Université Paris Diderot – Paris 7

École Doctorale de Sciences Mathématiques de Paris Centre
Laboratoire d'Informatique Algorithmique: Fondements et Applications

THÈSE

en vue d'obtenir le grade de
Docteur de l'Université Paris Diderot
Spécialité informatique

présentée et soutenue par

Adeline Pierrot

Combinatoire et algorithmique dans les classes de permutations

Thèse dirigée par Dominique Rossin

Soutenue publiquement le
26 juin 2013

devant le jury composé de:

Directeur de thèse:	Dominique Rossin
Rapporteurs:	Michael Albert
	Mireille Bousquet-mélou
Examinateurs:	Frédérique Bassino
	Robert Cori
	Frédéric Magniez
	Cyril Nicaud

Contents

Résumé

Cette thèse porte sur l'étude des classes de permutations à motifs exclus. Une analyse combinatoire des permutations via leur décomposition par substitution permet d'obtenir des résultats algorithmiques. La première partie de la thèse étudie la structure des classes de permutations. Plus précisément on donne un algorithme pour calculer une spécification combinatoire pour une classe de permutations donnée par sa base de motifs exclus. La spécification est obtenue si et seulement si la classe contient un nombre fini de permutations simples, cette condition étant testée par l'algorithme lui-même. Cet algorithme puise sa source dans les travaux de Albert et Atkinson établissant qu'une classe ayant un nombre fini de permutations simples a une base finie et une série génératrice algébrique, et dans les travaux suivants de Brignall et al. Les méthodes développées utilisent la théorie des langages et des automates, les ensembles partiellement ordonnés, l'introduction de motifs obligatoires.

La seconde partie de la thèse donne un algorithme polynomial décidant si une permutation donnée en entrée est triable par deux piles connectées en série. L'existence d'un algorithme polynomial résolvant cette question est un problème longtemps resté ouvert, que l'on clôt dans cette thèse en introduisant une nouvelle notion, le tri par sas, qui est une restriction du tri par piles général. On résout d'abord le problème de décision dans le cas particulier du tri par sas, en utilisant un codage des procédures de tri par un bicoloriage du diagramme des permutations. Puis on résout le problème général en montrant qu'une procédure de tri général correspond à plusieurs étapes de tri par sas qui doivent être compatibles.

Abstract

This work is dedicated to the study of pattern closed classes of permutations. Algorithmic results are obtained thanks to a combinatorial study of permutation classes through their substitution decomposition. The first part of the thesis focuses on the stucture of permutation classes. More precisely, we give an algorithm which derives a combinatorial specification for a permutation class given by its basis of excluded patterns. The specification is obtained if and only if the class contains a finite number of simple permutations, this condition being tested algorithmically. This algorithm takes its root in the theorem of Albert and Atkinson stating that every permutation class containing a finite number of simple permutations has a finite basis and an algebraic generating function, and its developments by Brignall and al.

The second part of the thesis gives a polynomial algorithm deciding whether a permutation given as input is sortable trough two stacks in series. The existence of a polynomial algorithm answering this question is a problem that stayed open for a long time, which is solved in this thesis by introducing a new notion, the pushall sorting, which is a restriction of the general stack sorting. We first solve the decision problem in the particular case of the pushall sorting, by encoding the sorting procedures through a bicoloring of the diagrams of the permutations. Then we solve the general case by showing that a sorting procedure in the general case corresponds to several steps of pushall sorting.

Remerciements

Je remercie Dominique, tout d'abord pour m'avoir donné l'envie de faire une thèse, ce que je ne comptais pas faire avant mon stage de M2 sous sa direction, et aussi pour son encadrement décontracté mais attentif. Sa disponibilité malgré l'éloignement relatif de nos laboratoires respectifs, et son dynamisme lors de nos séances de travail ont été une grande motivation. Je le remercie enfin de la liberté et de la confiance qu'il m'a toujours accordées, et du fait que l'on ait réellement travaillé ensemble, presque comme deux collègues plutôt que comme un directeur et un dirigé.

Je remercie mes rapporteurs, Mireille Bousquet-Mélou et Michael Albert, pour avoir accepté la lourde tâche de rapporter ma thèse, pour leur relecture attentive et leurs commentaires précis et utiles pour améliorer le manuscrit.

Merci également à Frédérique Bassino, Robert Cori, Frédéric Magniez et Cyril Nicaud d'avoir accepté de faire partie du jury.

De façon peut-être un peu étrange, je voudrais remercier deux lieux : l'ENS et le LIAFA, deux lieux qui ont en commun d'être extrêmement riches aussi bien sur le plan scientifique que sur le plan humain et dans lesquels je me suis sentie dans mon élément.

J'ai eu la chance de bénéficier de l'environnement extrêmement favorable que constitue le LIAFA pour effectuer ma thèse, que ce soit du point de vue de son positionnement géographique stratégique au cœur de Paris, de l'efficacité de ses secrétaires Noëlle et Nathalie (qu'elles soient ici remerciées), de la cohérence de ses thématiques scientifiques et de sa relative petite taille, qui avec la convivialité de ses membres en font une grande famille dans laquelle on a un réel plaisir à échanger entre membres de toutes les équipes, de ses nombreux séminaires et groupes de travail, en particulier le séminaire thésard commun avec PPS, à la fois une grande source de culture générale, une occasion de mieux comprendre ses voisins de bureau, et un moment de détente partagée avec les inconditionnels croissants, sans oublier le fameux gâteau du vendredi, une institution coup de pouce à la convivialité et aux échanges entre équipes, et surtout l'excellente ambiance qui règne entre les thésards (les repas tous ensemble à midi, le "poulpe", les sorties thésards, et surtout l'entraide permanente, que ce soit à propos de questions scientifiques, de LaTeX, de l'anglais, des enseignements ou de l'administration !). Je voudrais remercier plus particulièrement Mathilde, ma grande sœur de thèse, que j'ai toujours considérée un peu comme un modèle, Cezara, d'une certaine façon un peu une grande sœur elle aussi, Jérémie, Antoine et Bruno, mes dieux du système, Élie, source de culture scientifique inépuisable, Luc, Jehanne, Irène, Timo, Jad, Axel, ADG, Denis... Je voudrais aussi remercier l'équipe combinatoire du LIAFA, et surtout les femmes qui en font partie ou qui viennent à son séminaire, en particulier Anne, Enrica, Sylvie, Marie et Dominique, bien sûr pour leur convivialité, mais aussi d'être une preuve que l'on peut être une femme accomplie et faire de la recherche. Je ne l'aurais probablement pas envisagé pour moi-même si je n'avais pas connu leur exemple.

Mais je ne serais jamais arrivée au LIAFA si je n'avais pas eu auparavant la chance de passer par l'ENS. Ce qui pour moi fait la richesse de l'ENS, ce sont avant tout les personnes qui la rendent vivante, et en particulier ses élèves. Bien sûr par leurs qualités scientifiques, mais surtout par leurs qualités humaines. Mon entrée à l'ENS a été un peu comme la découverte d'une seconde famille, et je suis certaine que les amitiés qui s'y sont nouées le seront pour la vie. Véritable ville dans la ville, entre les cours, l'internat, la cantine, et surtout les multiples activités, tellement nombreuses et attirantes que l'on ne sait où donner de la tête ! En ce qui me concerne j'ai trouvé mon bonheur avec le club cirque, que je remercie pour les vendredis soirs, les week-ends et les conventions, en particulier Émilia et Florent, Christophe et Séverine, Denise et Jérôme, Chloé et Rémi, Ismaël, et Claude. Merci à Émilia et Claude de m'avoir fait découvrir le cirque par leur enthousiasme et leur patience, et aux autres de m'avoir donné envie de continuer grâce à leur complicité et la bonne ambiance qu'ils font régner au club ! Je n'oublie pas non plus les multiples clubs de danse : rock, valse, west coast swing, pompoms, danse indienne... Merci à tous les bons cavaliers ;) et à tous ceux qui ont contribué à faire de ces clubs des endroits chaleureux et vivants ! En particulier un grand merci à David pour m'avoir fait découvrir le west coast swing, pour les confidences autour d'un verre et pour les balades dans Paris. Merci à Rachel, blonde foncée (ben oui ça se voit extérieurement) pour avoir éveillé la danseuse qui sommeillait en moi et pour m'avoir écrit des rôles sur mesure dans ses pièces de théâtre, sans compter son apport à ma culture cinématographique de blonde (avec la complicité d'Émilia, Denise, Irène et Manon !). Merci aussi à Camille, Simon, Fathi, Marie, Daphné, Rémi, Béa et Matthias, pour la danse bien sûr, mais pas uniquement. Enfin je garderai toujours un excellent souvenir de la courô, oasis de calme et de beauté verdoyante au milieu du tourbillon parisien !

Au delà de mon ancienne école et de mon laboratoire, j'ai l'impression d'appartenir à une communauté extrêmement chaleureuse dans laquelle je me suis épanouie durant ma thèse, qui est la communauté de l'informatique-mathématique, et plus particulièrement sa composante Aléa. Je remercie le GDR-IM pour les journées nationales et les écoles jeunes chercheurs, une occasion bienvenue de rencontrer des thésards d'autres horizons et de thématiques proches, et d'échanger nos expériences respectives. Merci aux membres de l'ANR Magnum pour nos réunions et discussions extrêmement enrichissantes, et à l'équipe combinatoire du LaBRI pour les JBC, placées sous les auspices de la bonne chère. Enfin et surtout, je remercie tous les organisateurs successifs et participants à la conférence Aléa, dont la réputation n'est plus à faire. J'apprécie chaque année sa qualité scientifique, son cadre idyllique au bord des calanques, l'hébergement au CIRM, les pauses midi à rallonge, le ping-pong, les joyeuses séances d'exercices, la fameuse soirée bouillabaisse, et plus généralement l'excellente ambiance qui y règne ! Je voudrais remercier plus particulièrement Alice (pour le partage de chambres à Aléa et à Montréal, les soirées jeux et le cirque sur les pelouses d'Oxford !), Basile (pour la musique, le Pérou, et les discussions existentielles), Julien C. (pour la complicité et le squattage; pratique l'appart près de la gare !), Gwendal (pour le Japon, le chabada et son flegme), Julien D. (pour les remparts de Séville...), Valentin (pour son organisation sans faille de magnifiques voyages), Jérémie, Matthieu, Axel et Adrien, chacun unique à sa façon, et je n'oublie pas Florent, les balades dans les calanques et les loups-garous.

Au delà de ces exemples particuliers, je voudrais remercier plus généralement le milieu de la recherche et de l'enseignement supérieur français en mathématiques et informatique fondamentale, ou du moins la partie que j'ai eu l'occasion de côtoyer. Une fois de plus je voudrais attirer l'attention sur les qualités humaines plutôt que sur les qualités scientifiques, qui d'ailleurs ne sont pas en reste. J'ai vraiment l'impression que les gens essaient de s'entraider et de se comprendre pour faire ensemble de belles choses, plutôt que de tenter de tirer la couverture à soi comme cela peut malheureusement être parfois le cas ailleurs ou dans d'autres domaines. De plus la confiance et l'autogestion des horaires de travail et des choix scientifiques laissés aux chercheurs sont extrêmement stimulants.

Bien entendu la recherche n'est tout de même pas totalement un monde de bisounours, et tout n'est pas toujours rose. La liberté implique des choix qui ne sont pas toujours évidents, voire anxiogènes. De plus le caractère extrêmement pointu de la recherche fait que même nos voisins de bureau ne comprennent pas vraiment ce que nous faisons, d'où parfois le sentiment d'être vraiment seul face à ses problèmes, et totalement incompris. Et aussi l'impression de ne pas être à la hauteur lorsqu'on cherche et que l'on ne trouve pas, les moments de découragement lorsqu'on s'aperçoit que notre piste ne mène nulle part, et l'irritante petite question dans les moments de doute : "Mais à quoi ça sert tout ça ?".

Heureusement dans ce cas il suffit de s'ouvrir un peu, et il se trouve toujours un collègue compatissant qui prête une oreille attentive à nos doléances, et qui nous explique pourquoi même si tout cela est bien vrai, il n'échangerait sa place contre rien au monde !

Je voudrais remercier mes coauteurs, en particulier Dominique bien sûr, mais aussi Mathilde, Frédérique, Carine, Guillaume et Mathieu. Le travail en groupe est toujours extrêmement convivial et stimulant, et la confrontation avec des points de vues et des méthodes et personnalités variées est extrêmement enrichissant.

Parmi les gens que je ne sais où placer car ils rentrent dans plusieurs cases, je voudrais remercier Sandrine pour les discussions existentielles, ainsi que Hervé.

Je remercie aussi tous mes amis qui n'ont aucun rapport avec le monde de l'enseignement supérieur et de la recherche, pour m'aider à rester ouverte au "monde réel" et à garder les pieds sur terre. J'ai une pensée particulière pour mes deux amies d'enfance Claire et Danaé. On se voit de moins en moins souvent, mais c'est chaque fois un grand plaisir !

Merci à tous les anonymes ou amis qui m'ont demandé au détour d'une conversation en quoi consiste la recherche en maths et si ça sert à quelque chose, ce qui m'a permis de prendre du recul en tentant de leur répondre.

Je remercie aussi les professeurs qui ont participé à développer mon goût pour le savoir et les sciences, et j'en profite pour faire un pied de nez à ceux qui auraient pu m'en dégoûter – spéciale dédicace à mes professeurs de mathématiques de 4^{e} et de seconde !

Je dois beaucoup à ma famille, que je remercie de m'avoir toujours soutenue. Je sais que quels que soient les problèmes dans le grand vilain monde extérieur, je pourrai toujours trouver à la maison un cocon protecteur, havre de paix et de quiétude. En particulier je remercie mon père de me comprendre et pour ses conseils, ma mère d'être une source inépuisable d'optimisme, toujours à voir coûte que coûte le bon côté des choses et à chercher des solutions (souvent astucieuses !), ma sœur d'être ma complice depuis tant d'années (on ne compte plus les crises de fous rires et les confidences !), et mon frère simplement de continuer à être là de temps en temps malgré la distance.

J'ai une petite pensée pour mes grands-parents, dont je devine la fierté.

Je remercie plus globalement la famille plus élargie pour tous les bons moments passés ensemble. Quelle chance d'avoir une famille si soudée !

Enfin, merci à toi qui te reconnaîtra d'être tombé du ciel au moment où j'avais le plus besoin de toi. Ta présence à mes côtés, ta confiance et ta patience me rendent plus forte et me poussent à donner le meilleur de moi-même, en particulier durant cette période intense de fin de thèse.

Introduction

L'un des objectifs des mathématiques est de fournir une base théorique à l'étude du monde réel à travers l'utilisation de ses modèles abstraits en physique, chimie ou biologie. En effet le réel est intrinsèquement compliqué et il est nécessaire de le simplifier, de l'abstraire et de le modéliser pour le comprendre. Dans cette optique, on souhaite que les résultats théoriques obtenus puissent avoir des applications sur les objets réels dont les modèles mathématiques sont issus. L'équilibre est difficile à trouver entre trop abstraire et modéliser, au risque que les résultats théoriques ne soient plus applicables aux objets réels, ou au contraire ne pas assez abstraire et modéliser et avoir des objets mathématiques trop compliqués pour pouvoir obtenir le moindre résultat théorique. Il faut trouver un compromis entre puissance des résultats théoriques et applicabilité desdit résultats.

Au fil des siècles, tant de connaissances et de données scientifiques ont été amassées qu'il n'est plus possible d'être un spécialiste à la pointe de la recherche dans toutes les disciplines scientifiques: en effet une vie entière ne suffit pas à s'approprier tous les résultats obtenus au cours de l'histoire. Pour réussir à traiter la quantité de données amassées, et pour automatiser les calculs et ainsi gagner du temps, l'homme à eu recours à la machine, donnant naissance à l'informatique.

La combinatoire et l'algorithmique sont deux grands domaines de l'informatique théorique, situés à la frontière entre les mathématiques et l'informatique. L'idée est d'utiliser les théories mathématiques pour obtenir des résultats qui puissent être utilisés par des machines. Le principe de l'algorithmique est de donner des méthodes pour automatiser le calcul de la réponse à certains problèmes, afin qu'il puisse être effectué par une machine. Il est important que ces méthodes soient efficaces. En effet, devant la multitude de données amassées et la taille immense de ces données, par exemple en biologie concernant le séquençage de l'ADN, trouver un algorithme efficace permet dans certains cas de faire passer le temps de calcul de plusieurs centaines d'années à quelques minutes, ce qui rend envisageable en pratique des calculs qui auparavant restaient dans le domaine du théorique.

La combinatoire, elle, se concentre sur l'étude des "configurations", c'est-à-dire des structures discrètes respectant certaines contraintes bien formalisées. Ces structures sont des modèles d'objets réels que l'on cherche à décrire avec des règles formelles simples.

L'une des branches les plus anciennes de la combinatoire est le dénombrement : lorsque les structures étudiées sont munies d'une notion de taille telle que pour tout entier n le nombre d'objets de taille n est fini (on appelle *classe combinatoire* un ensemble de structures vérifiant une telle propriété), on cherche à calculer le nombre d'objets de taille n pour tout entier n. Idéalement, on espère trouver une formule close en fonction de n, mais l'existence d'une telle formule n'est pas garantie. On peut aussi s'intéresser à la série génératrice de la classe combinatoire, qui est une fonction codant le nombre d'objets de taille n pour tout n. Dans ce cas on cherche à obtenir une expression de la série génératrice, ou un système d'équations satisfait par la série génératrice. Enfin on peut chercher un équivalent asymptotique du nombre d'objets de taille n lorsque n tend vers l'infini.

L'étude du dénombrement s'est beaucoup développée avec l'essor des probabilités et des statistiques, au point que certains scientifiques ont tendance à assimiler combinatoire et dénombrement.

Cependant la combinatoire, loin de se limiter au dénombrement, est un domaine où les problèmes sont extrêmement riches et divers, et comporte de nombreuses autres branches.

On peut citer par exemple la génération exhaustive ou la génération aléatoire des objets étudiés, qui permettent entre autres de vérifier des conjectures ou de tester des algorithmes.

La combinatoire s'attache aussi à décrire les propriétés intrinsèques des structures étudiées, à caractériser les objets étudiés de la façon la plus simple possible, diverses caractérisations pour les mêmes structures pouvant se révéler adaptées à différents aspects de l'étude de ces structures, à étudier les transformations d'une structure en une autre (par exemple au moyen de bijections entre classes combinatoires), ou à étudier les sous-structures que l'on peut extraire d'une structure donnée.

Pour plus de détails à propos des objectifs de la combinatoire, se référer à l'introduction intitulée *Qu'est ce que la combinatoire ?* du livre [Ber68].

Combinatoire et algorithmique sont naturellement liées. En effet la combinatoire est très utile à l'algorithmique : connaître la structure des objets étudiés et leurs propriétés combinatoires permet d'obtenir des algorithmes plus efficaces. De plus le dénombrement, qui est une branche de la combinatoire, permet d'analyser la complexité des algorithmes, et la génération aléatoire permet de tester les algorithmes en pratique.

Inversement, l'algorithmique est utile à la combinatoire. En effet, lorsqu'un théorème de combinatoire permet d'affirmer l'existence d'une certaine structure, il est naturel de chercher un algorithme qui permet de tester si un objet donné vérifie les hypothèses du théorème assurant l'existence de la structure, et aussi de chercher un algorithme qui calcule ladite structure.

Cette thèse illustre ces deux aspects du lien entre algorithmique et combinatoire, en se concentrant sur le cas des classes de permutations : dans la première partie on utilise l'algorithmique pour décider si une classe de permutations contient un nombre fini de permutations simples, ce qui est l'hypothèse du théorème principal de [AA05], qui assure alors que la série génératrice de la classe permutations est algébrique. De plus on donne un algorithme permettant de calculer une spécification combinatoire de la classe de permutations lorsque cette hypothèse est vérifiée, ce qui permet entre autres de calculer la série génératrice. Dans la seconde partie, on utilise une étude combinatoire des permutations afin d'obtenir un algorithme polynomial résolvant un problème conjecturé NP-complet dans la littérature, à savoir le problème de décision de l'appartenance d'une permutation à la classe des permutations triables par deux piles connectées en série.

Les objets combinatoires au cœur de cette thèse sont les permutations, vues comme des mots composés de lettres distinctes et ordonnées. Ici on ne s'intéressera pas aux propriétés algébriques des permutations, en particulier on n'utilisera pas la notion de groupe, ni même la composition des permutations. On se concentrera plutôt sur les permutations vues comme une généralisation bi-dimensionnelle des mots : les lettres ne sont pas seulement ordonnées entre elles par leur ordre d'apparition dans le mot, mais aussi par leur valeur. Plus précisément, dans cette thèse une permutation de taille n sera considérée comme un mot de n lettres contenant une unique fois chaque lettre de 1 à n.

On s'intéresse en particulier à la notion de motif dans les permutations, qui est une généralisation naturelle de la notion de sous-mot : un sous-mot est obtenu à partir d'un mot en sélectionnant les éléments apparaissant à certains indices ; le sous-mot est obtenu en renormalisant les indices : l'élément apparaissant au premier indice sélectionné dans le mot devient l'élément d'indice 1 du sous-mot, l'élément apparaissant au second indice sélectionné dans le mot devient l'élément d'indice 2 du sous-mot, et ainsi de suite. Les permutations étant une généralisation bi-dimensionnelle des mots, avec un ordre à la fois sur les indices et sur les valeurs des lettres, il est logique que la notion de motif, généralisant celle de sous-mot, subisse une double normalisation, à la fois sur les indices et sur les valeurs. Ainsi si dans une permutation on sélectionne les éléments apparaissant à certains indices, disons k indices distincts, qu'on les garde dans le même ordre d'apparition en renormalisant les indices de 1 à k, et qu'on renormalise aussi les valeurs de 1 à k en gardant l'ordre relatif des valeurs des lettres, on obtient une seconde permutation qui est motif de la première. Par exemple 132 est un motif de la permutation 31487265 qu'on peut obtenir en sélectionnant les éléments 4, 7 et 5 apparaissant aux indices 3, 5 et 8 et en effectuant la double normalisation en indice et en valeur.

La relation de motif étant une relation d'ordre partiel, il est naturel de s'intéresser aux ensembles fermés par le bas pour cette relation, que l'on appelle classes de permutations. En d'autres termes, si une permutation σ appartient à une classe de permutations $\mathcal{C}$, alors tous les motifs de σ appartiennent aussi à $\mathcal{C}$; de plus cette propriété caractérise les classes de permutations.

L'ensemble de toutes les permutations est trivialement une classe de permutations. Le premier exemple non trivial de classe de permutations que l'on trouve dans la littérature est l'ensemble des permutations triables par une pile, étudié par Donald Knuth dans le volume 1 de *The Art of Computer Programming* [Knu68] en 1968 avant que la notion de motif n'ait été formalisée.

La combinatoire des classes de permutations est maintenant un domaine bien établi, en pleine expansion depuis les années 90. Jusque dans les années 2000, la plupart des articles se concentrent sur l'étude d'une classe de permutations donnée, dont la base est finie et explicite et contient souvent des motifs de taille seulement 3 ou 4. Depuis une dizaine d'années, une nouvelle ligne de recherche a émergé, essayant d'obtenir les résultats les plus généraux possibles sur les classes de permutations. La première partie de cette thèse s'inscrit dans cette dernière ligne de recherche. La deuxième partie de la thèse revient aux origines des classes de permutations, à savoir l'étude du tri par piles.

L'outil principal utilisé dans cette thèse est la décomposition par substitution des permutations. La décomposition par substitution est un principe général, qui s'applique à d'autres structures que les permutations, par exemple aux graphes. L'idée, comme pour la décomposition des nombres entiers en facteurs premiers, est de décomposer l'objet en structures plus simples à manipuler car plus petites que l'objet initial et ayant des propriétés plus fortes : les structures premières (appelées aussi simples ou indécomposables).

Contributions et plan détaillé de la thèse

Cette thèse s'inscrit dans la cadre de la combinatoire des classes de permutations. Les principaux objectifs de la thèse sont de nature algorithmique. Plus précisément la première partie de la thèse fournit une chaîne entièrement algorithmique pour calculer une spécification combinatoire pour une classe de permutations de base finie (sous certaines conditions testées algorithmiquement). La seconde partie donne un algorithme polynomial pour résoudre un problème ayant été conjecturé NP-complet dans la littérature, à savoir le problème d'appartenance d'une permutation à la classe des permutations triables par deux piles connectées en série.

On commence dans une partie préliminaire par rappeler les définitions et notations ainsi que quelques résultats généraux concernant les classes de permutations.

Partie I

La partie 1 de la thèse puise sa source dans l'article [AA05], qui établit le théorème suivant :

Theorem 0.1. *Si une classe de permutations contient un nombre fini de permutations simples, alors sa base est finie et sa série génératrice est algébrique.*

Comme remarqué dans [AA05], la preuve de ce théorème est constructive, et permet d'obtenir un sytème d'équations pour la série génératrice en utilisant le principe d'inclusion-exclusion. En adaptant cette preuve en contournant le principe d'inclusion-exclusion, on obtient une *spécification combinatoire* de la classe, à savoir un système d'équations combinatoires non-ambigu qui décrit récursivement les permutations de la classe à partir de la permutation de taille 1 en utilisant seulement des constructions combinatoires (union disjointe, produit cartésien, suite...). Cette spécification combinatoire donne une meilleure compréhension de la classe puisqu'elle en décrit les éléments, de plus elle permet par des méthodes automatiques d'obtenir un système d'équations positives satisfait par la série génératrice de la classe, et de faire de la génération aléatoire de permutations de la classe.

La première partie de la thèse décrit une chaîne algorithmique qui teste si une classe contient un nombre fini de permutations simples et le cas échéant en calcule une spécification combinatoire.

Chapitre 1

Le chapitre 1 étudie le *poset* (c'est-à-dire l'ensemble partiellement ordonné) des permutations simples pour l'ordre donné par la relation de motif entre permutations. Les permutations simples sont les briques de base dans la décomposition par substitution des permutations, et elles jouent un rôle crucial dans l'étude de la structure des classes de permutations (voir [Bri10] pour un résumé sur les permutations simples).

Dans le chapitre 1, on part des résultats généraux obtenus par Schmerl et Trotter sur les structures de relations binaires irréflexives [ST93] que l'on raffine dans le cas des permutations. Parmi les structures indécomposables (qui correspondent aux permutations simples), Schmerl et Trotter distinguent les structures indécomposables critiques, qui correspondent aux permutations exceptionnelles définies dans [AA05]. En utilisant les résultats de [ST93], on montre que si σ et π sont des permutations simples telles que π est motif de σ (ce qu'on note $\pi < \sigma$), alors il existe une chaîne de permutations $\sigma^{(0)}, \sigma^{(1)}, \ldots, \sigma^{(k)}$ telle que $\sigma^{(0)} = \sigma$, $\sigma^{(k)} = \pi$, $\sigma^{(i+1)} < \sigma^{(i)}$ et $|\sigma^{(i)}| - |\sigma^{(i+1)}| \in \{1, 2\}$ pour tout i, $|\sigma|$ désignant la taille de la permutation σ. En utilisant les propriétés des permutations simples, on renforce ce résultat pour obtenir le résultat principal de ce chapitre : si σ n'est pas exceptionnelle, alors on peut trouver une telle chaîne de permutations telle que les différences de taille soient toutes de 1, et si σ est exceptionnelle, alors les différences de taille soient toutes de 2.

