d_0$, alors $(\text{EXTEND} \circ \text{SHRINK})(S) = S$.

(iii) Si $|w_0| = d_0$, alors $(\text{L-SHRINK} \circ \text{L-EXTEND})(S) = S$. En outre, si $|w_1| > g$, alors $(\text{L-EXTEND} \circ \text{L-SHRINK})(S) = S$.

(iv) Si $|w_0| = d_0$, alors $(\text{R-SHRINK} \circ \text{R-EXTEND})(S) = S$. En outre, si $|w_7| > g$, alors $(\text{R-EXTEND} \circ \text{R-SHRINK})(S) = S$.

(v) Si u_{i+1}, u_{i+3}, u_{i+5}, u_{i+7} sont tous non vides, alors $(\text{SWAP}\circ\text{SWAP})(S) = \text{SWAP}^2(S) = S$.

Démonstration. (i) Évident.

(ii) Il est immédiat que les opérateurs SHRINK et EXTEND ont des effets inverses sur les facteurs u_0, v_0, u_4 et v_4.

(iii) Soit $w'_0 = pw_0$, $w'_1 = pw_1$, $w'_4 = qw_4$ et $w'_5 = qw_5$. Alors

$$S' = \text{L-EXTEND}(S) = (w'_0, w'_1, w_2, w_3, w'_4, w'_5, w_6, w_7).$$

Tout d'abord, nous avons que L-SHRINK(S') est défini puisque la condition $|w'_0| = d'_0$ est vérifiée. En effet, $|w'_0| = |w_0| + |p| = d_0 + |p| = |w_7| + |pw_1| = d'_0$. Nous obtenons alors

$$\begin{aligned} \text{L-SHRINK}(S') &= ((p')^{-1}w'_0, (p')^{-1}w'_1, w_2, w_3, (q')^{-1}w'_4, (q')^{-1}w'_5, w_6, w_7) \\ &= (p^{-1}pw_0, p^{-1}pw_1, w_2, w_3, q^{-1}qw_4, q^{-1}qw_5, w_6, w_7) \\ &= S \end{aligned}$$

puisque $p' = \text{Pref}_{|p|}(w'_1 w_2) = p$ et $q' = \text{Pref}_{|p|}(w'_5 w_6) = q$. Alors L-SHRINK$(S') = (\text{L-SHRINK} \circ \text{L-EXTEND})(S) = S$.

(iv) La démonstration est semblable à celle présentée en (iii).

(v) Découle directement de la définition de l'opérateur SWAP. □

Bien que les opérateurs EXTEND, SWAP, L-EXTEND et R-EXTEND fournissent un algorithme de génération de tous les 2-carrés (premiers ou composés), il est possible d'améliorer grandement sa performance en observant certaines propriétés de commutativité (voir figure 6.8).

Proposition 26. Soit $\Theta \in \{\text{EXTEND}, \text{SWAP}, \text{L-EXTEND}, \text{R-EXTEND}\}$ et $i \in \mathbb{Z}_8$. Alors

(i) $\Theta_i = \Theta_{i+4}$;

(ii) $\text{SWAP}_i = \text{SWAP}_{i+2}$;

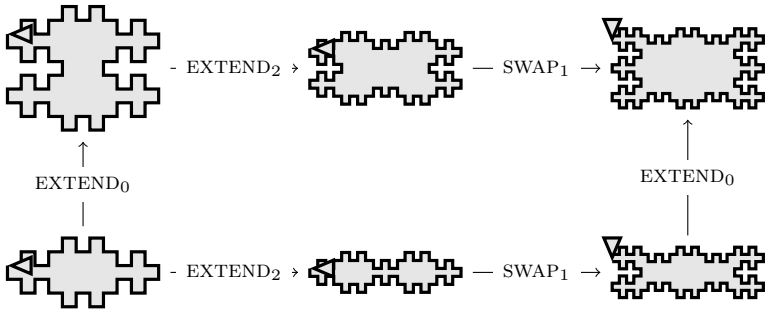


Figure 6.8: Deux façons distinctes de générer la même tuile 2-carrée. Le diagramme commute en vertu de la proposition 26(iii) et (iv).

(iii) $\textsc{extend}_{i+2} \circ \textsc{extend}_i = \textsc{extend}_i \circ \textsc{extend}_{i+2}$;

(iv) $\textsc{extend}_{i+1} \circ \textsc{swap}_i = \textsc{swap}_i \circ \textsc{extend}_{i+1}$;

(v) $\textsc{l-extend}_i \circ \textsc{r-extend}_i = \textsc{r-extend}_i \circ \textsc{extend}_i$;

Démonstration. Se vérifie à l'aide de la définition des opérateurs. □

En nous basant sur les résultats de cette section, il est possible d'écrire un algorithme générant tous les 2-carrés de périmètre au plus n (voir algorithme 5).

Algorithme 5 Génération des tuiles 2-carrées

fonction GENERATE(n)
  Entrée : Le périmètre maximal n des 2-carrés
  Sortie : L'ensemble de toutes les tuiles 2-carrées de périmètre au plus n
  $T \leftarrow \emptyset$
  $Q \leftarrow \{P : P$ est une croix composée de périmètre au plus $n\}$
  tant que $Q \neq \emptyset$ **faire**
    $t \leftarrow$ POP(Q)
    si $[t]$ est un polyomino **alors** $T \leftarrow T \cup \{[t]\}$
    $C \leftarrow \{\textsc{extend}_i(t) : i = 0,1,2,3\}$
    $C \leftarrow C \cup \{\textsc{swap}_i(t) : i = 0,1\}$
    $C \leftarrow C \cup \{\textsc{l-extend}_i(t) : i = 0,1,2,3 \text{ et } |w_i| = d_i\}$
    $C \leftarrow C \cup \{\textsc{r-extend}_i(t) : i = 0,1,2,3 \text{ et } |w_i| = d_i\}$
      ▷ C contient toutes les tuiles pouvant être générées à partir de t
    $Q \leftarrow Q \cup \{c \in C : |t| < |c| \leq n\}$
  fin tant que
  retourner T ▷ T contient toutes les tuiles de périmètre au plus n
fin fonction

En revanche, l'algorithme 5 peut être considérablement amélioré à l'aide des observations suivantes :

1. La proposion 26 indique qu'il est possible d'éviter de générer plusieurs fois une même tuile en imposant une priorité aux opérateurs. Par exemple, nous pourrions permettre l'application de l'opérateur $\textsc{extend}_2$ seulement si le dernier opérateur appliqué est soit $\textsc{extend}_0$ ou $\textsc{swap}_1$, c'est-à-dire que ces deux derniers opérateurs auraient priorité sur $\textsc{extend}_2$;
2. Nous croyons également que les opérateurs L-EXTEND et R-EXTEND sont superflus. Plus précisément, il semble que dès qu'un de ces opérateurs est appliqué, alors aucun des chemins générés n'est auto-évitant (voir figure 6.9) ;
3. Les opérateurs de génération préservent la composition des chemins (ou des tuiles), c'est-à-dire que si on applique, par exemple, l'opérateur EXTEND à une tuile composée, alors le résultat sera aussi une tuile composée. Ce point est abordé dans la section qui suit.

La figure 6.9 illustre une trace partielle des DS-factorisations explorées par l'algorithme 5 à partir du pentamino premier ✚. Il nous est difficile pour le moment d'évaluer la complexité de l'algorithme 5. En effet, nous ne savons pas exactement combien de DS-factorisations donnent effectivement des 2-carrés, puisqu'il est difficile de déterminer si les chemins sont auto-évitants ou non. Par contre, notre algorithme est clairement plus efficace que la stratégie naïve qui consiste à énumérer tous les chemins fermés de longueur n sur $\mathcal{F}$ puis de vérifier s'ils décrivent effectivement un polyomino. Une analyse plus fine des lignes 5 et 8 de l'algorithme 5 s'impose. Finalement, remarquons qu'il n'est pas difficile d'énumérer les 2-carrés en ordre croissant de périmètres : il suffit de munir l'ensemble Q d'une structure de file à priorité.

6.7 Tuiles 2 carrées premières et composées

Rappelons qu'un morphisme φ sur $\mathcal{F}$ est dit homologue si

$$\varphi(\widehat{w}) = \widehat{\varphi(w)}, \quad \text{pour n'importe quel mot } w \in \mathcal{F}^*$$

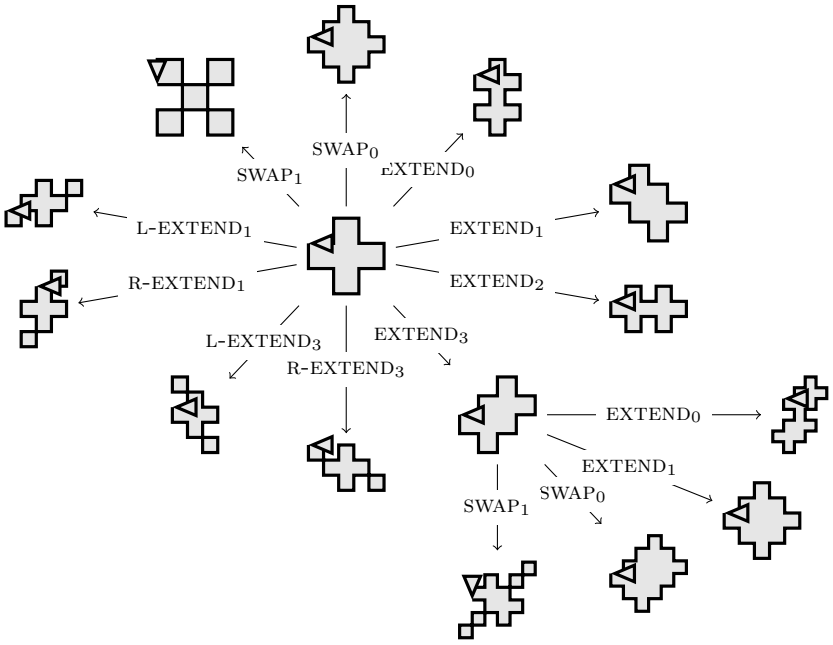


Figure 6.9: Sous-arbre de l'espace des DS-factorisations générées par l'algorithme 5 à partir du pentamino premier.

et qu'un polyomino est dit premier s'il n'existe aucune façon de le paver de façon carrée avec un autre polyomino plus petit autre que le carré unité. Pour le reste de la section, nous supposons que φ est un morphisme homologue, que $S = (w_i)_{i\in\mathbb{Z}_8}$ est une DS-factorisation et nous posons

$$\varphi(S) = (\varphi(w_0), \varphi(w_1), \varphi(w_2), \varphi(w_3), \varphi(w_4), \varphi(w_5), \varphi(w_6), \varphi(w_7)).$$

Nous montrons dans un premier temps que la composition de morphisme préserve les 2-carrés :

Lemme 38. Soit $S = (w_i)_{i\in\mathbb{Z}_8}$ une DS-factorisation. Alors $\varphi(S)$ est une DS-factorisation aussi.

Démonstration. Nous vérifions seulement la première condition de la définition de

DS-factorisation, les autres vérifications étant semblables. Nous trouvons

$$\widehat{\varphi(w_0)\varphi(w_1)} = \widehat{\varphi(w_0w_1)} = \varphi(\widehat{w_0w_1}) = \varphi(\widehat{w_4w_5}) = \widehat{\varphi(w_4w_5)} = \widehat{\varphi(w_4)\varphi(w_5)}. \quad \square$$

Soit $w'_i = \varphi(w_i)$ et $d'_i = |w'_{i+1}| + |w'_{i+3}|$ pour $i \in \mathbb{Z}_8$. Il suit du lemme 30 qu'il existe des mots u'_i, v'_i et un nombre n'_i tels que

$$\begin{aligned} \widehat{w'_{i-3}}w'_{i-1} &= u'_iv'_i \\ w'_i &= (u'_iv'_i)^{n'_i}u'_i \\ w'_{i+1}\widehat{w'_{i+3}} &= v'_iu'_i, \end{aligned}$$

où $0 \leq |u'_i| < d'_i$, pour chaque $i \in \mathbb{Z}_8$.

Le lemme suivant est particulièrement utile et montre que les facteurs u_i et v_i sont eux aussi préservés par φ.

Lemme 39. Pour tout $i \in \mathbb{Z}_8$, $\varphi(u_i) = u'_i$ et $\varphi(v_i) = v'_i$.

Démonstration. Nous présentons la démonstration pour $i = 0$. Premièrement, remarquons que

$$u'_0v'_0 = \widehat{w'_5}w'_7 = \widehat{\varphi(w_5)}\varphi(w_7) = \varphi(\widehat{w_5})\varphi(w_7) = \varphi(\widehat{w_5}w_7) = \varphi(u_0v_0).$$

Il ne reste qu'à montrer que $|u'_0| = |\varphi(u_0)|$. Nous avons aussi

$$(u'_0v'_0)^{n'_0}u'_0 = w'_0 = \varphi(w_0) = \varphi((u_0v_0)^{n_0}u_0) = \varphi(u_0v_0)^{n_0}\varphi(u_0) = (u'_0v'_0)^{n_0}\varphi(u_0).$$

Ainsi, si $L = |w'_0|$, alors

$$d'_0 \cdot n'_0 + |u'_0| = L = d'_0 \cdot n_0 + |\varphi(u_0)|$$

avec $0 \leq |u'_0| < d'_0$ et $0 \leq |\varphi(u_0)| < |\varphi(u_0v_0)| = |u'_0v'_0| = d'_0$. Or, il est connu que le quotient n'_0 et le reste $|u'_0|$ de la division euclidienne de L par d'_0 sont uniques, ce qui

entraîne que $n'_0 = n_0$ et $|u'_0| = |\varphi(u_0)|$. Nous concluons que $\varphi(u_0) = u'_0$ et $\varphi(v_0) = v'_0$. □

Soient $S = (w_i)_{i\in\mathbb{Z}_8}$ et $S' = \varphi(S)$ deux DS-factorisations, $g = \mathrm{pgcd}(|w_2|, d_2)$, $g' = \mathrm{pgcd}(|w'_2|, d'_2)$, $p = \mathrm{Pref}_g(w_1w_2w_3)$, $p' = \mathrm{Pref}_{g'}(w'_1w'_2w'_3)$ et $q = \mathrm{Pref}_g(w_5w_6w_7)$, $q' = \mathrm{Pref}_{g'}(w'_5w'_6w'_7)$ (voir la définition des opérateurs L-EXTEND et R-EXTEND). Comme dans le cas des mots u_i et v_i, la structure induite par φ est préservée sur p, p', q et q' :

Lemme 40. Soient p, p', q, q' les mots tels que définis pour les opérateurs L-EXTEND et R-EXTEND pour S et $\varphi(S)$. Alors, pour n'importe quel $i \in \mathbb{Z}_8$, $\varphi(p) = p'$ et $\varphi(q) = q'$.

Démonstration. Il suit directement du lemme 34(ii) que

$$\begin{aligned}(p')^k &= w'_0w'_1w'_2 = \varphi(w_0w_1w_2) = \varphi(p^k) = (\varphi(p))^k\\ (q')^k &= w'_4w'_5w'_6 = \varphi(w_4w_5w_6) = \varphi(q^k) = (\varphi(q))^k\end{aligned}$$

et le résultat en découle. □

Théorème 17. Pour tout $i \in \mathbb{Z}_8$, φ commute avec EXTEND_i, SWAP_i, L-EXTEND et R-EXTEND.

Démonstration. Il suffit d'utiliser le fait que les facteurs u_i, v_i, p et q sont préservés par φ. À titre d'exemple, nous montrons que SWAP_0 et φ commutent. Nous trouvons

$$\begin{aligned}\varphi(\mathrm{SWAP}_0(S)) &= (\varphi(\widehat{w_4}), \varphi((v_1u_1)^{n_1}v_1), \cdots)\\ &= (\widehat{\varphi(w_4)}, (\varphi(v_1)\varphi(u_1))^{n_1}\varphi(v_1), \cdots)\\ &= (\widehat{w'_4}, (v'_1u'_1)^{n'_1}v'_1, \cdots)\\ &= \mathrm{SWAP}_0(w'_0, w'_1, w'_2, w'_3, w'_4, w'_5, w'_6, w'_7)\\ &= \mathrm{SWAP}_0(\varphi(S)),\end{aligned}$$

tel que voulu. La démonstration des autres égalités est similaire. □

Le fait suivant découle directement de ce qui précède :

Corollaire 6. Soit $i \in \mathbb{Z}_8$. Alors

(i) EXTEND$_i$, L-EXTEND, R-EXTEND et SWAP$_i$ préservent les tuiles composées.