On donne ensuite plusieurs conséquences de ce résultat, qui permettent de mieux comprendre la structure du poset des permutations simples. Tout d'abord on sait que lorsque l'on supprime un point d'une permutation simple, le motif obtenu (en renormalisant le résultat) est une permutation qui n'est pas forcément simple. Étant donnée une permutation simple fixée, on s'intéresse aux points qui, lorsqu'ils sont supprimés, donnent un motif simple. On montre que ces points sont quasiment tous les points de la permutation ;

plus précisément on montre que le nombre moyen D_n de tels points pour une permutation simple de taille n vérifie $D_n = n - 4 - \frac{4}{n} + O(\frac{1}{n^2})$. De plus, les motifs simples obtenus en supprimant ces points sont tous distincts. On étudie symétriquement le nombre de points que l'on peut ajouter à une permutation simple de taille n de sorte qu'elle reste simple : on montre qu'il vaut exactement $(n+1)(n-3)$ et que les sur-permutations obtenues sont toutes distinctes.

Par ailleurs, le résultat principal du chapitre nous permet également de concevoir un algorithme polynomial (par rapport à la taille de la sortie) pour générer les permutations simples d'une classe de permutations close par substitution ayant une base finie, en prenant en entrée la base. Cet algorithme commence par considérer les permutations simples de taille 4 et itère sur la taille des permutations. Les résultats obtenus sur le poset des permutations simples permettent de trouver en temps polynomial les permutations simples de taille $n+1$ de la classe en connaissant seulement les permutations simples de taille n et $n-1$ de la classe. Notons que notre algorithme ne requiert aucun test de motif. Bien que nous décrivions cet algorithme dans le cadre général des classes closes par substitution, nous l'appliquons aux classes ne contenant qu'un nombre fini de permutations simples. En effet l'algorithme termine si et seulement si la classe contient un nombre fini de permutations simples. Il peut également être utilisé pour générer les permutations simples de toute classe close par substitution jusqu'à une taille donnée, même si le nombre total de permutations simples de la classe est infini. La complexité globale de cet algorithme est polynomiale en le nombre de permutations simples de la classe. Enfin on adapte cet algorithme pour les classes qui ne sont pas closes par substitution, au prix d'une perte d'efficacité.

Le chapitre 1 permet donc de mieux comprendre la structure du poset des permutations simples, et fournit un algorithme qui est l'une des étapes de la chaîne algorithmique décrite dans la partie 1.

Chapitre 2

Le chapitre 2 est consacré à une autre étape de la chaîne algorithmique décrite dans la partie 1, à savoir décider si une classe donnée par sa base (finie) contient un nombre fini de permutations simples. Remarquons que le fait de considérer uniquement des bases finies n'est pas une restriction, car on sait d'après un résultat de [AA05] que si la base est infinie, alors la classe contient un nombre infini de permutations simples.

Notre algorithme suit la structure de la méthodologie donnée par [BRV08], où il est prouvé qu'on peut décider en 3 étapes si une classe contient un nombre fini de permutations simples, en testant si la classe contient un nombre fini de permutations de 3 types : les permutations parallèles, les permutations simples en chevron, et les permutations en épingles propres. Cependant l'article [BRV08] s'attache uniquement à montrer que le problème est décidable, et non à fournir un algorithme effectif pour le résoudre.

En étudiant en détails la procédure de décision donnée par [BRV08], on peut montrer qu'elle est algorithmisable, mais de complexité fortement exponentielle. Plus précisément, en utilisant un résultat de [AAAH01] et la caractérisation des permutations parallèles et simples en chevron donnée dans [BRV08], on montre que les 2 premières étapes peuvent s'effectuer en temps $\mathcal{O}(\ell.m \log m)$, où ℓ est le nombre de permutations de la base, et m la taille maximale d'une permutation dans la base. Par contre en suivant la procédure de [BRV08], la dernière étape s'effectue en temps au moins $\mathcal{O}(2^{m.\ell.2^m})$. De plus, cette dernière étape nécessite d'identifier les éléments de la base qui sont des permutations en épingle et de calculer l'ensemble des mots d'épingles qui leur sont associés, deux problèmes dont l'algorithmisation n'est pas traitée dans [BRV08].

En se basant sur l'étude des permutations simples réalisées dans [BBR11], on donne un algorithme effectif pour cette dernière étape, de complexité $\mathcal{O}(n \log n + s^{2k})$ où n est la somme des tailles des permutations de la base B, s est la taille maximale d'une permutation en épingle de B et k est le nombre de permutations en épingle de B. On utilise le même codage des permutations en épingles par des mots sur un alphabet fini que dans [BRV08], et on utilise la théorie des automates. Notre construction d'automates est algorithmique et efficace.

De plus dans le cas particulier des classes closes par substitution, on donne une variante de notre algorithme qui est linéaire en n sur cette dernière étape, et de complexité $\mathcal{O}(n \log n)$ au total.

Chapitre 3

Dans le chapitre 3, on donne un algorithme pour calculer une spécification combinatoire pour une classe de permutations à partir de sa base et de son ensemble de permutations simples, en supposant que ce dernier est fini (ce qui d'après le théorème 0.1 implique que la base elle aussi est finie). Remarquons qu'avec les résultats des chapitres 1 et 2, il est suffisant de connaître la base pour pouvoir décider si le nombre de permutations simples de la classe est fini et pour pouvoir les calculer le cas échéant. Le chapitre 3 clôt donc la chaîne algorithmique de la partie 1.

Après quelques rappels sur les structures combinatoires et la génération aléatoire, le chapitre 3 part de la preuve du théorème 0.1 de [BBR11] : en explicitant et détaillant cette preuve, on la transforme en un algorithme effectif qui utilise la décomposition par substitution des permutations pour obtenir une grammaire algébrique décrivant les arbres de décomposition des permutations de la classe. Cependant la grammaire obtenue peut être ambiguë. Dans [BBR11], l'ambiguïté est levée en utilisant l'inclusion-exclusion, ce qui peut faire apparaître des termes négatifs que nous ne voulons éviter car ils n'ont pas de sens combinatoire. On utilise donc une autre méthode qui nous permet d'obtenir une spécification combinatoire : on transforme les unions non disjointes en unions disjointes de termes étant définis à la fois par des motifs exclus et par des motifs obligatoires (et non plus uniquement par des motifs exclus comme c'est le cas des classes de permutations). De plus on donne un exemple complet du déroulement de cette méthode.

Obtenir une spécification combinatoire permet d'avoir une description combinatoire récursive de nos objets, d'obtenir un système d'équations pour la série génératrice, et surtout de faire de la génération aléatoire de permutations de la classe. Notre chaîne algorithmique permet donc d'obtenir un outil pouvant servir à tester des conjectures, et à pousser plus loin l'étude des classes de permutations. Cependant cette chaîne algorithmique ne calcule son résultat que lorsque le nombre de permutations simples de la classe est fini.

Notons que l'article [BHV08b] étend le résultat de [AA05] en prouvant que la série génératrice de certains sous-ensembles d'une classe contenant un nombre fini de permutations simples est algébrique. Cette preuve est elle aussi constructive et n'utilise pas d'inclusion-exclusion. Il est donc possible que les résultats du chapitre 3 puissent être généralisés en transformant la preuve de [BHV08b] en un véritable algorithme. Enfin, la complexité de l'algorithme donné dans le chapitre 3 reste à analyser.

Partie II

La partie II est consacrée au tri des permutations avec 2 piles connectées en série. Plus précisément, on donne un algorithme polynomial pour décider si une permutation est triable par 2 piles en série. Cet algorithme s'appuie sur une nouvelle notion introduite dans cette thèse, qui est la notion de *tri par sas* (*pushall sorting* en anglais).

Chapitre 4

Le chapitre 4 introduit une nouvelle restriction du tri avec deux piles en série, appelé tri par sas. Il s'agit d'un tri en deux étapes, une première étape où l'on doit d'abord mettre tous les éléments dans les piles sans rien écrire en sortie, et une seconde étape où l'on sort les éléments des piles (on n'a plus rien le droit de prendre dans l'entrée). On montre que pour un tel type de tri, la connaissance de l'état des piles (appelé configuration de piles) à la fin de la première étape caractérise l'ensemble du processus de tri.

En utilisant les configurations de piles et la décomposition par substitution des permutations, on donne des conditions nécessaires et suffisantes récursives pour qu'une permutation à racine linéaire (notion définie par la suite dans les préliminaires) soit triable par deux piles en série (abrégé en 2-triable) ou triable par sas (abrégé en sas-triable). On remarque que l'ensemble des permutations sas-triables est une sous-classe de la classe des permutations 2-triables, et que ces deux classes sont très liées. On caractérise les permutations à racine linéaire de la base de chacune des deux classes, et on montre que la base des permutations sas-triables est infinie (on sait aussi d'après [Mur02] que la base des permutations 2-triables est infinie).

Enfin on donne un algorithme polynomial pour décider si une permutation est sas-triable. On commence par prouver qu'une permutation est sas-triable si et seulement si son diagramme admet un certain type de bicoloriage, appelé bicoloriage valide et défini par des motifs colorés exclus. On montre qu'on peut décider en temps linéaire par rapport à la taille n de la permutation si un bicoloriage est valide, et on montre qu'une permutation indécomposable a au plus $9n$ coloriages valides (pour une notion d'indécomposabilité définie par la suite et notée $\ominus$-indécomposable). Grâce à la décomposition des permutations, ceci nous donne un algorithme en $\mathcal{O}(n^2)$ qu'on prouve optimal pour calculer un codage de tous les processus de tri par sas d'une permutation donnée. En particulier cet algorithme permet de décider si une permutation est triable par sas.

Chapitre 5

Le chapitre 5 décrit un algorithme polynomial permettant de décider si une permutation est triable par 2 piles en série. Ceci clôt une question restée longtemps ouverte ; par ailleurs ce problème a été conjecturé NP-complet dans la littérature.

Décider si une permutation est triable par une unique pile est simple car toute permutation triable admet exactement un tri. Par contre avec deux piles en série, une permutation peut être triable de nombreuses façons différentes. Par exemple la permutation décroissante de taille n admet 2^{n-1} procédures de tri. Il est difficile de décider si une permutation est 2-triable car tester tous les tris possibles est exponentiel, et il y a pas de tri canonique connu : les tris glouton connus ne trient qu'une partie des permutations 2-triable.

L'une des clés de l'algorithme polynomial du chapitre 5 est de limiter le nombre de tris à tester en définissant une propriété P sur les tris, et en prouvant que toute permutation 2-triable admet un tri respectant une propriété appelée P. Cette propriété P correspond en quelque sorte à faire sortir les petits éléments des piles le plus vite possible. Par exemple la propriété P implique que si σ_{k_i} est un minimum droite-gauche (appelé aussi élément saillant inférieur droit), alors tous les éléments plus petits que σ_{k_i} sont déjà sortis des piles lorsque σ_{k_i} y entre.

L'algorithme procède en s étapes, où s est le nombre de minima droite-gauche de σ. Soit σ_{k_i} le i-ème minimum droite-gauche de σ, à l'étape i l'algorithme calcule tous les tris vérifiant la propriété P du préfixe de σ finissant par σ_{k_i}. Pour décider si σ est triable, il suffit alors de tester si l'ensemble calculé à la dernière étape est non vide.

Un second point clé de l'algorithme est de coder l'ensemble de tris calculés, qui peuvent être en nombre exponentiel, sous la forme d'un graphe de taille polynomiale qu'on appelle un graphe de tri. Le graphe de l'étape $i+1$ est calculé à partir du graphe de l'étape i et de l'ensemble des tris par sas d'une sous-permutation définie par $\sigma_{k_{i+1}}$ qui sont calculés en faisant appel à l'algorithme du chapitre 4.

La notion de tri par sas définie dans cette thèse est essentielle à notre algorithme polynomial.

Cette notion de tri par sas peut se généraliser pour t piles en série avec $t > 2$, mais il semble difficile de l'utiliser pour obtenir un algorithme polynomial décidant si une permutation est triable par t piles en série.

Preliminaries: Definitions and some background

0.1 Permutation patterns and permutation classes

In this section we define the concepts used throughout the thesis. In particular, we start by defining the one main object, namely the permutations.

Definition 0.2. For any integers i and j, let us denote by $[i..j]$ the interval of integers between i and j: $[i..j] = \{k \in \mathbb{N} \mid i \leq k \leq j\}$. A *permutation* of size n is a bijective function from $[1..n]$ onto itself. We denote by $|\sigma|$ the size of a permutation σ, $\mathcal{S}_n$ the set of permutations of size n and $\mathcal{S} = \cup_{n\geq 1}\mathcal{S}_n$ the set of all permutations.

For all $i \in [1..n]$, we denote by $\sigma_i = \sigma(i)$ the image of i by σ. We say that the element σ_i of σ has index i and value σ_i. We write a permutation $\sigma \in \mathcal{S}$ as the word $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ where $n = |\sigma|$. A permutation of size n is then a word containing each integer from 1 to n exactly once.

Example 0.3. The permutation $\sigma = 3\ 5\ 1\ 4\ 2$ is the bijective function such that $\sigma(1) = 3$, $\sigma(2) = 5$, $\sigma(3) = 1$, $\sigma(4) = 4$ and $\sigma(5) = 2$.

We usually write permutations as words, but it will also be useful to have a pictorial description of permutations, which we now introduce.

Definition 0.4. The *diagram* of a permutation $\sigma \in S_n$ is the set of points in the plane at coordinates $(1,\sigma_1),(2,\sigma_2),\ldots,(n,\sigma_n)$. Figure 1 shows an example of such a diagram.

The leftmost (resp. rightmost, lowest, topmost) point of σ is the point $(1,\sigma_1)$ (resp. (n,σ_n), $(\sigma^{-1}(1),1)$, $(\sigma^{-1}(n),n)$) in the diagram of σ. We say that (i,σ_i) is in a corner of the diagram of σ if $(i,\sigma_i) \in \{(1,1),(1,n),(n,1),(n,n)\}$.

To shorten notations, we sometimes write σ_i to refer to the point (i,σ_i).

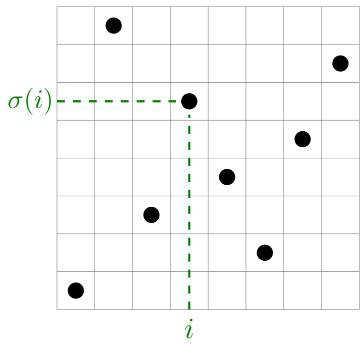


Figure 1: Diagram of $\sigma = 1\ 8\ 3\ 6\ 4\ 2\ 5\ 7$

Definition 0.5. The *bounding box* of a set of points E in the plane is the smallest axis-parallel rectangle containing the set E (see Figure 2). This box defines nine regions in the plane that form a partition of the plane:

- The bounding box itself.
- The sides of the bounding box (L, R, U, D on Figure 2 for left, right, up, down).
- The corners of the bounding box ($1, 2, 3, 4$ on Figure 2).

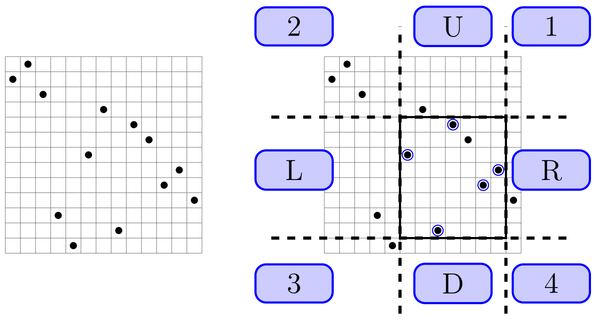


Figure 2: Diagram of $\sigma = 12\,13\,11\,3\,1\,7\,10\,2\,9\,8\,5\,6\,4$ and the bounding box of $\{7, 2, 9, 5, 6\}$.

Since we consider permutations as words, it is natural to consider subwords of permutations. But a subword w of a permutation $\sigma \in \mathcal{S}$ is not necessarily an element of $\mathcal{S}$ as there is no reason for w to contain each integer from 1 to $|w|$ exactly once. We have to renormalize w to obtain a permutation.

Definition 0.6. Two sequences $w = w_1 \dots w_k$ and $v = v_1 \dots v_k$ of distinct integers are *order isomorphic* if they have the same relative comparisons, that is, for all $i, j \in [1..n]$, $w_i < w_j$ if and only if $v_i < v_j$.

The *normalization* of w is the unique $\sigma \in \mathcal{S}_k$ such that w is order isomorphic to σ.

Example 0.7. The normalization of $2\,5\,4\,9$ is $1\,3\,2\,4$.

Using normalization, we can define a relation on permutations similar to the notion of subwords.

Definition 0.8. A permutation $\pi = \pi_1 \dots \pi_k$ is a *pattern* of the permutation $\sigma = \sigma_1 \dots \sigma_n$ if σ has a subsequence that is order isomorphic to π, that is, $k \leq n$ and there exist indices $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ such that $\sigma_{i_\ell} < \sigma_{i_m}$ whenever $\pi_\ell < \pi_m$.

Then $\sigma_{i_1} \dots \sigma_{i_k}$ is called an occurrence of π in σ and we say that σ contains π. Otherwise σ is said to *avoid* π.

Example 0.9. The permutation $\sigma = 1\ 4\ 2\ 5\ 6\ 3$ contains the pattern $1\ 3\ 4\ 2$ whose occurrences in σ are $1\ 5\ 6\ 3$, $1\ 4\ 6\ 3$, $2\ 5\ 6\ 3$ and $1\ 4\ 5\ 3$. But σ avoids the pattern $3\ 2\ 1$ as none of its subsequences of length 3 is order isomorphic to $3\ 2\ 1$, i.e. is decreasing.

Remark 0.10. The pattern containment relation is a partial order on $\mathcal{S}$. We then write $\pi \leq \sigma$ to denote that π is a pattern of σ, and we write $\pi < \sigma$ when $\pi \leq \sigma$ and $\pi \neq \sigma$.

The notion of pattern in permutation can be visualised using diagrams. To obtain a pattern of a permutation σ, start with the diagram of σ, select some points and delete the others, finally delete blank rows and columns from the diagram. The diagram obtained is then the diagram of a pattern of σ (see Figure 3).

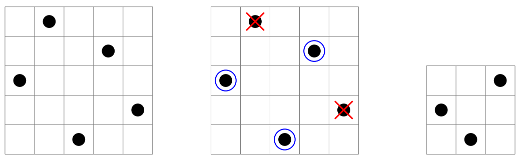


Figure 3: The permutation $3\,5\,1\,4\,2$ contains $2\,1\,3$ as a pattern.

Although the notion of pattern comes from the concept of subword, normalization makes it much more complicated. For instance, deciding whether a word is a subword of another can be done in linear time w.r.t. the sizes of the words, while the same decision problem is NP-complete for permutation patterns (see [BBL98]).

The concept of pattern is used in particular in the definition of permutation classes. Permutation classes arise naturally in a variety of settings, ranging from sorting (see, e.g., Bóna's survey [B0́3]) to algebraic geometry (see, e.g., Lakshmibai and Sandhya [LS90]). This thesis focuses on the study of permutation classes, which we now define.

Definition 0.11. A *permutation class* is a set of permutations closed downwards under pattern relation. In other words, a set $\mathcal{C}$ is a permutation class if and only if for any $\sigma \in \mathcal{C}$, if $\pi \leq \sigma$, then $\pi \in \mathcal{C}$.

Example 0.12. The set $\cup_{n=1}^{\infty}\{1\,2\ldots n\}$ is a permutation class whereas $\{4\,2\,3\,5\,1, 3\,2\,1\}$ is not a permutation class.

A permutation class may be given by a property stable for $<$, or may be defined with excluded patterns.

Definition 0.13. Let E be any set of permutations, we denote by $Av(E)$ the set of permutations avoiding every element of E: $Av(E) = \{\sigma \in \mathcal{S} \mid \forall \pi \in E, \pi \not\leq \sigma\}$

Proposition 0.14. *For any set E of permutations, $Av(E)$ is a permutation class.*

Two distinct sets of permutations E and F may give the same set (i.e. $Av(E) = Av(F)$) when there are relations between elements of E or of F. To avoid this, we use the concept of antichains.

Definition 0.15. An *antichain* is a set of pairwise incomparable elements. More precisely, a set E of permutations is an antichain if for any $\sigma \neq \pi$ in E, $\pi \not\leq \sigma$.

Theorem 0.16. *For any permutation class $\mathcal{C}$, there is a unique (possibly infinite) antichain B such that $\mathcal{C} = Av(B)$. This antichain B, which consists of the minimal permutations not in $\mathcal{C}$, is called the* basis *of $\mathcal{C}$. More formally, $B = \{\sigma \notin \mathcal{C} \mid \forall \pi < \sigma, \pi \in \mathcal{C}\}$.*

The name *basis* may be confusing: contrary to basis of vector spaces for instance, elements of the basis of a permutation class do not belong to the class.

Permutation classes are characterized by their bases. But some permutation classes have infinite bases. This is the case for example for the class of 2-stack sortable permutations studied in Part 2.

We denote by $\mathcal{C}_n$ the set $\mathcal{C} \cap \mathcal{S}_n$, that is, the permutations in $\mathcal{C}$ of size n, and we refer to $\sum |\mathcal{C}_n| x^n$ as the generating function for $\mathcal{C}$.

Permutation classes have been widely studied in the literature, mainly from a pattern-avoidance point of view. See [BM03, Eli04, KM03, Vat08] among many others. The main result about the enumeration of permutation classes is the recent proof of the Stanley-Wilf conjecture by Marcus and Tardos [MT04], who established that for any class $\mathcal{C}$, there is a constant c such that the number of permutations of size n in $\mathcal{C}$ is at most c^n.

0.2 Substitution decomposition of permutations

Substitution decomposition is a general framework, adapted to various families of discrete objects [MR84], that is based on core items and relations and in which every object can be recursively decomposed into core objects using relations. In the case of permutations, core elements are simple permutations and the relations are substitutions.

The substitution decomposition of permutations is somehow similar to prime factorization of integers. There is indeed existence and uniqueness of the decomposition. The ground permutations playing the role of prime numbers are the simple permutations, which are defined thanks to the notion of interval (or block).

Definition 0.17. An *interval* of a permutation σ is a set of contiguous indices $I = [k..k+\ell-1]$ such that the set of values $\sigma(I) = \{\sigma(i) \mid i \in I\}$ also forms an interval of natural numbers.

A *block* of σ is a factor $\sigma_k \sigma_{k+1} \dots \sigma_{k+\ell-1}$ of σ such that the set of values $\{\sigma_k, \sigma_{k+1}, \dots, \sigma_{k+\ell-1}\}$ is an interval.

The integer ℓ is called the *size* of the interval or of the block.

Intervals and blocks convey the same notion: a set of elements $\{(k_1, \sigma_{k_1}), (k_2, \sigma_{k_2}), \dots, (k_\ell, \sigma_{k_\ell})\}$ whose set of indices $\{k_1, \dots, k_\ell\}$ and set of values $\{\sigma_{k_1}, \sigma_{k_2}, \dots, \sigma_{k_\ell}\}$ both form intervals of natural numbers. However this notion can be defined from the point of view of indices or from the point of view of values.

Example 0.18. 5 7 4 6 is a block of 2 5 7 4 6 1 3 whose corresponding interval is [2..5].

Remark 0.19. A factor $\sigma_k \sigma_{k+1} \dots \sigma_{k+\ell-1}$ is a block of σ if and only if on the diagram of σ, the bounding box of $\{\sigma_k, \sigma_{k+1}, \dots, \sigma_{k+\ell-1}\}$ has no point on its sides.

More generally in the diagram of σ, any bounding box of points without any point on its sides corresponds to a block of σ. Then this bounding box is a square which is itself a diagram of a permutation (if translated to the origin).

This is illustrated by Figure 4.

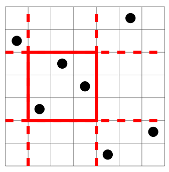


Figure 4: The block 3 5 4 in the permutation 6 3 5 4 1 7 2.

Every permutation $\sigma \in \mathcal{S}_n$ has blocks (resp. intervals) of sizes 1 and n: the singletons $\{i\}$ with $1 \leq i \leq n$ and σ itself (resp. $[1..n]$).

Definition 0.20. An interval or block of a permutation $\sigma \in \mathcal{S}_n$ is *trivial* if it is of size 1 or n, i.e. if it is either a singleton or the whole permutation.

Definition 0.21. A permutation is *simple* if it contains no block, except the trivial ones and if it is not $1, 12$ or 21.

Notice that the permutations 1, 12 and 21 also have only trivial intervals, nevertheless they are *not* considered to be simple in this thesis. Just as 1 is not considered as a prime number in the factorization of integers, it is natural not to consider the permutation 1 as simple. The choice of considering 12 and 21 as not simple is not widely shared in the literature. However 12 and 21 have a different behaviour than other simple permutations, as we will see for example in Theorem 0.27. Moreover this choice will simplify many statements.

As no permutation of size 3 has only trivial intervals, the smallest simple permutations are of size 4. There are two of them: 2413 and 3142.

We can check on the diagram of a permutation whether it is simple: A permutation different from 1, 1 2 and 2 1 is simple if and only if on its diagram, every non-trivial bounding box has at least a point on its side (see Remark 0.19).

Example 0.22. The permutation 6 3 5 4 1 7 2 is not simple as shown in Figure 4, whereas 3 1 7 4 6 2 5 is a simple permutation (see Figure 5).

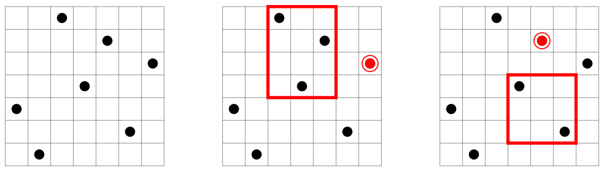


Figure 5: The permutation 3 1 7 4 6 2 5 is simple: each non-trivial bounding box has a point on its side.

For a detailed study of simple permutations, in particular from an enumerative point of view, we refer the reader to [AA05, AAK03, Bri10]. Let us only mention that the number of simple permutations of size n is asymptotically equivalent to $\frac{n!}{e^2}$ as n grows:

Theorem 0.23 (Theorem 5 of [AAK03])**.** *Let s_n be the number of simple permutations of size n. Then:*

$$s_n = \frac{n!}{e^2}\Big(1 - \frac{4}{n} + \frac{2}{n(n-1)} + O(n^{-3})\Big)$$

Blocks and simple permutations are the two key concepts involved in substitution decomposition. We now define substitution.

If σ is a permutation of S_n and $\pi \in S_p$ then substituting π in σ at position i leads to the permutation $\alpha = \bar{\sigma}_1\bar{\sigma}_2 \dots \bar{\sigma}_{i-1}(\pi_1 + \sigma_i - 1)\dots(\pi_p + \sigma_i - 1)\bar{\sigma}_{i+1}\dots\bar{\sigma}_{n+p-1}$ where $\bar{\sigma}_j = \begin{cases} \sigma_j \text{ if } \sigma_j \leq \sigma_i, \\ p + \sigma_j - 1 \text{ otherwise.} \end{cases}$ For convenience, as multiple substitutions can occur in a permutation we will denote by $\sigma[1, 1, \dots, 1, \underbrace{\pi}_{i}, 1, \dots, 1]$ this substitution.

Consider for example the substitution of $\pi = 3\ 1\ 2\ 4$ in $\sigma = 2\ 5\ 4\ 6\ 7\ 1\ 3$ at position 3 (i.e. replacing $\sigma_3 = 4$). We obtain the permutation $\alpha = 2\ 8\ \mathbf{6}\ \mathbf{4}\ \mathbf{5}\ \mathbf{7}\ 9\ 10\ 1\ 3$ and write $\alpha = 2\ 5\ 4\ 6\ 7\ 1\ 3[1, 1, 3\ 1\ 2\ 4, 1, 1, 1, 1]$.

This notation naturally generalizes to $\sigma[\pi^1, \pi^2, \ldots, \pi^n]$, and it has been defined in [AA05] under the name of *inflation*. Let σ be a permutation of size n and $\pi^1, \ldots, \pi^n$ be n permutations of size $p_1, \ldots, p_n$ respectively. Define the substitution $\sigma[\pi^1, \pi^2, \ldots, \pi^n]$ of $\pi^1, \pi^2, \ldots, \pi^n$ in σ to be the permutation obtained by concatenation of n sequences of integers $S^1, \ldots, S^n$ from left to right, such that for every i, j, the integers of S^i form an interval, are ordered in a sequence order isomorphic to π^i, and S^i consists of integers smaller than S^j if and only if $\sigma_i < \sigma_j$.

This operation of substitution is easier to describe on the diagram of permutations as presented in Figures 6 and 7: the diagram of $\sigma[\pi^1, \pi^2, \ldots, \pi^n]$ is obtained from the one of σ by replacing each point σ_i by a block containing the diagram of π^i.

Example 0.24. The substitution $1\,3\,2[2\,1, 1\,3\,2, 1]$ gives the permutation $2\,1\ 4\,6\,5\ 3$ (see Figure 6).

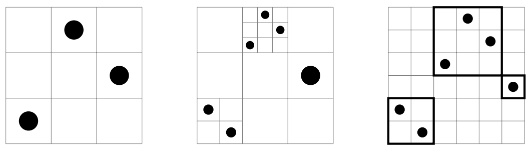


Figure 6: The permutation $1\,3\,2[2\,1, 1\,3\,2, 1] = 2\,1\ 4\,6\,5\ 3$

More formally:

Definition 0.25. Let $\sigma \in \mathcal{S}_n$ and $\pi^1, \ldots, \pi^n$ be n permutations of $\mathcal{S}_{p_1}, \ldots, \mathcal{S}_{p_n}$ respectively. The *substitution* $\tau = \sigma[\pi^1, \pi^2, \ldots, \pi^n]$ of $\pi^1, \pi^2, \ldots, \pi^n$ in σ is

$$\sigma[\pi^1, \pi^2, \ldots, \pi^n] = \mathit{shift}(\pi^1, \sigma_1) \ldots \mathit{shift}(\pi^k, \sigma_k)$$

where $\mathit{shift}(\pi^i, \sigma_i) = \mathit{shift}(\pi^i, \sigma_i)(1) \ldots \mathit{shift}(\pi^i, \sigma_i)(p_i)$ and

$\mathit{shift}(\pi^i, \sigma_i)(x) = \pi^i(x) + p_{\sigma^{-1}(1)} + \ldots + p_{\sigma^{-1}(\sigma_i - 1)}$ for any x between 1 and p_i.

We also say that $\sigma[\pi^1, \pi^2, \ldots, \pi^n]$ provides a *block decomposition* of τ.

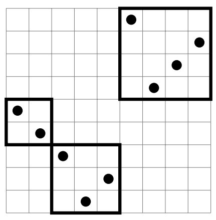


Figure 7: The permutation $213[21, 312, 4123] = 5\,4\ 3\,1\,2\ 9\,6\,7\,8$.

Two different substitutions may give the same permutation. For example the permutation $5\,4\,3\,1\,2\,9\,6\,7\,8$ of Figure 7 can be obtained as $213[21, 312, 4123]$ or $12[54312, 4123]$. To ensure uniqueness, we have to impose further restrictions.

For any $n \geq 2$, let $\mathcal{I}_n$ be the permutation $12\ldots n$ and $\mathcal{D}_n$ be $n(n-1)\ldots 1$. We use the notations $\oplus$ and $\ominus$ for denoting respectively $\mathcal{I}_n$ and $\mathcal{D}_n$, for any $n \geq 2$. This allows to write substitutions of the form $\oplus[\pi^1, \pi^2, \ldots, \pi^n] = \mathcal{I}_n[\pi^1, \pi^2, \ldots, \pi^n]$ or $\ominus[\pi^1, \pi^2, \ldots, \pi^n] = \mathcal{D}_n[\pi^1, \pi^2, \ldots, \pi^n]$ without ambiguity, the integer n being determined by the number of blocks π^i.

Definition 0.26. A permutation σ is *$\oplus$-indecomposable* (resp. *$\ominus$-indecomposable*) if it cannot be written as $\oplus[\pi^1, \pi^2, \dots, \pi^n]$ (resp. $\ominus[\pi^1, \pi^2, \dots, \pi^n]$), for any $n \geq 2$. Otherwise σ is *$\oplus$-decomposable* (resp. *$\ominus$-decomposable*).

Simple permutations, together with the families $(\mathcal{I}_n)$ and $(\mathcal{D}_n)$, are enough to describe all permutations through their *substitution decomposition*:

Theorem 0.27. (first appeared implicitly in [HS01]) *Every permutation $\sigma \in \mathcal{S}_n$ with $n > 1$ can be uniquely decomposed as either:*

- $\oplus[\pi^1, \pi^2, \dots, \pi^k]$*, with* $\pi^1, \pi^2, \dots, \pi^k$ *$\oplus$-indecomposable,*
- $\ominus[\pi^1, \pi^2, \dots, \pi^k]$*, with* $\pi^1, \pi^2, \dots, \pi^k$ *$\ominus$-indecomposable,*
- $\alpha[\pi^1, \dots, \pi^k]$ *with α a simple permutation.*

For example, $\sigma = 1\,3\,2\,4$ can be written either as $12[1, 2\,1\,3] = \oplus[1, 2\,1\,3]$ or $1\,2\,3[1, 2\,1, 1] = \oplus[1, 2\,1, 1]$ but in the first form, $\pi^2 = 2\,1\,3$ is not $\oplus$-indecomposable, thus we use the second decomposition.

It is important for stating Theorem 0.27 that 12 and 21 are not considered as simple permutations. An equivalent version of this theorem, which includes 12 and 21 among simple permutations, is given in [AA05] and reproduced below:

Theorem 0.28 (Proposition 2 of [AA05]). *Let $\sigma \in \mathcal{S}$. There is a unique simple permutation α and sequence $\pi^1, \pi^2, \dots, \pi^k \in \mathcal{S}$ such that $\sigma = \alpha[\pi^1, \pi^2, \dots, \pi^k]$. If $\alpha \neq 12, 21$, then $\pi^1, \pi^2, \dots, \pi^k$ are also uniquely determined by σ. If $\alpha = 12$ or 21, then π^1, π^2 are unique so long as we require that π^1 is $\oplus$-indecomposable or $\ominus$-indecomposable respectively.*

Since we do not consider 12 and 21 to be simple, this can be reformulated by:

Theorem 0.29. *Every permutation σ of size $n > 1$ can be uniquely decomposed as either:*

- $12[\pi^1, \pi^2]$*, with* π^1 *$\oplus$-indecomposable,*
- $21[\pi^1, \pi^2]$*, with* π^1 *$\ominus$-indecomposable,*
- $\alpha[\pi^1, \pi^2, \dots, \pi^k]$ *with α a simple permutation.*

Even though Theorems 0.27 and 0.29 are equivalent, they give two different points of view on substitution decomposition, that are complementary. Theorem 0.27 provides a decomposition that is intrinsically unique but with an infinite family of skeletons $\oplus$ and $\ominus$. On the contrary, Theorem 0.29 uses skeletons 12 and 21 only, but somehow breaks the symmetry by favouring one representation of $\oplus[\pi^1, \pi^2, \dots, \pi^n]$ as $12[\tau^1, \tau^2]$, among the many it may have (it is only the choice of the representative where τ^1 is $\oplus$-indecomposable, i.e. $\tau^1 = \pi^1$, that allows us to achieve uniqueness). Theorem 0.27 is therefore preferred for a use in combinatorics, where uniqueness is essential. This is the point of view that is further developed in this section. On the other hand, Theorem 0.29 is more adapted for a use of substitution decomposition in algorithms, as it allows to deal only with 12 and 21 instead of skeletons $\oplus$ and $\ominus$ of arbitrary arity $n \geq 2$. This formalism for substitution decomposition will be used in Chapter 3.

Remark 0.30. The simple permutation α in the third item of Theorems 0.27 and 0.29 is a pattern of the permutation σ. Hence, as soon as σ belongs to some permutation class $\mathcal{C}$, then so does α.

Notice that the π^i's correspond to blocks in the permutation σ. Another important remark is that:

Remark 0.31. Any block of $\sigma = \alpha[\pi^1, \dots, \pi^k]$ (with α a simple permutation) is either σ itself, or is included in one of the π^i's.

Theorem 0.27 provides the first step in the decomposition of a permutation σ. To obtain its full decomposition, we can recursively decompose the permutations π^i in the same fashion, until we reach permutations of size 1, leading to a complete decomposition where each permutation which appears is either $\mathcal{I}_k, \mathcal{D}_k$ (denoted by $\oplus, \ominus$ respectively) or a simple permutation. This recursive decomposition can naturally be represented by a tree, that is called the substitution decomposition tree (or *decomposition tree* for short) of σ, where a substitution $\alpha[\pi^1, \dots, \pi^k]$ is represented by a node V labeled α with k ordered children, and the subtrees rooted at the children of V represents the π^i's. In the sequel we will say "a child of the node V" instead of "a permutation corresponding to the subtree rooted at a child of the node V".

Example 0.32. Let $\sigma = 10\,13\,12\,11\,14\,1\,18\,19\,20\,21\,17\,16\,15\,4\,8\,3\,2\,9\,5\,6\,7$. Its recursive decomposition can be written as

$3\,1\,4\,2[\oplus[1, \ominus[1, 1, 1], 1], 1, \ominus[\oplus[1, 1, 1, 1], 1, 1, 1], 2\,4\,1\,5\,3[1, 1, \ominus[1, 1], 1, \oplus[1, 1, 1]]]$.

and its diagram and decomposition tree are given in Figure 8.

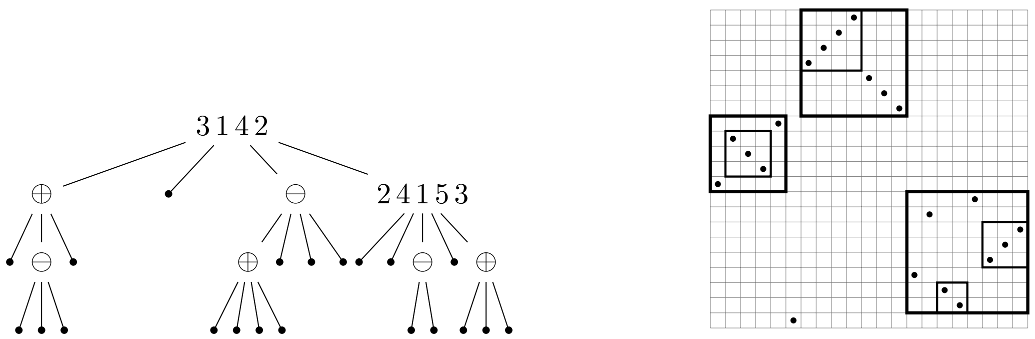


Figure 8: The substitution decomposition tree and the diagram (where non-trivial blocks corresponding to internal nodes are marked by rectangles) of the permutation $\sigma = 10\,13\,12\,11\,14\,1\,18\,19\,20\,21\,17\,16\,15\,4\,8\,3\,2\,9\,5\,6\,7$.

Definition 0.33. The *substitution decomposition tree* T of a permutation σ is the unique labeled ordered tree encoding the substitution decomposition of σ, where each internal node is either labeled by $\oplus, \ominus$ or by a simple permutation σ. The nodes labeled by $\oplus$ or $\ominus$ are called *linear* nodes. The nodes labeled by a simple permutation σ are called *prime* nodes.

Notice that in decomposition trees, each node labeled by α has arity $|\alpha|$, each subtree maps onto a block of σ, and there are no edges between two nodes labeled by $\oplus$, nor between two nodes labeled by $\ominus$, since the π^i are $\oplus$-indecomposable (resp. $\ominus$-indecomposable) in the first (resp. second) item of Theorem 0.27.

From Theorem 0.27 we have:

Theorem 0.34. *Permutations are in one-to-one correspondence with substitution decomposition trees.*

Theorem 0.35. *The substitution decomposition tree of a permutations of size n can be computed in time $\mathcal{O}(n)$. Conversely given a substitution decomposition tree, we can compute the permutation it encodes in linear time w.r.t. its number of leaves.*

Of course, especially for algorithmic use, equivalent decomposition trees can be defined using Theorem 0.29 instead of Theorem 0.27.

Example 0.36. The permutation $\sigma = 10\,12\,14\,11\,13\,1\,21\,19\,16\,18\,20\,17\,15\,4\,8\,3\,2\,9\,5\,6\,7$ can be recursively decomposed as

$\sigma = 3142\,[13524, 1, 7524631, 37218456]$

$= 3142\,[12[1, 2413], 1, 21[1, 21[41352[1, 1, 1, 1, 1], 1], 24153[1, 1, 21[1, 1], 1, 12[1, 12[1, 1]]]$

and its decomposition tree is given in Figure 9.

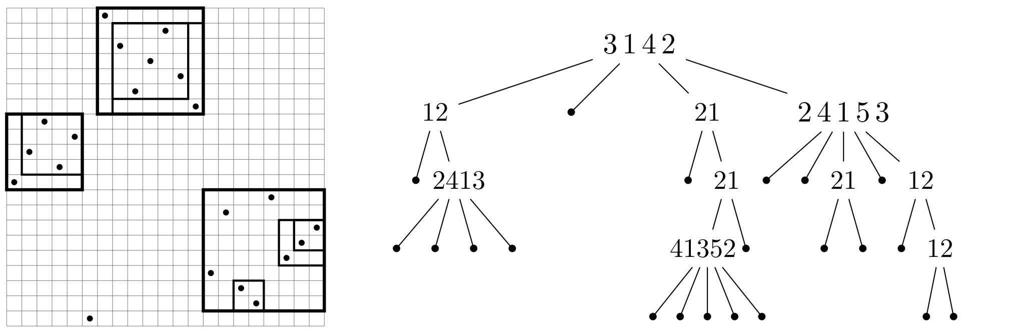


Figure 9: Decomposition tree of $\sigma = 10\,12\,14\,11\,13\,1\,21\,19\,16\,18\,20\,17\,15\,4\,8\,3\,2\,9\,5\,6\,7$.

Substitution decomposition allows to define decomposition trees, but also substitution closure:

Definition 0.37. The substitution closure $\hat{\mathcal{C}}$ of a permutation class $\mathcal{C}$ is defined as $\cup_{k\geq 1}\mathcal{C}^k$ where $\mathcal{C}^1 = \mathcal{C}$ and $\mathcal{C}^{k+1} = \{\sigma[\pi^1, \dots, \pi^n] \mid \sigma \in \mathcal{C} \text{ and } \pi^i \in C^k \text{ for any } i \text{ from 1 to } n = |\sigma|\}$.

Because simple permutations contain no intervals, we have:

Proposition 0.38. *For any class $\mathcal{C}$, the simple permutations in $\hat{\mathcal{C}}$ are exactly the simple permutations in $\mathcal{C}$.*

Consequently, for any permutation class $\mathcal{C}$ containing 12 and 21, its closure $\hat{\mathcal{C}}$ is the class of all permutations whose decomposition trees can be built on the set of nodes $\{\oplus, \ominus\} \cup \mathcal{S}_{\mathcal{C}}$, where $\mathcal{S}_{\mathcal{C}}$ denotes the set of simple permutations in $\mathcal{C}$. If $\mathcal{C}$ does not contain 12 (resp. 21), we have to remove $\oplus$ (resp. $\ominus$) from the set of nodes. Note that in this latter case, $\mathcal{S}_{\mathcal{C}} = \emptyset$ and for each n, $\mathcal{C}$ contains exactly one permutation of size n: the decreasing permutation $\mathcal{D}_n$ (resp. the identity $\mathcal{I}_n$)

Definition 0.39. A permutation class $\mathcal{C}$ is *substitution-closed* if $\mathcal{C} = \hat{\mathcal{C}}$, or equivalently if for any permutation σ of $\mathcal{C}$, and any permutations $\pi^1, \pi^2, \dots, \pi^n$ of $\mathcal{C}$ (with $n = |\sigma|$), the permutation $\sigma[\pi^1, \pi^2, \dots, \pi^n]$ also belongs to $\mathcal{C}$.

The terminology *wreath-closed* permutation class is also used, derived from the term *wreath product* sometimes used to denote the substitution operation in the context of permutations [AS02, AA05].

Example 0.40. The class $Av(231)$ of permutations sortable by a stack is not substitution-closed. Indeed $231 = 21[12, 1] \notin Av(231)$ while $1, 12$ and 21 belong to $Av(231)$. The substitution closure of $Av(231)$ is the class $Av(2413, 3142)$ of separable permutations: both classes contain 12 and 21 but no simple permutation.

Note that the substitution closure of a permutation class $\mathcal{C}$ is the smallest substitution-closed class containing $\mathcal{C}$.

To decide whether a permutation class is substitution-closed, we can use this simple characterization:

Proposition 0.41 (Proposition 1 of [AA05]). *A class is substitution-closed if and only if its basis consists entirely of simple permutations (or maybe* $1, 12$ *or* 21 *for trivial classes).*

As noticed for substitution closure, a substitution-closed permutation class can be seen as the set of decomposition trees built on the set of nodes $\{\oplus, \ominus\} \cup \mathcal{S}_{\mathcal{C}}$.

Substitution-closed classes are useful since many results are easier to prove for substitution-closed classes, and can then be extended to any permutation class using substitution closure.

This is for example the case of the main theorem of [AA05] stating that any permutation class containing only finitely many simple permuations has an algebraic generating function.

Part I

Structure of permutation classes

Foreword: A fully algorithmic method to make explicit the structure of a permutation class

Since the definition of the pattern relation among permutations by Knuth in the 70's [Knu73b], the study of permutation patterns and permutation classes in combinatorics has been a quickly growing research field, and is now well-established. Most of the research done in this domain concerns *enumeration* questions on permutation classes (see [BM03, Eli04, KM03] and their references among many others). Most articles are focused on a given class $\mathcal{C} = Av(B)$ where the basis B of excluded patterns characterizing $\mathcal{C}$ is finite, explicit, and in most cases contains only patterns of size 3 or 4. Another line of research on permutation classes has been emerging for almost a decade: it is interested in properties or results that are less precise but apply to *families* of permutation classes that are as wide as possible. Examples of such general results may regard enumeration of permutation classes that fall into general frameworks [AA05, ALR05, MT04, Vat08], properties of the corresponding generating functions [AA05, BHV08a, BHV08b, BRV08], growth rates of permutation classes [AAB$^+$10, Cib09, MT04, Vat10, Vat11], order-theoretic properties of permutation classes [AAB$^+$13, ARS11, Bri12, VW11], and so on. This second point of view is not purely combinatorial but instead is intimately linked with algorithms. Indeed, when stating general structural results on families of permutation classes, it is natural to associate to an existential theorem an algorithm that *tests* whether a class given in input falls into the family of classes covered by the theorem, and in this case to *compute* the result whose existence is guaranteed by the theorem.

Certainly the best illustration of this paradigm that can be found in the literature is the result of Albert and Atkinson [AA05], stating that every permutation class containing a finite number of simple permutations has a finite basis and an algebraic generating function, and its developments by Brignall *et al.* in [BHV08b, BHV08a, BRV08]. A possible interpretation of this result is that the simple permutations that are contained in a class somehow determine how structured the class is. Indeed, the algebraicity of the generating function is an echo of a deep structure of the class that appears in the proof of the theorem of [AA05]: the permutations of the class (or rather their decomposition trees) can be described by a context-free grammar.

In this theorem, as well as in other results obtained in this field [AA05, AAK03, BBPR10, Bri10, BHV08b, BRV08, PR12, Vat08], it appears that *simple permutations* play a crucial role. They can be seen as encapsulating most of the difficulties in the study of permutation classes considered in their generality, both in algorithms and combinatorics.

The first part of this thesis is about these general results that can be obtained for large families of permutation classes, and is resolutely turned towards algorithmic considerations. It takes its root in [AA05] and in particular in the theorem of Albert and Atkinson that we already mentioned.

Because an algebraic generating function is a witness of the combinatorial structure of a permutation class, we may interpret testing whether a permutation class has a finite number of simple permutations as testing a sufficient condition for a permutation class to be well structured.

The study of the intrinsic structure of some permutation classes lies at the boundary between algorithmics and combinatorics. The goal is indeed not only to give combinatorial criteria on the class so that one can talk about a structured class (which is not a formally defined concept, but means properties like having an rational algebraic generating function, having a finite base, having a natural recursive description) but also having algorithms to test those criteria.

In addition more could and should be done on the algorithmization of finding structure in permutation classes. In particular, the first part of this thesis provides efficient algorithms that not only *test* that there is an underlying structure in a permutation class, but that also *compute* this structure. In particular, our goal in this part is to provide a general algorithmic method to obtain a combinatorial specification for any permutation class $\mathcal{C}$ from its basis B, assuming this set is finite.

By a *combinatorial specification* of a class (see [FS09]), we mean an unambiguous system of combinatorial equations that describes recursively the permutations of $\mathcal{C}$ using only combinatorial constructors (disjoint union, cartesian product, sequence, ...) and permutations of size 1.

As discussed in [AA05], the proof of the main theorem therein is constructive. Namely, given the basis B of a class $\mathcal{C}$, and the set $\mathcal{S}_\mathcal{C}$ of simple permutations in $\mathcal{C}$ (assuming that both are finite), the proof of the main theorem of [AA05] describes how to compute an algebraic system of equations satisfied by the generating function of $\mathcal{C}$, proving thereby that it is algebraic. The main step is actually to compute a (possibly ambiguous) context-free grammar of trees for the permutations of $\mathcal{C}$, or rather their decomposition trees.

Such a context-free grammar of trees almost captures the combinatorial structure of a permutation class. The only reason why it does not do so completely is because the grammar may be ambiguous, and thus may generate several times the same permutation in the class. On the other hand, unambiguous context-free grammars of trees fall exactly in the context of the combinatorial specifications of [FS09], and describing a permutation class by such a combinatorial specification is undoubtedly demonstrating the structure of the class. Consequently, we aim at describing an algorithm to compute this combinatorial specification.

This part of the thesis provides a full algorithmic chain from the basis (when finite) of a permutation class $\mathcal{C}$ to a specification for $\mathcal{C}$. This procedure may fail to compute its result, when $\mathcal{C}$ contains an infinite number of simple permutations, this condition being tested algorithmically.

We now describe the different steps of this algorithmic chain, which is illustrated by Figure 10.

First, we should ensure that $\mathcal{C}$ falls into the set of permutation classes we can handle, i.e. ensure that $\mathcal{C}$ contains a finite number of simple permutations. Chapter 2 gives an algorithm for this first step, following the line opened by [BRV08].

Second, when finite, we compute the set $\mathcal{S}_\mathcal{C}$ of simple permutations in $\mathcal{C}$. An algorithm for this second step is described in Chapter 1, and its complexity analyzed. It should be noticed that the complexity of this algorithm also depends on the size of its output, namely on $|\mathcal{S}_\mathcal{C}|$ and on $\max\{|\pi| : \pi \in \mathcal{S}_\mathcal{C}\}$.

Third, from B and $\mathcal{S}_\mathcal{C}$, we turn the constructive proof of [AA05] into an actual algorithm, that computes the (possibly ambiguous) context-free grammar of trees describing the decomposition trees of the permutations of $\mathcal{C}$.

Finally, we transform this (possibly ambiguous) context-free grammar into an unambiguous combinatorial specification for $\mathcal{C}$. We describe in Chapter 3 an algorithm for these last two steps, whose complexity is still to be analyzed.

Combining these four steps provides an algorithm to obtain from a basis B of excluded patterns a combinatorial specification for the permutation class $\mathcal{C} = Av(B)$ when $\mathcal{C}$ contains a finite number of simple permutations. We are not only convinced of the importance of this result from a theoretical point of view, but also (and maybe more importantly) we are confident that it will be of practical use to the permutation patterns community. Indeed, from a combinatorial specification, it is of course possible with the methodology of [FS09] to immediately deduce a system of equations for the generating function of $\mathcal{C}$. But other algorithmic developments can be considered. In particular, this opens the way to obtaining systematically Boltzmann random samplers of permutations in a class, or to the automatic evaluation of the Stanley-Wilf growth rate of a class.

Since simple permutations play a crucial role in describing the structure of a permutation class, we start by studying in Chapter 1 the poset of simple permutations with respect to the pattern containment relation.

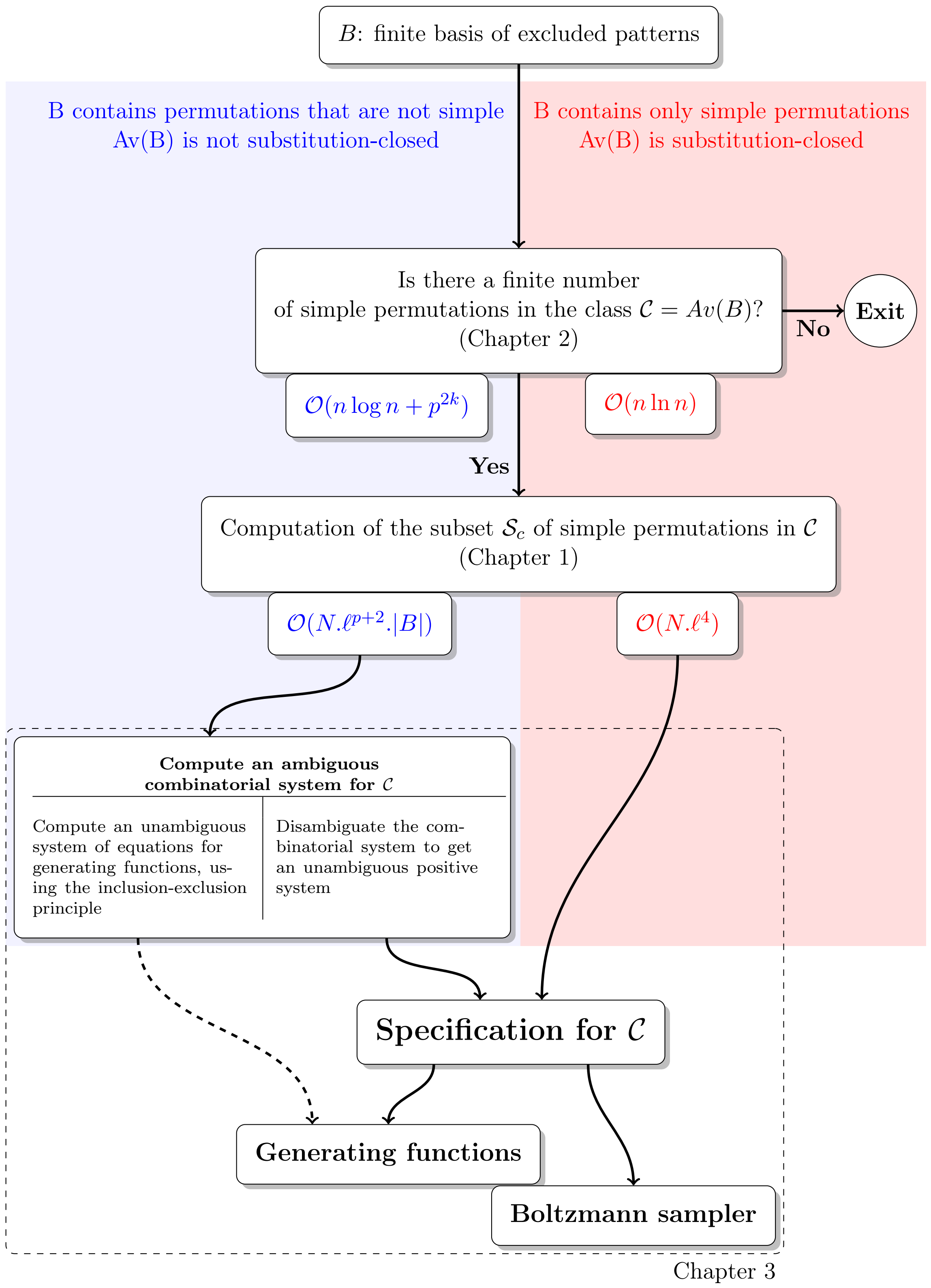

B: finite basis of excluded patterns
B contains permutations that are not simple
Av(B) is not substitution-closed
B contains only simple permutations
Av(B) is substitution-closed
Is there a finite number of simple permutations in the class $\mathcal{C} = Av(B)$? (Chapter 2)
No
Exit
$\mathcal{O}(n \log n + p^{2k})$
$\mathcal{O}(n \ln n)$
Yes
Computation of the subset $\mathcal{S}_c$ of simple permutations in $\mathcal{C}$ (Chapter 1)
$\mathcal{O}(N.\ell^{p+2}.|B|)$
$\mathcal{O}(N.\ell^4)$
Compute an ambiguous combinatorial system for $\mathcal{C}$
Compute an unambiguous system of equations for generating functions, using the inclusion-exclusion principle
Disambiguate the combinatorial system to get an unambiguous positive system
Specification for $\mathcal{C}$
Generating functions
Boltzmann sampler
Chapter 3

Figure 10: Automatic process from the basis of a permutation class to generating function and Boltzmann sampler. The complexities are given w.r.t. $n = \sum_{\beta \in B} |\beta|$, $k = \sharp B$, $p = \max\{|\beta| : \beta \in B\}$, $N = \sharp \mathcal{S}_{\mathcal{C}}$ and $\ell = \max\{|\pi| : \pi \in \mathcal{S}_{\mathcal{C}}\}$ where $\mathcal{S}_{\mathcal{C}}$ is the set of simple permutations of $\mathcal{C}$.