(ii) SHRINK$_i$, L-SHRINK, R-SHRINK et SWAP$_i$ préservent les tuiles premières.

En outre, nous savons maintenant générer toutes les tuiles premières et exactement chacune d'entre elles à partir du pentamino ✚.

Corollaire 7. Soit D une tuile 2-carrée. Si D est première, alors D se réduit au pentamino ✚.

6.8 Propriétés centrosymétriques des 2-carrés

Dans la suite, nous nous intéressons à la démonstration de la conjecture 3 qui affirme que tout 2-carré premier admet des factorisations palindromiques ou, de façon équivalente, est fixé par rotation d'angle π. Nous disons d'un polyomino satisfaisant ces conditions qu'il est *centrosymétrique*.

La palindromicité des DS-factorisations se traduit simplement sur les facteurs w_i, u_i et v_i grâce aux différentes équations sur les mots utilisés à travers ce chapitre.

Lemme 41. Les conditions suivantes sont équivalentes :

(i) $w_i w_{i+1}$ est un palindrome pour tout $i \in \mathbb{Z}_8$;

(ii) $w_i = \overline{w_{i+4}}$ pour tout $i \in \mathbb{Z}_8$;

(iii) $u_i = \overline{u_{i+4}}$ et $v_i = \overline{v_{i+4}}$ pour tout $i \in \mathbb{Z}_8$.

Démonstration. On montre d'abord que (i) et (ii) sont équivalents. Puisque $w_i w_{i+1} = \widehat{w_{i+5}}\widehat{w_{i+4}}$, nous avons que $w_i w_{i+1}$ est un palindrome si et seulement si $\widetilde{w_{i+1}}\widetilde{w_i} = w_i w_{i+1} = \widehat{w_{i+5}}\widehat{w_{i+4}}$. Or, $|w_i| = |w_{i+4}|$, ce qui entraîne le résultat.

Vérifions maintenant que (ii) et (iii) sont équivalents. Puisque $|u_i| = |u_{i+4}|$ et $|v_i| = |v_{i+4}|$ pour tout i, nous avons $w_i = \overline{w_{i+4}}$ pour tout i si et seulement si

$$u_i v_i = \widehat{w_{i-3}} w_{i-1} = \widetilde{w_{i+1}}\,\overline{w_{i+3}} = \overline{\widehat{w_{i+1}} w_{i+3}} = \overline{u_{i+4} v_{i+4}}.$$

□

Par ailleurs, les opérateurs sur les 2-carrés présentés plus haut préservent tous les palindromes dans le sens suivant.

Lemme 42. Soit S une DS-factorisation telle que $w_i w_{i+1}$ est un palindrome pour tout $i \in \mathbb{Z}_8$,

$$\Theta \in \{\text{SHIFT}, \text{EXTEND}, \text{SHRINK}, \text{SWAP}, \text{L-SHRINK}, \text{R-SHRINK}, \text{L-EXTEND}, \text{R-EXTEND}\}$$

et $S' = \Theta(S)$ une autre DS-factorisation. Alors $w'_i w'_{i+1}$ est un palindrome pour tout $i \in \mathbb{Z}_8$.

Démonstration. (SHIFT) L'opérateur SHIFT préserve exactement les facteurs $w_i w_{i+1}$, le résultat est immédiat dans ce cas.

(EXTEND) Considérons maintenant l'opérateur EXTEND. Nous devons démontrer que $w'_0 w'_1 = w_0 \cdot w_1 v_1 u_1$ et $w'_1 w'_2 = w_1 u_1 v_1$ sont des palindromes. Par les équations (6.5), (6.8) et (6.11), nous pouvons écrire $w_0 w_1 v_1 u_1 = \widehat{u_5}\widehat{v_5} w_0 w_1$. En outre, par le lemme 41(iii), $\widehat{u_5}\widehat{v_5} = \widetilde{u_1}\widetilde{v_1}$, ceci démontre que $w_0 w_1 v_1 u_1$ est un palindrome. On démontre de façon semblable que $w_1 v_1 u_1 w_2 = w_1 w_2 \widehat{v_5}\widehat{u_5} = u_1 v_1 w_1 w_2$ est un palindrome.

(SHRINK) Les idées sont les mêmes que pour l'opérateur EXTEND, mais on doit supprimer un motif répété plutôt qu'en ajouter un.

(SWAP) Il suit du lemme 41 et des équations (6.6) et (6.11) que

$$\begin{aligned}
\widetilde{w'_0 w'_1} &= \widetilde{v_1}(\widetilde{u_1}\widetilde{v_1})^{n_1}\overline{w_4} \\
&= \widehat{v_5}(\widehat{u_5}\widehat{v_5})^{n_1} w_0 \\
&= \widehat{v_5} w_0 (u_1 v_1)^{n_1} \\
&= \widehat{w_4} v_1 (u_1 v_1)^{n_1} \\
&= w'_0 w'_1.
\end{aligned}$$

L'argument est le même pour démontrer que les autres facteurs $w'_i w'_{i+1}$ sont aussi des palindromes.

(R-SHRINK) Considérons les opérateurs R-SHRINK$_0$ et R-SHRINK$_2$. Alors $u_1 = \varepsilon$ et $w_1 = v_1$ et donc $w_1 = w_2 w_0$. En particulier, $w_0 w_1$ et $w_1 w_2$ sont des palindromes si et seulement si w_0 et w_2 sont des palindromes puisque $w_0 w_1 = w_0 w_2 w_0$ et $w_1 w_2 = w_2 w_0 w_2$. Il suffit donc de montrer que w_0' et w_2' sont aussi des palindromes. Or, $w_2 w_3 w_4 = q^{k+\ell}$ où

$$w_3 = \widehat{w_0} w_2 = q^{\ell}$$

où $\ell \geq 2$. Écrivons $\widehat{w_0} = (q_1 q_2)^{\alpha} q_1$ et $w_2 = (q_2 q_1)^{\beta} q_2$, où $\alpha + \beta = \ell$. Comme $\ell \geq 2$, nous avons $\alpha \neq 0$ ou $\beta \neq 0$. Ceci implique que q_1 et q_2 sont des palindromes puisque w_0 et w_2 le sont. Nous concluons finalement que $w_0' = w_0 (\widehat{q_2}\widehat{q_1})^{-1} = (\widehat{q_2}\widehat{q_1})^{\alpha-1}\widehat{q_2}$ et $w_2' = p^{-1} w_2 = (q_2 q_1)^{\beta-1} q_2$ sont des palindromes.

La démonstration pour les opérateurs L-SHRINK, R-EXTEND, L-EXTEND se fait de façon semblable. □

Nous sommes maintenant en mesure de répondre à la conjecture 3 :

Théorème 18. Toute tuile 2-carrée première est centrosymétrique.

Démonstration. Si D est une tuile 2-carrée première, alors elle se réduit à la croix ✚ qui est constituée de palindromes. Or, les opérateurs inverses préservent tous les palindromes $w_i w_{i+1}$, d'où le résultat. □

6.9 Problèmes ouverts

L'énumération des tuiles 2-carrées révèle une structure combinatoire riche et complexe qu'on est en mesure de décrire à l'aide d'équations sur les mots. Bien que nous ayons étudié en détail la génération de celles-ci à l'aide d'opérateurs possédant des propriétés remarquables, certains éléments restent à approfondir.

Par exemple, nous avons observé informatiquement que tous les chemins fermés générés en utilisant au moins une fois les opérateurs R-SHRINK et L-SHRINK contiennent un point de croisement. Ceci nous mène naturellement à conjecturer que seuls les opérateurs

SHRINK et SWAP suffisent à réduire un tuile 2-carrée, de sorte que seuls les opérateurs EXTEND et SWAP sont nécessaires dans un but de génération exhaustive.

Conjecture 4. Soit S une DS-factorisation codant un 2-carré. Alors

(i) S est un pentamino premier ou composé ou

(ii) $\text{SHRINK}_i(S)$ est un 2-carré plus petit que S pour un certain $i \in \mathbb{Z}_8$ ou

(iii) $\text{SWAP}_i(S)$ est un 2-carré plus petit que S pour un certain $i \in \mathbb{Z}_8$.

Il serait aussi intéressant de vérifier si tout 2-carré peut être généré de façon unique à partir du pentamino ✚ en tenant compte des propriétés commutatives des opérateurs énoncées à la proposition 26.

En dernier lieu, il semble naturel d'étendre les opérateurs de réduction et de génération aux tuiles n-hexagonales, qui présentent également des périodicité locale, bien qu'il existe des tuiles n-hexagonales pour tout entier $n \geq 1$.

CONCLUSION

Dans cette thèse, nous avons étudié différents problèmes se trouvant à l'intersection de la combinatoire des mots et de la géométrie discrète. Nous avons mis en lumière l'intérêt et les avantages d'une telle approche, en présentant la solution à problèmes portant sur les pavages. Nous avons également contribué à l'étude de la complexité palindromique dans les mots infinis et facilité la compréhension de la structure des codages de rotations et des mots pseudostandards généralisés.

Il est intéressant de mentionner le fait que notre article sur les codages de rotations (Blondin Massé et al., 2011) ainsi que l'un des arbitres l'ayant examiné ont mis en évidence une erreur mineure de l'article de Katok (Katok, 1980). Il faut en effet remplacer toutes les occurrences de $m+1$ par $m+2$ dans le lemme 2.

Il reste de nombreux problèmes à résoudre dans les domaines étudiés et nous sommes convaincus que les outils présentés dans cette thèse favoriseront la résolution de certains d'entre eux. Par exemple, nous connaissons bien peu de choses sur la famille des mots pseudostandards généralisés qui est pourtant naturelle et contient plusieurs mots célèbres tels que le mot de Thue-Morse et les suites sturmiennes. Il existe également plusieurs problèmes secondaires à considérer sur la clôture palindromique itéré. Par exemple, étant donné un pseudopalindrome p, quel est le mot u le plus court dont p est la clôture pseudopalindromique ?

Il reste de nombreuses avenues à explorer concernant les pavages bidimensionnels par translation. Une idée serait de vérifier quels résultats et propriétés peuvent être étendus au cas où le contour des tuile est délimité par des courbes lisses par morceaux. Aussi, il est fort probable que les idées permettant de caractériser les tuiles n-carrées mènent à une caractérisation des tuiles n-hexagonales ou des tuiles sur d'autres réseaux, tels que

la grille hexagonale.

Aussi, bien que nous ayons considéré des problèmes en dimension deux, il semble raisonnable de croire que certaines idées peuvent être plongées dans un espace de dimension trois. C'est d'ailleurs un de mes objectifs dans les années à venir. Plus précisément, il s'agit d'étudier les pavages réguliers engendrés par les *polycubes*, une génération naturelle des polyominos en dimension trois. Bien que nous ne puissions pas décrire ces objets par un mot de contour, différentes représentations en trois dimensions basées sur la combinatoire des mots ont été proposées dans la littérature (Vuillon, 1998; Arnoux, Berthé et Siegel, 2004; Fernique, 2006; Berthé et Labbé, 2011). Cette nouvelle vague de publications prometteuse nous mènera sans doute vers une meilleure compréhension des problèmes de pavages tridimensionnels, qui présentent des applications intéressantes, entre autres dans le domaine de la cristallographie.

APPENDICE A

CODE SOURCE

Dans cet annexe, on trouve le code source de quatre fichiers utilisés dans le cadre de mon doctorat, principalement à des fins d'exploration informatique. Les trois premiers fichiers ont été conçus par moi alors que le dernier a été créé conjointement avec Ariane Garon et Sébastien Labbé. Ils sont tous sous license GPL.

A.1 Fichier palindromic_closure.sage

```
#*****************************************************************************
#       Copyright (C) 2011 Alexandre Blondin Masse <ablondin@uqac.ca>,
#
#  Distributed under the terms of the GNU General Public License
#  version 2 (GPLv2)
#
#  The full text of the GPLv2 is available at:
#
#                  http://www.gnu.org/licenses/
#*****************************************************************************

def _iterated_right_palindromic_closure_iterator(self, f=None):
    r"""
    Returns an iterator over the iterated (‘f‘-)palindromic closure of self.

    INPUT:

    -  ‘‘f‘‘ - involution (default: None) on the alphabet of self. It must
       be callable on letters as well as words (e.g. WordMorphism).

    OUTPUT:

        iterator -- the iterated (‘f‘-)palindromic closure of self

    EXAMPLES::

        sage: w = Word('abc')
        sage: it = w._iterated_right_palindromic_closure_iterator()
        sage: Word(it)
        word: abacaba

    ::

        sage: w = Word('aaa')
        sage: it = w._iterated_right_palindromic_closure_iterator()
        sage: Word(it)
        word: aaa

    ::
```

```
            sage: w = Word('abbab')
            sage: it = w._iterated_right_palindromic_closure_iterator()
            sage: Word(it)
            word: ababaababababaababa

        An infinite word::

            sage: t = words.ThueMorseWord('ab')
            sage: it = t._iterated_right_palindromic_closure_iterator()
            sage: Word(it)
            word: ababaababababaababaababababaababaababababaabab...

        TESTS:

        The empty word::

            sage: w = Word()
            sage: it = w._iterated_right_palindromic_closure_iterator()
            sage: it.next()
            Traceback (most recent call last):
            ...
            StopIteration

        REFERENCES:

        -   [1] A. de Luca, A. De Luca, Pseudopalindrome closure operators
            in free monoids, Theoret. Comput. Sci. 362 (2006) 282--300.
        """
        par = self.parent()
        w = self[:0]
        for letter in self:
            length_before = w.length()
            w = (w*par([letter])).palindromic_closure(f=f)
            length_after = w.length()
            d = length_after - length_before
            for a in w[-d:]:
                yield a

    def _iterated_right_palindromic_closure_recursive_iterator(self, f=None):
        r"""
        Returns an iterator over the iterated (`f`-)palindromic closure of
        self.
```

```
    INPUT:

    -  ``f`` - involution (default: None) on the alphabet of self.
       It must be callable on letters as well as words
       (e.g. WordMorphism).

    OUTPUT:

        iterator -- the iterated (`f`-)palindromic closure of self

    ALGORITHM:

        For the case of palindromes only, it has been shown in [2] that
        the iterated right palindromic closure of a given word `w`,
        denoted by `IRPC(w)`, may be obtained as follows.
        Let `w` be any word and `x` be a letter. Then

        #. If `x` does not occur in `w`,
           `IRPC(wx) = IRPC(w) \cdot x \cdot IRPC(w)`
        #. Otherwise, write `w = w_1xw_2` such that `x` does not
           occur in `w_2`. Then `IRPC(wx) = IRPC(w) \cdot IRPC(w_1)^{-1}
           \cdot IRPC(w)`

        This formula is directly generalized to the case of
        `f`-palindromes. See [1] for more details.

    EXAMPLES::

        sage: w = Word('abc')
        sage: it = w._iterated_right_palindromic_closure_recursive_iterator()
        sage: Word(it)
        word: abacaba

    ::

        sage: w = Word('aaa')
        sage: it = w._iterated_right_palindromic_closure_recursive_iterator()
        sage: Word(it)
        word: aaa

    ::
```

```
    sage: w = Word('abbab')
    sage: it = w._iterated_right_palindromic_closure_recursive_iterator()
    sage: Word(it)
    word: ababaababababaababa

An infinite word::

    sage: t = words.ThueMorseWord('ab')
    sage: it = t._iterated_right_palindromic_closure_recursive_iterator()
    sage: Word(it)
    word: ababaababababaababaababababaababaababababaabab...

TESTS:

The empty word::

    sage: w = Word()
    sage: it = w._iterated_right_palindromic_closure_recursive_iterator()
    sage: it.next()
    Traceback (most recent call last):
    ...
    StopIteration

REFERENCES:

-   [1] A. de Luca, A. De Luca, Pseudopalindrome closure operators
    in free monoids, Theoret. Comput. Sci. 362 (2006) 282--300.
-   [2] J. Justin, Episturmian morphisms and a Galois theorem on
    continued fractions, RAIRO Theoret. Informatics Appl. 39 (2005)
    207-215.
"""
parent = self.parent()
ipcw = self[:0]
lengths = []
for i, letter in enumerate(self):
    lengths.append(ipcw.length())
    w = self[:i]
    pos = w.rfind(parent([letter]))
    if pos == -1:
        to_append = parent([letter]).palindromic_closure(f=f) + ipcw
    else:
```

```
            to_append = ipcw[lengths[pos]:]
        ipcw += to_append
        for a in to_append:
            yield a

def iterated_right_palindromic_closure(self, f=None, algorithm='recursive'):
    r"""
    Returns the iterated (‘f‘-)palindromic closure of self.

    INPUT:

    - ‘‘f‘‘ - involution (default: None) on the alphabet of self. It must
      be callable on letters as well as words (e.g. WordMorphism).

    - ‘‘algorithm‘‘ - string (default: ‘‘’recursive’‘‘) specifying which
      algorithm to be used when computing the iterated palindromic closure.
      It must be one of the two following values:

      - ‘‘’definition’‘‘ - computed using the definition
      - ‘‘’recursive’‘‘ - computation based on an efficient formula
        that recursively computes the iterated right palindromic closure
        without having to recompute the longest ‘f‘-palindromic suffix
        at each iteration [2].

    OUTPUT:

        word -- the iterated (‘f‘-)palindromic closure of self

    EXAMPLES::

        sage: w = Word('abc')
        sage: w.iterated_right_palindromic_closure()
        word: abacaba

    ::

        sage: w = Word('aaa')
        sage: w.iterated_right_palindromic_closure()
        word: aaa

    ::
```

```
    sage: w = Word('abbab')
    sage: w.iterated_right_palindromic_closure()
    word: ababaababababaababa

A right 'f'-palindromic closure::

    sage: f = WordMorphism('a->b,b->a')
    sage: w = Word('abbab')
    sage: w.iterated_right_palindromic_closure(f=f)
    word: abbaabbaababbaabbaabbaababbaabbaab

An infinite word::

    sage: t = words.ThueMorseWord('ab')
    sage: t.iterated_right_palindromic_closure()
    word: ababaababababaababaababababaababaababababaabab...

There are two implementations computing the iterated right
'f'-palindromic closure, the latter being much more efficient::

    sage: w = Word('abaab')
    sage: u = w.iterated_right_palindromic_closure(algorithm='definition')
    sage: v = w.iterated_right_palindromic_closure(algorithm='recursive')
    sage: u
    word: abaabaababababaaba
    sage: u == v
    True
    sage: w = words.RandomWord(8)
    sage: u = w.iterated_right_palindromic_closure(algorithm='definition')
    sage: v = w.iterated_right_palindromic_closure(algorithm='recursive')
    sage: u == v
    True

TESTS:

The empty word::

    sage: w = Word()
    sage: w.iterated_right_palindromic_closure()
    word:

If the word is finite, so is the result::
```

```
        sage: w = Word([0,1]*7)
        sage: c = w.iterated_right_palindromic_closure()
        sage: type(c)
        <class 'sage.combinat.words.word.FiniteWord_iter_with_caching'>

    REFERENCES:

    -   [1] A. de Luca, A. De Luca, Pseudopalindrome closure operators
        in free monoids, Theoret. Comput. Sci. 362 (2006) 282--300.
    -   [2] J. Justin, Episturmian morphisms and a Galois theorem on
        continued fractions, RAIRO Theoret. Informatics Appl. 39 (2005)
        207-215.
    """
    from sage.combinat.words.word import FiniteWord_class, InfiniteWord_class
    if isinstance(self, FiniteWord_class):
        length = "finite"
    elif isinstance(self, InfiniteWord_class):
        length = None
    else:
        length = "unknown"
    if algorithm == 'definition':
        it = self._iterated_right_palindromic_closure_iterator(f=f)
    elif algorithm == 'recursive':
        it = self._iterated_right_palindromic_closure_recursive_iterator(f=f)
    else:
        raise ValueError, "algorithm (=%s) must be either 'definition' or\
                           'recursive'"
    return self._parent(it, length=length)
```

A.2 Fichier iterated_palindromic_closure .sage

```
#*****************************************************************************
#       Copyright (C) 2011 Alexandre Blondin Masse <ablondin@uqac.ca>,
#
#  Distributed under the terms of the GNU General Public License
#  version 2 (GPLv2)
#
#  The full text of the GPLv2 is available at:
#
#                  http://www.gnu.org/licenses/
#*****************************************************************************

def _iterated_right_palindromic_closure_iterator(self, f=None):
    r"""
    Returns an iterator over the iterated (‘f‘-)palindromic closure of self.

    INPUT:

    -  ‘‘f‘‘ - involution (default: None) on the alphabet of self. It must
       be callable on letters as well as words (e.g. WordMorphism).

    OUTPUT:

        iterator -- the iterated (‘f‘-)palindromic closure of self

    EXAMPLES::

        sage: w = Word('abc')
        sage: it = w._iterated_right_palindromic_closure_iterator()
        sage: Word(it)
        word: abacaba

    ::

        sage: w = Word('aaa')
        sage: it = w._iterated_right_palindromic_closure_iterator()
        sage: Word(it)
        word: aaa

    ::
```

```
        sage: w = Word('abbab')
        sage: it = w._iterated_right_palindromic_closure_iterator()
        sage: Word(it)
        word: ababaababababaababa

    An infinite word::

        sage: t = words.ThueMorseWord('ab')
        sage: it = t._iterated_right_palindromic_closure_iterator()
        sage: Word(it)
        word: ababaababababaababaababababaababaababababaabab...

    TESTS:

    The empty word::

        sage: w = Word()
        sage: it = w._iterated_right_palindromic_closure_iterator()
        sage: it.next()
        Traceback (most recent call last):
        ...
        StopIteration

    REFERENCES:

    -   [1] A. de Luca, A. De Luca, Pseudopalindrome closure operators
        in free monoids, Theoret. Comput. Sci. 362 (2006) 282--300.
    """
    par = self.parent()
    w = self[:0]
    for letter in self:
        length_before = w.length()
        w = (w*par([letter])).palindromic_closure(f=f)
        length_after = w.length()
        d = length_after - length_before
        for a in w[-d:]:
            yield a

def _iterated_right_palindromic_closure_recursive_iterator(self, f=None):
    r"""
    Returns an iterator over the iterated (`f`-)palindromic closure of
    self.
```

```
INPUT:

-  ``f`` - involution (default: None) on the alphabet of self.
   It must be callable on letters as well as words
   (e.g. WordMorphism).

OUTPUT:

    iterator -- the iterated (`f`-)palindromic closure of self

ALGORITHM:

    For the case of palindromes only, it has been shown in [2] that
    the iterated right palindromic closure of a given word `w`,
    denoted by `IRPC(w)`, may be obtained as follows.
    Let `w` be any word and `x` be a letter. Then

    #. If `x` does not occur in `w`,
       `IRPC(wx) = IRPC(w) \cdot x \cdot IRPC(w)`
    #. Otherwise, write `w = w_1xw_2` such that `x` does not
       occur in `w_2`. Then `IRPC(wx) = IRPC(w) \cdot IRPC(w_1)^{-1}
       \cdot IRPC(w)`

    This formula is directly generalized to the case of
    `f`-palindromes. See [1] for more details.

EXAMPLES::

    sage: w = Word('abc')
    sage: it = w._iterated_right_palindromic_closure_recursive_iterator()
    sage: Word(it)
    word: abacaba

::

    sage: w = Word('aaa')
    sage: it = w._iterated_right_palindromic_closure_recursive_iterator()
    sage: Word(it)
    word: aaa

::
```

```
        sage: w = Word('abbab')
        sage: it = w._iterated_right_palindromic_closure_recursive_iterator()
        sage: Word(it)
        word: ababaababababaababa

    An infinite word::

        sage: t = words.ThueMorseWord('ab')
        sage: it = t._iterated_right_palindromic_closure_recursive_iterator()
        sage: Word(it)
        word: ababaababababaababaababababaababaababababaababab...

    TESTS:

    The empty word::

        sage: w = Word()
        sage: it = w._iterated_right_palindromic_closure_recursive_iterator()
        sage: it.next()
        Traceback (most recent call last):
        ...
        StopIteration

    REFERENCES:

    -   [1] A. de Luca, A. De Luca, Pseudopalindrome closure operators
        in free monoids, Theoret. Comput. Sci. 362 (2006) 282--300.
    -   [2] J. Justin, Episturmian morphisms and a Galois theorem on
        continued fractions, RAIRO Theoret. Informatics Appl. 39 (2005)
        207-215.
    """
    parent = self.parent()
    ipcw = self[:0]
    lengths = []
    for i, letter in enumerate(self):
        lengths.append(ipcw.length())
        w = self[:i]
        pos = w.rfind(parent([letter]))
        if pos == -1:
            to_append = parent([letter]).palindromic_closure(f=f) + ipcw
        else:
```

```
            to_append = ipcw[lengths[pos]:]
        ipcw += to_append
        for a in to_append:
            yield a

def iterated_right_palindromic_closure(self, f=None, algorithm='recursive'):
    r"""
    Returns the iterated (‘f‘-)palindromic closure of self.

    INPUT:

    -  ‘‘f‘‘ - involution (default: None) on the alphabet of self. It must
       be callable on letters as well as words (e.g. WordMorphism).

    -  ‘‘algorithm‘‘ - string (default: ‘‘’recursive’‘‘) specifying which
       algorithm to be used when computing the iterated palindromic closure.
       It must be one of the two following values:

       - ‘‘’definition’‘‘ - computed using the definition
       - ‘‘’recursive’‘‘ - computation based on an efficient formula
         that recursively computes the iterated right palindromic closure
         without having to recompute the longest ‘f‘-palindromic suffix
         at each iteration [2].

    OUTPUT:

        word -- the iterated (‘f‘-)palindromic closure of self

    EXAMPLES::

        sage: w = Word('abc')
        sage: w.iterated_right_palindromic_closure()
        word: abacaba

    ::

        sage: w = Word('aaa')
        sage: w.iterated_right_palindromic_closure()
        word: aaa

    ::
```

```
    sage: w = Word('abbab')
    sage: w.iterated_right_palindromic_closure()
    word: ababaababababaababa

A right 'f'-palindromic closure::

    sage: f = WordMorphism('a->b,b->a')
    sage: w = Word('abbab')
    sage: w.iterated_right_palindromic_closure(f=f)
    word: abbaabbaababbaabbaabbaababbbaabbaab

An infinite word::

    sage: t = words.ThueMorseWord('ab')
    sage: t.iterated_right_palindromic_closure()
    word: ababaababababaababaababababaababababababaab...

There are two implementations computing the iterated right
'f'-palindromic closure, the latter being much more efficient::

    sage: w = Word('abaab')
    sage: u = w.iterated_right_palindromic_closure(algorithm='definition')
    sage: v = w.iterated_right_palindromic_closure(algorithm='recursive')
    sage: u
    word: abaabaababababaaba
    sage: u == v
    True
    sage: w = words.RandomWord(8)
    sage: u = w.iterated_right_palindromic_closure(algorithm='definition')
    sage: v = w.iterated_right_palindromic_closure(algorithm='recursive')
    sage: u == v
    True

TESTS:

The empty word::

    sage: w = Word()
    sage: w.iterated_right_palindromic_closure()
    word:

If the word is finite, so is the result::
```

```
    sage: w = Word([0,1]*7)
    sage: c = w.iterated_right_palindromic_closure()
    sage: type(c)
    <class 'sage.combinat.words.word.FiniteWord_iter_with_caching'>

REFERENCES:

-   [1] A. de Luca, A. De Luca, Pseudopalindrome closure operators
    in free monoids, Theoret. Comput. Sci. 362 (2006) 282--300.
-   [2] J. Justin, Episturmian morphisms and a Galois theorem on
    continued fractions, RAIRO Theoret. Informatics Appl. 39 (2005)
    207-215.
"""
from sage.combinat.words.word import FiniteWord_class, InfiniteWord_class
if isinstance(self, FiniteWord_class):
    length = "finite"
elif isinstance(self, InfiniteWord_class):
    length = None
else:
    length = "unknown"
if algorithm == 'definition':
    it = self._iterated_right_palindromic_closure_iterator(f=f)
elif algorithm == 'recursive':
    it = self._iterated_right_palindromic_closure_recursive_iterator(f=f)
else:
    raise ValueError, "algorithm (=%s) must be either 'definition' or\
                       'recursive'"
return self._parent(it, length=length)
```