Chapter 1

Simple permutations as a poset

This chapter studies the poset of simple permutations with respect to the pattern involvement. We specialize results on critically indecomposable posets obtained by Schmerl and Trotter in [ST93] to simple permutations and prove that if σ and π are two simple permutations such that $\pi < \sigma$ then there exists a chain of simple permutations $\sigma^{(0)} = \sigma, \sigma^{(1)}, \dots, \sigma^{(k)} = \pi$ such that $|\sigma^{(i)}| - |\sigma^{(i+1)}| = 1$, or 2 when permutations are exceptional, and $\sigma^{(i+1)} < \sigma^{(i)}$. This structural result on permutations and pattern involvement has many consequences. First, it gives the average number of points in a simple permutation that can be removed and give another simple permutation. Second, it gives rise to an algorithm polynomial in the size of the output to generate the set of simple permutations in a substitution-closed class of permutations. This algorithm can be adapted for non-substitution-closed classes but with a loss of efficiency.

1.1 Introduction

Simple permutations play a key role in the study of permutation classes. More precisely, they are core objects in the substitution decomposition of permutations. For example, if a class contains a finite number of simple permutations then it is finitely based, meaning that the class can be expressed as the set of permutations that do not contain as pattern any permutation of a finite set B. Moreover, the generating function of the permutation class is then algebraic. Also, even though the pattern involvement problem is NP-complete in general, there exists an FPT algorithm [BR06] whose parameter is the length of the largest simple permutation that appears as a pattern of the involved permutations.

In this chapter, we study the set of simple permutations with respect to the pattern containment relation, starting from results of [ST93]. In [ST93], Schmerl and Trotter study finite irreflexive binary relational structures and prove many structural results on critically indecomposable structures. They notice that their results holds for graphs, partially ordered sets (posets), tournaments, and oriented graphs.

We focus on permutations and show that the general results on binary relational structures can be refined in our case, with simple permutations playing the role of indecomposable structures. More precisely, if σ and π are two simple permutations such that π is pattern of σ, results of [ST93] imply that there exists a chain of simple permutations $\sigma^{(0)} = \sigma, \sigma^{(1)}, \ldots, \sigma^{(k)} = \pi$ such that $|\sigma^{(i)}| - |\sigma^{(i+1)}| = 1$ or 2 and $\sigma^{(i+1)} < \sigma^{(i)}$. Using the structure of simple permutations, we strengthen the result and show that except when σ is exceptional, we can find a chain with all size differences of 1. Moreover when σ is exceptional there exists a chain with all size differences of 2.

This structural result on permutations and pattern involvement has many consequences. First, it allows us to compute the average number of points in a simple permutation that can be removed, one at each time, in order to obtain another simple permutation, showing that it is almost the case of each point of the permutation. We also give the exact number of points in a simple permutation that can be added to give another simple permutation. Second, it gives rise to a polynomial algorithm (polynomial in this size of the output) to generate simple permutations of a finitely based substitution-closed class of permutations. This algorithm roughly starts by looking to simple permutations of size 4 and iterates over the size of permutations. The characterization of the preceding chain translates into a polynomial time algorithm for finding simple permutations of size $n+1$ in $Av(B)$ knowing only simple permutations of size n and $n-1$ in this class. Note that our algorithm requires no pattern involvement test. Although we give this algorithm in the general framework of substitution-closed permutation classes, we apply it for classes containing only a finite number of simple permutations. It can be also used to generate the simple permutations of any substitution-closed class up to a given size, even if the number of simple permutations in the class is infinite. The overall complexity of this algorithm is polynomial in the number of simple permutations. This algorithm is then adapted for non-substitution-closed classes but with a loss of efficiency.

1.2 Preliminaries

1.2.1 Exceptional permutations

A subset of simple permutations, called exceptional and introduced in [AA05], plays a key role in this chapter.

Definition 1.1. Exceptional permutations are permutations defined below for every $m \geq 2$

(see Figure 1.1):

- 2 4 6 8 . . . $(2m)$ 1 3 5 . . . $(2m-1)$ — type 1
- $(2m-1)$ $(2m-3)$ . . . 1 $(2m)$ $(2m-2)$ . . . 2 — type 2
- $(m+1)$ 1 $(m+2)$ 2 . . . $(2m)$ m — type 3
- m $(2m)$ $(m-1)$ $(2m-1)$ . . . 1 $(m+1)$ — type 4

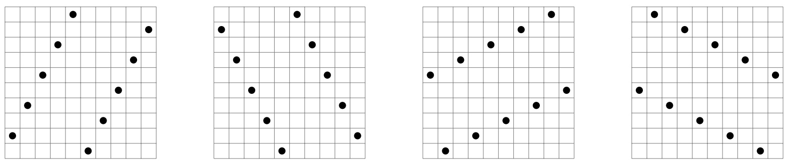


Figure 1.1: Exceptional permutations of type 1, 2, 3 and 4

Remark 1.2. For $m \geq 3$, there are exactly 4 exceptional permutations of size $2m$. But there are only 2 exceptional permutations of size 4: 2413, which is of type 1 and of type 4, and 3142, which is of type 2 and of type 3.

Notice also that, if we remove the symbols $2m-1$ and $2m$ from the first two types, we obtain another exceptional permutation of the same type; and likewise if we remove the symbols in the last two positions from the types 3 and 4 and renormalize the result.

Proposition 1.3. *Let σ, σ' be two exceptional permutations with $|\sigma| \leq |\sigma'|$. Then $\sigma \leq \sigma'$ if and only if σ and σ' are exceptional permutations of the same type.*

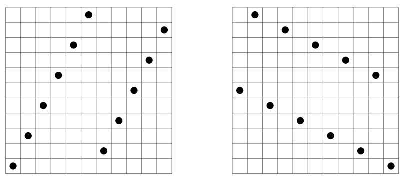


Figure 1.2: Parallel alternations

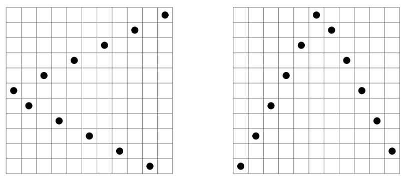


Figure 1.3: Wedge alternations

A more general kind of permutations containing exceptional permutations will appear naturally in this chapter. An *alternation* is a permutation in which every odd entry lies to the left of every even entry, or any symmetry of such a permutation. A *parallel* alternation is an alternation in which these two sets of entries form monotone subsequences, either both increasing or both decreasing. A *wedge* alternation is an alternation in which the two sets of entries form monotone subsequences pointing in opposite directions. See Figures 1.2 and 1.3 for examples.

Exceptional permutations are simple alternations:

Proposition 1.4. *Exceptional permutations are simple permutations. Moreover among parallel or wedge alternations, only exceptional permutations are simple.*

1.2.2 General results on binary relational structures

In this subsection, we recall results of [ST93] that we use in the sequel. First we recall the terminology used in [ST93]. We restrict ourselves to structures of type 1 while [ST93] deals generally with structures of type k.

A *binary relational structure* $\mathcal{A}$ *of type* 1 consists of a non-empty underlying set A together with a binary relation $R \subseteq A \times A$. We write $\mathcal{A}$ as $(A; R)$. The order of $\mathcal{A}$ is $|A|$, the number of elements of A; in particular $\mathcal{A}$ is *finite* if A is finite. If R is irreflexive (that is, if aRa for no $a \in A$) then $\mathcal{A}$ is *irreflexive*.

The only structures that we will be encountering are finite, irreflexive, binary structures of type 1, so we will henceforth use the word 'structure' to refer just to these. Graphs, partially ordered sets (posets), tournaments, and oriented graphs are examples of such structures. For instance if $\mathcal{A} = (A; R)$, then $\mathcal{A}$ is a poset if and only if aRc whenever aRb and bRc.

A structure $\mathcal{B}$ is a *substructure* of $\mathcal{A}$ if $\mathcal{A} = (A; R)$, $\mathcal{B} = (B; S)$, $B \subseteq A$, and $S = R \cap (B \times B)$. If $\mathcal{B}$ is isomorphic to a substructure of $\mathcal{A}$ then $\mathcal{B}$ is *embeddable* in $\mathcal{A}$.

The *lexicographic product* $\mathcal{A} * \mathcal{B}$ of two structures $\mathcal{A} = (A; R)$ and $\mathcal{B} = (B; S)$ is defined to be $(A \times B; T)$, where $(a_1, b_1)T(a_2, b_2)$ if and only if either a_1Ra_2 or else $a_1 = a_2$ and b_1Sb_2.

A structure $\mathcal{A}$ is *indecomposable* if, whenever $\mathcal{A}$ is embeddable in $\mathcal{B}_1 * \mathcal{B}_2$, then $\mathcal{A}$ is embeddable in $\mathcal{B}_1$ or in $\mathcal{B}_2$. This definition of indecomposable has a useful equivalent in terms of intervals.

Let $\mathcal{A} = (A; R)$ be a structure, and let $I \subseteq A$ be a subset. Then I is an *interval* of $\mathcal{A}$ if whenever $a, b \in I$ and $c \in A \setminus I$, then both aRc if and only if bRc and cRa if and only if cRb. The interval I is *nontrivial* if $2 \leq |I| < |A|$.

Proposition 1.5 (Proposition 1.1 of [ST93]). *A structure $\mathcal{A}$ is indecomposable if and only if it contains no nontrivial intervals.*

Notice that this definition of indecomposable structures is very similar to the definition of simple permutations (Definition 0.21).

We now state main results of [ST93] that we use in the sequel.

Proposition 1.6 (Theorem 2.1 of [ST93]). *Suppose $\mathcal{A} = (A; R)$ is indecomposable and $|A| \geq 3$. Then there is an indecomposable $B \subseteq A$ such that either $|B| = 3$ or $|B| = 4$.*

Proposition 1.7 (Theorem 2.2 of [ST93]). *Suppose $\mathcal{A} = (A; R)$ is indecomposable and $B \subseteq A$ is also indecomposable, where $3 \leq |B| \leq |A| - 2$. Then there is an indecomposable C such that $B \subseteq C \subseteq A$ and $|C| = |B| + 2$.*

An indecomposable structure $\mathcal{A} = (A; R)$ is *critically indecomposable* if whenever $a \in A$, then $A \setminus \{a\}$ is not indecomposable.

Propositions 1.6 and 1.7 imply a strong result about critically indecomposable structures.

Proposition 1.8 (Corollary 3.1 of [ST93]). *Suppose that $\mathcal{A}$ is a critically indecomposable structure of order n and that $3 \leq m \leq n$. Then $\mathcal{A}$ has an indecomposable substructure of order m if and only if $n - m$ is even.*

For indecomposable structures which are not critically indecomposable, Schmerl and Trotter gives the following result:

Proposition 1.9 (Corollary 5.10 of [ST93]). *Suppose $\mathcal{A}$ is an indecomposable structure of order n which is not critically indecomposable, and suppose $5 \leq m \leq n$. Then $\mathcal{A}$ has an indecomposable substructure of order m.*

Finally, Theorem 5.1. of [ST93] gives the list of critically indecomposable structures (up to isomorphism of skeletons). Then this result is specialized to graphs, posets, tournaments and oriented graphs in Corollary 5.8 of [ST93]. The result that we use is the one about posets. To state it we first need to define two particular families of posets.

Definition 1.10. For each $r \geq 2$ we define two posets $\mathcal{P}_r$ and $\mathcal{P'}_r$ of order $2r$. Let $V_r = \{a_1, a_2, \dots a_r, b_1, b_2, \dots b_r\}$, we set $\mathcal{P}_r = (V_r; P_r)$ and $\mathcal{P'}_r = (V_r; P'_r)$ where

$$xP_ry \Leftrightarrow x = a_i \text{ and } y = b_j \text{ for some } i \geq j, \text{ and}$$
$$xP'_ry \Leftrightarrow (x, y) \in \{(a_i, b_j), (a_i, a_j), (b_i, b_j)\} \text{ for some } i < j.$$

Proposition 1.11 (Corollary 5.8 (2) of [ST93]). *A structure $\mathcal{A}$ is a critically indecomposable poset if and only if it is isomorphic to either $\mathcal{P}_r$ or $\mathcal{P'}_r$ for some $r \geq 2$.*

1.3 Pattern containment on simple permutations

1.3.1 Simple patterns of simple permutations

In this section, we specialize and extend results from [ST93] to permutations.

The idea of applying results of [ST93] to permutations comes from [AA05], where it is used to prove the following theorem:

Theorem 1.12 (Theorem 5 of [AA05]). *If π is simple, then either there is a one-point deletion that is also simple or π is exceptional (in which case it has a two point deletion that is simple).*

We develop and make more precise this idea in this section.

To each permutation σ is associated a poset $P(\sigma) = ([1..n], <)$ where $i < j \Leftrightarrow (i < j \text{ and } \sigma_i < \sigma_j)$. Recall that a poset on $[1..n]$ is indecomposable if it does not contain any non-trivial interval with respect to the relation, $<$ in our case. It is critically indecomposable if furthermore whenever an element is removed, the resulting poset is not indecomposable. In the specific case of permutations those poset characteristics can be translated as stated in the following propositions:

Proposition 1.13. *The poset $P(\sigma)$ is indecomposable if and only if σ is simple or is equal to* 1*,* 12 *or* 21*.*

Proof. By definition of simple permutations (Definition 0.21), this is a direct consequence of Proposition 1.5 noticing that intervals of σ corresponds to intervals of $P(\sigma)$. ■

Proposition 1.14. *$P(\sigma)$ is critically indecomposable if and only if σ is exceptional.*

Proof. This is a consequence of Corollary 5.8(2) of [ST93] (Proposition 1.11). ■

The standard poset isomorphism can be transposed to integer posets.

Definition 1.15. Let $A, B \subset \mathbb{N}$ be posets, then $A \equiv B$ if A and B are isomorphic as posets and the isomorphism keeps the natural integer ordering on $\mathbb{N}$.

Proposition 1.16. *Let σ and π be permutations. Then $\pi \leq \sigma$ if and only if there exists $A \subset P(\sigma)$ such that $A \equiv P(\pi)$.*

The next four propositions are mere translations of results from [ST93] on permutations, using the three propositions above.

Proposition 1.17. *Let σ be a simple permutation, then either* 2 4 1 3 *or* 3 1 4 2 *is a simple pattern of σ.*

Proof. This is a direct consequence of Proposition 1.6, noticing that there are no simple permutations of size 3 and there are only two simple permutations of size 4. ■

Proposition 1.18. *Let π, σ be two simple permutations with $\pi \leq \sigma$. If $|\pi| \leq |\sigma| - 2$, then there exists a simple permutation τ such that $\pi \leq \tau \leq \sigma$ and $|\tau| = |\pi| + 2$.*

Proof. This is the mere translation of Theorem 2.2 of [ST93] (Proposition 1.7). ■

Proposition 1.19. *Let σ be an exceptional permutation. If $3 \leq m \leq |\sigma|$, then σ has a simple pattern of size m if and only if m is even.*

Proof. This is a direct consequence of Proposition 1.8 as every exceptional permutation is of even size. ■

The preceding result gives the sizes of simple patterns for exceptional permutations. For non-exceptional ones, the following proposition concludes:

Proposition 1.20. *Let σ be a non-exceptional simple permutation. If $4 \leq m \leq |\sigma|$ then σ has a simple pattern of size m.*

Proof. This is the mere translation of Proposition 1.9 extended to size 4 using Proposition 1.17. ■

Using Proposition 1.20, we can refine the result of Proposition 1.19:

Proposition 1.21. *If σ is an exceptional permutation, then for every m such that $3 \leq m \leq |\sigma|$:*

- *If m is odd, then σ has no simple pattern of size m.*
- *Otherwise m is even and σ has exactly one simple pattern of size m which is the exceptional permutation of the same type as σ.*

Proof. The first item (m is odd) is a direct consequence of Proposition 1.19. For the second point, σ has at least one simple pattern π of size m by Proposition 1.19. Suppose now that π is not exceptional, then $m \geq 5$ as simple permutations of size 4 are exceptional. Then, using Proposition 1.20, π has a simple pattern τ of size 5, thus τ is a pattern of σ but of odd size which is forbidden by Proposition 1.19. So π is exceptional and of the same type as σ from Proposition 1.3. ■

A direct consequence of the preceding proposition is that all simple patterns of exceptional permutations are exceptional. Moreover the biggest proper simple pattern of an exceptional permutation σ has size $|\sigma| - 2$. For non-exceptional simple permutations, we can find simple patterns by deleting only one point:

Proposition 1.22. *Let σ be a simple permutation. Then σ has a simple pattern of size $|\sigma| - 1$ if and only if σ is not exceptional.*

Proof. Consequence of Proposition 1.19 and Proposition 1.20 above. ■

1.3.2 Simple pattern containing a given simple permutation

The results obtained in the preceding section describe how a simple permutation can give other simple permutations by deleting elements. In the sequel, we add another constraint on patterns, that is we want to delete an element in a simple permutation σ containing a simple permutation π as a pattern to obtain another simple permutation σ' such that $\pi \leq \sigma'$.

Proposition 1.24 deals with the case $|\pi| = |\sigma| - 2$ and relies on the following intermediate result:

Proposition 1.23. *Let τ be a non-simple permutation such that $\tau \setminus \{\tau_i\}$ is simple. Then τ_i belongs to an interval of size 2 of τ or is in a corner of the diagram of τ.*

Proof. As τ is not simple, τ contains at least one non-trivial interval I. As I is an interval of τ, $I \setminus \{\tau_i\}$ is an interval of $\tau \setminus \{\tau_i\}$. But $\tau \setminus \{\tau_i\}$ is simple thus $I \setminus \{\tau_i\}$ is a trivial interval of $\tau \setminus \{\tau_i\}$, hence is a singleton $\{\tau_k\}$ or the whole permutation $\tau \setminus \{\tau_i\}$. But I is non-trivial so in the first case $I = \{\tau_i, \tau_k\}$ and τ_i belongs to an interval of size 2 of τ, and in the second case $I = \tau \setminus \{\tau_i\}$ and τ_i is in a corner of the diagram of τ. ■

Proposition 1.24. *Let σ be a non-exceptional simple permutation of size n and π a simple permutation of size $n-2$ such that $\pi \leq \sigma$. Then there exists a simple permutation τ of size $n-1$ such that $\pi \leq \tau \leq \sigma$.*

Proof. Assume that such a permutation τ does not exist. We prove that this leads to a contradiction. Let i, j such that $\pi = \sigma \setminus \{\sigma_i, \sigma_j\}$. If $\sigma \setminus \{\sigma_i\}$ is simple then $\tau = \sigma \setminus \{\sigma_i\}$ would contradict our hypothesis. Thus $\sigma \setminus \{\sigma_i\}$ is not simple, but $\pi = \sigma \setminus \{\sigma_i, \sigma_j\}$ is simple. From Proposition 1.23, σ_j belongs to an interval of size 2 of $\sigma \setminus \{\sigma_i\}$ or is in a corner of the bounding box of the diagram of $\sigma \setminus \{\sigma_i\}$ thanks to π. By symmetry between i and j the same results holds when exchanging these two indices. So there are 3 different cases:

- σ_i and σ_j are both in a corner thanks to π. In that case π is a non-trivial interval of σ, which contradicts the fact that σ is simple.
- σ_i belongs to an interval I of size 2 of $\sigma \setminus \{\sigma_j\}$ and σ_j is in a corner of $\sigma \setminus \{\sigma_i\}$ thanks to π (the same proof holds when exchanging i and j). As σ is simple, σ_j is not in a corner of σ, but is in a corner of $\sigma \setminus \{\sigma_i\}$. Thus σ_i is the only point separating σ_j from a corner (see Figure 1.4 for an example). Let i_1 such that $I = \{i, i_1\}$, then σ_j is the only point separating σ_{i_1} from σ_i, so $\pi = \sigma \setminus \{\sigma_{i_1}, \sigma_j\}$. If $\sigma \setminus \{\sigma_{i_1}\}$ is simple then $\tau = \sigma \setminus \{\sigma_{i_1}\}$ would contradict our hypothesis that such a permutation τ does not exist. Thus $\sigma \setminus \{\sigma_{i_1}\}$ is not simple, but $\pi = \sigma \setminus \{\sigma_{i_1}, \sigma_j\}$ is simple. Hence from Proposition 1.23 σ_j belongs to an interval J of size 2 of $\sigma \setminus \{\sigma_{i_1}\}$, or is in a corner of $\sigma \setminus \{\sigma_{i_1}\}$ which is impossible as σ_i separate it from one corner and $|\sigma| \geq 4$. Let j_1 such that $J = \{j, j_1\}$, then $\pi = \sigma \setminus \{\sigma_{i_1}, \sigma_{j_1}\}$. If $\sigma \setminus \{\sigma_{j_1}\}$ is simple then $\tau = \sigma \setminus \{\sigma_{i_1}\}$ gives a contradiction. Letting $i_0 = i$ and $j_0 = j$, we recursively build this way $i_0, j_0, i_1, j_1, \ldots$ such that $\forall k, \pi = \sigma \setminus \{\sigma_{i_k}, \sigma_{j_k}\} = \sigma \setminus \{\sigma_{j_k}, \sigma_{i_{k+1}}\}$ and $\sigma \setminus \{\sigma_{i_k}\}$ and $\sigma \setminus \{\sigma_{j_k}\}$ are not simple, until reaching all points of σ, i.e. until $i_k = n$ or $j_k = n$ in the case depicted in Figure 1.4. Points σ_i and σ_{i_1} are in increasing or decreasing order. For each case, σ_{i_1} has a determined position (see Figure 1.4). Then the positions of σ_{i_k} and σ_{j_k} are fixed for all k. Indeed only $\sigma_{j_{k-1}}$ separates σ_{i_k} from $\sigma_{i_{k-1}}$, and σ_{i_k} does not separate $\sigma_{i_{k-1}}$ from $\sigma_{i_{k-2}}$. The same reasoning goes for σ_{j_k}. Therefore, depending on the position of σ_{i_1}, σ is either a parallel alternation or a wedge alternation, and thus is exceptional or not simple, a contradiction.

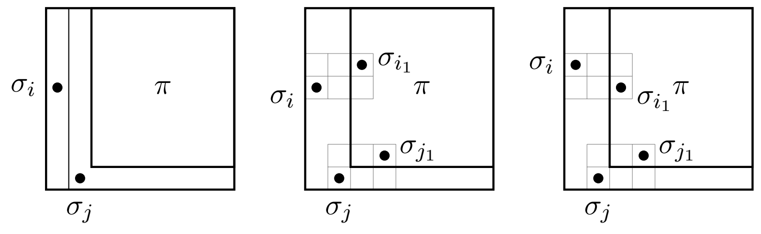


Figure 1.4: Diagram of σ in the second case of the proof of Proposition 1.24.

- σ_i belongs to an interval I of size 2 of $\sigma \setminus \{\sigma_j\}$ and σ_j belongs to an interval J of size 2 of $\sigma \setminus \{\sigma_i\}$. Letting i_1 be such that $I = \{i, i_1\}$, I is an interval of $\sigma \setminus \{\sigma_j\}$ but σ is simple thus σ_j is the only point separating σ_i from σ_{i_1}. Letting j_1 be such that $J = \{j, j_1\}$, J is an interval of $\sigma \setminus \{\sigma_i\}$ but σ is simple thus σ_i is the only point separating σ_j from σ_{j_1}. This indeed is one of the two cases depicted in Figure 1.5 (up to symmetry). Let $i'_0 = i$ and $j'_0 = j$, we recursively build $j'_0, i_1, j'_1, i_2 \ldots$ (resp. $i'_0, j_1, i'_1, j_2 \ldots$) until reaching all points of σ, i.e. until $i_k = n$ or $j'_k = n$ (resp. $j_k = 1$ or $i'_k = 1$) in the case depicted in Figure 1.5. We have $\pi = \sigma \setminus \{\sigma_i, \sigma_j\} = \sigma \setminus \{\sigma_{i_1}, \sigma_j\}$. If $\sigma \setminus \{\sigma_{i_1}\}$ is simple then $\tau = \sigma \setminus \{\sigma_{i_1}\}$ contradicts our hypothesis that such a permutation τ does not exist. Thus $\sigma \setminus \{\sigma_{i_1}\}$ is not simple, but $\pi = \sigma \setminus \{\sigma_{i_1}, \sigma_j\}$ is simple. Therefore from Proposition 1.23, σ_j belongs to an interval J' of size 2 of $\sigma \setminus \{\sigma_{i_1}\}$ or lies in a corner of $\sigma \setminus \{\sigma_{i_1}\}$ which is impossible if $i_1 \neq n$ (up to symmetry). Let $J' = \{j, j'_1\}$ then $\pi = \sigma \setminus \{\sigma_{i_1}, \sigma_j\} = \sigma \setminus \{\sigma_{i_1}, \sigma_{j'_1}\}$. If $\sigma \setminus \{\sigma_{j'_1}\}$ is simple then $\tau = \sigma \setminus \{\sigma_{i_1}\}$ contradicts our hypothesis. Moreover $\pi = \sigma \setminus \{\sigma_i, \sigma_j\} = \sigma \setminus \{\sigma_i, \sigma_{j_1}\}$. If $\sigma \setminus \{\sigma_{j_1}\}$ is simple then $\tau = \sigma \setminus \{\sigma_{j_1}\}$ fulfill the theorem, contradiction. Thus $\sigma \setminus \{\sigma_{j_1}\}$ is not simple but $\pi = \sigma \setminus \{\sigma_{j_1}, \sigma_i\}$ is simple so that σ_i belongs to an interval I' of size 2 of $\sigma \setminus \{\sigma_{j_1}\}$ or lies in a corner of $\sigma \setminus \{\sigma_{j_1}\}$, which is impossible if $j_1 \neq 1$ (up to symmetry). Letting i'_1 be such that $I' = \{i, i'_1\}$, then $\pi = \sigma \setminus \{\sigma_i, \sigma_{j_1}\} = \sigma \setminus \{\sigma_{i'_1}, \sigma_{j_1}\}$. If $\sigma \setminus \{\sigma_{i'_1}\}$ is simple then $\tau = \sigma \setminus \{\sigma_{i'_1}\}$ fulfill our theorem, contradiction. Similarly to the preceding case, we can prove by induction until reaching all points of σ that either σ is a parallel alternation or a wedge permutation so that σ is exceptional or not simple, providing the desired contradiction. ∎

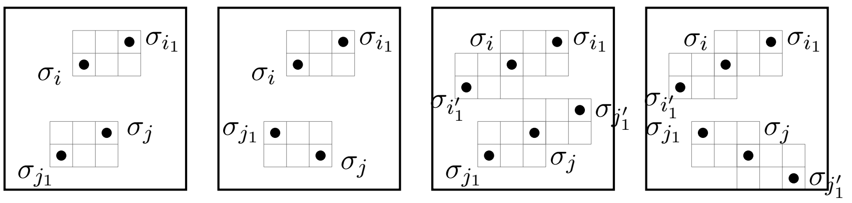


Figure 1.5: Diagram of σ in the third case of the proof of Proposition 1.24.

Thanks to Proposition 1.24 and a straightforward induction, we are able to state our main result on pattern involvement of simple permutations.

Theorem 1.25. *Let $\pi < \sigma$ be two simple permutations with σ non-exceptional. Then there exists a simple permutation τ such that $\pi \leq \tau < \sigma$ and $|\tau| = |\sigma| - 1$.*

Proof. We prove this result by induction on $|\sigma| - |\pi|$ using Proposition 1.18. If $|\sigma| - |\pi|$ is odd, using recursively Proposition 1.18 we find a simple permutation τ such that $\pi \leq \tau \leq \sigma$ and $|\tau| = |\sigma| - 1$. If $|\sigma| - |\pi|$ is even, we find a simple permutation τ' such that $\pi \leq \tau' \leq \sigma$ and $|\tau'| = |\sigma| - 2$ and we apply Proposition 1.24 which ensures the existence of a simple permutation τ such that $\pi \leq \tau' \leq \tau \leq \sigma$ and $|\tau| = |\sigma| - 1$. ∎

Theorem 1.25 has many implications. First, it gives structural results on the poset of simple permutations, studied in the next section. Second it leads to an efficient algorithm for computing the set of simple permutations in a permutation class (see Section 1.5).

1.4 Simple permutations as a poset

In this section we study the poset of simple permutations with respect to the pattern containment relation. We represent this poset by an oriented graph G, whose vertices are the simple permutations, and with an edge from a simple permutation σ to a simple permutation π if and only if $\pi < \sigma$ and there is no simple permutation τ such that $\pi < \tau < \sigma$. Then $\pi < \sigma$ if and only if there is a path from σ to π in G. From Theorem 1.25, if σ is not exceptional there is an edge from σ to π if and only if we can obtain π from σ by deleting one point, and from Proposition 1.21 if σ is exceptional, there is an edge from σ to π if and only if π is exceptional of the same type of σ and $|\sigma| = |\pi| + 2$. In this section we study other properties of G. We first focus on paths of G, then we study the degree of vertices in G.

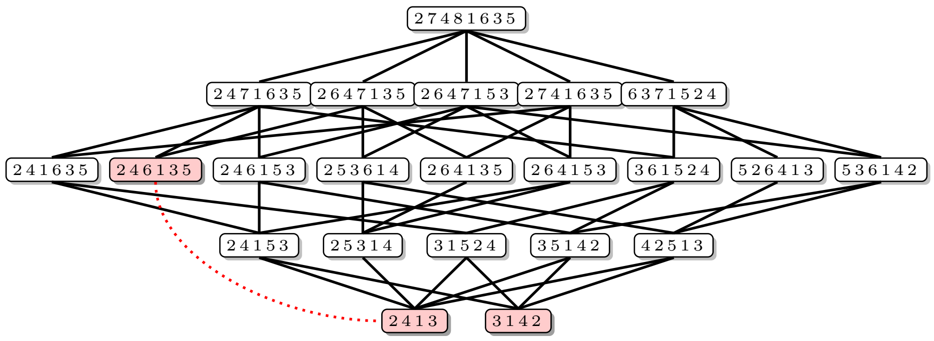


Figure 1.6: The poset of simple permutations which are pattern of 2 7 4 8 1 6 3 5. Colored nodes are exceptional ones.

1.4.1 Paths in the poset of simple permutations

In Theorem 1.26, we prove that if a simple permutation σ has a simple pattern π, then there is a path in G from σ to π in the graph whose first part consists of non-exceptional simple permutations of consecutive sizes and second part of exceptional permutations (one of the parts can be empty). Conversely, all paths from σ to π are of this form. Then we extend this result in Theorem 1.29 to prove that whenever σ is not exceptional, there is such a path such that the second part of the path is empty, that is we can reach π from σ by deleting one element at a time and all the permutations involved are simple.

Theorem 1.26. *Let $\pi < \sigma$ be simple permutations, then there exists a chain of simple permutations $\sigma^{(0)} = \sigma, \sigma^{(1)}, \dots, \sigma^{(k-1)}, \sigma^{(k)} = \pi$ and $m \in [0..k]$ such that for all $i \in [1..k]$, $\sigma^{(i)} < \sigma^{(i-1)}$ and:*

- $|\sigma^{(i-1)}| - |\sigma^{(i)}| = 1$ *if* $1 \le i \le m$ *(in particular $\sigma^{(i-1)}$ is not exceptional),*
- $|\sigma^{(i-1)}| - |\sigma^{(i)}| = 2$ *if* $m + 1 \le i \le k$
- *if $m < k$ then $\sigma^{(i)}$ is exceptional for $m \le i \le k$.*

Conversely, all paths from σ to π are of this form.

Proof. If σ is exceptional, then π is exceptional of the same type as σ (from Proposition 1.21). Then we set $m = 0$ and $k = (|\sigma| - |\pi|)/2$, and $\sigma^{(i)}$ are exceptional permutations of the same type as σ and size between $|\pi|$ and $|\sigma|$.

If σ is not exceptional, we set $\sigma^{(0)} = \sigma$, and using Theorem 1.25, for $i \ge 1$ we construct by induction $\sigma^{(i)}$ such that $\sigma^{(i)}$ is simple, $\pi \le \sigma^{(i)} < \sigma^{(i-1)}$ and $|\sigma^{(i)}| = |\sigma^{(i-1)}| - 1$ while

$\sigma^{(i-1)}$ is not exceptional and $\pi \neq \sigma^{(i-1)}$. We iterate until $\sigma^{(j)} = \pi$, then $m = k = |\sigma| - |\pi|$ and we have the result, or until $\sigma^{(j)}$ is exceptional. Then π is exceptional of the same type as $\sigma^{(j)}$. Then we set $m = j$ and $k = m + (|\sigma^{(j)}| - |\pi|)/2$, and $\sigma^{(i)}$ for $j \leq i \leq k$ are exceptional permutations of the same type as π and size between $|\pi|$ et $|\sigma^{(j)}|$.

From Proposition 1.21 it is obvious that, conversely, all paths from σ to π are of this form. ■

Note that the paths in G between two simple permutations can be of different lengths. As an example, with $\sigma = 5263714$ and $\pi = 3142$, we have a path of length 2 (by 415263, exceptional, see Figure 1.7) and a path of length 3 (by 526314 and 42613, non-exceptional, see Figure 1.8).

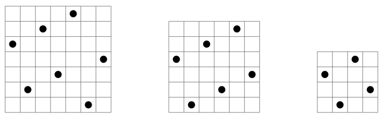


Figure 1.7: Path of length 2 from $\sigma = 5263714$ to $\pi = 3142$.

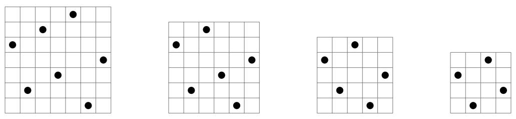


Figure 1.8: Path of length 3 from $\sigma = 5263714$ to $\pi = 3142$.

However if σ and π are both non-exceptional, all paths from σ to π have length $|\sigma| - |\pi|$, and if σ and π are both exceptional, all paths from σ to π have length $(|\sigma| - |\pi|)/2$.

The last case is σ not exceptional and π exceptional. Then Theorem 1.29 below shows that we can always choose a path with only one exceptional permutation: π. The proof of Theorem 1.29 relies on the two following propositions:

Proposition 1.27. *Let π be an exceptional permutation of size $2(n+1)$ where $n \geq 2$, P be a set of n points of π and π' the exceptional permutation of size $2n$ of the same type as π. Then there exists a pattern π' in π which contains all points of P.*

Proof. We have only to prove the result for exceptional permutations of type 3; the result for the other types follows by symmetry. Suppose that π is of type 3. Then by deleting two points of π of consecutive indices, we obtain a pattern π'. So we have only to prove that there exist two points of consecutive indices neither of which is in P. If two such points do not exist, then P contains at least half of points of π i.e. $n+1$ points, a contradiction. ■

Proposition 1.28. *Let $\pi' < \pi$ be exceptional permutations with $|\pi'| = |\pi| - 2$ and σ a non-exceptional simple permutation such that $\pi < \sigma$ and $|\sigma| = |\pi| + 1$. Then there exists a simple permutation τ such that $\pi' < \tau < \sigma$ and $|\tau| = |\pi'| + 1$.*

Notice that Figures 1.7 and 1.8 provide an example of this proposition: with $\pi' = 3142$, $\pi = 415263$ and $\sigma = 5263714$, we can take $\tau = 42513$.

Proof. Let $n = |\sigma| = |\pi| + 1$, as $\pi < \sigma$ there exists an index $k \in [1..n]$ such that we obtain π from σ by deleting σ_k. As σ is simple, we cannot have both $k \in \{1, n\}$ and $\sigma_k \in \{1, n\}$. So there exist i and j in $[1..n]$ such that $i = k - 1$ and $j = k + 1$, or $\sigma_i = \sigma_k - 1$ and $\sigma_j = \sigma_k + 1$. As σ is simple, there exists a point $\sigma_{i'}$ which separates σ_i from σ_k and a point $\sigma_{j'}$ which separates σ_j from σ_k.

If $|\pi| \geq 10$, from Proposition 1.27 there exists a pattern π' in π (thus in σ) which contains σ_i, σ_j, $\sigma_{i'}$ and $\sigma_{j'}$. Let τ be the permutation obtained from this pattern π' in σ and point σ_k. Then $\pi' < \tau < \sigma$ and $|\tau| = |\pi'| + 1$. Moreover if τ were not simple, as $\pi' = \tau \setminus \{\sigma_k\}$ is simple, from Proposition 1.23 σ_k belongs to an interval I of size 2 of τ. Indeed σ_k is not in a corner of the diagram of τ because $i < k < j$ or $\sigma_i < \sigma_k < \sigma_j$. Set $I = \{\sigma_\ell, \sigma_k\}$, then $\ell = i$ or $\ell = j$, excluded because $\sigma_{i'}$ separates σ_i from σ_k and $\sigma_{j'}$ separate σ_j from σ_k. Thus τ is simple and we have the result expected.

As π' is exceptional, $|\pi'| \geq 4$ so it remains only to prove the result when $|\pi| = 6$ or $|\pi| = 8$. We can prove by exhaustive verification that in this case there also exists a set P of points of π building a pattern π' such that $\tau = P \cup \sigma_k$ fulfills our proposition. ∎

Theorem 1.29. *Let $\pi < \sigma$ be two simple permutations with σ non-exceptional and $\ell = |\sigma| - |\pi|$. Then there exists a chain of simple permutations $\sigma^{(0)} = \sigma, \sigma^{(1)}, \ldots, \sigma^{(\ell-1)}, \sigma^{(\ell)} = \pi$ such that for all $i \in [1..\ell]$, $\sigma^{(i)} < \sigma^{(i-1)}$ and $|\sigma^{(i-1)}| - |\sigma^{(i)}| = 1$.*

Proof. Let $(\sigma^{(i)})_{0 \leq i \leq k}$ be a chain of simple permutations given by Theorem 1.26 with m maximum. If $m < k$ then $\sigma^{(m)}$ and $\sigma^{(m+1)}$ are exceptional, and $\sigma^{(m-1)}$ is not exceptional ($m > 0$ because σ is not exceptional). We can apply Proposition 1.28, and we have a simple permutation τ such that $\sigma^{(m+1)} \leq \tau \leq \sigma^{(m-1)}$ and $|\tau| = |\sigma^{(m+1)}| + 1$. By Proposition 1.24, we have a simple permutation ρ such that $\tau \leq \rho \leq \sigma^{(m-1)}$ and $|\rho| = |\sigma^{(m-1)}| - 1$. Then we set $\pi^{(i)} = \sigma^{(i)}$ if $1 \leq i \leq m-1$, $\pi^{(m)} = \rho$, $\pi^{(m+1)} = \tau$ and $\pi^{(i)} = \sigma^{(i-1)}$ if $m+2 \leq i \leq k+1$. Then we have $|\pi^{(i-1)}| - |\pi^{(i)}| = 1$ if $1 \leq i \leq m+1$, which give us a chain of simple permutations verifying conditions of Theorem 1.26 so m is not maximum, a contradiction. Thus $m = k$ and we have the result expected. ∎

1.4.2 Degree of vertices in the poset

The preceding section proves that if we omit exceptional permutations the poset is ranked, meaning that each level of the poset corresponds to simple permutations of given size and there exist only edges between permutations of contiguous levels. In this section, we study the possible edges between two contiguous levels, which provide statistics on simple permutations. More precisely, Proposition 1.22 states that if σ is a non-exceptional simple permutation, then there exists a simple permutation σ' of size $|\sigma| - 1$ such that $\sigma' < \sigma$. In other words, there exists a point σ_i of σ such that the permutation obtained when deleting σ_i and renormalizing is simple. But how many points of σ have this property? To answer this question, we look at the multigraph where vertices are simple permutations and there are as many edges between σ and σ' as the number of possible ways to insert an element in σ' to obtain σ. Proving that this indeed is a graph (instead of a multigraph) shows that our question is equivalent to counting the number of edges between two consecutive levels in the original poset.

Proposition 1.30. *Let $\sigma < \tau$ be simple permutations such that $|\tau| = |\sigma| + 1$. Then there exists only one way to obtain τ by adding a point in σ.*

Proof. Suppose that there exist at least 2 ways to do it. Then with $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n$ there are integers $a \neq b$ and $i < k$ such that:
$\tau = \sigma'_1 \ldots \sigma'_{i-1}\ a\ \sigma'_i \ldots \sigma'_{k-1}\ \sigma'_k \ldots \sigma'_n$ and
$\tau = \sigma''_1 \ldots \sigma''_{i-1}\ \sigma''_i \ldots \sigma''_{k-1}\ b\ \sigma''_k \ldots \sigma''_n$
where $\sigma'_j = \begin{cases} \sigma_j & \text{if } \sigma_j < a \\ \sigma_j + 1 & \text{otherwise} \end{cases}$ and $\sigma''_j = \begin{cases} \sigma_j & \text{if } \sigma_j < b \\ \sigma_j + 1 & \text{otherwise.} \end{cases}$

In particular the equality between these two ways to write τ implies that if $i < k-1$, then $\sigma'_i = \tau_{i+1} = \sigma''_{i+1}$, but $\sigma_i \neq \sigma_{i+1}$ thus $|\sigma_i - \sigma_{i+1}| = 1$ which is impossible because

σ is simple. Therefore $i = k - 1$, but then the equality implies that $a = \tau_i = \sigma''_i$ and $b = \tau_{i+1} = \sigma'_i$, so $\{a, b\} = \{\sigma_i, \sigma_i{+}1\}$, which is impossible because τ is simple. Consequently there is only one way to write τ from σ. ■

Recalling that G is the graph representing the poset of simple permutations defined at the beginning of Section 1.4, we consider the graph G_1 obtained from G by deleting edges between exceptional permutations. Notice that there is an edge in G_1 from a simple permutation σ to a simple permutation π if and only if we can obtain π from σ by deleting one point. Then from Proposition 1.30, the number of points we can delete (resp. insert) in σ in order to obtain a simple permutation is the outdegree (resp. indegree) of σ in G_1.

Definition 1.31. Let π be a simple permutation. We denote by $S_{\pi+}$ (resp. $S_{\pi-}$) the set of parents (resp. children) of π in G_1:
$S_{\pi+} = \{\sigma \mid \sigma \text{ is simple, } \pi < \sigma \text{ and } |\sigma| = |\pi| + 1\}$ and
$S_{\pi-} = \{\sigma \mid \sigma \text{ is simple, } \sigma < \pi \text{ and } |\sigma| = |\pi| - 1\}$

Proposition 1.32. *Let π be a simple permutation of size n, then $|S_{\pi+}| = (n + 1)(n - 3)$. Moreover we obtain these $(n+1)(n-3)$ simple permutations from π by adding a point of the grid of the diagram of π which is neither a corner of the diagram nor of a cell containing a point of π.*

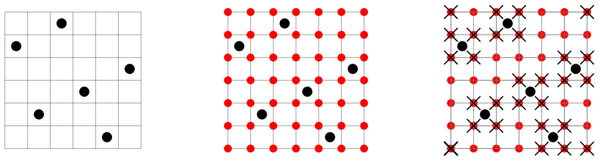


Figure 1.9: $(n + 1)^2$ ways to add a point in a simple permutation of size n, among them $4n$ lead to a permutation with an interval of size 2 and 4 lead to a permutation with an interval of size n.

Proof. Permutations of $S_{\pi+}$ are simple permutations obtained from π by adding one point. There are $(n+1)^2$ ways to insert a point in π (giving permutations not necessarily different): if we consider the diagram of π in a grid, adding one point to π corresponds to choosing one point in the grid, which is of size $(n + 1)^2$. But we want only simple permutations, which excludes $4(n + 1)$ points in the grid: for one given point π_i of π, we cannot take any of the 4 corners of the cell where it is, this excludes $4n$ points, which are all different because π is simple so there are no points in contiguous cells. And we cannot take any of the 4 corners of the grid, and these 4 points have not been excluded yet because π is simple so there is no point in a corner. There are $4(n+1)$ points excluded among $(n+1)^2$ possibilities and Proposition 1.23 ensures that they are the only points to exclude, so we have $(n + 1)(n - 3)$ points left which give simple permutations. We have now to ensure that we cannot have the same simple permutation from two different points, which is given by Proposition 1.30. ■

So $|S_{\pi+}|$, which is the indegree of π in the graph G_1, is independent of π. We are now interested in $|S_{\pi-}|$, the outdegree of π in G_1. We know that it depends on π, and especially that $|S_{\pi-}| = 0$ if and only if π is exceptional. We know also that $|S_{\pi-}| \leq |\pi|$. We consider the average outdegree in G_1.

Proposition 1.33. *Let D_n be the average outdegree of simple permutations of size n in G_1. Then $D_n = n - 4 - \frac{4}{n} + O(\frac{1}{n^2})$.*

Proof. In G_1, there is an edge from a simple permutation σ to a simple permutation π if and only if we can obtain π from σ by deleting one point. So edges that come from permutations of size n are those which go to permutations of size $n-1$. Let s_n be the number of simple permutations of size n. From Proposition 1.32 we know that there are $s_{n-1} \times n(n-4)$ such edges. So $D_n = \dfrac{s_{n-1} \times n(n-4)}{s_n}$. But from Theorem 5 of [AAK03] (Theorem 0.23) we know that $s_n = \dfrac{n!}{e^2}\left(1 - \dfrac{4}{n} + \dfrac{2}{n(n-1)} + O(n^{-3})\right)$ and a straightforward computation allows us to conclude. ■

Thus in general in a simple permutation, almost every point gives another simple permutation when deleting it. We are now interested in the number S_n^k of simple permutations of size n and of outdegree k fixed. For example we know that $S_n^0 = 4$ for every even n and $S_n^0 = 0$ for every odd n (number of exceptional permutations). Figure 1.10 shows the percentage of simple permutations which have outdegree k. Each plot shows this distribution for a given size of permutations as indicated in the caption. Notice that Figure 1.10 also illustrates the result of Proposition 1.33.

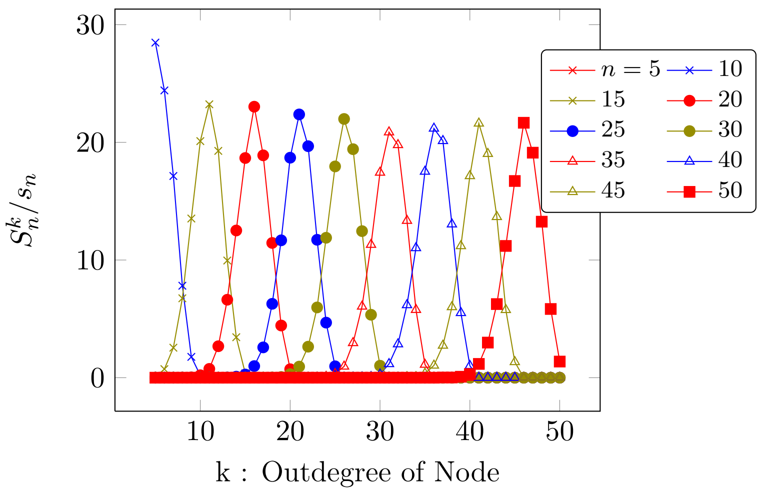


Figure 1.10: Proportion S_n^k/s_n (in percentage) of simple permutations with outdegree k in G_1 among simple permutations of size n

Proposition 1.34. *Let $S_n^k = |\{\pi : |S_{\pi-}| = k\}|$ be the number of simple permutations of size n and of outdegree k in G_1. Then for every fixed k, the proportion $\dfrac{S_n^k}{s_n}$ of simple permutations of outdegree k among simple permutations of size n tends to zero when n tends to infinity.*

Proof. By definition $s_n \times D_n = \sum_{i=0}^n i \times S_n^i$. Suppose that there exists k such that $\dfrac{S_n^k}{s_n}$ does not tend to zero, then there exists $\epsilon > 0$ such that $\forall n_0$, $\exists n \geq n_0$ such that $\dfrac{S_n^k}{s_n} > \epsilon$. But then

$$D_n = \sum_{i=0}^{n} i \times \frac{S_n^i}{s_n} = k \times \frac{S_n^k}{s_n} + \sum_{i \neq k, i=0}^{n} i \times \frac{S_n^i}{s_n} \leq k + \sum_{i \neq k, i=0}^{n} n \times \frac{S_n^i}{s_n} = k + n\left(1 - \frac{S_n^k}{s_n}\right) \leq k + n(1-\epsilon)$$

but from Proposition 1.33, for n_0 large enough $D_n \geq n-5$, a contradiction. ■

1.5 Generating simple permutations in a permutation class

1.5.1 An algorithm for substitution-closed classes

Recall that from Proposition 0.41, a permutation class $Av(B)$ is substitution-closed if and only if B contains only simple permutations (and maybe 1, 12 or 21, however when B contains 1, 12 or 21, there is no simple permutation in $Av(B)$, and we exclude this to avoid trivial cases).

Theorem 1.25 ensures that if σ is a non-exceptional simple permutation and $Av(B)$ a substitution-closed class of permutations, then σ does not belong to $Av(B)$ if and only if it is equal to a permutation of B or contains as a pattern a simple permutation of size $|\sigma| - 1$ which does not belong to $Av(B)$. Indeed by definition σ does not belong to $Av(B)$ if and only if it contains a permutation of B as a pattern, and elements of B are simple.

This leads to an efficient iterative algorithm to generate simple permutations in a finitely based substitution-closed class (see Algorithm 1 whose validity is proved in Proposition 1.35).

Algorithm 1: Generating simple permutations in a finitely based substitution-closed class of permutations

```
Data: $B$ a finite set of simple permutations
Result: Si ou **Si** ou ***Si*** ou Si ou $Si$ the set of simple permutations in $Av(B)$
$Si_3 \leftarrow \emptyset, Si_4 \leftarrow \{2413, 3142\} \setminus B$;
$n \leftarrow 5$;
while $Si_{n-1} \neq \emptyset$ or $Si_{n-2} \neq \emptyset$ do
    Si$_n$ ou **Si**$_n$ ou ***Si***$_n$ ou Si$_n$ ou $Si_n \leftarrow \emptyset$;
    for $\pi \in Si_{n-1}$ do
        for each admissible way to insert a point into $\pi$ and obtain a simple permutation $\sigma$ do
            if $\sigma \notin B$ then
                inS = true;
                for each of the $n$ ways to delete a point of $\sigma$ do
                    Compute the obtained permutation $\tau$;
                    if $\tau$ is simple then
                        if $\tau \notin Si_{n-1}$ then
                            inS = false ;
                if inS = true then
                    $Si_n \leftarrow Si_n \bigcup \sigma$
    for $\pi$ exceptional of type $i \in Si_{n-2}$ do
        if $\sigma$ exceptional of type $i$ and of size $n \notin B$ then
            $Si_n \leftarrow Si_n \bigcup \sigma$
    $n \leftarrow n + 1$;
```

Proposition 1.35. *For all $n \geq 3$, the set Si_n computed by Algorithm 1 is the set of simple permutations of size n contained in $Av(B)$.*

Proof. The result holds for $n = 3$ and $n = 4$. For $n \geq 5$, we show it by induction assuming it holds for $n - 1$ and $n - 2$.

We have to prove that every simple permutation σ of size n in $Av(B)$ belongs to Si_n. Let σ be a simple permutation of size n in $Av(B)$. If σ is not exceptional, there exists π simple such that $\pi \leq \sigma$ and $|\pi| = |\sigma| - 1$ (Proposition 1.22). By the induction hypothesis, $\pi \in Si_{n-1}$ so that σ is considered at line 6 of our algorithm. As $\sigma \in Av(B)$, $\sigma \notin B$ and every simple pattern τ of σ of length $n-1$ is in $Av(B)$ and by the induction hypothesis lies in Si_{n-1}. Thus line 15 is reached and σ is added to Si_n. If σ is exceptional, σ is considered at line 17 of our algorithm and is added to Si_n by the induction hypothesis.