A.3 Fichier equations.sage

```
#*****************************************************************************
#       Copyright (C) 2011 Alexandre Blondin Masse <ablondin@uqac.ca>,
#
#  Distributed under the terms of the GNU General Public License
#  version 2 (GPLv2)
#
#  The full text of the GPLv2 is available at:
#
#                  http://www.gnu.org/licenses/
#*****************************************************************************

#----------#
# Constant #
#----------#

Freeman = WordPaths('abAB')
BAR     = lambda n: -n
F_BAR   = WordMorphism('a->A,b->B,A->a,B->b', codomain=Freeman)
F_HAT   = lambda w: F_BAR(w.reversal())
F_RHO   = WordMorphism('a->b,b->A,A->B,B->a', codomain=Freeman)
F_SIGMA = WordMorphism('a->A,b->b,A->a,B->B', codomain=Freeman)

def PHI(n):
    """
    This morphism is used to transform general words into
    words on the Freeman alphabet
    """
    if n == 1:
        return 'a'
    elif n == -1:
        return 'A'
    elif n == 2:
        return 'b'
    elif n == -2:
        return 'B'

#---------------------#
# Partition functions #
#---------------------#
```

```
def get_generic_partition(generic_expressions, delays):
    """
    Returns the equivalence classes induced by overlaping
    the given generic expressions.

    EXAMPLES::

        sage: get_generic_partition([[1,2,3,4,5],[5,4,3,2,1]], [0])
        [set([1,5]),set([2,4]),set([3])]
    """
    l = min(delays + [0])
    u = len(generic_expressions[0])
    i = 1
    while i < len(generic_expressions):
        u = max(u, len(generic_expressions[i]) + delays[i - 1])
        i += 1
    partition = []
    for i in range(l, u):
        if i in range(len(generic_expressions[0])):
            c = set([generic_expressions[0][i]])
        else:
            c = set([])
        for j in range(1, len(generic_expressions)):
            if i in range(delays[j - 1],\
                          len(generic_expressions[j]) + delays[j - 1]):
                c |= set([generic_expressions[j][i - delays[j - 1]]])
        i = 0
        partition = merge_group_in_partition(c, partition)
        partition = merge_group_in_partition(set(map(lambda n:-n, c)), partition)
    return partition

def merge_group_in_partition(group, partition):
    """
    Merges the given group to the partition
    If this group creates bigger classes, they are merged as well.

    EXAMPLES:
        sage: merge_group_in_partition([set([2,3]), [set([1,2]),set([3,4])])
        [set([1,2,3,4])]
        sage: merge_group_in_partition([set([1,2]), [set([1,2]),set([3,4])])
        [set([1,2]),set([3,4])]
        sage: merge_group_in_partition([set([5]), [set([1,2]),set([3,4])])
```

```
        [set([1,2]),set([3,4]),set([5])]
    """
    if len(group) == 0:
        return copy(partition)
    else:
        merged_partition = []
        merged_class = group.copy()
        for c in partition:
            if len(c & group) == 0:
                merged_partition.append(c)
            else:
                merged_class |= c
        merged_partition.append(merged_class)
        return merged_partition

def get_representants(partition):
    """
    Returns a dictionary giving, for each number its representant
    according to the given partition.
    For sake of simplicity, the integer close to 0 is chosen.

    EXAMPLES:
        sage: get_representants([set([1,2]),set([3,4,5])])
        {1 : 1, 2 : 1, 3 : 3, 4 : 3, 5 : 3}
    """
    representants = {}
    for p in partition:
        r = min(p, key = lambda n:abs(n))
        for q in p:
            representants[q] = r
    return representants

def is_realizable(partition):
    """
    Returns True if no letter is equivalent to its complement

    EXAMPLES:
        sage: is_realizable([set([1,2]),set([3,4])])
        True
        sage: is_realizable([set([1,2,-1]),set([3,4])])
        False
    """
```

```
    realizable = True
    i = 0
    while realizable and i < len(partition):
        realizable = len(set(map(lambda x:abs(x), partition[i]))) ==\
                     len(partition[i])
        i += 1
    return realizable

#----------------------------#
# Solving equations on words #
#----------------------------#

def get_generic_words(lengths):
    """
    Returns the general solution of the given equation on words.

    EXAMPLES:
        sage: get_generic_words({'u' : 4})
        {'u' : [1,2,3,4]}
        sage: get_generic_words({'u' : 2, 'v' : 6})
        {'u' : [1,2], 'v' : [3,4,5,6,7,8]}
    """
    i = 1
    generic_words = {}
    for w in lengths:
        generic_words[w] = range(i, i + lengths[w])
        i += lengths[w]
    return generic_words

def translate_generic_expression(generic_words, expression):
    """
    Translates an expression involving words into its generic word
    Three special symbols may be used in an expression, -, ~ and ^,
    playing respectively the role of the complement operator, the
    reversal operator and the hat operator. They have to be placed
    in front of the word on which it applies.

    EXAMPLES:
        sage: translate_generic_expression({'u' : [1,2,3,4]}, 'uu')
        [1,2,3,4,1,2,3,4]
        sage: translate_generic_expression({'u' : [1,2,3,4]}, '-u~u^u')
```

```
        [-1,-2,-3,-4,4,3,2,1,-4,-3,-2,-1]
    """
    i = 0
    generic_expression = []
    while i < len(expression):
        if expression[i] == '~':
            reversal = copy(generic_words[expression[i+1]])
            reversal.reverse()
            generic_expression += reversal
            i += 2
        elif expression[i] == '-':
            generic_expression += map(lambda n:-n,\
                                      generic_words[expression[i+1]])
            i += 2
        elif expression[i] == '^':
            reversal = copy(generic_words[expression[i+1]])
            reversal.reverse()
            generic_expression += map(lambda n:-n, reversal)
            i += 2
        else:
            generic_expression += generic_words[expression[i]]
            i += 1
    return generic_expression

def reduce_solution(solution):
    """
    Transforms the solution by reducing the numbers appearing
    For instance, if numbers 1, 3, 4 and 6 appears they are replaced
    respectively by 1, 2, 3 and 4.

    EXAMPLES:
        sage: reduce_solution({'u' : [1,2,2,1], 'v' : [1,5,1]})
        {'u' : [1,2,2,1], 'v' : [1,3,1]}
    """

    numbers = sorted(list(set(map(lambda x:abs(x),\
              reduce(lambda s, t: s + t, solution.values())))))
    reduced = {}
    for w in solution:
        reduced[w] = map(lambda n:(-1) ** int(n < 0) *\
                     (numbers.index(abs(n)) + 1), solution[w])
    return reduced
```

```
def overlap(lengths, equations, delays):
    """
    Returns the general solution of the given equation on words

    EXAMPLES::

        sage: overlap({'u' : 4}, ['u', '~u'], [0])
        {'u' : [1,2,2,1]}
        sage: overlap({'u' : 4, 'v' : 4}, ['u', '~u', 'v'], [0, 2])
        {'u' : [1,2,2,1], 'v' : [2,1,3,4]}
    """
    generic_words = get_generic_words(lengths)
    generic_expressions = map(lambda e:translate_generic_expression\
                              (generic_words, e), equations)
    partition = get_generic_partition(generic_expressions, delays)
    if not is_realizable(partition):
        return None
    else:
        representants = get_representants(partition)
        solution = generic_words.copy()
        for gw in solution:
            solution[gw] = map(lambda w:representants[w], generic_words[gw])
        return reduce_solution(solution)

#---------------------------#
# Enumerating all solutions #
#---------------------------#

def overlap_dps_distinct_lengths(length1, length2, length3, delay):
    return overlap({'A' : length1, 'B' : length2, 'X' : length3,\
                    'Y' : length1 + length2 - length3},\
                    ['AB-A-BAB-A-B', '~A~B', 'XY-X-Y', '~X~Y'],\
                    [0, delay, delay])

def get_all_distinct_lengths_dps_particular_solutions(l, m, n):
    solutions = []
    for length1 in range(0, l):
        for length2 in range(0, m):
            for length3 in set(range(0, length1 + length2 + 1)):
                for delay in range(1, length1):
                    solution = overlap_dps_distinct_lengths\
```

```
                            (length1, length2, length3, delay)
                    if solution != None:
                        boundary_general = solution['A'] + solution['B']\
                                           + map(lambda n:-n, solution['A'])\
                                           + map(lambda n:-n, solution['B'])
                        if len(set(map(lambda n:abs(n), boundary_general)))\
                           == 2 and seems_prime(boundary_general):
                            solutions.append(Freeman(map(lambda n:PHI(n),\
                                                         boundary_general)))
    return solutions

def get_all_distinct_lengths_dps(l, m, n):
    return filter(lambda p:p[1:].is_simple(),\
           get_all_distinct_lengths_dps_particular_solutions(l, m, n))

def double_square_iterator(max_length=None,\
                           max_iteration=None,\
                           merge_isometric=True):
    i = 0
    p = 2
    double_squares = set([])
    if max_length is None:
        max_length = Infinity
    if max_iteration is None:
        max_iteration = Infinity
    while p <= max_length / 2:
        for a in range(1, p - 1):
            b = p - a
            for d in range(1, min(a - 1, p / 2)):
                for x in range(a + 1 - d, p - 1):
                    y = p - x
                    solution = overlap_dps_distinct_lengths(a, b, x, d)
                    if solution:
                        boundary_general = solution['A'] + solution['B'] + \
                        map(BAR, solution['A']) + map(BAR, solution['B'])
                        if len(set(map(lambda n:abs(n), boundary_general)))\
                           == 2 and seems_prime(boundary_general):
                            word = min(Freeman(map(lambda n:PHI(n),\
                                   boundary_general)).conjugates())
                            if word.is_simple() and word not in double_squares:
                                yield word
                                inv_word = min(F_HAT(word).conjugates())
```

```
                                if merge_isometric:
                                    iso_tiles = map(lambda p:\
                                                    min(p.conjugates()),\
                                                    isometric_paths(word))
                                    iso_tiles += map(lambda p:\
                                                     min(p.conjugates()),\
                                                     isometric_paths(inv_word))
                                else:
                                    iso_tiles = [word, inv_word]
                                i += 1
                                if i >= max_iteration:
                                    raise StopIteration
                                double_squares |= set(iso_tiles)
        p += 2

#------------------#
# Useful functions #
#------------------#

def seems_prime(boundary_general):
    i = 0
    prime = True
    while prime and i < len(boundary_general):
        prime = (i % 2 == 0 and boundary_general[i] in [-1, 1])\
                or (i % 2 == 1 and boundary_general[i] in [-2, 2])
        i += 1
    return prime

def plot_tiles(tiles):
    G = [p.plot(pathoptions=dict(rgbcolor='red',thickness=1),\
        endarrow=False,startpoint=False) for p in tiles]
    cols = int(sqrt(len(tiles)))
    rows = len(tiles) / cols + 1
    g = graphics_array(G, rows, cols)
    g.show(figsize=[15, 15 * rows/cols])

def latex_table(tiles, figure_width=2, num_cols=8):
    perimeter_to_tiles = {}
    for tile in tiles:
        if len(tile) not in perimeter_to_tiles:
            perimeter_to_tiles[len(tile)] = [tile]
        else:
```

```
            perimeter_to_tiles[len(tile)].append(tile)
    s = '\\begin{supertabular}{|c|l|}\n'
    s += '  \hline\n'
    s += "  P\\'erim\'etre & Tuiles \\\\\n"
    s += '  \hline\n'
    perimeters = sorted(perimeter_to_tiles.keys())
    for p in perimeters:
        s += '  ' + str(p) + ' & \\begin{tikzpicture}\n'
        s += '          \\matrix [ampersand replacement=\&,\n'
        s += '                    every cell/.style={anchor=base},\n'
        s += '                    column sep=5mm,row sep=7mm]\n'
        s += '          {\n'
        i = 0
        for t in perimeter_to_tiles[p]:
            figure_height = figure_width
            h = t.height()
            w = t.width()
            step = 1.0 * figure_width / max(h, w)
            x = min(map(lambda p: p[0], t.points())) * step
            y = min(map(lambda p: p[1], t.points())) * step
            s += '            \\filldraw[draw=black, fill=black!20,\
                                x=%scm, y=%scm, xshift=%scm,\
                                yshift=%scm] '\
                 % (step, step, -x + (figure_width - w * step) / 2.0, -y +\
                   (figure_height - h * step) / 2.0)
            s += t.tikz_trajectory() + '; '
            if i % num_cols == num_cols - 1:
                s += '\\\\\n'
            elif i < len(perimeter_to_tiles[p]) - 1:
                s += '\\&\n'
            else:
                s += '\\\\\n'
            i += 1
        s += '          };\n'
        s += '       \\end{tikzpicture}\n'
        s += '       \\\\\n'
        s += '       \hline\n'
    s += '\\end{supertabular}'
    from sage.misc.latex import LatexExpr
    return LatexExpr(s)

def isometric_paths(path):
```

```
return [path, F_RHO(path), F_RHO(F_RHO(path)), F_RHO(F_RHO(F_RHO(path))),\
        F_SIGMA(path), F_RHO(F_SIGMA(path)), F_RHO(F_RHO(F_SIGMA(path))),\
        F_RHO(F_RHO(F_RHO(F_SIGMA(path))))]
```

A.4 Fichier configuration.sage

```
# -*- coding: utf-8 -*-
#*****************************************************************************
#       Copyright (C) 2011 Alexandre Blondin Masse <ablondin@uqac.ca>,
#                          Ariane Garon <ariane.garon@gmail.com>,
#                          Sebastien Labbe <slabqc@gmail.com>
#
#  Distributed under the terms of the GNU General Public License
#  version 2 (GPLv2)
#
#  The full text of the GPLv2 is available at:
#
#                  http://www.gnu.org/licenses/
#*****************************************************************************
r"""
Class Configuration used to study the enumeration
of double square tiles.

AUTHORS:
- Alexandre Blondin Masse
- Sebastien Labbe
"""
class Configuration(object):
    r"""
    Configuration object.

    INPUT:

    - ``A`` - a word path having a 4-letter alphabet parent
    - ``B`` - a word path having a 4-letter alphabet parent
    - ``d1`` - integer
    - ``d2`` - integer

    or

    - ``A`` - a word path having a 4-letter alphabet parent
    - ``B`` - a word path having a 4-letter alphabet parent
    - ``d1`` - integer
    - ``d2`` - integer
    - ``bar`` - the bar operator
```

```
or

- ``ds`` - a double square

or

- ``l0`` - the length of w0
- ``l1`` - the length of w1
- ``l2`` - the length of w2
- ``l3`` - the length of w3

or

- ``t`` - a tuple of 4 elements (w0,w1,w2,w3) or 8 elements
  (w0,w1,w2,w3,w4,w5,w6,w7)
- ``bar`` - the bar operator

EXAMPLES:

From a configuration (A, B, d1, d2)::

    sage: P = WordPaths('abAB')
    sage: Configuration(P('aba'), P('bAb'),1,1)
    Configuration(w0, w1, w2, w3):
      w0 = Path: a                              w4 = Path: A
      w1 = Path: ba                             w5 = Path: BA
      w2 = Path: b                              w6 = Path: B
      w3 = Path: Ab                             w7 = Path: aB
    (|w0|, |w1|, |w2|, |w3|) = (1, 2, 1, 2)
    (n0, n1, n2, n3)         = (0, 1, 0, 1)

From a double square::

    sage: fibo = words.fibonacci_tile
    sage: Configuration(fibo(1))
    Configuration(w0, w1, w2, w3):
      w0 = Path: 3                              w4 = Path: 1
      w1 = Path: 03                             w5 = Path: 21
      w2 = Path: 0                              w6 = Path: 2
      w3 = Path: 10                             w7 = Path: 32
    (|w0|, |w1|, |w2|, |w3|) = (1, 2, 1, 2)
    (n0, n1, n2, n3)         = (0, 1, 0, 1)
```

```
        sage: Configuration(fibo(2))
        Configuration(w0, w1, w2, w3):
          w0 = Path: 30323                          w4 = Path: 12101
          w1 = Path: 21232303                       w5 = Path: 03010121
          w2 = Path: 23212                          w6 = Path: 01030
          w3 = Path: 10121232                       w7 = Path: 32303010
        (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
        (n0, n1, n2, n3)         = (0, 0, 0, 0)

      From four integers::

        sage: c = Configuration(4,2,4,2)
        sage: c
        Configuration(w0, w1, w2, w3):
          w0 = Path: DCab                           w4 = Path: cdBA
          w1 = Path: DC                             w5 = Path: cd
          w2 = Path: BADC                           w6 = Path: abcd
          w3 = Path: BA                             w7 = Path: ab
        (|w0|, |w1|, |w2|, |w3|) = (4, 2, 4, 2)
        (n0, n1, n2, n3)         = (1, 0, 1, 0)
        sage: c = Configuration(2,1,2,1)
        sage: c
        Configuration(w0, w1, w2, w3):
          w0 = Path: Ab                             w4 = Path: aB
          w1 = Path: A                              w5 = Path: a
          w2 = Path: BA                             w6 = Path: ba
          w3 = Path: B                              w7 = Path: b
        (|w0|, |w1|, |w2|, |w3|) = (2, 1, 2, 1)
        (n0, n1, n2, n3)         = (1, 0, 1, 0)

    """
    def __init__(self, *args):
        r"""
        Constructor.

        See :Configuration: for documentation.

        EXAMPLES::

            sage: Configuration(2,2,2,2)
            Configuration(w0, w1, w2, w3):
              w0 = Path: ab                             w4 = Path: ab
```

```
      w1 = Path: BA                                w5 = Path: BA
      w2 = Path: ab                                w6 = Path: ab
      w3 = Path: BA                                w7 = Path: BA
    (|w0|, |w1|, |w2|, |w3|) = (2, 2, 2, 2)
    (n0, n1, n2, n3)         = (0, 0, 0, 0)
"""
if len(args) == 4 and all(isinstance(a, (int, Integer)) for a in args):
    msg = "les delais doivent etre strictement positifs"
    assert args[0] + args[2] > 0 and args[1] + args[3] > 0, msg
    ((w0,w1,w2,w3,w4,w5,w6,w7), self.bar) = overlap_moi_ca(*args)
    self._w = (w0,w1,w2,w3,w4,w5,w6,w7)
    alphabet = self.A.parent().alphabet()
    if not hasattr(alphabet,'cardinality')\
    or alphabet.cardinality() != 4:
        pass
elif len(args) == 4:
    A, B, d1, d2 = args
    w0 = A[:d1]
    w1 = A[d1:]
    w2 = B[:d2]
    w3 = B[d2:]
    self._w = (w0,w1,w2,w3)
    self.bar = None
elif len(args) == 5:
    A, B, d1, d2, self.bar = args
    w0 = A[:d1]
    w1 = A[d1:]
    w2 = B[:d2]
    w3 = B[d2:]
    self._w = (w0,w1,w2,w3)
elif len(args) == 2 :
    (self._w, self.bar) = args
    assert len(self._w) in (4, 8)
elif len(args) == 1:
    ds = args[0]
    f = find_good_ds_factorisation(ds, delay='minimize')
    startA,endA,startX,endX = f
    demiper = ds.length()//2
    twice = ds * ds
    w0 = twice[startA:startX]
    w1 = twice[startX:endA]
    w2 = twice[endA:endX]
```

```
            w3 = twice[endX:startA+demiper]
            w4 = twice[startA+demiper:startX+demiper]
            w5 = twice[startX+demiper:endA+demiper]
            w6 = twice[endA+demiper:endX+demiper]
            w7 = twice[endX+demiper:startA+demiper+demiper]
            self._w = (w0,w1,w2,w3,w4,w5,w6,w7)
            self.bar = None
        else:
            raise TypeError, "Configuration takes one argument (a double square)\
                  or four arguments (A, B, d1, d2) not %s."%len(args)

        if self.bar is None:
            if self.A.parent() != self.B.parent():
                raise ValueError, "A and B must have the same parent"
            alphabet = self.A.parent().alphabet()
            if not hasattr(alphabet,'cardinality')\
            or alphabet.cardinality() != 4:
                raise ValueError, "The parent of A must have a 4-letter\
                                   alphabet."
            e,n,w,s = alphabet
            self.bar = WordMorphism({e:w,w:e,n:s,s:n},codomain=self.A.parent())

        self.verify_definition()
        self.verify_conjecture()

    @lazy_attribute
    def hat(self):
        return lambda x:self.bar(x).reversal()

    @lazy_attribute
    def A(self):
        return self._w[0] * self._w[1]

    @lazy_attribute
    def B(self):
        return self._w[2] * self._w[3]

    @lazy_attribute
    def d1(self):
        return len(self._w[0])

    @lazy_attribute
```

```
def d2(self):
    return len(self._w[2])

@cached_method
def __getitem__(self, i):
    r"""
    Return the factor w_i

    This corresponds to the new definition of configuration (solution).

    EXAMPLES::

        sage: fibo = words.fibonacci_tile
        sage: c = Configuration(fibo(1))
        sage: [c[i] for i in range(8)]
        [Path: 3, Path: 03, Path: 0, Path: 10, Path: 1, Path: 21, Path: 2,
         Path: 32]

    ::

        sage: c = Configuration(fibo(2))
        sage: [c[i] for i in range(8)]
        [Path: 30323, Path: 21232303, Path: 23212, Path: 10121232,
         Path: 12101, Path: 03010121, Path: 01030, Path: 32303010]
    """
    if 0 <= i < len(self._w):
        return self._w[i]
    elif i == 4:
        return self.hat(self.A)[:self.d1]
    elif i == 5:
        return self.hat(self.A)[self.d1:]
    elif i == 6:
        return self.hat(self.B)[:self.d2]
    elif i == 7:
        return self.hat(self.B)[self.d2:]
    else:
        raise ValueError, 'i (=%s) must be between 0 and 7.'%i

w = __getitem__

def __eq__(self, other):
    r"""
```

```
        Returns True if A, B, d1 and d2 are the same.

        EXAMPLES::

            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: c == c
            True
            sage: c == c.conjugate()
            False
            sage: c == c.old_extend(2,4).old_shrink(2,4)
            True
        """
        return isinstance(other, Configuration) and\
               self.A == other.A and\
               self.B == other.B and\
               self.d1 == other.d1 and\
               self.d2 == other.d2

    def __cmp__(self, other):
        self_bw = self.boundary_word()
        other_bw = other.boundary_word()
        if len(self_bw) != len(other_bw):
            return len(self_bw) - len(other_bw)
        else:
            return self_bw.__cmp__(other_bw)

    def __len__(self):
        return len(self.boundary_word())

    def __hash__(self):
        r"""
        hash

        EXAMPLES::

            sage: c = Configuration(words.fibonacci_tile(2))
            sage: hash(c)
            368453717
            sage: hash(c.exchange(0).exchange(0))
            368453717
        """
```

```
        return hash((self.A,self.B,self.d1,self.d2))

    def verify_definition(self):
        r"""
        Checks that the solution verify the definition.
        """
        for i in range(4):
            msg = "wiwi+1 = hat(wi+4,wi+5) is broken for i=%s"%i
            assert self[i] * self[i+1] ==\
                    self.hat(self[i+4] * self[(i+5)%8]), msg

    def verify_conjecture(self):
        r"""
        Verification de conjectures.

        Si une conjecture nest pas verifie, une AssertionError est lancee.
        """
        #Verification que chapeau de uivi est facteur de w pour tout i
        bw = self.boundary_word()
        for i in range(8):
            factor = self.hat(self.u(i)*self.v(i))
            a = factor.is_factor(bw**2)
            assert a, "hat(uivi)=(%s) nest pas facteur du contour pour i=%s\
                       et pour %s"%(factor,i,self)
        #Verification que chapeau de viui est facteur de w pour tout i
        for i in range(8):
            factor = self.hat(self.v(i)*self.u(i))
            a = factor.is_factor(bw**2)
            assert a, "hat(viui)=(%s) nest pas facteur du contour pour i=%s\
                       et pour %s"%(factor,i,self)

    def alphabet(self):
        r"""
        Returns the python set of the letters that occurs in the boundary
        word.
        """
        return set(self.boundary_word())

    def boundary_word(self):
        r"""
        EXAMPLES::
```

```
            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323                          w4 = Path: 12101
              w1 = Path: 21232303                       w5 = Path: 03010121
              w2 = Path: 23212                          w6 = Path: 01030
              w3 = Path: 10121232                       w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
            sage: c.boundary_word()
            Path: 303232123230323212101212321210103010121O...
        """
        return self.A*self.B*self.hat(self.A)*self.hat(self.B)

    def turning_number(self):
        r"""
        """
        boundary = self.boundary_word()
        boundary = boundary + boundary[:1]
        boundary = boundary.to_integer_word()
        turns = boundary.finite_differences(mod=4)
        ev = turns.evaluation_dict()
        return QQ(((ev[1] - ev[3]), 4))

    def __repr__(self):
        r"""
        EXAMPLES::

            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323                          w4 = Path: 12101
              w1 = Path: 21232303                       w5 = Path: 03010121
              w2 = Path: 23212                          w6 = Path: 01030
              w3 = Path: 10121232                       w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
        """
        s = []
        s.append("Configuration(w0, w1, w2, w3):")
```

```
        s += ["  w%s = %s"%(i,repr(self[i])) for i in range(4)]
        t = ['] + ["  w%s = %s"%(i,repr(self[i])) for i in range(4,8)]
        s.append("(|w0|, |w1|, |w2|, |w3|) = %s"%(tuple(map(len,\
                 (self[i] for i in range(4)))),))
        s.append("(n0, n1, n2, n3)         = %s"%(tuple(self.n(i)\
                                                  for i in range(4)),))
        return tableau_de_col(s,t)

    def _latex_(self):
        r"""
        Returns a 8-tuple representing the configuration in Latex

        EXAMPLES::

            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: latex(c)
            (30323,21232303,23212,10121232,12101,03010121,01030,32303010)

        """
        from sage.misc.latex import LatexExpr
        w = [self[i].string_rep() for i in range(8)]
        w = map(lambda s: '\\' + '\\'.join(s), w)
        return LatexExpr("(%s)"%", ".join(w))

    @cached_method
    def u(self, i):
        r"""
        EXAMPLES::

            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: c.u(0)
            Path: 30323
            sage: c.v(0)
            Path: 21232303010
            sage: c[0]
            Path: 30323
        """
        p = self.hat(self[(i-3)%8]) * self[(i-1)%8]
        return p[:len(self[i])%len(p)]
```

```
    @cached_method
    def v(self, i):
        r"""
        EXAMPLES::
        """
        p = self.hat(self[(i-3)%8]) * self[(i-1)%8]
        return p[len(self[i])%len(p):]

    @cached_method
    def n(self, i):
        r"""
        EXAMPLES::

            sage: c = Configuration(words.fibonacci_tile(2))
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323                          w4 = Path: 12101
              w1 = Path: 21232303                       w5 = Path: 03010121
              w2 = Path: 23212                          w6 = Path: 01030
              w3 = Path: 10121232                       w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
            sage: [c.n(i) for i in range(8)]
            [0, 0, 0, 0, 0, 0, 0, 0]

        """
        p = self.hat(self[(i-3)%8]) * self[(i-1)%8]
        return len(self[i]) // len(p)

    @cached_method
    def d(self, i):
        r"""
        EXAMPLES::

            sage: c = Configuration(words.fibonacci_tile(2))
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323                          w4 = Path: 12101
              w1 = Path: 21232303                       w5 = Path: 03010121
              w2 = Path: 23212                          w6 = Path: 01030
              w3 = Path: 10121232                       w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
```

```
        (n0, n1, n2, n3)          = (0, 0, 0, 0)
        sage: [c.d(i) for i in range(8)]
        [16, 10, 16, 10, 16, 10, 16, 10]
    """
    return len(self.w((i - 1) % 8)) + len(self.w((i + 1) % 8))

def thickness(self):
    r"""
    Returns the parameters (i,j) of the configuration.
    Note that thickness depends directly on the configuration,
    not only on the associated double square
    (see Words2009 article)

    EXAMPLES::

        sage: c = Configuration(words.fibonacci_tile(2))
        sage: c.thickness()
        (0, 0)
        sage: c.old_extend(2,3).thickness()
        (2, 3)

        sage: c = Configuration(words.fibonacci_tile(2)).old_extend(2,3)
        sage: for i in range(8):
        ...       c = c.conjugate()
        ...       print c.thickness()
        (0, 0)
        (3, 2)
        (0, 0)
        (2, 3)
        (0, 0)
        (3, 2)
        (0, 0)
        (2, 3)
    """
    i = int((len(self.A) - self.d1) / (self.d1 + self.d2))
    j = int((len(self.B) - self.d2) / (self.d1 + self.d2))
    return (i,j)

def permute(self):
    r"""
    EXAMPLES::
```

```
            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323                          w4 = Path: 12101
              w1 = Path: 21232303                       w5 = Path: 03010121
              w2 = Path: 23212                          w6 = Path: 01030
              w3 = Path: 10121232                       w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
            sage: c.permute()
            Configuration(w0, w1, w2, w3):
              w0 = Path: 23212                          w4 = Path: 01030
              w1 = Path: 10121232                       w5 = Path: 32303010
              w2 = Path: 12101                          w6 = Path: 30323
              w3 = Path: 03010121                       w7 = Path: 21232303
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
            sage: c.permute().permute().permute().permute() == c
            True
        """
        return Configuration(self.B, self.hat(self.A),\
                             self.d2, self.d1, self.bar)

    def conjugate(self):
        r"""
        EXAMPLES::

            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323                          w4 = Path: 12101
              w1 = Path: 21232303                       w5 = Path: 03010121
              w2 = Path: 23212                          w6 = Path: 01030
              w3 = Path: 10121232                       w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
            sage: c.conjugate()
            Configuration(w0, w1, w2, w3):
              w0 = Path: 21232303                       w4 = Path: 03010121
              w1 = Path: 23212                          w5 = Path: 01030
```

```
            w2 = Path: 10121232                          w6 = Path: 32303010
            w3 = Path: 12101                             w7 = Path: 30323
        (|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
        (n0, n1, n2, n3)         = (0, 0, 0, 0)
        sage: c.conjugate().conjugate().conjugate().conjugate().
              conjugate().conjugate().conjugate().conjugate() == c
        True
        sage: c.conjugate().conjugate().conjugate().conjugate() == c
        False
    """
    (w0,w1,w2,w3,w4,w5,w6,w7) = tuple(self.w(i) for i in range(8))
    return Configuration((w1,w2,w3,w4,w5,w6,w7,w0), self.bar)

def conjugates(self):
    conjugates = [self]
    for i in range(7):
        conjugates.append(conjugates[-1].conjugate())
    return conjugates

def rotate(self):
    freeman = self.boundary_word().parent()
    a,b,A,B = freeman.alphabet()
    rho = WordMorphism({a:b, b:A, A:B, B:a}, codomain=freeman)
    return Configuration(tuple(rho(self.w(i)) for i in range(8)), self.bar)

def reflect(self):
    freeman = self.boundary_word().parent()
    a,b,A,B = freeman.alphabet()
    sigma = WordMorphism({a:a, b:B, A:A, B:b}, codomain=freeman)
    return Configuration(tuple(sigma(self.w(i)) for i in range(8)), self.bar)

def isometric_representant(self):
    if self.turning_number() != 1:
        candidates = [self.reverse()]
    else:
        candidates = [self]
    for i in range(3):
        candidates.append(candidates[-1].rotate())
    if self.turning_number() != 1:
        candidates.append(self.reflect())
    else:
        candidates.append(self.reverse().reflect())
```

```
        for i in range(3):
            candidates.append(candidates[-1].rotate())
        return min(map(lambda c: min(c.conjugates()), candidates))


    def reverse(self):
        r"""
        EXAMPLES::

            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323                          w4 = Path: 12101
              w1 = Path: 21232303                       w5 = Path: 03010121
              w2 = Path: 23212                          w6 = Path: 01030