Conversely, let us prove that every permutation $\sigma \in Si_n$ is a simple permutation of size n of $Av(B)$. If $\sigma \in Si_n$, notice first that σ is simple and of size n. Suppose now that $\sigma \notin Av(B)$; then there exists $\pi \in B$ (π simple) such that $\pi \leq \sigma$. We have $\sigma \neq \pi$; otherwise $\sigma \in B$, but there is no permutation of B in Si_n (because of lines 7 and 17 of the algorithm). If σ is not exceptional, using Theorem 1.25, we can find τ simple of size $n-1$ such that $\pi \leq \tau \leq \sigma$, thus $\tau \notin Av(B)$ and by the induction hypothesis $\tau \notin Si_{n-1}$. But our algorithm tests every pattern of σ of size $n-1$ in line 9 so σ is not added to Si_n. If σ is exceptional, then $|\pi|$ is even (Proposition 1.21) so $\pi \leq \sigma'$ where σ' is the exceptional permutation of the same type as σ of size $|\sigma| - 2$. By induction hypothesis $\sigma' \notin Si_{n-2}$ so σ is not added to Si_n and we have the result. ■

Proposition 1.36. *Algorithm 1 terminates if and only if $Av(B)$ contains only a finite number of simple permutations. In this case it gives all simple permutations in $Av(B)$.*

Proof. If Algorithm 1 terminates, there exists $n \geq 5$ such that $Av(B)$ contains no simple permutation of size $n-1$ or $n-2$. Suppose that $Av(B)$ contains a simple permutation σ of size $k \geq n$, then from Proposition 1.19 and Proposition 1.20 σ has a simple pattern of size $n-1$ or $n-2$ in $Av(B)$, a contradiction. So $Av(B)$ contains no simple permutation of size greater than $n-2$ and Proposition 1.35 ensures that the algorithm gives all simple permutations in $Av(B)$.

Conversely if $Av(B)$ contains only a finite number of simple permutations, let k be the size of the greater simple permutation in $Av(B)$. From Proposition 1.35, the algorithm computes $Si_{k+1} = Si_{k+2} = \emptyset$ and the algorithm terminates. ■

Before running Algorithm 1 we can test whether $Av(B)$ contains a finite number of simple permutations in time $\mathcal{O}(n \log n)$ where $n = \sum_{\pi \in B} |\pi|$ thanks to the algorithm given in the next chapter. If $Av(B)$ contains a finite number of simple permutations, Algorithm 1 gives all simple permutations in $Av(B)$. If $Av(B)$ contains an infinite number of simple permutations, we can use a modified version of the algorithm to obtain all simple permutations in $Av(B)$ of size less than a fixed integer k: it is sufficient to replace in the algorithm "while $Si_{n-1} \neq \emptyset$ or $Si_{n-2} \neq \emptyset$" by "for $n \leq k$". Notice that this modified version of the algorithm can be used even if the basis B is infinite to obtain all simple permutations in $Av(B)$ up to size k, by taking in input the subset of permutations of size at most k of B, which is necessarily finite.

Let us now evaluate the complexity of our algorithm.

Proposition 1.37. *The complexity of Algorithm 1 is $\mathcal{O}\big(\sum_{n=5}^{k+1} n^4 |Si_{n-1}|\big)$ where k is the size of the longest simple permutation in $Av(B)$.*

Proof. First, we encode every set of permutations as a trie, allowing a linear algorithm to check if a permutation is in the set. The *while* loop beginning at line 3 is done for n from 5 to $k+2$. The inner loop beginning at line 5 is repeated $|Si_{n-1}|$ times (with $|Si_{k+1}| = 0$). The loop of line 6 is repeated $n(n-4)$ times (see Proposition 1.32; notice that we don't have to check whether σ is simple because we know where to insert the points to obtain a simple permutation). This loop performs the following tests:

- Compute $\sigma \to \mathcal{O}(n)$
- Test whether σ is in $B \to \mathcal{O}(n)$ using tries.
- Loop at line 9 is performed n times and performs each time the following operations:
  - Compute $\tau \to \mathcal{O}(n)$
  - Test whether τ is simple $\to \mathcal{O}(n)$ (consequence of Theorem 0.35)
  - Test whether $\tau \in Si_{n-1} \to \mathcal{O}(n)$ using tries.
- Add if necessary σ into $Si_n \to \mathcal{O}(n)$ as we use tries.

Thus the inner part of loop in line 3 has a complexity of order $(n(n-4)(n+n+n(n+n+n)+n))$ that is $\mathcal{O}(n^4)$ leading to the claimed result. Indeed, the loop for the exceptional case is easy to implement in $\mathcal{O}(n)$ time as there are at most 4 exceptional permutations of a given size. ■

1.5.2 An algorithm for not substitution-closed classes

Algorithm 1 cannot be applied directly to general permutation classes. However we can adapt it by replacing the part from line 7 to line 15 by testing if $\sigma \in Av(B)$ and in that case adding it to Si_n, leading to Algorithm 2 whose validity is proved by adapting the proofs of the previous section.

Algorithm 2: Generating simple permutations in a finitely based permutation class

```
Data: $B$ a finite set of permutations
Result: $Si$ the set of simple permutations in $Av(B)$
$Si_3 \leftarrow \emptyset, Si_4 \leftarrow \{2413, 3142\} \setminus B$;
$n \leftarrow 5$;
while $Si_{n-1} \neq \emptyset$ or $Si_{n-2} \neq \emptyset$ do
    $Si_n \leftarrow \emptyset$;
    for $\pi \in Si_{n-1}$ do
        for each admissible way to insert a point into $\pi$ and obtain a simple permutation $\sigma$ do
            if $\sigma \in Av(B)$ then
                $Si_n \leftarrow Si_n \bigcup \sigma$
    for $\pi$ exceptional of type $i \in Si_{n-2}$ do
        if $\sigma$ exceptional of type $i$ and of size $n \in Av(B)$ then
            $Si_n \leftarrow Si_n \bigcup \sigma$
    $n \leftarrow n + 1$;
```

Thus we have the following proposition, adapting the proof of Proposition 1.37:

Proposition 1.38. *For every class $Av(B)$ containing a finite number of simple permutations, Algorithm 2 computes the simple permutations in $Av(B)$ in time $\sum_{n=5}^{k+1} |Si_{n-1}| n^2 f_{Av(B)}(n)$ where k is the size of the longest simple permutation in $Av(B)$ and $f_{Av(B)}(n)$ the complexity of testing if a permutation of size n belongs to $Av(B)$.*

Unfortunately, in general there is no efficient algorithm to test if a permutation is in a class $Av(B)$. The only known general algorithm is to test if σ avoids every permutation in B. Then in the preceding proposition the function $f_{Av(B)}(n)$ is bounded by $\sum_{\tau \in B} n^{|\tau|+1}$. In some particular cases, this test can be highly improved, see for example [AAAH01, BRV07]. Finding an efficient algorithm in the general case is a current research topic.Note that in our algorithm we test the membership of $Av(B)$ only for permutations obtained from a permutation of $Av(B)$ by inserting a point. In this case the complexity is $\mathcal{O} \sum_{\tau \in B} n^{|\tau|}$ since the point added has to belong to the pattern.

In any case, putting these results together we are now able to state our main result about computing simple permutations in a permutation class:

Theorem 1.39. *For every class $Av(B)$ containing a finite number of simple permutations and given by its basis, Algorithm 2 computes the set Si_B of simple permutations in $Av(B)$ in time $\mathcal{O}\big(|B|.|Si_B|.k^{p+2}\big)$, with $p = \max\{|\tau| : \tau \in B\}$ and $k = \max\{|\pi| : \pi \in Si_B\}$.*

Moreover if $Av(B)$ is substitution-closed, this can be done in time $\mathcal{O}\big(|Si_B|.k^4\big)$ using Algorithm 1.

Indeed when a class contains a finite number of simple permutations, we know that its basis is finite. Thus the need of a finite basis is not a restriction : the only restriction is the finiteness of the number of simple permutations.

It is then interesting to be able to test algorithmically whether a permutation class contains a finite number of simple permutations. This is the topic of the next chapter.

Chapter 2

Finitely many simple permutations?

In this chapter, we describe an algorithm to determine whether a permutation class $\mathcal{C}$ given by a finite basis B of excluded patterns contains a finite number of simple permutations. This is a continuation of the work initiated in [BRV08] (Brignall, Ruškuc, Vatter, *Simple permutations: decidability and unavoidable substructures*, 2008), and shares several aspects with it. As in this article, the main difficulty is to decide whether $\mathcal{C}$ contains a finite number of proper pin-permutations, and this decision problem is solved using automata theory. Moreover, we use an encoding of proper pin-permutations by words over a finite alphabet, introduced by Brignall *et al.* However, unlike in their article, our construction of automata is fully algorithmic and efficient. It is based on the study of pin-permutations in [BBR11] (Bassino, Bouvel, Rossin, *Enumeration of pin-permutations*, 2011). The complexity of the overall algorithm is $\mathcal{O}(n \log n + s^{2k})$ where n denotes the sum of the sizes of permutations in the basis B, s is the maximal size of a pin-permutation in B and k is the number of pin-permutations in B.

2.1 Introduction

The work reported here follows the line opened by [AA05] and continued by [BRV08]. In [AA05], the main theorem provides (in particular) a sufficient condition for a permutation class $\mathcal{C}$ to have an algebraic generating function: namely, that $\mathcal{C}$ contains a finite number of simple permutations.

Then in [BRV08], Brignall, Ruškuc and Vatter introduce new objects (most importantly, pin-permutations) to provide a criterion deciding whether a permutation class contains a finite number of simple permutations. To this criterion, [BRV08] associates a decision procedure testing from a finite basis B whether $\mathcal{C} = Av(B)$ contains a finite number of simple permutations. Both in the criterion and in the procedure, the set of *proper pin-permutations* introduced in [BRV08] plays a crucial part. The procedure is based on the construction of automata that accept languages of words on a finite alphabet (that are called *pin words*) that encode such permutations that do not belong to the class. This procedure is however not fully algorithmic, and its complexity is a double exponential, as we explain in Subsection 2.2.3.

Our goal is to solve the decision problem of [BRV08] with an actual algorithm, whose complexity should be kept as low as possible. For this purpose, we heavily rely on [BBR11] in which is performed a detailed study of the class of *pin-permutations*, which contains the proper pin-permutations of [BRV08]. These results allow us to precisely characterize the set of pin words corresponding to any given pin-permutation, and subsequently to modify the automata construction of [BRV08], leading to our algorithm deciding whether a permutation class given by a finite basis B contains a finite number of simple permutations. The resulting algorithm is efficient: it is polynomial w.r.t. the sizes of the patterns in B and simply exponential w.r.t. their number, which is a significant improvement to the first decidability procedure of [BRV08]. More precisely we give a $\mathcal{O}(n \log n + s^{4k})$ algorithm to decide if a finitely based permutation class $Av(\pi_1, \pi_2, \ldots, \pi_k)$ contains a finite number of simple permutations where $n = \sum |\pi_i|$ and $s \leq \max |\pi_i|$, and we describe a variant of our algorithm, whose complexity is $\mathcal{O}(n \log n + s^{2k})$. We also give an algorithm solving the same problem on substitution-closed permutation classes (equivalently, classes of permutations whose bases contain only simple permutations). The complexity of our algorithm in this special framework is $\mathcal{O}(n \log n)$ where $n = \sum |\pi_i|$.

The chapter is organized as follows. Section 2.2 is a reminder of previous definitions and results: pin-permutations and their pin words, brief description of the characterization and the procedure of [BRV08], and special families of pin-permutations playing an important role in our work. Section 2.3 recalls and refines the interpretation of the pattern containment relation between pin-permutations on their pin words. The main result of Section 2.3 is Theorem 2.27, which is the basis of the automata construction of Section 2.5. Section 2.4 focuses on the language of pin words encoding a given pin-permutation. Theorems 2.48, 2.53, 2.60, 2.66 and 2.72 describe, for any pin-permutation π, the language $P(\pi)$ of pin words of π. These languages are described recursively following the recursive characterization of the decomposition trees of pin-permutations obtained in [BBR11], and refining the ideas used in its proof. Then, based on these results and following the same recursive approach, Section 2.5 describes a recursive algorithm that builds, for any pin-permutation π, an automaton accepting the language $\mathcal{L}_\pi$ of pin words that encode proper pin-permutations containing π as a pattern (or rather, for technical reasons that will be explained later, a slight modification of this language). By Theorem 2.27, checking whether $\mathcal{C} = Av(B)$ contains finitely many proper pin-permutations is equivalent to checking whether the complement set of $\cup_{\pi \in B} \mathcal{L}_\pi$ is finite. Finally, Section 2.6 explains how to decide this question using the automata of Section 2.5, and combines this procedure with existing algorithms from [AAAH01, BBPR10, BCMR08, BRV08]: this provides an algorithm of reasonable complexity to decide, given a finite basis B, whether the class $\mathcal{C} = Av(B)$ contains a finite number of simple permutations. We also give a $\mathcal{O}(n \log n)$ algorithm solving the same problem on substitution-closed permutation classes.

2.2 Preliminaries

Our goal is to check whether a permutation class contains a finite number of simple permutations, ensuring in this way that its generating function is algebraic [AA05]. In this chapter, permutation classes are given by their bases. We are further interested in classes having finite bases, since otherwise, from [AA05], they contain infinitely many simple permutations. As we shall see in the following, a class of particular permutations, called the pin-permutations, plays a central role in the decision procedure of this problem. For this reason, we record in this section a few definitions and results related with these pin-permutations. We also recall the main steps of the decision procedure of [BRV08]. More details can be found in [BBR11, BHV08b, BRV08].

2.2.1 Pin-permutations and pin representations

A *pin* is a point in the plane. A pin p *separates* – horizontally or vertically – the set of pins P from the set of pins Q if and only if a horizontal – resp. vertical – line drawn across p separates the plane into two parts, one containing P and the other one containing Q. Recall that the *bounding box* of a set of points P is the smallest axis-parallel rectangle containing the set P. A *pin sequence* is a sequence $(p_1, \ldots, p_k)$ of pins in the plane such that no two points are horizontally or vertically aligned and for all $i \geq 2$, p_i lies outside the bounding box of $\{p_1, \ldots, p_{i-1}\}$ and satisfies one of the following conditions:

- *separation condition*: p_i separates p_{i-1} from $\{p_1, \ldots, p_{i-2}\}$;
- *independence condition*: p_i is independent from $\{p_1, \ldots, p_{i-1}\}$, i.e., it does not separate this set into two non-empty sets.

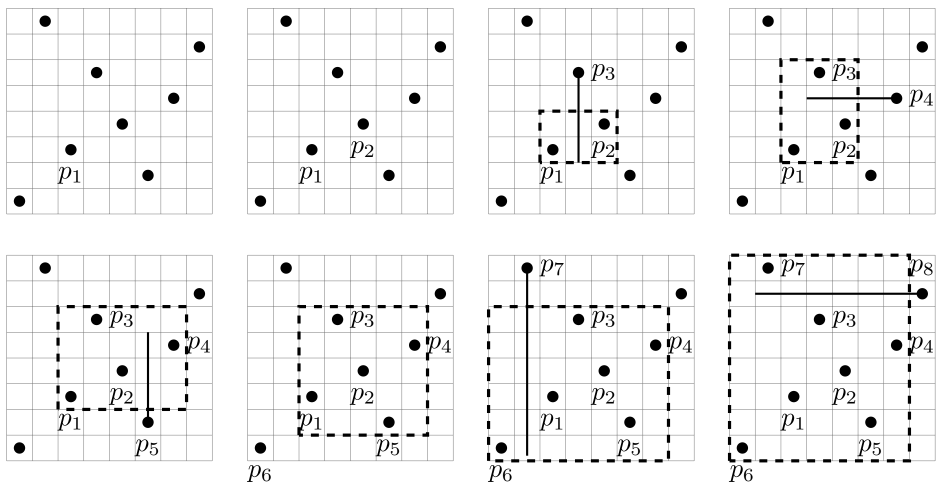


Figure 2.1: A pin representation of 1 8 3 6 4 2 5 7. In this pin representation, each pin satisfies the separation condition, except p_6 which satisfies the independence condition.

A pin sequence represents a permutation σ if and only if it is order isomorphic to its diagram. We say that a permutation σ is a *pin-permutation* if it can be represented by a pin sequence, which is then called a *pin representation* of σ (see Figure 2.1). Not all permutations are pin-permutations (see for example the permutation $\sigma = 4\,7\,2\,6\,3\,1\,5$ of Figure 2.2: we can check that no ordering of its points gives a pin representation).

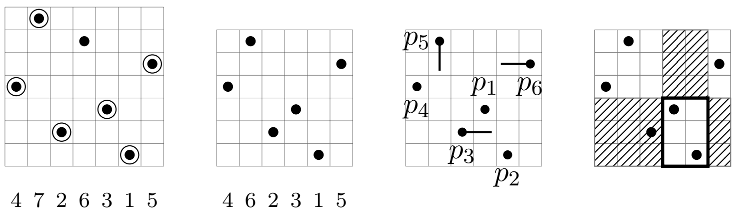


Figure 2.2: The permutation $\sigma = 4\,7\,2\,6\,3\,1\,5$, its pattern $\pi = 4\,6\,2\,3\,1\,5$, a pin representation p of π, and the bounding box of $\{p_1, p_2\}$ with its sides shaded.

Lemma 2.17 of [BBR11] is used several times in our proofs, and we state it here:

Lemma 2.1. *Let $(p_1, \ldots, p_n)$ be a pin representation of $\sigma \in \mathcal{S}_n$. Then for each $i \in \{2, \ldots, n-1\}$, if there exists a point x of σ on the sides of the bounding box of $\{p_1, \ldots, p_i\}$, then it is unique and $x = p_{i+1}$.*

A *proper* pin representation is a pin representation in which every pin p_i, for $i \geq 3$, separates p_{i-1} from $\{p_1, \ldots, p_{i-2}\}$. A *proper* pin-permutation is a permutation that admits a proper pin representation.

The set of pin-permutation is a permutation class, but it is not the case of the set of proper pin-permutations. Indeed by deleting some points in a pin sequence and keeping the remaining points in the same order, we still have a pin sequence, but some separating pins become independent (see Figure 2.3 where the pattern $6\,2\,4\,3\,1\,5$ admits no proper pin representation).

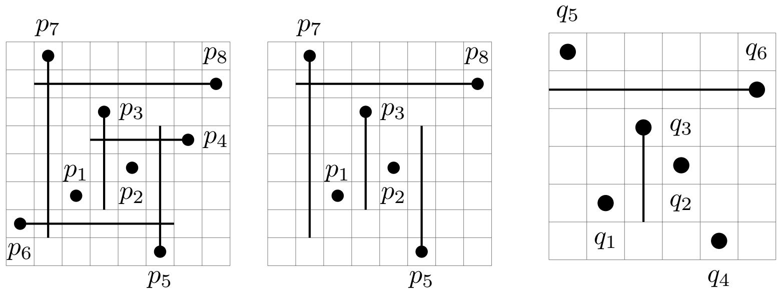


Figure 2.3: A pin representation of the pattern $6\,2\,4\,3\,1\,5$ obtained from a proper pin representation of $2\,8\,3\,6\,4\,1\,5\,7$.

Remark 2.2. A pin representation of a simple pin-permutation is always proper as any independent pin p_i with $i \geq 3$ creates a block corresponding to $\{p_1, \ldots, p_{i-1}\}$.

2.2.2 Pin words

Pin representations can be encoded on the alphabet $\{1, 2, 3, 4, U, D, L, R\}$ by words called *pin words*. Consider a pin representation $(p_1, \ldots, p_n)$ and choose an origin p_0 in the plane such that $(p_0, p_1, \dots, p_n)$ is a pin sequence. Then every pin $p_1, \dots, p_n$ is encoded by a letter according to the following rules:

- The letter associated with p_i is U – resp. D, L, R – if p_i separates p_{i-1} and $\{p_0, p_1, \dots, p_{i-2}\}$ from the top – resp. bottom, left, right.
- The letter associated with p_i is 1 – resp. 2, 3, 4 – if p_i is independent from $\{p_0, p_1, \dots, p_{i-1}\}$ and is located in the up-right – resp. up-left, bottom-left, bottom-right – corner of the bounding box of $\{p_0, p_1, \dots, p_{i-1}\}$.

This encoding is summarized by Figure 2.4. The region encoded by 1 is called the first *quadrant* with respect to the box ■. The same goes for $2, 3, 4$. The letters U, D, L, R are called *directions*, while $1, 2, 3$ and 4 are *numerals*.

2	U	1
L	■	R
3	D	4

Figure 2.4: Encoding of pins by letters.

Example 2.3. $14L2UR$ (if p_0 is between p_3 and p_1) and $3DL2UR$ (if p_0 is horizontally between p_1 and p_4 and vertically between p_2 and p_6) are pin words corresponding to the pin representation of $\pi = 4\,6\,2\,3\,1\,5$ shown in Figure 2.2.

Example 2.3 shows in particular that several pin words encode the same pin representation, depending on the choice of the origin p_0. We may actually describe the number of these pin words:

Remark 2.4. Because of the choice of the origin p_0, each pin-permutation of size greater than 1 has at least 6 pin words. More precisely each pin representation p is encoded by 6 pin words if p_3 is a separating pin and 8 pin words otherwise (see Figure 2.5). Indeed, once a pin representation p is fixed, the letters encoding p_i for $i \geq 3$ in a pin word encoding p are uniquely determined.

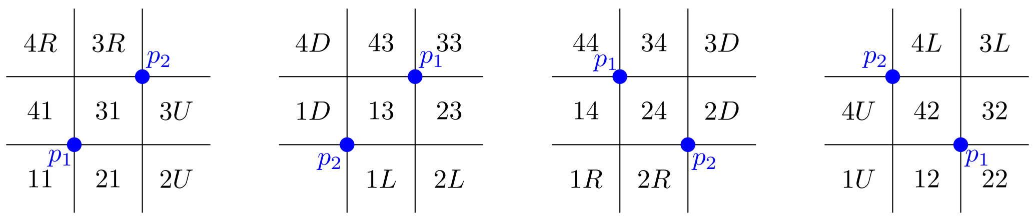


Figure 2.5: The two letters in each cell indicate the first two letters of the pin word encoding $(p_1, \dots, p_n)$ when p_0 is taken in this cell.

This remark has a direct algorithmic consequence:

Remark 2.5. Given a pin representation p, we can compute its set of corresponding pin words in linear time w.r.t. $|p|$.

Conversely, pin words indeed encode pin-permutations since to each pin word corresponds a unique pin representation, hence a unique permutation.

From the definition of pin words we can deduce the following remarks that will be useful in the following:

Remark 2.6. The definition of pin sequences implies that pin words do not contain any of the factors $UU, UD, DU, DD, LL, LR, RL$ and RR.

Remark 2.7. There are at most 8^n and at least 6^n pin words of size n. Indeed pin words are words on a 8 letter alphabet, and taking the word from the beginning, for each letter we have 8 or 6 choices depending whether the preceding letter is a direction.

Finally we end this section about pin words by defining strict and quasi-strict pin words and by discussing their link with proper pin representations:

A *strict* (resp. *quasi-strict*) pin word is a pin word that begins with a numeral (resp. two numerals) followed only by directions.

Remark 2.8 (Proper pin representations, strict and quasi-strict pin words)**.** Every pin word encoding a proper pin representation is either strict or quasi-strict. Conversely if a pin word is strict or quasi-strict, then the pin representation it encodes is proper. Finally a pin-permutation is proper if and only if it admits a strict pin word.

2.2.3 Decidability procedure

In [BRV08], Brignall *et al.* study conditions for a class to contain a finite number of simple permutations. Introducing three new kinds of permutations they show that this problem is equivalent to ensuring that the class contains a finite number of permutations of each of these three simpler kinds.

Theorem 2.9. *[BRV08] A permutation class $Av(B)$ contains a finite number of simple permutations if and only if it contains:*

- *a finite number of wedge simple permutations, and*
- *a finite number of parallel alternations, and*
- *a finite number of proper pin-permutations.*

In Theorem 2.9 above, the proper pin sequences of [BRV08] have been replaced by proper pin-permutations, which is equivalent. Indeed, containing a finite number of proper pin-permutations is equivalent to containing a finite number of proper pin sequences, since the encoding of proper pin-permutations by proper pin sequences provides a finite-to-one correspondence. More precisely, each proper pin sequence corresponds to a unique proper pin-permutation. Conversely every proper pin-permutation of size n is associated with at least one and very loosely at most 8^n pin sequences, since pin sequences are encoded by pin words on an 8-letter alphabet.

The definition of the wedge simple permutations and the parallel alternations are not crucial to our work. Let us only mention that parallel alternations are defined in the previous chapter and that wedge simple permutations are obtained from wedge alternations by adding an extremal point which split their interval of size 2. (see Figure 1.3 p.37). Wedge simple permutations can be of type 1 or 2 since there are two different ways to add a single point to a wedge alternation to form a simple permutation. We refer the reader to [BRV08] for more details. What is however important for our purpose is to be able to test whether a class given by a finite basis contains a finite number of parallel alternations and wedge simple permutations.

Parallel alternations and wedge simple permutations are well characterized in [BRV08]. This characterization leads to the following lemmas:

Lemma 2.10. *[BRV08] The permutation class $Av(B)$ contains only finitely many parallel alternations if and only if B contains an element of every symmetry of the class $Av(123, 2413, 3412)$.*

Lemma 2.11. *[BRV08] The permutation class $Av(B)$ contains only finitely many wedge simple permutations of type 1 if and only if B contains an element of every symmetry of the class $Av(1243, 1324, 1423, 1432, 2431, 3124, 4123, 4132, 4231, 4312)$.*

Lemma 2.12. *[BRV08] The permutation class $Av(B)$ contains only finitely many wedge simple permutations of type 2 if and only if B contains an element of every symmetry of the class $Av(2134, 2143, 3124, 3142, 3241, 3412, 4123, 4132, 4231, 4312)$.*

Using these lemmas together with a result of [AAAH01] we have:

Lemma 2.13. *Testing whether a finitely based class $Av(B)$ contains a finite number of wedge simple permutations and parallel alternations can be done in $\mathcal{O}(n \log n)$ time, where $n = \sum_{\pi \in B} |\pi|$.*

Proof. From Lemmas 2.10 to 2.12, deciding if $Av(B)$ contains a infinite number of wedge simple permutations and parallel alternations is equivalent to checking if elements of its basis B involve patterns of size at most 4. From [AAAH01] checking whether a permutation π involves a fixed set of patterns of length at most 4 can be done in $\mathcal{O}(|\pi| \log |\pi|)$. As we have to check for each permutation of B the involvement of fixed sets of permutations of size at most 4, this leads to a $\mathcal{O}(n \log n)$ algorithm for deciding whether the number of parallel alternations and of wedge simple permutations in the class is finite. ■

In [BRV08] Brignall *et al.* also proved that it is decidable to know if $\mathcal{C} = Av(B)$ contains a finite number of proper pin-permutations using language-theoretic arguments. More precisely they define an order relation $\preccurlyeq$ on pin words, and denoting by $\mathcal{SP}$ the set of all strict pin words, and by $P(B)$ the set of pin words encoding a permutation of B, they prove that $\mathcal{C}$ contains a finite number of proper pin-permutations if and only if the set $\mathcal{L}$ is finite, where

$$\mathcal{L} = \mathcal{SP} \setminus \bigcup_{u \in P(B)} \{w \mid u \preccurlyeq w\}$$

.

They also notice that using automata theory (see [HU79] among other references), given a finite automaton recognizing $\mathcal{L}$, it can be decided whether $\mathcal{L}$ is finite.

Then given a pin word u, they explain how to build a finite automaton $\mathcal{A}^{(u)}$ recognizing a language $\mathcal{L}^{(u)}$ such that $\mathcal{SP} \cap \mathcal{L}^{(u)} = \mathcal{SP} \cap \{w \mid u \preccurlyeq w\}$, and how to build a finite automaton recognizing $\mathcal{SP}$, and they conclude by referring to automata theory.