              w3 = Path: 10121232                       w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
            sage: c.reverse()
            Configuration(w0, w1, w2, w3):
              w0 = Path: 23212101                       w4 = Path: 01030323
              w1 = Path: 21232                          w5 = Path: 03010
              w2 = Path: 30323212                       w6 = Path: 12101030
              w3 = Path: 32303                          w7 = Path: 10121
            (|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
            sage: c.reverse().reverse() == c
            True
        """
        (w0,w1,w2,w3,w4,w5,w6,w7) = map(self.w, range(8))
        return Configuration((self.hat(w7),self.hat(w6),self.hat(w5),\
                              self.hat(w4),self.hat(w3),self.hat(w2),\
                              self.hat(w1),self.hat(w0)), self.bar)

    def extend(self, i):
        r"""
        EXAMPLES::
        """
        (w0,w1,w2,w3,w4,w5,w6,w7) = [self[j] for j in range(i % 8, 8) +\
                                     range(0, i % 8)]
        w = (w0*w1*self.hat(w3),w1,w2,w3,w4*w5*self.hat(w7),w5,w6,w7)
```

```
        w = tuple(w[j] for j in range(-i % 8, 8) + range(0, -i % 8))
        return Configuration(w, self.bar)

    def old_extend(self, k, l):
        r"""
        EXAMPLES::

            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323                          w4 = Path: 12101
              w1 = Path: 21232303                       w5 = Path: 03010121
              w2 = Path: 23212                          w6 = Path: 01030
              w3 = Path: 10121232                       w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
            (n0, n1, n2, n3)         = (0, 0, 0, 0)
            sage: c.old_extend(2,2)
            Configuration(w0, w1, w2, w3):
              w0 = Path: 3032321232303010303232123230301030323
              w1 = Path: 21232303
              w2 = Path: 232121012123230323212101212323030323212
              w3 = Path: 10121232
              w4 = Path: 12101030101212321210103010121232121012
              w5 = Path: 03010121
              w6 = Path: 0103032303010121010303230301012101030
              w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (37, 8, 37, 8)
            (n0, n1, n2, n3)         = (2, 0, 2, 0)
            sage: c.old_extend(3,5).old_shrink(3,5) == c
            True
        """
        c = self
        for _ in range(k): c = c.extend(0)
        for _ in range(l): c = c.extend(2)
        return c

    def shrink(self, i):
        r"""
        EXAMPLES::

            sage: fibo = words.fibonacci_tile
```

```
            sage: c = Configuration(fibo(2))
            sage: d = c.old_extend(2,5)
            sage: d
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323212323030103032321232303010303 23
              w1 = Path: 21232303
              w2 = Path: 232121012123230323212101212323030323212101...
              w3 = Path: 10121232
              w4 = Path: 12101030101212321210103010121232123212101
              w5 = Path: 03010121
              w6 = Path: 01030323030101210103032303010121010303 23...
              w7 = Path: 32303010
            (|w0|, |w1|, |w2|, |w3|) = (37, 8, 85, 8)
            (n0, n1, n2, n3)         = (2, 0, 5, 0)
            sage: c == d.old_shrink(2,5) == d.old_shrink(1,3).old_shrink(1,2)
            True
        """
        d = len(self[(i-1)%8]) + len(self[(i+1)%8])
        if len(self[i]) > d:
            (w0,w1,w2,w3,w4,w5,w6,w7) = [self[j] for j in range(i % 8, 8)\
                                         + range(0, i % 8)]
            w = (w0[d:],w1,w2,w3,w4[d:],w5,w6,w7)
            w = tuple(w[j] for j in range(-i % 8, 8) + range(0, -i % 8))
            return Configuration(w, self.bar)
        else:
            raise ValueError, 'shrink cannot be applied on index %s\
                               of the following configuration\n%s'%(i,self)

    def old_shrink(self, k, l):
        r"""
        EXAMPLES::

            sage: fibo = words.fibonacci_tile
            sage: c = Configuration(fibo(2))
            sage: d = c.old_extend(2,5)
            sage: d
            Configuration(w0, w1, w2, w3):
              w0 = Path: 30323212323030103032321232303010303 23
              w1 = Path: 21232303
              w2 = Path: 232121012123230323212101212323030323212101...
              w3 = Path: 10121232
              w4 = Path: 12101030101212321210103010121232123212101
```

```
            w5 = Path: 03010121
            w6 = Path: 0103032303010121010303230301012101030323...
            w7 = Path: 32303010
        (|w0|, |w1|, |w2|, |w3|) = (37, 8, 85, 8)
        (n0, n1, n2, n3)         = (2, 0, 5, 0)
        sage: c == d.old_shrink(2,5) == d.old_shrink(1,3).old_shrink(1,2)
        True
    """
    c = self
    for _ in range(k): c = c.shrink(0)
    for _ in range(l): c = c.shrink(2)
    return c

def exchange(self, i):
    r"""
    EXAMPLES::
    """
    (w0,w1,w2,w3,w4,w5,w6,w7) = [self[j] for j in range(i % 8, 8)\
                                 + range(0, i % 8)]
    indexes = range(i % 8 + 1, 8, 2) + range((1 + i) % 2, i % 8, 2)
    (n1,n3,n5,n7) = [self.n(j) for j in indexes]
    (u1,u3,u5,u7) = [self.u(j) for j in indexes]
    (v1,v3,v5,v7) = [self.v(j) for j in indexes]
    w = (self.hat(w4),(v1*u1)**n1*v1,self.hat(w6),\
         (v3*u3)**n3*v3,self.hat(w0),(v5*u5)**n5*v5,\
         self.hat(w2),(v7*u7)**n7*v7)
    w = tuple(w[j] for j in range(-i % 8, 8) + range(0, -i % 8))
    return Configuration(w, self.bar)

def old_exchange(self):
    r"""
    EXAMPLES::

        sage: fibo = words.fibonacci_tile
        sage: c = Configuration(fibo(2))
        sage: c
        Configuration(w0, w1, w2, w3):
          w0 = Path: 30323                          w4 = Path: 12101
          w1 = Path: 21232303                       w5 = Path: 03010121
          w2 = Path: 23212                          w6 = Path: 01030
          w3 = Path: 10121232                       w7 = Path: 32303010
        (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
```

```
    (n0, n1, n2, n3)         = (0, 0, 0, 0)
    sage: c.old_exchange()
    Configuration(w0, w1, w2, w3):
      w0 = Path: 32303                          w4 = Path: 10121
      w1 = Path: 23                             w5 = Path: 01
      w2 = Path: 21232                          w6 = Path: 03010
      w3 = Path: 12                             w7 = Path: 30
    (|w0|, |w1|, |w2|, |w3|) = (5, 2, 5, 2)
    (n0, n1, n2, n3)         = (1, 0, 1, 0)
    sage: c.old_exchange().old_exchange()
    Configuration(w0, w1, w2, w3):
      w0 = Path: 30323                          w4 = Path: 12101
      w1 = Path: 21232303                       w5 = Path: 03010121
      w2 = Path: 23212                          w6 = Path: 01030
      w3 = Path: 10121232                       w7 = Path: 32303010
    (|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
    (n0, n1, n2, n3)         = (0, 0, 0, 0)
    sage: c.old_exchange().old_exchange() == c
    True
"""
#shorter names
A = self.A
B = self.B
d1 = self.d1
d2 = self.d2
d = d1 + d2

A1 = A[:d1]
A2 = A[-d1:]
B1 = B[:d2]
B2 = B[-d2:]
r = (B2+A1)[:(len(A)-d1)%d]
s = (B1+A2)[:d-len(r)]
m = (self.hat(A1)+B1)[:(len(B)-d2)%d]
n = (self.hat(A2)+B2)[:d-len(m)]
i = int((len(A) - d1 - len(r))) / d
j = int((len(B) - d2 - len(m))) / d

Ap = A2 + (s + r) ** i + s
Bp = B2 + (n + m) ** j + n

return Configuration(Ap, Bp, d1, d2, self.bar)
```

```
def old_mini_shrink(self):
    r"""
    Reduces the configuration in the case ``(|A| - d_1) \bmod (d_1 + d_2) =
    0`` and ``(|B| - d_2) \bmod (d_1 + d_2) \neq 0`` by using the smaller
    periodicity in the overlaps.

    EXAMPLES::

        sage: FREEMAN = WordPaths('abAB')
        sage: c = Configuration(FREEMAN('abAba'), FREEMAN('bAbAb'), 1, 3); c
        Configuration(w0, w1, w2, w3):
          w0 = Path: a                              w4 = Path: A
          w1 = Path: bAba                           w5 = Path: BaBA
          w2 = Path: bAb                            w6 = Path: BaB
          w3 = Path: Ab                             w7 = Path: aB
        (|w0|, |w1|, |w2|, |w3|) = (1, 4, 3, 2)
        (n0, n1, n2, n3)         = (0, 1, 0, 0)
        sage: c.old_mini_shrink()
        Configuration(w0, w1, w2, w3):
          w0 = Path: a                              w4 = Path: A
          w1 = Path: ba                             w5 = Path: BA
          w2 = Path: b                              w6 = Path: B
          w3 = Path: Ab                             w7 = Path: aB
        (|w0|, |w1|, |w2|, |w3|) = (1, 2, 1, 2)
        (n0, n1, n2, n3)         = (0, 1, 0, 1)
    """
    (A,B,d1,d2) = (self.A,self.B,self.d1,self.d2)
    d = d1 + d2
    r = A[d1:d1+(len(A)-d1)%d]
    m = B[d2:d2+(len(B)-d2)%d]

    if len(r) == 0 and len(m) != 0:
        k = gcd(len(m), d - len(m))
        A1 = A[:d1]
        B1 = B[:d2]


        w = A1[:len(A1)%k]
        z = B1[:len(B1)%k]
        Ap = w + z + w
        Bp = z + m
```

```
            return Configuration(Ap, Bp, d1, len(z), self.bar)
        else:
            return Configuration(A, B, d1, d2, self.bar)

    def mini_shrink(self, i):
        r"""
        Reduces the configuration in the case ``|u_1| = 0``
        and ``|u_3| \neq 0`` by using the smaller
        periodicity in the overlaps.

        EXAMPLES::

            sage: c = Configuration(FREEMAN('abAba'), FREEMAN('bAbAb'), 1, 3).
            conjugate()
            sage: c
            Configuration(w0, w1, w2, w3):
              w0 = Path: bAba                          w4 = Path: BaBA
              w1 = Path: bAb                           w5 = Path: BaB
              w2 = Path: Ab                            w6 = Path: aB
              w3 = Path: A                             w7 = Path: a
            (|w0|, |w1|, |w2|, |w3|) = (4, 3, 2, 1)
            (n0, n1, n2, n3)         = (1, 0, 0, 0)
            sage: c.u(0)
            Path:
            sage: c.mini_shrink(2)
            Traceback (most recent call last):
            ...
            NotImplementedError: On ne peut enlever g(=2) a w_2, car |w_2|<=g
        """
        assert self.u(0).is_empty(), "u0 doit etre vide"
        w = w0,w1,w2,w3 = map(self.w, range(4))
        g = gcd(len(w2), len(w1) + len(w3))
        if g < len(w[i]):
            w[i] = w[i][g:]
        else:
            msg = "On ne peut enlever g(=%s) a w_%s, car |w_%s|<=g"%(g,i,i)
            raise NotImplementedError, msg
        return Configuration(w, self.bar)

    def l_shrink(self, i):
        r"""
        TODO
```

```
        """
        #assert len(self.w(i)) == self.d(i), 'on doit avoir w_%s == d_%s'%(i,i)
        w = [self.w((j+i)%8) for j in range(8)]
        d = [self.d((j+i)%8) for j in range(8)]
        g = gcd(len(w[2]),d[2])
        wp = [w[0][g:], w[1][g:], w[2], w[3], w[4][g:], w[5][g:], w[6], w[7]]
        return Configuration(tuple(wp[(j-i)%8] for j in range(8)), self.bar)

    def r_shrink(self, i):
        r"""
        TODO
        """
        #assert len(self.w(i)) == self.d(i), 'on doit avoir w_%s == d_%s'%(i,i)
        w = [self.w((j+i)%8) for j in range(8)]
        d = [self.d((j+i)%8) for j in range(8)]
        g = gcd(len(w[2]),d[2])
        wp = [w[0][:-g], w[1], w[2], w[3][:-g], w[4][:-g], w[5], w[6], w[7][:-g]]
        print tuple(wp[(j-i)%8] for j in range(8))
        return Configuration(tuple(wp[(j-i)%8] for j in range(8)), self.bar)

    def l_extend(self, i):
        r"""
        EXAMPLES::

            sage: c = Configuration(words.fibonacci_tile(1))
            sage: c.l_extend(0)
            Traceback (most recent call last):
            ...
            AssertionError: on doit avoir w_0 == d_0
            sage: c.l_extend(1)
            Configuration(w0, w1, w2, w3):
              w0 = Path: 3                              w4 = Path: 1
              w1 = Path: 0103                           w5 = Path: 2321
              w2 = Path: 010                            w6 = Path: 232
              w3 = Path: 10                             w7 = Path: 32
              (|w0|, |w1|, |w2|, |w3|) = (1, 4, 3, 2)
              (n0, n1, n2, n3)         = (0, 1, 0, 0)
        """
        assert len(self.w(i)) == self.d(i), 'on doit avoir w_%s == d_%s'%(i,i)
        w = [self.w((j+i)%8) for j in range(8)]
        d = [self.d((j+i)%8) for j in range(8)]
        g = gcd(len(w[2]),d[2])
```

```
        p = (w[1] + w[2] + w[3])[:g]
        q = (w[5] + w[6] + w[7])[:g]
        wp = [p + w[0], p + w[1], w[2], w[3], q + w[4], q + w[5], w[6], w[7]]
        return Configuration(tuple(wp[(j-i)%8] for j in range(8)), self.bar)

    def r_extend(self, i):
        r"""
        EXAMPLES::

            sage: c = Configuration(words.fibonacci_tile(1))
            sage: c.r_extend(0)
            Traceback (most recent call last):
            ...
            AssertionError: wiwi+1 = hat(wi+4,wi+5) is broken for i=0
            sage: c.r_extend(1)
            Configuration(w0, w1, w2, w3):
                w0 = Path: 323                          w4 = Path: 101
                w1 = Path: 0323                         w5 = Path: 2101
                w2 = Path: 0                            w6 = Path: 2
                w3 = Path: 10                           w7 = Path: 32
                (|w0|, |w1|, |w2|, |w3|) = (3, 4, 1, 2)
                (n0, n1, n2, n3)         = (0, 1, 0, 0)
        """
        #assert len(self.w(i)) == self.d(i), 'on doit avoir w_%s == d_%s'%(i,i)
        w = [self.w((j+i)%8) for j in range(8)]
        d = [self.d((j+i)%8) for j in range(8)]
        g = gcd(len(w[2]),d[2])
        p = (w[1] + w[2] + w[3])[:g]
        q = (w[5] + w[6] + w[7])[:g]
        wp = [w[0] + q, w[1], w[2], w[3] + p, w[4] + p, w[5], w[6], w[7] + q]
        return Configuration(tuple(wp[(j-i)%8] for j in range(8)), self.bar)

    def reduction(self, iteration=1, verbose=True):
        r"""
        Reduces the current configuration if it is possible

        EXAMPLES::

            sage: c = Configuration(words.fibonacci_tile(3))
            sage: c.reduction(7)[1]
            exchange(0) applied
            shrink(0) applied
```

```
            shrink(2) applied
            exchange(0) applied
            shrink(0) applied
            shrink(2) applied
            not reducible, this is a morphic pentamino
            ['\\EXCHANGE_0', '\\SHRINK_0', '\\SHRINK_2', '\\EXCHANGE_0',\
             '\\SHRINK_0', '\\SHRINK_2']
        """
        # We start with the recursive call
        if iteration >= 2:
            conf,op = self.reduction(1, verbose=verbose)
            conf2,op2 = conf.reduction(iteration - 1, verbose)
            return (conf2, op + op2)

        # Now we check if SHRINK may be applied
        for i in range(8):
            if len(self.w(i)) > self.d(i):
                if verbose:
                    print 'shrink(%s) applied'%i
                return (self.shrink(i), ['\\SHRINK_{%s}'%i])

        # Then we verify if we have a morphic pentamino
        if (len(self.u(0)) == 0 and len(self.u(2)) == 0) or\
           (len(self.u(1)) == 0 and len(self.u(3)) == 0):
            if verbose:
                print 'not reducible, this is a morphic pentamino'
            return (self, [])

        # Otherwise, we try with EXCHANGE
        for i in range(2):
            if len(self.v((i + 1) % 8)) + len(self.v((i + 3) % 8)) <\
               len(self.u((i + 1) % 8)) + len(self.u((i + 3) % 8)):
                if verbose:
                    print 'exchange(%s) applied'%i
                return (self.exchange(i), ['\\EXCHANGE_{%s}'%i])

        raise ValueError, 'case not treated yet !!!\n%s'%self

    def old_reduction(self, iteration=1, verbose=True):
        r"""
        Reduces the current configuration to a smaller one, if possible
        """
```

```
if iteration >= 2:
    return self.old_reduction(1, verbose=verbose).old_reduction\
           (iteration - 1, verbose)
c = self.find_thickest_equivalent_configuration()
(A,B,d1,d2) = (c.A,c.B,c.d1,c.d2)
d = d1 + d2
dp = len(A) + len(B) - d
r = (len(A) - d1) % d
m = (len(B) - d2) % d
(i,j) = c.thickness()
if r == 0: i = max(0, i - 1)
if m == 0: j = max(0, j - 1)
if i >= 1 or j >= 1:
    rep = c.shrink(i,j)
    if verbose:
        print 'shrink applied with parameters', i, 'and', j
        rep = (rep, "$\\SHRINK(%s, %s)$"%(i,j))
elif r == 0 and m == 0:
    rep = None
    if verbose:
        print 'not reductible : r and m both empty'
        rep = (rep, None)
elif d == (len(A) + len(B)) / 2:
    rep = c
    if verbose:
        print 'not reductible : case d = d\''
        rep = (rep, None)
elif r + m > d:
    rep = c.exchange()
    if verbose:
        print 'exchange applied'
        rep = (rep, "\\EXCHANGE")
elif d2 % dp + d1 % dp > dp:
    rep = c.conjugate().exchange()
    if verbose:
        print 'exchange applied to conjugate'
        rep = (rep, "$\\EXCHANGE\\circ\\CONJUGATE$")
elif r == 0:
    rep = c.old_mini_shrink()
    if verbose:
        print 'mini shrink applied'
        rep = (rep, "$\\MSHRINK$")
```

```
        elif m == 0:
            rep = c.conjugate().conjugate().old_mini_shrink()
            if verbose:
                print 'mini shrink applied to conjugate'
                rep = (rep, "$\\MSHRINK\\circ\\CONJUGATE\\circ\\CONJUGATE$")
        else:
            raise Exception, "missing case in reduction."
        return rep

    def find_thickest_equivalent_configuration(self):
        r"""
        Returns the thickest configuration equivalent to this
        one.
        More precisely, using the operators reverse() and conjugate(),
        it chooses the configuration having thickness (i,j) such that
        i + j is maximized

        EXAMPLES::

            sage: c = Configuration(words.fibonacci_tile(1)).conjugate().\
            old_extend(1,2).conjugate()
            sage: c.thickness()
            (0, 0)
            sage: c.find_thickest_equivalent_configuration().thickness()
            (2, 1)
        """
        candidates = [self, self.conjugate(), self.reverse(),\
                      self.reverse().conjugate()]
        distinct_thicknesses = set(map(lambda c:c.thickness(), candidates))
        if len(distinct_thicknesses) > 2:
            print 'this seems to be a counter-example to the constant thickness\
                   hypothesis.'
            print distinct_thicknesses
        return max(candidates, key=lambda c:sum(c.thickness()))

    def factorization_points(self):
        r"""
        Returns the eight factorization points of this configuration
        """
        return [0, self.d1, len(self.A), len(self.A)+self.d2,\
                len(self.A)+len(self.B), len(self.A)+len(self.B)+self.d1,\
                2*len(self.A)+len(self.B), 2*len(self.A)+len(self.B)+self.d2]
```

```
    def plot(self, pathoptions=dict(rgbcolor='black',thickness=3),
             fill=True, filloptions=dict(rgbcolor='black',alpha=0.2),
             startpoint=True, startoptions=dict(rgbcolor='black',pointsize=100),
             endarrow=True, arrowoptions=dict(rgbcolor='black',arrowsize=5,width=3),
             gridlines=False, gridoptions=dict(),
             axes=False):
        r"""
        Returns a 2d Graphics illustrating the double square tile associated to
        this configuration including the factorizations points. The options are
        the same as for instances of WordPaths

        INPUT:

        - ``pathoptions`` - (dict,
          default:dict(rgbcolor='red',thickness=3)), options for the
          path drawing

        - ``fill`` - (boolean, default: True), if fill is True and if
          the path is closed, the inside is colored

        - ``filloptions`` - (dict,
          default:dict(rgbcolor='red',alpha=0.2)), ptions for the
          inside filling

        - ``startpoint`` - (boolean, default: True), draw the start point?

        - ``startoptions`` - (dict,
          default:dict(rgbcolor='red',pointsize=100)) options for the
          start point drawing

        - ``endarrow`` - (boolean, default: True), draw an arrow end at the end?

        - ``arrowoptions`` - (dict,
          default:dict(rgbcolor='red',arrowsize=20, width=3)) options
          for the end point arrow

        - ``gridlines``- (boolean, default: False), show gridlines?

        - ``gridoptions`` - (dict, default: {}), options for the gridlines

        - ``axes`` - (boolean, default: False), options for the axes
```

```
        EXAMPLES:

        The cross of area 5 together with its double square factorization
        points::

            sage: c = Configuration(words.fibonacci_tile(1))
            sage: c.plot()
        """
        path = self.boundary_word()
        points = list(path.points())
        points = [map(RR, x) for x in points]
        G = path.plot(pathoptions, fill, filloptions, startpoint, startoptions,\
                      endarrow, arrowoptions, gridlines, gridoptions)
        i = 0
        for p in self.factorization_points():
            if i % 2 == 0: G += point(points[p],\
                                      pointsize=startoptions['pointsize'],\
                                      rgbcolor="red")
            else: G += point(points[p],\
                             pointsize=startoptions['pointsize'],\
                             rgbcolor="blue")
            i += 1
        return G

    def tikz_trajectory(self, step=1):
        r"""
        Returns a tikz string describing the double square induced by
        this configuration together with its factorization points

        The factorization points respectively get the tikz attribute 'first'
        and 'second' so that when including it in a tikzpicture environment,
        it is possible to modify the way those points appear.

        EXAMPLES::

            sage: c = Configuration(words.fibonacci_tile(1))
            sage: c.tikz_trajectory()
            \filldraw[-triangle 45, very thick, draw=black, fill=black!10]
            (0.000, 0.000) -- (0.000, -1.00) -- (1.00, -1.00) -- (1.00, -2.00)
            -- (2.00, -2.00) -- (2.00, -1.00) -- (3.00, -1.00) -- (3.00, 0.000)
```

```
            -- (2.00, 0.000) -- (2.00, 1.00) -- (1.00, 1.00) -- (1.00, 0.000)
            -- (0.000, 0.000);
            \foreach \i in {(0.0000, 0.0000), (1.000, -2.000), (3.000, -1.000),
            (2.000, 1.000)}
              \node at \i[first] {};
            \foreach \i in {(0.0000, -1.000), (2.000, -2.000), (3.000, 0.0000),
            (1.000, 1.000)}
              \node at \i[second] {};
        """
        from sage.all import n
        from sage.misc.latex import LatexExpr
        f = lambda x: n(x,digits=3)
        step = n(step, digits=4)
        points = map(lambda (x,y):(x*step,y*step),\
                     list(self.boundary_word().points()))
        l = [str(tuple(map(f, pt))) for pt in points]
        s = '\\filldraw[-triangle 45, very thick, draw=black, fill=black!10] '\
            + ' -- '.join(l) + ';'
        [a1, b1, a2, b2, a3, b3, a4, b4] = self.factorization_points()
        #s += '\n\\foreach \\x / \\y in {'
        #for p in [a1, a2, a3]:
        #    x,y = points[p]
        #    s += '%s/%s, ' % (str(x), str(y))
        #x,y = points[a4]
        #s += '%s/%s' % (str(x), str(y))
        #s += '}'
        #s += '\n  \\node[first] at (\\x, \\y) {};\n'
        #s += '\n\\foreach \\x / \\y in {'
        #for p in [b1, b2, b3]:
        #    x,y = points[p]
        #    s += '%s/%s, ' % (str(x), str(y))
        #x,y = points[b4]
        #s += '%s/%s' % (str(x), str(y))
        #s += '}'
        #s += '\n  \\node[second] at (\\x, \\y) {};\n'
        return LatexExpr(s)

    def tikz_reduction(self, size=6, nbcolonnes=3):
        r"""
        INPUT:

        - ``nbcolonnes`` - le nombre de colonnes de l'affichage
```

```
EXAMPLES::

    sage: fibo = words.fibonacci_tile
    sage: c = Configuration(fibo(1))
    sage: c.tikz_reduction()
    \node (q0) at (0, 0.000000000000000) {\begin{tikzpicture}
    \filldraw[-triangle 45, very thick, draw=black, fill=black!10]
    (0.000, 0.000) -- (0.000, -2.00) -- (2.00, -2.00) -- (2.00, -4.00)
    -- (4.00, -4.00) -- (4.00, -2.00) -- (6.00, -2.00) -- (6.00, 0.000)
    -- (4.00, 0.000) -- (4.00, 2.00) -- (2.00, 2.00) -- (2.00, 0.000)
    -- (0.000, 0.000);
    \foreach \i in {(0.0000, 0.0000), (2.000, -4.000), (6.000, -2.000),
    (4.000, 2.000)}
      \node at \i[first] {};
    \foreach \i in {(0.0000, -2.000), (4.000, -4.000), (6.000, 0.0000),
    (2.000, 2.000)}
      \node at \i[second] {};\end{tikzpicture}};
"""
c = self
s = ''
i = 0
fns = []
while True:
    (w, h) = (c.width(), c.height())
    t = c.tikz_trajectory(step=size/max(w,h))
    x,y = serpent(i, nbcolonnes)
    s += '\\node (q%s) '%i
    s += 'at %s '% ( (x*1.5*size, y*1.5*size), )
    s += '{\\begin{tikzpicture}%s\\end{tikzpicture}};\n'%t
    c,func = c.reduction(iteration=1, verbose=False)
    i += 1
    if func == []: break
    fns.append(func[0])
for j,func in zip(range(1, i), fns):
    rotate90 = "" if j%3==0 else ", rotate=90"
    edge = "edge node[midway, rectangle, fill=white%s] "%rotate90
    edge += "{$%s$}"%func
    s += "\\path[->] (q%s) %s (q%s);\n"%(j-1, edge, j)
from sage.misc.latex import LatexExpr
return LatexExpr(s)
```

```
    def width(self):
        r"""
        Returns the width of this polyomino, i.e. the difference
        between its rightmost and leftmost coordinates
        """
        points = list(self.boundary_word().points())
        return max(map(lambda p:p[0], points)) - min(map(lambda p:p[0], points))

    def height(self):
        r"""
        Returns the width of this polyomino, i.e. the difference
        between its uppermost and lowermost coordinates
        """
        points = list(self.boundary_word().points())
        return max(map(lambda p:p[1], points)) - min(map(lambda p:p[1], points))

    def uv_non_simple_dict(self):
        r"""
        Retourne un dictionnaire de la forme
        {'u': liste d'entiers, 'v': liste d'entiers}
        ou les liste d'entiers donne les indices i tels que ui ou vi se
        croisent.

        EXAMPLES::

            sage: c = Configuration(1,4,3,2)
            sage: c.uv_non_simple_dict()
            {'u': [], 'v': [0, 4]}
        """
        d = {}
        d['u'] = [i for i in range(8) if not self.u(i).is_simple() ]
        d['v'] = [i for i in range(8) if not self.v(i).is_simple() ]
        return d

    def uv_closed_dict(self):
        r"""
        Retourne un dictionnaire de la forme
        {'u': liste d'entiers, 'v': liste d'entiers}
        ou les liste d'entiers donne les indices i tels que ui ou vi sont
        fermes (mais non vide!).

        EXAMPLES::
```

```
            sage: c = Configuration(1,4,3,2)
            sage: c.uv_closed_dict()
            {'u': [], 'v': []}

        ::

            sage: c = Configuration(words.fibonacci_tile(1))
            sage: d = c.conjugate().exchange(0)
            sage: d.uv_closed_dict()
            {'u': [], 'v': [0, 2, 4, 6]}
        """
        d = {}
        d['u'] = [i for i in range(8) if self.u(i).is_closed() and\
                  not self.u(i).is_empty() ]
        d['v'] = [i for i in range(8) if self.v(i).is_closed() and\
                  not self.v(i).is_empty() ]
        return d

    def latex_table(self):
        r"""
        Returns a Latex expression of a table containing
        the parameters A, B, X, Y, d1, d2, d, d'1, d'2,
        d', r, m, i and j of this configuration
        """
        from sage.misc.latex import LatexExpr
        remove_coma = lambda s:s.translate(None, ',')
        if_empty = lambda s:'\\varepsilon' if len(s) == 0 else s
        u = [if_empty(remove_coma(self.u(i).string_rep())) for i in range(4)]
        v = [if_empty(remove_coma(self.v(i).string_rep())) for i in range(4)]
        #uv = [self.u(i) for i in range(4)] + [self.v(i) for i in range(4)]
        x = self.uv_non_simple_dict()
        y = self.uv_closed_dict()
        secroisent = ', '.join(['u_%s'%i for i in x['u']] + ['v_%s'%i\
                     for i in x['v']])
        fermes = ', '.join(['u_%s'%i for i in y['u']] + ['v_%s'%i\
                     for i in y['v']])

        s = '\\begin{tabular}{|c|}\n\\hline\n\\\\\n'
        s += '\\begin{tikzpicture}\n'
        s += '  [first/.style={circle,draw=black,fill=gray, inner sep=0pt,\
             minimum size=3pt},\n'
```

```
        s += '   second/.style={rectangle,draw=black,fill=white, inner sep=0pt,\
             minimum size=3pt}]\n'
        s += self.tikz_trajectory(step=5.0/max(self.width(), self.height()))
        s += '\n\\end{tikzpicture} \\\\[1ex] \n\\hline\\\\\n'
        #for i in range(4):
        #    s += '$w_{%s} = %s$\\\\\n'%(i, remove_coma(self[i].string_rep()))
        s += '$(w_0,w_1,w_2,w_3) = (%s,%s,%s,%s)$ \\\\\n'%\
             tuple(len(self[i]) for i in range(4))
        s += '$u_0 = %s$\quad $u_1 = %s$\\\\$u_2 = %s$\quad $u_3 = %s$\\\\\n'%\
             tuple(u[i] for i in range(4))
        s += '$v_0 = %s$\quad $v_1 = %s$\\\\$v_2 = %s$\quad $v_3 = %s$\\\\\n'%\
             tuple(v[i] for i in range(4))
        s += '$(n_0,n_1,n_2,n_3) = (%s,%s,%s,%s)$ \\\\\n'%\
             tuple(self.n(i) for i in range(4))
        s += 'Turning number = %s\\\\\n'%self.turning_number()
        s += 'Self-avoiding = %s\\\\\n'%self.boundary_word().is_simple()
        s += 'Se croisent $=%s$\\\\\n'%secroisent
        s += 'Sont fermes $=%s$\\\\\n'%fermes
        s += '\\hline\n\\end{tabular}\n'
        return LatexExpr(s)

    ####################################
    # Fonctions sur les configurations #
    ####################################

    def is_factor_of_ui_or_vi(self, w):
        r"""
        Returns True if w is a factor of one of the ui or vi of self.
        """
        return any(w.is_factor(self.u(j)) or w.is_factor(self.v(j))\
                   for j in range(8))

    def uv_conservation_for_exchange_dict(self):
        r"""
        Retourne un dictionnaire qui donne les indices i tels que ui, vi et
        leurs chapeaux sont conserves par l'operateur exchange.

        OUTPUT:

            Un dictionaire de la forme
            {'u': liste d'entiers, 'v': liste d'entiers, 'hatu': liste
            d'entiers, 'hatv': liste d'entiers}
```

```
        EXAMPLES::

            sage: c = Configuration(FREEMAN('abAba'), FREEMAN('bAbAb'),1,3).
            old_extend(1,1).conjugate().old_extend(1,2)
            sage: c.uv_conservation_for_exchange_dict()
            {'hatu': [0, 1, 2, 3, 4, 5, 6, 7], 'hatv': [0, 2, 4, 6],
            'u': [0, 1, 2, 3, 4, 5, 6, 7], 'v': [0, 1, 2, 3, 4, 5, 6, 7]}
        """
        e = self.exchange(0)
        hat = self.hat
        d = {}
        d['u'] = [i for i in range(8) if e.is_factor_of_ui_or_vi(self.u(i))]
        d['v'] = [i for i in range(8) if e.is_factor_of_ui_or_vi(self.v(i))]
        d['hatu'] = [i for i in range(8)\
                     if e.is_factor_of_ui_or_vi(hat(self.u(i)))]
        d['hatv'] = [i for i in range(8)\
                     if e.is_factor_of_ui_or_vi(hat(self.v(i)))]
        return d

    def checks_uv_conservation_for_exchange_operator(self, hat_allowed=True):
        r"""
        Checks which `u_i` and `v_i` of the given configuraiton
        are preserved when applying the exchange operator

        INPUT:

        - ``self`` - a configuration.
        - ``hat_allowed`` - a boolean (default: ``True``). If True, then cheks
          also if `\hat{u_i}` and `\hat{v_i}` occur.

        EXAMPLES::

            sage: c = Configuration(FREEMAN('abAba'), FREEMAN('bAbAb'),1,3).
            old_extend(1,1).conjugate().old_extend(1,2)
            sage: c.checks_uv_conservation_for_exchange_operator()
            ['u_0', 'u_1', 'u_2', 'u_3', 'u_4', 'u_5', 'u_6', 'u_7', 'v_0',\
             'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6', 'v_7', '\\hat{u_0}',\
             '\\hat{u_1}', '\\hat{u_2}', '\\hat{u_3}', '\\hat{u_4}',\
             '\\hat{u_5}', '\\hat{u_6}', '\\hat{u_7}', '\\hat{v_0}',\
             '\\hat{v_2}', '\\hat{v_4}', '\\hat{v_6}']
```

```
        """
        uv = []
        d = self.uv_conservation_for_exchange_dict()
        uv.extend('u_%s'%j for j in d['u'])
        uv.extend('v_%s'%j for j in d['v'])
        if hat_allowed:
            uv.extend('\\hat{u_%s}'%j for j in d['hatu'])
            uv.extend('\\hat{v_%s}'%j for j in d['hatv'])
        return uv

################################
# String manipulation helpers #
################################

def tableau_de_col(col1, col2, espace=3):
    r"""
    EXAMPLES::

        sage: col1 = ['ab','asdfasdf', 'adf']
        sage: col2 = ['11', '1313', '131313', '1313']
        sage: print tableau_de_col(col1, col2)
        ab         11
        asdfasdf   1313
        adf        131313
                   1313
    """
    from itertools import izip_longest
    largeur = max(map(len, col1))
    it = izip_longest(col1,col2, fillvalue='')
    espace = ' '*espace
    L = [a.ljust(largeur) + espace + b for (a,b) in it]
    return '\n'.join(L)

##################################
# Finding a square factorization #
##################################

def find_square_factorisation(ds, factorisation=None, alternate=True):
    r"""
    Return a square factorisation of the double square ds, distinct from
    factorisation.
```

```
INPUT:

- ds - word, a tile
- factorisation - tuple (optional), a known factorisation
- alternate - bool (optional, default True), if True the search for the
  second factorisation is restricted to those who alternates with the
  first factorisation

OUTPUT:

tuple of four positions of a square factorisation

EXAMPLES::

    sage: fibo = words.fibonacci_tile
    sage: find_square_factorisation(fibo(1))
    (0, 3, 6, 9)
    sage: find_square_factorisation(fibo(0))
    (0, 1, 2, 3)
    sage: find_square_factorisation(fibo(1))
    (0, 3, 6, 9)
    sage: find_square_factorisation(fibo(2))
    (0, 13, 26, 39)
    sage: find_square_factorisation(fibo(3))
    (0, 55, 110, 165)

::

    sage: f = find_square_factorisation(fibo(3));f
    (0, 55, 110, 165)
    sage: find_square_factorisation(fibo(3),f)
    (34, 89, 144, 199)
    sage: find_square_factorisation(fibo(3),f,False) #optional long
    (34, 89, 144, 199)

::

    sage: find_square_factorisation(christo_tile(4,5))
    (0, 7, 28, 35)
    sage: find_square_factorisation(christo_tile(4,5),_)
    (2, 27, 30, 55)
```

```
::

    sage: find_square_factorisation(Words('abcd')('aaaaaa'))
    Traceback (most recent call last):
    ...
    ValueError: pas de factorisation carree
    sage: find_square_factorisation(Words('abcd')('aaaaaa'),(1,2,3,4))
    Traceback (most recent call last):
    ...
    ValueError: pas de seconde factorisation carree

"""
e,n,w,s = ds.parent().alphabet()
bar = WordMorphism({e:w,w:e,n:s,s:n},codomain=ds.parent())
hat = lambda x:bar(x).reversal()

l = ds.length()
demiper = l/2
aucarre = ds * ds

if factorisation and alternate:
    a,b,c,d = factorisation
    it = ((debutA,finA) for debutA in range(a+1,b)\
          for finA in range(b+1,a+demiper))
else:
    it = ((debutA,finA) for debutA in range(demiper)\
          for finA in range(debutA+1,debutA+demiper+1) )

for debutA,finA in it:
    new = (debutA,finA,(debutA+demiper)%l,(finA+demiper)%l)
    if factorisation and set(factorisation) == set(new):
        continue
    A = aucarre[debutA:finA]
    B = aucarre[finA:debutA+demiper]
    A2 = aucarre[debutA+demiper:finA+demiper]
    B2 = aucarre[finA+demiper:l+debutA]
    if A == hat(A2) and B == hat(B2):
        return new

if factorisation is None:
    raise ValueError, 'pas de factorisation carree'
else:
```

```
        raise ValueError, 'pas de seconde factorisation carree'

def find_good_ds_factorisation(ds, delay='minimize'):
    r"""
    Returns the factorizations such that d(A,X) + d(B,Y) is
    minimized (or maximized) where d(x,y) is the distance between
    the start of the block x and the start of the block y.

    INPUT:

    - ``ds`` - double square
    - ``delay`` - 'minimize' or 'maximize'

    OUTPUT:

    tuple of four integers (start of A, end of A, start of X, end of X)

    EXAMPLES::

        sage: fibo = words.fibonacci_tile
        sage: find_good_ds_factorisation(fibo(1))
        (2, 5, 3, 6)
        sage: find_good_ds_factorisation(fibo(2))
        (8, 21, 13, 26)
        sage: find_good_ds_factorisation(fibo(3))
        (34, 89, 55, 110)
        sage: find_good_ds_factorisation(fibo(1),delay='maximize')
        (0, 3, 2, 5)
        sage: find_good_ds_factorisation(fibo(2),delay='maximize')
        (0, 13, 8, 21)
        sage: find_good_ds_factorisation(fibo(3),delay='maximize') #optional long
        (0, 55, 34, 89)

    """
    f = find_square_factorisation(ds)
    g = find_square_factorisation(ds, f, alternate=True)
    #print f,g
    debutA,finA,debutAhat,finAhat = sorted(f)
    debutX,finX,debutXhat,finXhat = sorted(g)

    l = ds.length()
    demiper = l//2
```

```
    d1 = debutX - debutA
    d2 = finX - finA
    #print d1,d2,demiper

    if (delay == 'minimize' and d1 + d2 > demiper/2) or\
       (delay == 'maximize' and d1 + d2 < demiper/2):
        #change the factorisation: B becomes X, X becomes A
        debutX,finX,debutA,finA = finA,debutA+demiper,debutX,finX
        d1 = debutX - debutA
        d2 = finX - finA
    #print d1,d2,demiper
    return debutA,finA,debutX,finX

#------------------------------#
# Enumeration of double squares #
#------------------------------#

Freeman = words.fibonacci_tile(1).parent()
F_BAR   = WordMorphism({0:2, 1:3, 2:0, 3:1}, codomain=Freeman)
F_HAT   = lambda w: F_BAR(w.reversal())
F_RHO   = WordMorphism({0:1, 1:2, 2:3, 3:0}, codomain=Freeman)
F_SIGMA = WordMorphism({0:2, 1:1, 2:0, 3:3}, codomain=Freeman)

def isometric_paths(path):
    return [path, F_RHO(path), F_RHO(F_RHO(path)), F_RHO(F_RHO(F_RHO(path))),\
            F_SIGMA(path), F_RHO(F_SIGMA(path)), F_RHO(F_RHO(F_SIGMA(path))),\
            F_RHO(F_RHO(F_RHO(F_SIGMA(path))))]

def all_double_squares_iterator(max_iteration=None,\
                                max_length=None,\
                                verbose=False):
    if max_iteration is None:
        max_iteration = Infinity
    if max_length is None:
        max_length = Infinity
    import heapq
    queue = [Configuration(words.fibonacci_tile(1))]
    visited = set([])
    i = 0
    while queue and i < max_iteration:
        tile = heapq.heappop(queue).isometric_representant()
        if not tile in visited:
```

```
if tile.boundary_word().is_simple():
    if verbose:
        print i, tile.boundary_word()
    yield tile.boundary_word()
    i += 1
visited |= set([tile])
# Extend operator
for j in range(4):
    t = tile.extend(j)
    if len(t.boundary_word()) <= max_length:
        heapq.heappush(queue, t)
# Extend operator
for j in range(2):
    t = tile.exchange(j)
    if len(t) >= len(tile) and len(t.boundary_word()) <= max_length:
        heapq.heappush(queue, t)
```