The proof of the decidability in [BRV08] is constructive and establishes that this can be done algorithmically. However the authors don't give an actual algorithm since many steps are not given explicitly. In particular, the computation of the set $P(B)$ is not addressed. Moreover, they don't bother about the complexity of their procedure.

Analyzing the procedure of [BRV08], we can prove that it has a doubly exponential complexity due to the resolution of a co-finiteness problem for a regular language given by a non-deterministic automaton.

More precisely, if we turn into an actual algorithm the procedure of [BRV08] deciding whether $\mathcal{C} = Av(B)$ contains a finite number of proper pin-permutations, the main steps are:

1. Compute the set $P(B)$ of pin words encoding permutations of B.
2. For each $u \in P(B)$, build the automaton $\mathcal{A}^{(u)}$ recognizing $\mathcal{L}^{(u)}$.
3. Build an automaton $\mathcal{A}$ recognizing $\mathcal{SP} \setminus \bigcup_{u \in P(B)} \mathcal{L}^{(u)}$.
4. Test whether $\mathcal{A}$ contains a cycle that can be reached from an initial state and can lead to a final state.

As already explained, the first step is not adressed in [BRV08]. However there is a naive way to do it: for each permutation π in B, test each ordering of the points of π and check whether it gives a pin representation of π. In this case, compute the set of pin words encoding this pin representation. But this naive procedure leads to a complexity of $\mathcal{O}(\sum_{\pi \in B} |\pi|!)$ which is a very high complexity since $n!$ is roughly n^n.

A more clever way to compute $P(B)$ would be: for each permutation π in B, for each pin word u of size $|\pi|$, check if the permutation encoded by u is π. Recalling that there are at most 8^n pin words of size n (and at least 6^n of them, see Remark 2.7), this yield a complexity of $\mathcal{O}(\sum_{\pi \in B} |\pi| \cdot 8^{|\pi|})$ which is $\mathcal{O}(n \cdot 8^m)$ where $n = \sum_{\pi \in B} |\pi|$ and $m = \max\{|\pi| : \pi \in B\}$. This complexity is better but still exponential. In our work we explain how to replace this step by a step of complexity $\mathcal{O}(n)$.

Regarding the second step, it is explained in [BRV08] and produced non-deterministic automata $\mathcal{A}^{(u)}$ whose number of state is $\mathcal{O}(|u|)$, and is indeed at least $|u|$.

Concerning the third step, [BRV08] only refers to automata theory and doesn't give any detail. Without any further assumptions, the more effective way to do this step would be to build by juxtaposition an intermediate automaton $\mathcal{A}^U$ recognizing $\bigcup_{u \in P(B)} \mathcal{L}^{(u)}$, to determinize this automaton in order to complement it, and then to compute the intersection with an automaton recognizing $\mathcal{SP}$. But the determinization of an automaton with q states has a complexity $\mathcal{O}(2^q)$ and there are some automata for which this bound is reached. Since the number of states of $\mathcal{A}^U$ is $\mathcal{O}(\sum_{u \in P(B)} |u|)$, and is indeed at least $\sum_{u \in P(B)} |u|$, this leads to a complexity of $\mathcal{O}(2^{\sum_{u \in P(B)} |u|})$ which may be reached for this third step.

Denoting k the number of pin-permutations in B and s the maximal size of a pin-permutation of B, $\sum_{u \in P(B)} |u| \leq k \cdot 8^s \cdot s$. This bound is maybe not tight. However we can notice that the number of pin words encoding the identity of size s is at least 2^s, since any word on the alphabet $\{1, 3\}$ is a pin word of the identity. Finally the complexity of this third step is of order at least $\mathcal{O}(2^{k \cdot s \cdot 2^s})$, which is doubly exponential w.r.t. s.

The goal of the present work is to give an actual algorithm of complexity as low as possible to determine whether a permutation class $\mathcal{C}$ given by a finite basis B contains a finite number of simple permutations. To do so, we complete and modify the procedure of [BRV08] and obtain an algorithm whose total complexity is $\mathcal{O}(n \log n + s^{2k})$, where as above n denotes the sum of the sizes of permutations in the basis B, s is the maximal size of a pin-permutation in B and k is the number of pin-permutations in B.

As in [BRV08], the main difficulty is to decide whether $\mathcal{C}$ contains a finite number of proper pin-permutations, and this decision problem is solved using automata theory, by first computing a description of the set $P(B)$ and then building an automaton depending on $P(B)$ in which we look for a cycle. Thanks to a detailled study of pin-permutations in [BBR11] and in Section 2.4, the decription of $P(B)$ we need can be computed in time $\mathcal{O}(n)$ and then the automaton built in time $\mathcal{O}(s^{2k})$, which is to be compared with the respective complexities $\mathcal{O}(n8^m)$ and $\mathcal{O}(2^{k\cdot s\cdot 2^s})$ in the procedure of [BRV08]. Writing that $\mathcal{O}(s^{2k}) = \mathcal{O}(2^{k\cdot 2\log s})$ enables us to measure the complexity improvement w.r.t. $\mathcal{O}(2^{k\cdot s\cdot 2^s})$: this improvement is doubly exponential w.r.t. s.

Indeed we obtain an algorithm of total complexity $\mathcal{O}(n\log n + s^{2k})$, which is reasonable to use in practice, contrary to a doubly exponential algorithm.

2.3 Characterization of classes with finitely many proper pin-permutations

First, following [BRV08], we interpret the pattern containment relation on pin-permutations by a piecewise factor relation between their pin words. This interpretation then leads to the characterization of the relation $\pi \leq \sigma$, for σ a pin-permutation, by a set inclusion of the form $\mathcal{L}_\sigma \subseteq \mathcal{L}_\pi$ for languages we introduce in the following. Subsequently, taking $\pi \in B$, we show how these languages $\mathcal{L}_\pi$ can be used to characterize permutation classes $Av(B)$ that contain finitely many proper pin-permutations.

2.3.1 Pattern containment and piecewise factor relation

Recall the definition of the partial order $\preccurlyeq$ on pin words introduced in [BRV08].

Definition 2.14. Let u and w be two pin words. We decompose u in terms of its strong numeral-led factors as $u = u^{(1)} \dots u^{(j)}$, *a strong numeral-led factor* being a strict pin word. We then write $u \preccurlyeq w$ if w can be chopped into a sequence of factors $w = v^{(1)}w^{(1)} \dots v^{(j)}w^{(j)}v^{(j+1)}$ such that for all $i \in \{1, \dots, j\}$:

- if $w^{(i)}$ begins with a numeral then $w^{(i)} = u^{(i)}$, and
- if $w^{(i)}$ begins with a direction, then $v^{(i)}$ is nonempty, the first letter of $w^{(i)}$ corresponds to a point lying in the quadrant – w.r.t. the origin of the encoding w – specified by the first letter of $u^{(i)}$, and all other letters in $u^{(i)}$ and $w^{(i)}$ agree.

Example 2.15. The strong numeral-led factor decomposition of $u = 31U42R$ is $u = 3 \cdot 1U \cdot 4 \cdot 2R$. Moreover, $u \preccurlyeq w = 31URDLUR$, because w may be decomposed as $w = \mathbf{3} \cdot \mathbf{1U} \cdot R\mathbf{D} \cdot L\mathbf{UR}$, where the factors $w^{(i)}$ satisfying the conditions of Definition 2.14 are emphasized by bold letters.

This order is closely related to the pattern containment order $\leq$ on permutations.

Lemma 2.16. *[BRV08] If the pin word w encodes the permutation σ and $\pi \leq \sigma$ then there is a pin word u encoding π with $u \preccurlyeq w$. Conversely if $u \preccurlyeq w$ then the permutation corresponding to u is contained in the one corresponding to w.*

Example 2.17. Taking $\sigma = 28364157$ and $\pi = 624315$, then we have $\pi \leq \sigma$ as illustrated in Figure 2.3. Moreover taking the origin p_0 horizontally between p_1 and p_3 and vertically between p_1 and p_2, the word $w = 31URDLUR$ encodes the pin representation $p_1, \dots, p_8$ of σ and the word $u = 31U42R$ encodes the pin representation $q_1, \dots, q_6$ of π given in Figure 2.3, and we have $u \preccurlyeq w$ as explained in Example 2.15.

We may notice that the number of letters of the words $v^{(i)}$ which appear in a decomposition of w satisfying the conditions of Definition 2.14 corresponds to the number of points we have to delete from σ (encoded by w) to obtain π (encoded by u).

In what follows, σ is a proper pin-permutation (recall that our goal is to test whether there are finitely many proper pin-permutations in a class). So we can choose a strict pin word w_σ that encodes σ (see Remark 2.8 p.58). As a consequence of Lemma 2.16, checking whether a permutation π is a pattern of σ is equivalent to checking whether there exists a pin word u corresponding to π with $u \preccurlyeq w_\sigma$.

The relation $u \preccurlyeq w$ on pin words is nearly a piecewise factor relation, the factors being determined by the strong numeral-led factors of u. We use an encoding of pin words that we introduced in [BBPR10] and recall hereafter: it maps the relation $\preccurlyeq$ on an actual piecewise factor relation. As explained above, for our purpose it will be enough to consider in Lemma 2.26 the case where w is a strict pin word.

Denote by $\mathcal{SP}$ the language of strict pin words. Let $\mathcal{SP}_{\geq 2} = \mathcal{SP} \setminus \{1, 2, 3, 4\}$ be the set of strict pin words of length at least 2. Denote by $\mathcal{M}$ (resp. $\mathcal{M}_{\geq 3}$) the set of words of length at least 2 (resp. at least 3) over the alphabet L, R, U, D such that L, R is followed by U, D and conversely. We define below a bijection that sends strict pin words to words of $\mathcal{M}$. It consists of replacing the only numeral in a strict pin word by two directions. Intuitively, given a numeral q and a box $\blacksquare$, inserting two pins in the two directions prescribed by the bijection ends up in a pin lying in quadrant q with respect to the box $\blacksquare$.

Definition 2.18. We define a bijection ϕ from $\mathcal{SP}_{\geq 2}$ to $\mathcal{M}_{\geq 3}$ as follows. For any strict pin word $u \in \mathcal{SP}_{\geq 2}$ such that $u = u'u''$ with $|u'| = 2$, we set $\phi(u) = \varphi(u')u''$ where φ is given by:

$1R \mapsto RUR$	$2R \mapsto LUR$	$3R \mapsto LDR$	$4R \mapsto RDR$
$1L \mapsto RUL$	$2L \mapsto LUL$	$3L \mapsto LDL$	$4L \mapsto RDL$
$1U \mapsto URU$	$2U \mapsto ULU$	$3U \mapsto DLU$	$4U \mapsto DRU$
$1D \mapsto URD$	$2D \mapsto ULD$	$3D \mapsto DLD$	$4D \mapsto DRD$

For any $n \geq 2$, the map ϕ is a bijection from the set $\mathcal{SP}_n$ of strict pin words of length n to the set $\mathcal{M}_{n+1}$ of words of $\mathcal{M}$ of length $n + 1$. Furthermore, it satisfies, for any $u = u_1u_2 \ldots \in \mathcal{SP}_{\geq 2}$, $u_i = \phi(u)_{i+1}$ for any $i \geq 2$.

In the above table, we may notice that, for any $u \in \mathcal{SP}_{\geq 2}$, the first two letters of $\phi(u)$ are sufficient to determine the first letter of u (which is a numeral). Thus it is natural to extend the definition of ϕ to $\mathcal{SP}$ by setting for words of length 1: $\phi(1) = \{UR, RU\}, \phi(2) = \{UL, LU\}, \phi(3) = \{DL, LD\}$ and $\phi(4) = \{RD, DR\}$, and by defining consistently $\phi^{-1}(v) \in \{1, 2, 3, 4\}$ for any v in $\{LU, LD, RU, RD, UL, UR, DL, DR\}$.

Lemma 2.19 below shows that for each pin word w, we know in which quadrant (w.r.t. the origin of the encoding) lies each pin of the pin representation p corresponding to w. More precisely for each $i \leq |w|$, knowing only w_i and w_{i-1}, we can determine in which quadrant p_i lies.

Lemma 2.19. *Let w be a pin word and p be the pin representation corresponding to w. For any $i \geq 2$, set*

$$q(w_{i-1}, w_i) = \begin{cases} w_i \text{ if } w_i \text{ is a numeral} \\ \phi^{-1}(w_{i-1}w_i) \text{ if } w_{i-1} \text{ and } w_i \text{ are directions} \\ \phi^{-1}(Bw_i) \text{ otherwise, with } \phi(w_{i-1}w_i) = ABw_i \end{cases}$$

Then for any $i \geq 2$, $q(w_{i-1}, w_i)$ is a numeral indicating the quadrant in which p_i lies with respect to $\{p_0, \ldots, p_{i-2}\}$.

Notice that in the third case w_{i-1} is a numeral and w_i is a direction; consequently, $ABw_i \in \mathcal{M}_3$ is given by the table of Definition 2.18.

Proof. It is obvious that $q(w_{i-1}, w_i)$ is a numeral. Moreover as $(p_0, \dots, p_{|w|})$ is a pin sequence, p_i lies outside the bounding box of $\{p_0, \dots, p_{i-2}\}$ and it does not lies on its sides because it respects the separation or the independence condition, thus its lies on one corner. The fact that $q(w_{i-1}, w_i)$ indicates the claimed quadrant is proved by case examination.

If w_i is a numeral, then by definition of the encoding, $q(w_{i-1}, w_i) = w_i$ indicates the quadrant in which p_i lies w.r.t. $\{p_0, \dots, p_{i-1}\}$ and a fortiori w.r.t. $\{p_0, \dots, p_{i-2}\}$.

If w_{i-1} and w_i are directions we can check by a comprehensive study that given a numeral ℓ and a bounding box $\mathcal{B}$, inserting two pins in the directions given by the two letters of $\phi(\ell)$ ends up in a pin lying in quadrant ℓ w.r.t. $\mathcal{B}$. Thus the quadrant in which p_i lies with respect to $\{p_0, \dots, p_{i-2}\}$ is $\phi^{-1}(w_{i-1}w_i)$. For example if $w_{i-1} = L$ and $w_i = U$, then p_i lies in the quadrant 2 and $\phi^{-1}(LU) = 2$.

Otherwise w_{i-1} is a numeral, w_i is a direction and $\phi(w_{i-1}w_i)$ ends with w_i. We can check by a comprehensive study that given a bounding box $\mathcal{B}$, inserting two pins according to w_{i-1} and w_i ends up in a pin lying in quadrant $\phi^{-1}(BC)$ with $\phi(w_{i-1}w_i) = ABC$ w.r.t. $\mathcal{B}$. For example if $w_{i-1} = 1$ and $w_i = L$, then p_i lies in the quadrant 2 and we have $\phi(1L) = RUL$ and $\phi^{-1}(UL) = 2$. ∎

Lemma 2.19 is used in the proofs of Lemma 2.21 and Theorem 2.22. Their statement also requires that we extend some definitions from $\mathcal{SP}$ to $\mathcal{M}$.

Remark 2.20. Words of $\mathcal{M}$ may also be seen as encodings of pin sequences (as in Subsection 2.2.2), taking the origin p_0 to be a box instead of a point. Moreover, the relation $u \preccurlyeq w$ can be extended to $w \in \mathcal{M}$, and the map ϕ can be defined on words of $\mathcal{M}$ as the identity map.

By definition, strong numeral-led factors of any pin word u are strict pin words. Therefore we first study how the relation $u \preccurlyeq w$ is mapped on $\phi(u), \phi(w)$ when u is a strict pin word.

Lemma 2.21. *Let u be a strict pin word and w be a word of $\mathcal{SP} \cup \mathcal{M}$. If $|u| \geq 2$ then $u \preccurlyeq w$ if and only if $\phi(u)$ is a factor of $\phi(w)$. If $|u| = 1$ then $u \preccurlyeq w$ if and only $\phi(w)$ has a factor in $\phi(u)$.*

Proof. If $u \preceq w$, as u is a strict pin word, writing u in terms of its strong numeral-led factors leads to $u = u^{(1)}$; thus w can be decomposed into a sequence of factors $w = v^{(1)}w^{(1)}v^{(2)}$ as in Definition 2.14.

If $v^{(1)}$ is empty then $w^{(1)}$ begins with a numeral, $w^{(1)} = u^{(1)}$ and u is a prefix of w. Consequently $\phi(u)$ is a prefix of $\phi(w)$.

Otherwise $i = |v^{(1)}| \geq 1$ and $w^{(1)}$ begins with a direction. By Definition 2.14, the first letter w_{i+1} of $w^{(1)}$ corresponds to a point p_{i+1} lying in the quadrant specified by u_1 (the first letter of $u^{(1)}$), and all other letters (which are directions) in $u^{(1)}$ and $w^{(1)}$ agree: $u_2 \dots u_{|u|} = w_{i+2} \dots w_{i+|u|}$.

By Lemma 2.19, $q(w_i, w_{i+1})$ is the quadrant in which p_{i+1} lies, i.e. $u_1 = q(w_i, w_{i+1})$. If $|u| \geq 2$, $\phi(u) = \phi(q(w_i, w_{i+1})w_{i+2} \dots w_{i+|u|})$ is a factor of $\phi(w)$ by definition of q. If $|u| = 1$, $\phi(w)$ has a factor in $\phi(u) = \phi(q(w_i, w_{i+1}))$ by definition of q.

Conversely if $|u| \geq 2$ and $\phi(u)$ is a factor of $\phi(w)$ or if $|u| = 1$ and $\phi(w)$ has a factor in $\phi(u)$, then $\phi(w) = v\, u'\, v'$ with $u = \phi^{-1}(u')$. Let $i = |v|$, we set $w = v^{(1)}w^{(1)}v^{(2)}$ with $|v^{(1)}| = i$ if $w \in \mathcal{SP}$, $|v^{(1)}| = i + 1$ if $w \in \mathcal{M}$ and $|w^{(1)}| = |u|$. Let $\ell = |v^{(1)}| + 1$.

If $\ell = 1$ then v is empty and $w \in \mathcal{SP}$. Thus u' is a prefix of $\phi(w)$ and $u = \phi^{-1}(u')$ is a prefix of w. Hence $u \preceq w$.

Otherwise $\ell \geq 2$. Then w_ℓ is a direction and we prove that the point p_ℓ corresponding to w_ℓ lies in the quadrant indicated by u_1. By Lemma 2.19, p_ℓ lies in quadrant $q(w_{\ell-1}, w_\ell)$.

But $\phi(w_{\ell-1}, w_\ell) = \begin{cases} \phi(w)_{\ell-1}\phi(w)_\ell\phi(w)_{\ell+1} \text{ if } w \in \mathcal{SP} \text{ and } l = 2 \\ \phi(w)_\ell\phi(w)_{\ell+1} \text{ if } w \in \mathcal{SP} \text{ and } l \neq 2 \\ \phi(w)_{\ell-1}\phi(w)_\ell \text{ if } w \in \mathcal{M} \end{cases}$

Thus $\phi(w_{\ell-1}, w_\ell) = \begin{cases} \phi(w)_i\phi(w)_{i+1}\phi(w)_{i+2} \text{ if } w \in \mathcal{SP} \text{ and } l = 2 \\ \phi(w)_{i+1}\phi(w)_{i+2} \text{ otherwise.} \end{cases}$

But $\phi(w)_{i+1}\phi(w)_{i+2} = u'_1u'_2$ hence $q(w_{\ell-1}, w_\ell) = \phi^{-1}(u'_1u'_2) = u_1$. If $|u| = 1$ this proves $u \preceq w$. Otherwise as $u = \phi^{-1}(u')$, $u_2 \dots u_{|u|} = u'_3 \dots u'_{|u|+1} = \phi(w)_{i+3} \dots \phi(w)_{i+|u|+1} = w_{\ell+1} \dots w_{\ell+|u|-1} = w^{(1)}_2 \dots w^{(1)}_{|u|}$. Hence $u \preceq w$. ■

In the statement of Lemma 2.21, we have distinguished the cases $|u| \geq 2$ and $|u| = 1$ since $\phi(u)$ is a word or a set of two words in these respective cases. However, to avoid such uselessly heavy statements, we do not make this distinction in the sequel, and we write indifferently "$\phi(u)$ is a factor of w" or "w has a factor in $\phi(u)$" meaning that $\begin{cases} \text{if } |u| = 1, w \text{ has a factor in } \phi(u) \\ \text{if } |u| \geq 2, \phi(u) \text{ is a factor of } w. \end{cases}$

When the pin word u is not strict, Lemma 2.21 can be extended formalizing the idea of piecewise factors mentioned at the beginning of this section.

Theorem 2.22. *Let u and w be two pin words and $u = u^{(1)} \dots u^{(j)}$ be the strong numeral-led factors decomposition of u. Then $u \preccurlyeq w$ if and only if w can be chopped into a sequence of factors $w = v^{(1)}w^{(1)} \dots v^{(j)}w^{(j)}v^{(j+1)}$ such that for all $i \in \{1, \dots, j\}$, $w^{(i)} \in \mathcal{SP} \cup \mathcal{M}$ and $\phi(w^{(i)})$ has a factor in $\phi(u^{(i)})$.*

Proof. We prove that $u \preccurlyeq w$ if and only if w can be chopped into a sequence of factors $w = v^{(1)}w^{(1)} \dots v^{(j)}w^{(j)}v^{(j+1)}$ such that for all $i \in \{1, \dots, j\}$, $w^{(i)} \in \mathcal{SP} \cup \mathcal{M}$ and $u^{(i)} \preccurlyeq w^{(i)}$. Then the result follows using Lemma 2.21.

If $u \preccurlyeq w$, then w can be chopped into $w = \bar{v}^{(1)}\bar{w}^{(1)} \dots \bar{v}^{(j)}\bar{w}^{(j)}\bar{v}^{(j+1)}$ as in Definition 2.14. We set $w^{(i)} = \bar{w}^{(i)}$ if $\bar{w}^{(i)}$ begins with a numeral, and we take $w^{(i)}$ to be the suffix of $\bar{v}^{(i)}\bar{w}^{(i)}$ of length $|\bar{w}^{(i)}| + 1$ otherwise. Then for all $i \in \{1, \dots, j\}$, $w^{(i)} \in \mathcal{SP} \cup \mathcal{M}$ and we have $u^{(i)} \preccurlyeq w^{(i)}$. Indeed, if $\bar{w}^{(i)}$ begins with a direction, by Lemma 2.19 the point corresponding to the first letter of $\bar{w}^{(i)}$ lies in the quadrant determined by the last letter of $\bar{v}^{(i)}$ and the first letter of $\bar{w}^{(i)}$ (w.r.t. the origin of the encoding w and also w.r.t. the origin of the encoding $w^{(i)}$).

Conversely if w can be chopped into $w = v^{(1)}w^{(1)} \dots v^{(j)}w^{(j)}v^{(j+1)}$ such that for all $i \in \{1, \dots, j\}$, $u^{(i)} \preccurlyeq w^{(i)}$ then from Definition 2.14 we can decompose $w^{(i)}$ as $y^{(i)}\bar{w}^{(i)}z^{(i)}$ and thanks to Lemma 2.19 it is sufficient to set $\bar{v}^{(i)} = z^{(i-1)}v^{(i)}y^{(i)}$ to have $w = \bar{v}^{(1)}\bar{w}^{(1)} \dots \bar{v}^{(j)}\bar{w}^{(j)}\bar{v}^{(j+1)}$ as in Definition 2.14. ■

2.3.2 Pattern containment and set inclusion

Definition 2.23. Let u be a pin word and $u = u^{(1)} \dots u^{(j)}$ be its strong numeral-led factor decomposition. We set

$$\mathcal{L}(u) = A^\star\phi(u^{(1)})A^\star\phi(u^{(2)}) \dots A^\star\phi(u^{(j)})A^\star$$

where $A = \{U, D, L, R\}$ (or $A = \{U, D, L, R, \sharp\}$ in Theorem 2.24(i)).

Let π be a permutation, and $P(\pi)$ be the set of pin words that encode π. We set
$$\mathcal{L}_\pi = \cup_{u \in P(\pi)} \mathcal{L}(u)$$

Note that $\mathcal{L}_\pi$ is nonempty if and only if $P(\pi)$ is nonempty or equivalently π is a pin-permutation. When π is not a pin-permutation, the results of Theorem 2.24 and Lemma 2.26 follow easily from the following statement (see for instance Lemma 3.3 of [BBR11]): if $\pi \leq \sigma$ and σ is a pin-permutation, then π is a pin-permutation.

In the following, we write $m = v^{(1)}\phi(u^{(1)})v^{(2)}\phi(u^{(2)})\dots v^{(j)}\phi(u^{(j)})v^{(j+1)}$ for $m \in A^\star$, meaning that $m = v^{(1)}w^{(1)}v^{(2)}w^{(2)}\dots v^{(j)}w^{(j)}v^{(j+1)}$ with $w^{(i)} \in \phi(u^{(i)})$ if $u^{(i)}$ has length 1 and $w^{(i)} = \phi(u^{(i)})$ otherwise.

The languages $\mathcal{L}_\pi$ have been introduced in Definition 2.23 because they describe the proper pin-permutations that contain π. Indeed $\mathcal{L}_\pi \cap \mathcal{M}$ is in one-to-one correspondence with pin words encoding proper pin-permutations that contain π, via ϕ^{-1}. Moreover the languages $\mathcal{L}_\pi$ somehow translate the pattern involvement between pin-permutations into set inclusion:

Theorem 2.24. *Let π and σ be permutations, σ being a pin-permutation. Then*

(i) for $A = \{U, D, L, R, \sharp\}$, $\pi \leq \sigma$ if and only if $\mathcal{L}_\sigma \subseteq \mathcal{L}_\pi$;

(ii) for $A = \{U, D, L, R\}$, if $\pi \leq \sigma$ then $\mathcal{L}_\sigma \subseteq \mathcal{L}_\pi$.

Remark 2.25. In the sequel, we always take $A = \{U, D, L, R\}$, and use only the second statement of Theorem 2.24. However, we find it interesting to have an equivalence between pattern containment of pin-permutations and set inclusion in the first statement of this theorem with $A = \{U, D, L, R, \sharp\}$. We do not know if the equivalence still holds when $A = \{U, D, L, R\}$.