BIBLIOGRAPHIE

Adamczewski, B. 2002. « Codages de rotations et phénomènes d'autosimilarité », *J. Théorie des nombres de Bordeaux*, vol. 14, p. 351–386.

Alessandri, P. 1995. Classification et représentation des mots de complexité $n + 2$. Rapport, Université Aix-Marseille II.

Allouche, J.-P. 1997. « Schrödinger operators with Rudin–Shapiro potentials are not palindromic. Quantum problems in condensed matter physics », vol. 38, no. 4, p. 1843–1848.

Allouche, J.-P. et J. Shallit. 1999. « The ubiquitous Prouhet-Thue-Morse sequence ». In *Sequences and their applications, Proceedings of SETA'98*, p. 1–16. Springer.

———. 2000. « Sums of digits, overlaps, and palindromes », *Discrete Math. & Theoret. Comput. Sci*, vol. 4, p. 1–10.

Allouche, J.-P. et J. O. Shallit. 2003. *Automatic Sequences - Theory, Applications, Generalizations*. Cambridge University Press.

Arnoux, P., V. Berthé, et A. Siegel. 2004. « Two-dimensional iterated morphisms and discrete planes », *Theor. Comput. Sci.*, vol. 319, no. 1-3, p. 145–176.

Beauquier, D. et M. Nivat. 1991. « On translating one polyomino to tile the plane », *Discrete Comput. Geom.*, vol. 6, p. 575–592.

Berstel, J. 1995. Axel Thue's papers on repetitions in words : a translation. Publications du LaCIM no. 20, Université du Québec à Montréal. 85 pages.

Berthé, V. et S. Labbé. 2011. « An arithmetic and combinatorial approach to three-dimensional discrete lines ». In *Actes de colloques de DGCI2001*, p. 47–58.

Blondin Massé, A. 2008. « Sur le défaut palindromique des mots finis et infinis ». Mémoire de maîtrise, Université du Québec à Montréal.

Blondin Massé, A., S. Brlek, A. Garon, et S. Labbé. 2008. « Palindromic lacunas of the Thue-Morse word », *Pure Mathematics and Applications*, vol. 19, no. 2-3, p. 39–52.

Blondin Massé, A., S. Brlek, A. Garon, et S. Labbé. 2011. « Every polyomino yields at most two square tilings ». In *Lattice Paths 2011, 7e Conférence internationale sur les chemins discrets et leurs applications, 4-7 juillet, 2011, Siene, Italie*, p. 57–61.

Blondin Massé, A., S. Brlek, A. Glen, et S. Labbé. 2007. « On the critical exponent of generalized Thue-Morse words », *Discrete Mathematics & Theoretical Computer Science*, vol. 9, no. 1.

Blondin Massé, A., S. Brlek, et S. Labbé. 2012. « A square tile fills the plane by translation in at most two distinct ways », *Discrete Applied Mathematics*. À paraître.

Blondin Massé, A., S. Brlek, S. Labbé, et L. Vuillon. 2009. « Codings of rotations on two intervals are full ». In Nešetřil, J. et A. Raspaud, éditeurs, *EuroComb 2009, 7e Conférence européenne de combinatoire*. T. 34, p. 289–293.

Blondin Massé, A., S. Brlek, S. Labbé, et L. Vuillon. 2011. « Return words in codings of rotations ». Sous presse.

Blondin Massé, A., A. Garon, et S. Labbé. 2011a. « Generation of double square tiles ». In *GASCOM 2010, 7e Conférence internationale sur la géométrie discrète et l'imagerie numérique*.

———. 2011b. « Generation of double square tiles », *Theoretical Computer Science*. Soumis.

Blondin Massé, A. et G. Paquin. 2009. Communication personnelle.

Blondin Massé, A., G. Paquin, et L. Vuillon. 2010. « A Fine and Wilf's theorem for pseudoperiods and Justin's formula for generalized pseudostandard words ». In *8e Journées montoises d'informatique théorique*.

Borel, J.-P. et C. Reutenauer. 2006. « On Christoffel classes », *RAIRO-Theoretical Informatics and Applications*, vol. 40, p. 15–28.

Brlek, S. 1989. « Enumeration of factors in the Thue-Morse word. », *Discrete Applied Mathematics*, p. 83–96.

Brlek, S., S. Hamel, M. Nivat, et C. Reutenauer. 2004. « On the palindromic complexity of infinite words », *Int. J. Found. Comput. Sci.*, vol. 15, no. 2, p. 293–306.

Brlek, S., M. Koskas, et X. Provençal. 2011a. « A linear time and space algorithm for detecting path intersection in $\mathbb{Z}^d$ », *Theor. Comput. Sci.* In press.

Brlek, S., M. Koskas, et X. Provençal. 2011b. « A linear time and space algorithm for detecting path intersection in $\mathbb{Z}^d$ », *Theor. Comput. Sci.*, vol. 412, no. 36, p. 4841–4850.

Brlek, S., G. Labelle, et A. Lacasse. 2005. « A note on a result of Daurat and Nivat ». In *Developments in Language Theory*. T. 3572, série *Lecture Notes in Comput. Sci.*, p. 189–198, Berlin. Springer.

———. 2006a. « Properties of contour path of discrete sets », *I. J. of Found. of Comput.*

Sci., vol. 17, no. 3, p. 543–556.

Brlek, S., G. Labelle, et A. Lacasse. 2006b. « Properties of the contour path of discrete sets », *Int. J. Found. Comput. Sci.*, vol. 17, no. 3, p. 543–556.

Brlek, S., J.-O. Lachaud, X. Provençal, et C. Reutenauer. 2009. « Lyndon + Christoffel = digitally convex », *Pattern Recognition*, vol. 42, no. 10, p. 2239–2246.

Brlek, S., X. Provençal, et J.-M. Fédou. 2009. « On the tiling by translation problem », *Discrete Applied Mathematics*, vol. 157, no. 3, p. 464–475.

Bucci, M., A. D. Luca, A. Glen, et L. Zamboni. 2009. « A connection between palindromic and factor complexity using return words », *Adv. in Appl. Math.*, vol. 42, p. 60–74.

Czeizler, E., E. Czeizler, L. Kari, et S. Seki. 2009. « An extension of the Lyndon Schützenberger result to pseudoperiodic words ». In *DLT '09 : Actes de colloques de la 13 Conférence internationale sur les récents résultats en théorie des langages*, p. 183–194, Berlin, Heidelberg. Springer-Verlag.

Damanik, D. et D. Lenz. 2000. « Uniform spectral properties of one-dimensional quasicrystals, iv. quasi-sturmian potentials », *I. Absence of eigenvalues, Commun. Math. Phys*, vol. 212, p. 687–696.

Daurat, A. et M. Nivat. 2003. « Salient and reentrant points of discrete sets ». In *9th International Workshop on Combinatorial Image Analysis*. T. 12, série *Electron. Notes Discrete Math.*, p. 12 pp. (electronic), Amsterdam. Elsevier.

De Luca, A. 1997. « Sturmian words : structure, combinatorics, and their arithmetics », *Theoret. Comput. Sci.*, vol. 183, p. 45–82.

de Luca, A. et A. De Luca. 2006. « Pseudopalindrome closure operators in free monoids », *Theoret. Comput. Sci.*, vol. 362, no. 1-3, p. 45–82.

Droubay, X., J. Justin, et G. Pirillo. 2001. « Episturmian words and some constructions of de Luca and Rauzy », *Theoret. Comput. Sci.*, vol. 255, p. 539–553.

Fernique, T. 2006. « Multidimensional sturmian sequences and generalized substitutions », *Int. J. Found. Comput. Sci.*, vol. 17, no. 3, p. 575–600.

Garon, A. 2010. « Équations sur les mots et tuiles doublement pavantes ». Mémoire de maîtrise, Université du Québec à Montréal.

Glen, A., A. Lauve, et F. V. Saliola. 2008. « A note on the Markoff condition and central words », *Inf. Process. Lett.*, vol. 105, p. 241–244.

Golomb, S. 1970. « Tiling with sets of polyominoes », *Journal of Combinatorial Theory*, vol. 9, p. 60–71.

Haselgrove, C. et J. Haselgrove. 1960. « A computer program for pentominoes », *Eureka*,

vol. 23, p. 16–18.

Hof, A., O. Knill, et B. Simon. 1995. « Singular continuous spectrum for palindromic schrödinger operators », *Commun. Math. Phys*, vol. 174, p. 149–159.

Justin, J. 2005. « Episturmian morphisms and a Galois theorem on continued fractions », *RAIRO-Theor. Inform. and Appl.*, vol. 39, no. 1, p. 207–215.

Justin, J. et L. Vuillon. 2000. « Return words in sturmian and episturmian words », *RAIRO Theoretical Informatics and Applications*, vol. 34, p. 343–356.

Kari, L. et K. Mahalingam. 2010. « Watson-Crick palindromes in DNA computing », vol. 9, p. 297–316.

Katok, A. 1980. « Interval exchange transformations and some special flows are not mixing », *Israel J. Math.*, vol. 35, no. 4, p. 301–310.

Keane, M. 1975. « Interval exchange transformations », *Math. Zeit.*, vol. 141, p. 25–31.

Lenz, D. 2003. « Hierarchical structures in Sturmian dynamical systems », *Theor. Comput. Sci.*, vol. 303, p. 463–490.

Lothaire, M. 1983. *Combinatorics on Words*. Addison-Wesley.

———. 1997. *Combinatorics on Words*. Cambridge University Press, 2e édition.

Marathe, A., A. E. Condon, et R. M. Corn. 1999. « On combinatorial dna word design », *Journal of Computational Biology*, vol. 8, p. 201–220.

Monnerot Dumaine, A. 2009. The Fibonacci Word fractal. 24 p.

Pelantová, E. et S. Starosta. 2011. « Infinite words rich and almost rich in generalized palindromes ». In *Developments in Language Theory*, p. 406–416.

Pirillo, G. 1999. « A new characteristic property of the palindrome of standard sturmian word », *Sém. Lothar. Combin.*, vol. 43, p. 1–3.

Provençal, X. 2008. « Combinatoire des mots, géométrie discrète et pavages ». Thèse de Doctorat, D1715, Université du Québec à Montréal.

Rauzy, G. 1979. « Echange d'intervalles et transformations induites », *Acta. Arith.*, vol. 34, p. 315–328.

Rhoads, G. 2003. « Planar tilings and the search for an aperiodic prototile ». Thèse de Doctorat, Rutgers University.

Rote, G. 1994. « Sequences with subword complexity $2n$ », *J. Number Theory*, vol. 46, p. 196–213.

Sloane, N. J. 2007. « The on-line encyclopedia of integer sequences ». In *Actes de conférence du 14th symposium on Towards Mechanized Mathematical Assistants :*

6th International Conference. Coll. « Calculemus '07 / MKM '07 », p. 130–130, Berlin, Heidelberg. Springer-Verlag.

Vuillon, L. 1998. « Combinatoire des motifs d'une suite sturmienne bidimensionnelle », *Theor. Comput. Sci.*, vol. 209, no. 1-2, p. 261–285.

Vuillon, L. 2001. « A characterization of sturmian words by return words », *European Journal of Combinatorics*, vol. 22, p. 263–275.

———. 2007. « On the number of return words in infinite words constructed by interval exchange transformations », *PU.M.A.*, vol. 18, p. 345–355.

Wang, H. 1961. « Proving theorems by pattern recognition ii », *Bell System Tech. Journal*, vol. 40, no. 1, p. 1–41.

INDEX