Proof. Suppose that $\pi \leq \sigma$. Let $m \in \mathcal{L}_\sigma$, we want to show that $m \in \mathcal{L}_\pi$. By definition of $\mathcal{L}_\sigma$, there exists $w \in P(\sigma)$ such that $m = v^{(1)}\phi(w^{(1)})v^{(2)}\dots\phi(w^{(i)})v^{(i+1)}$ where $w = w^{(1)}w^{(2)}\dots w^{(i)}$ is the strong numeral-led factor decomposition of w. From Lemma 2.16 there is $u \in P(\pi)$ such that $u \preccurlyeq w$. Let $u = u^{(1)}\dots u^{(j)}$ be the strong numeral-led factor decomposition of u. From Theorem 2.22, $w = \bar{v}^{(1)}\bar{w}^{(1)}\dots\bar{v}^{(j)}\bar{w}^{(j)}\bar{v}^{(j+1)}$ where for all $k \in \{1, \dots, j\}$, $\bar{w}^{(k)} \in \mathcal{SP} \cup \mathcal{M}$ and $\phi(\bar{w}^{(k)})$ has a factor in $\phi(u^{(k)})$. But $w^{(1)}w^{(2)}\dots w^{(i)}$ is the strong numeral-led factor decomposition of $w = \bar{v}^{(1)}\bar{w}^{(1)}\dots\ \bar{v}^{(j)}\bar{w}^{(j)}\bar{v}^{(j+1)}$. Therefore the factors $\bar{w}^{(1)}$, $\bar{w}^{(2)}$, $\dots\bar{w}^{(j)}$ appear in this order in $w^{(1)}w^{(2)}\dots w^{(i)}$, being non-overlapping and each inside one $w^{(\ell)}$, since each $w^{(\ell)}$ begins with a numeral and each $\bar{w}^{(k)}$ is in $\mathcal{SP} \cup \mathcal{M}$. Thus by definition of ϕ, the factors $\phi(\bar{w}^{(1)})$, $\phi(\bar{w}^{(2)})$, $\dots$, $\phi(\bar{w}^{(j)})$ appear in this order in $\phi(w^{(1)})\phi(w^{(2)})\dots\phi(w^{(i)})$, being non-overlapping and each inside one $\phi(w^{(\ell)})$. So $m = v^{(1)}\phi(w^{(1)})v^{(2)}\phi(w^{(2)})\dots v^{(i)}\phi(w^{(i)})v^{(i+1)} \in A^\star\phi(\bar{w}^{(1)})A^\star\phi(\bar{w}^{(2)})\dots A^\star\phi(\bar{w}^{(j)})A^\star$. But for all $k \in \{1, \dots, j\}$, $\phi(\bar{w}^{(k)})$ has a factor in $\phi(u^{(k)})$, thus $m \in \mathcal{L}(u) = A^\star\phi(u^{(1)})A^\star$ $\phi(u^{(2)})\dots A^\star\phi(u^{(j)})A^\star$ and so $m \in \mathcal{L}_\pi$.

Conversely, in the case $A = \{U, D, L, R, \sharp\}$, if $\mathcal{L}_\sigma \subseteq \mathcal{L}_\pi$, let us show that $\pi \leq \sigma$. From Lemma 2.16 it is sufficient to show that there is some $u \in P(\pi)$ and some $w \in P(\sigma)$ such that $u \preccurlyeq w$. As σ is a pin-permutation, $P(\sigma)$ is not empty. Let w be a word of $P(\sigma)$ and $w^{(1)}w^{(2)}\dots w^{(i)}$ its strong numeral-led factor decomposition. Let $m = \phi(w^{(1)})\,\sharp\,\phi(w^{(2)})\dots\,\sharp\,\phi(w^{(i)})$. Obviously, $m \in \mathcal{L}(w)$ so that $m \in \mathcal{L}_\sigma$ and $m \in \mathcal{L}_\pi$. By definition of $\mathcal{L}_\pi$ there is some $u \in P(\pi)$ such that $m = v^{(1)}\phi(u^{(1)})v^{(2)}\phi(u^{(2)})\dots\phi(u^{(j)})v^{(j+1)}$ where $u^{(1)}u^{(2)}\dots u^{(j)}$ is the strong numeral-led factor decomposition of u. But $m = \phi(w^{(1)})\,\sharp\,\phi(w^{(2)})\dots\,\sharp\,\phi(w^{(i)})$ and there is no letter $\sharp$ in the $\phi(u^{(k)})$. So the factors $\phi(u^{(1)})$, $\phi(u^{(2)})$, $\dots\phi(u^{(j)})$ appear in this order in $\phi(w^{(1)})\phi(w^{(2)})\dots\phi(w^{(i)})$, being non-overlapping and each inside one $\phi(w^{(k)})$. Therefore w can be chopped into a sequence of factors

$w = \bar{v}^{(1)}\bar{w}^{(1)} \dots \bar{v}^{(j)}\bar{w}^{(j)}\bar{v}^{(j+1)}$ such that for all $k \in \{1, \dots, j\}$, $\bar{w}^{(k)} \in \mathcal{SP} \cup \mathcal{M}$ and $\phi(\bar{w}^{(k)})$ has a factor in $\phi(u^{(k)})$. Thus from Theorem 2.22 $u \preccurlyeq w$, concluding the proof. ■

2.3.3 Characterizing when a class has a finite number of proper pin-permutations

We conclude Section 2.3 by explaining how the above definitions and results are related to our original problem: testing whether a permutation class contains finitely many proper pin-permutations.

Lemma 2.26. *Let σ be a proper pin-permutation, π be a permutation and w be a strict pin word encoding σ. Then $\pi \leq \sigma$ if and only if $\phi(w) \in \mathcal{L}_\pi$.*

Proof. Assume that $\pi \leq \sigma$, then from Theorem 2.24(*ii*) $\mathcal{L}_\sigma \subseteq \mathcal{L}_\pi$. As w is a strict pin word, $\phi(w) \in \mathcal{L}(w)$ thus $\phi(w) \in \mathcal{L}_\sigma$ and so $\phi(w) \in \mathcal{L}_\pi$.

Conversely, assume that $\phi(w) \in \mathcal{L}_\pi$. Then there exists a pin word u encoding π such that $\phi(w) \in \mathcal{L}(u)$. Let us denote by $u = u^{(1)} \dots u^{(j)}$ the strong numeral-led factor decomposition of u. By definition of $\mathcal{L}(u)$, $\phi(w)$ can be decomposed into $t^{(1)} \dots t^{(j+1)}$, with $t^{(i)} \in A^\star \phi(u^{(i)}) \cap \mathcal{M}$ for $i \in \{1, \dots, j\}$. By definition of ϕ and since w is a strict pin word, there exists a strict pin word t such that $w = t\, t^{(2)} \dots t^{(j+1)}$. Then $\phi(t) = t^{(1)}$ and $\phi(u^{(1)})$ is a factor of $\phi(t)$. Furthermore, for $i \in \{2, \dots, j\}$, $\phi(u^{(i)})$ is a factor of $\phi(t^{(i)}) = t^{(i)}$ (this equality holds because $t^{(i)} \in \mathcal{M}$). Consequently, from Theorem 2.22, $u \preccurlyeq w$. Finally from Lemma 2.16, we conclude that $\pi \leq \sigma$. ■

By ϕ^{-1}, each word of $\mathcal{M}$ is turned into a strict pin word and hence into a proper pin-permutation. As a consequence of Lemma 2.26, $\mathcal{L}_\pi \cap \mathcal{M}$ is the image by ϕ of the language of strict pin words encoding proper pin-permutations σ that contain π as a pattern: $\mathcal{L}_\pi \cap \mathcal{M} = \{\phi(w) \mid \exists \sigma \text{ such that } \pi \leq \sigma \text{ and } w \in \mathcal{SP} \cap P(\sigma)\}$. With the same idea we have the following theorem:

Theorem 2.27. *A permutation class $Av(B)$ contains a finite number of proper pin-permutations if and only if the set $\mathcal{M} \setminus \cup_{\pi \in B} \mathcal{L}_\pi$ is finite.*

Proof. Let S_B be the set of strict pin words encoding permutations of size at least 2 of $Av(B)$. Then ϕ is a bijection from S_B to $\mathcal{M}_{\geq 3} \setminus \cup_{\pi \in B} \mathcal{L}_\pi$. Indeed for any strict pin word w of size at least 2, let σ be the permutation encoded by w. Then σ is a proper pin-permutation and Lemma 2.26 implies that $\sigma \in Av(B)$ if and only if $\phi(w) \notin \cup_{\pi \in B} \mathcal{L}_\pi$. We conclude the proof observing that every proper pin-permutation σ of size n is associated to at least 1 and at most 8^n strict pin words, as any pin word encoding σ is a word of length n over an 8-letter alphabet. ■

From Theorem 2.27, our goal is now to find an algorithm checking if $\mathcal{M} \setminus \cup_{\pi \in B} \mathcal{L}_\pi$ is finite, given a finite basis B. To do this, in Section 2.5 we construct automata recognizing $\mathcal{L}_\pi$ for any pin-permutation π. As the language $\mathcal{L}_\pi$ is defined from the set $P(\pi)$ of pin words of π, we first need to explicitly describe $P(\pi)$ for any pin-permutation π. This is done in Section 2.4.

2.4 Pin words of pin-permutations

In this section, our goal is to describe the set $P(\pi)$ of pin words that encode a pin-permutation π. In [BBR11], a recursive characterization of the decomposition trees of pin-permutations is provided, and we follow it to recursively describe $P(\pi)$.

2.4.1 Characterization of pin-permutations

In this chapter, we use the characterization of pin-permutations obtained by Bassino, Bouvel and Rossin in [BBR11].

Some special permutations, called *oscillations* and *quasi-oscillations*, play a key role in the characterization of substitution decomposition trees associated with pin-permutations. Therefore, we start by defining these permutations.

Following [BRV08], let us consider the infinite oscillating sequence defined (on $\mathbb{N}\setminus\{0,2\}$ for regularity of the diagram) by $\omega = 4\ 1\ 6\ 3\ 8\ 5\ \ldots (2k+2)\ (2k-1)\ \ldots$. Figure 2.6 shows the diagram of a prefix of ω.

Definition 2.28 (oscillation)**.** An *increasing oscillation* of size $n \geq 4$ is a simple permutation of size n that is contained as a pattern in ω. For smaller sizes the increasing oscillations are 1, 21, 231 and 312. A *decreasing oscillation* is the reverse[1] of an increasing oscillation.

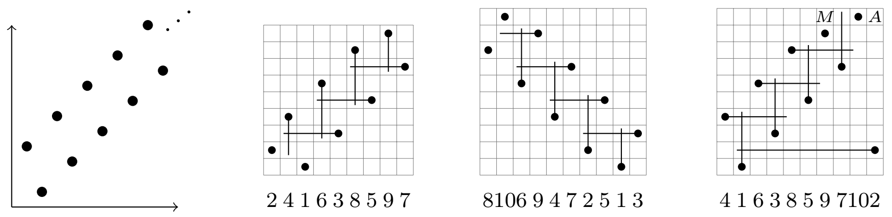


Figure 2.6: The infinite oscillating sequence ω, an increasing oscillation ξ of size 9, a decreasing oscillation of size 10, and the increasing quasi-oscillation of size 10 obtained from ξ by addition of a maximal element, with a pin representation for each.

As noticed in [BBR11] there are two increasing (resp. decreasing) oscillations of size n for any $n \geq 3$. Permutations 1, 2 4 1 3 and 3 1 4 2 are both increasing and decreasing oscillations, and are the only ones with this property.

Definition 2.29 (quasi-oscillations [BBR11])**.** An *increasing quasi-oscillation* of size $n \geq 6$ is obtained from an increasing oscillation ξ of size $n-1$ by the addition of either a minimal element at the beginning of ξ or a maximal element at the end of ξ, followed by a *flip* of an element of ξ according to the rules of Table 2.1[2]. We define the *auxiliary point* (A) to be the point added to ξ, and the *main substitution point* (M) to be an extremal point of ξ according to Table 2.1.

Furthermore, for $n = 4$ or 5, there are two increasing quasi-oscillations of size n: 2 4 1 3, 3 1 4 2, 2 5 3 1 4 and 4 1 3 5 2. Each of them has two possible choices for its pair of auxiliary and main substitution points. See Figure 2.7 for more details. Finally, a *decreasing quasi-oscillation* is the reverse of an increasing quasi-oscillation.

As noticed in [BBR11] there are four increasing (resp. decreasing) quasi-oscillations of size n for any $n \geq 6$, two of size 4 (2 4 1 3 and 3 1 4 2) and two of size 5 (2 5 3 1 4 and 4 1 3 5 2). It should be noticed that each quasi-oscillation of size 4 or 5 is both increasing

[1]The reverse of $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ is $\overleftarrow{\sigma} = \sigma_n\ldots\sigma_2\sigma_1$

[2]The first row of Table 2.1 reads as follows: If a maximal element is added to ξ, with $\xi \in S_{n-1}$ starting (resp. ending) with a pattern 231 (resp. 132), then the corresponding increasing quasi-oscillation β is obtained by flipping the left-most point of ξ to the right-most (in β), and the main substitution point is the largest point of ξ (see Figure 2.6).

Element inserted	Pattern $\xi_1\xi_2\xi_3$	Pattern $\xi_{n-3}\xi_{n-2}\xi_{n-1}$	Flipped element ...	... which becomes	Main substitution point
max	231	132	left-most	right-most	largest
max	231	312	left-most	right-most	right-most
max	213	132	smallest	largest	largest
max	213	312	smallest	largest	right-most
min	231	132	largest	smallest	left-most
min	231	312	right-most	left-most	left-most
min	213	132	largest	smallest	smallest
min	213	312	right-most	left-most	smallest

Table 2.1: Flips and main substitution points in increasing quasi-oscillations.

and decreasing. However, once its auxiliary point is chosen among the four possibilities, then its nature (increasing or decreasing) is determined without ambiguity, and so is its main substitution point. Moreover, knowing the (unordered) pair of points which are the auxiliary and main substitution points, we can deduce which one is the auxiliary point without ambiguity. These remarks will be useful in the following.

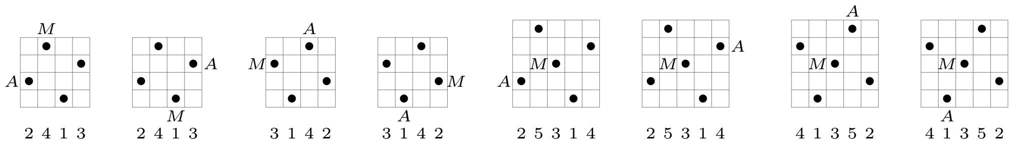


Figure 2.7: The diagrams of the increasing quasi-oscillations of size 4 and 5.

Remark 2.30. Oscillations of size at least 4 and quasi-oscillations are simple pin-permutations.

We refer the reader to [BBR11] for further properties of oscillations and quasi-oscillations.

We now have all the notions we need to state the characterization of pin-permutations obtained by Bassino, Bouvel and Rossin in [BBR11].

Recalling that we say "a child of a node V" instead of "the permutation corresponding to the subtree rooted at a child of node V", the characterization of [BBR11] is as follows:

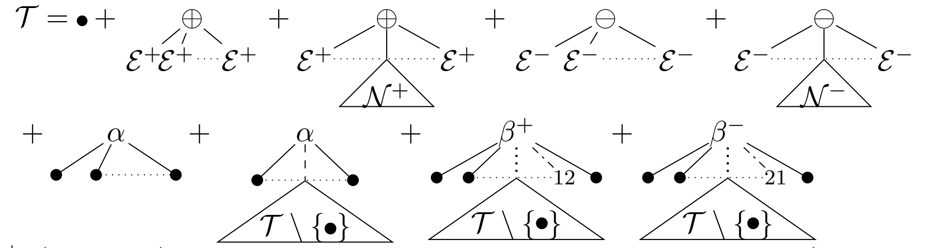
Theorem 2.31 (Theorem 3.1 of [BBR11])**.** *A permutation σ is a pin-permutation if and only if its substitution decomposition tree T_σ satisfies the following conditions:*

(C_1) *any linear node labeled by $\oplus$ (resp. $\ominus$) in T_σ has at most one child that is not an increasing (resp. decreasing) oscillation.*

(C_2) *any prime node in T_σ is labeled by a simple pin-permutation α and satisfies one of the following properties:*

- *it has at most one child that is not a singleton; moreover the point of α corresponding to the non-trivial child (if it exists) is an* active *point of α, i.e., a point that can be the first point of a pin representation of α.*
- *α is an increasing (resp.decreasing) quasi-oscillation, and the node has exactly two children that are not singletons: one of them expands the main substitution point of α and the other one is the permutation* 12 *(resp.* 21*), expanding the auxiliary substitution point of α.*

As stated in [BBR11], using Theorem 2.31 the set $\mathcal{T}$ of substitution decomposition trees of pin-permutations is recursively characterized by the following equation:



$\mathcal{T} = \bullet +$
$\oplus$
$\mathcal{E}^+ \mathcal{E}^+ \cdots \mathcal{E}^+$
$\mathcal{N}^+$
$\ominus$
$\mathcal{E}^- \mathcal{E}^- \cdots \mathcal{E}^-$
$\mathcal{N}^-$
α
β^+
12
β^-
21
$\mathcal{T} \setminus \{\bullet\}$

$(\star)$

where $\mathcal{E}^+$ (resp. $\mathcal{E}^-$) is the set of decomposition trees of increasing (resp. decreasing) oscillations, $\mathcal{N}^+$ (resp. $\mathcal{N}^-$) is the set of decomposition trees of pin-permutations that are not increasing (resp. decreasing) oscillations and whose root is not $\oplus$ (resp. $\ominus$), α is any simple pin-permutation and β^+ (resp. β^-) is any increasing (resp. decreasing) quasi-oscillation. Edges written - - - (resp. -.-.-, ...) correspond to an active point of α (resp. to a pair formed by an auxiliary point and a main substitution point of β). In this definition the only terms that are recursive are those containing a subtree labeled by $\mathcal{N}^+$,$\mathcal{N}^-$ or $\mathcal{T} \setminus \{\bullet\}$.

In [BBR11], the authors also compute the generating function of the class of pin-permutations, proving the rationality of this generating function. Moreover, they show that the basis of the pin-permutation class is infinite.

To prove Theorem 2.31, they established several lemmas. We record here the ones that we use in this section, recalling first a useful definition:

Definition 2.32. Let π be a pin-permutation and $p = (p_1, \ldots, p_n)$ be a pin representation of π. For any set D of points of π, if k is the number of maximal factors $p_i, p_{i+1}, \ldots, p_{i+j}$ of p that contain only points of D, we say that D is *read in k pieces by p*. If C is a set of points of π disjoint from D, we say that D is *read before* (resp. *read entirely before*) C if the first pin in D (resp. every pin belonging to D) appears in p before the first pin belonging to C.

Lemma 2.33 (Lemma 3.7 of [BBR11]). *Let σ be a pin-permutation whose substitution decomposition tree has a root that is a linear node V labeled by $\oplus$ (resp. by $\ominus$). Then at most one child of V is not an ascending (resp. descending) oscillation, this child (if it exists) is the first child of V that is read by any pin representation of σ, and all other children are read in one piece.*

Lemma 2.34 (Lemma 3.11 of [BBR11]). *Let σ be a pin-permutation whose substitution decomposition tree has a prime node V as root and let $p = (p_1, \ldots, p_n)$ be a pin representation of σ.*

(*i*) *If some child B of V is read in more than one piece by p, then it is read in exactly two pieces, the second part being the last point p_n of p.*

(*ii*) *At most one of the children of V can be read in two pieces by p and this child (if it exists) is the first or the second child of V to be read by p.*

Lemma 2.35 (Lemma 3.12 of [BBR11]). *Let σ be a pin-permutation whose substitution decomposition tree has a prime root V and $p = (p_1, \ldots, p_n)$ be a pin representation of σ.*

(*i*) *V has at most two children that are not singletons.*

(*ii*) *If there exists a child B of V that is not a singleton and that is not the first child of V to be read by p then B contains exactly two points, the first point of B read by p*

is an independent pin, the second one is p_n. Moreover the first child of V read by p in read in one piece and B is the second child read by p.

The characterization of $P(\pi)$ we provide in this section is naturally divided into several cases, depending on which term of Equation ($\star$) π belongs to. First, we study the non-recursive cases, then the recursive cases with a linear root and finally the recursive cases with a prime root. We start with a preliminary study of the ways children of decomposition trees with linear root can be read in a pin representation. These first results will be useful both in the non-recursive and the recursive cases.

Remark 2.36. In the study that follows, we never examine the case of decomposition trees with a linear root labeled by $\ominus$. Indeed, permutations with decomposition trees of this form are the reverse of permutations whose decomposition trees have a linear root labeled by $\oplus$, and every argument and result on the $\oplus$ case can therefore be transposed to the $\ominus$ case.

2.4.2 Reading of children of a linear node

Let π be a pin-permutation whose decomposition tree T has a linear root. W.l.o.g., assume that $T = \begin{array}{c}\oplus\\ T_1\ T_2\ \cdots\ T_r\end{array}$ and let $p = (p_1, \dots, p_n)$ be one of its pin representations. In the sequel, we denote by i_0 the index of the child which contains p_1.

Lemma 2.37. *Let $1 \le i, j \le r$ such that either $i < j < i_0$ or $i_0 < j < i$. Then T_j is read by p entirely before T_i.*

Proof. By definition of pin representations, for all $k \ge 2$, p_k lies outside the bounding box of $\{p_1, \dots, p_{k-1}\}$. More generally for all $m \ge 0$, p_{k+m} lies outside the bounding box of $\{p_1, \dots, p_{k-1}\}$. This observation is used many times in the following, sometimes implicitly, and is the funtamental argument of this proof. Indeed let $\ell = \min\{\ell', p_{\ell'} \in T_i\}$ and let $\mathcal{B}_{p_1,\dots,p_\ell}$ be the bounding box of $\{p_1, \dots, p_\ell\}$. As $p_1 \in T_{i_0}$ and $p_\ell \in T_i$, $T_j \subseteq \mathcal{B}_{p_1,\dots,p_\ell}$ (see Figure 2.8), hence it is entirely read before p_ℓ in p. ■

The previous lemma gives the possible orders in which children are read. Now we characterize the children T_i which may be read in several pieces. When this is the case, we will prove that the decomposition tree is of a specific shape. This can indeed be deduced from the two following lemmas.

Lemma 2.38. *For every $k \in \{1, \dots, n\}$ there is at most one child whose reading has started and is not finished after $(p_1, p_2, \dots, p_k)$.*

Proof. Suppose that pins $p_1, \dots, p_k$ have already been read and that there are two children T_i and T_m with $i < m$ whose readings have started and are not finished. By Lemma 2.37 there exists at most one child T_j with $j < i_0$ and at most one child T_j with $j > i_0$ whose readings have started and are not finished. Therefore $i \le i_0$ and $m \ge i_0$. Note that $i = \min\{\ell \mid \exists h \in \{1, \dots, k\}, p_h \in T_\ell\}$. The same goes for m changing the minimum into a maximum. If the reading of T_i is not finished, since T_i is $\oplus$-indecomposable, there must exist a pin p_q in zone ▨ (see Figure 2.9). Such a pin is on the side of the bounding box $\mathcal{B}_{p_1,\dots,p_k}$ of $\{p_1, \dots, p_k\}$, and the same remark goes for T_m. But from Lemma 2.1 (p.56) there is at most one pin lying on the sides of a bounding box, and this contradiction concludes the proof. ■

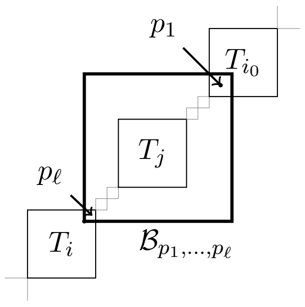


Figure 2.8: Proof of Lemma 2.37.

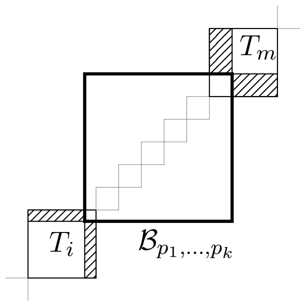


Figure 2.9: Proof of Lemmas 2.38 and 2.39.

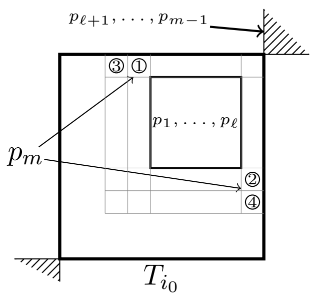


Figure 2.10: T_{i_0} is read in two pieces (Lemma 2.40).

Lemma 2.39. *Every child T_i is read in one piece by p, except perhaps T_{i_0}.*

Proof. Consider a child T_i with $i \neq i_0$ which is read in more than one piece by p. Consider the pin p_{k+1} which is the first pin outside T_i after p has started reading T_i. As p_1 is in T_{i_0}, p_1 is outside T_i and the bounding box of $\{p_1, p_2, \ldots, p_{k-1}, p_k\}$ allows to define a zone ▨ in T_i as shown in Figure 2.9. Since T_i is $\oplus$-indecomposable, there is at least one pin in this zone. This pin is on the side of the bounding box of $\{p_1, p_2, \ldots, p_k\}$ so it is p_{k+1} by Lemma 2.1 (p.56). Thus $p_{k+1} \in T_i$ which provides the desired contradiction. ■

When a child may be read in several pieces, the decomposition tree of the whole permutation π has a special shape given in the following lemma.

Lemma 2.40. *The only permutations π whose decomposition trees have a root $\oplus$ in which a child may be read in several pieces are those whose decomposition trees have one of the shapes given in Figure 2.11 where ξ^+ is an increasing oscillation of size at least 4.*

A given permutation π may match several shapes of Figure 2.11. However if a child is read in more than one piece, then it is necessarily the first child to be read (denoted T_{i_0}) and it is read in two pieces; in addition, there is exactly one shape of Figure 2.11 such that the first part of T_{i_0} to be read is S and the second part is the remaining leaves of T_{i_0} with only the point x read in between.

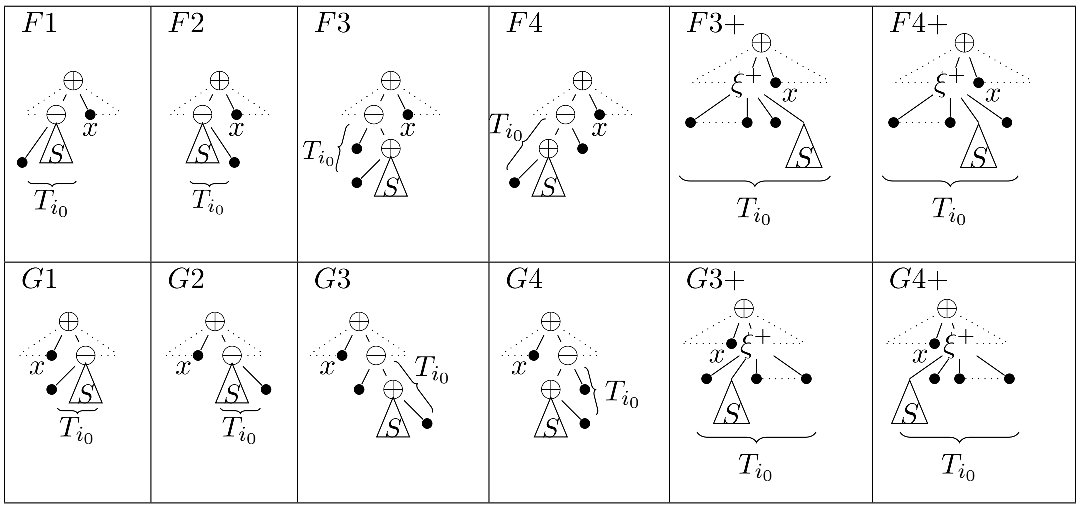
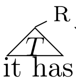
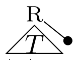


Figure 2.11: Decomposition tree of π when T_{i_0} may be read in several pieces.

In Figure 2.11 and in the sequel, we draw the attention of the reader to the difference between trees of the shape 

and 

: in the first case the root R has exactly 2 children, in the second one it has at least two children, T being a forest.

Proof. Let π be a pin-permutation whose decomposition tree has a root $\oplus$. Let $p = (p_1, p_2, \ldots, p_n)$ be a pin representation of π that reads one child in several pieces. Lemma 2.39 ensures that there is only one such child, which is necessarily T_{i_0}. Denote $p_1, \ldots, p_\ell$ the first part of the reading of T_{i_0}. Then $p_{\ell+1}, \ldots, p_{m-1}$ belong to other children until $p_m \in T_{i_0}$.

As T_{i_0} is a child of the root $\oplus$, each pin p_i with $i \in \{\ell+1, \ell+2, \ldots, m-1\}$ lies in one of the zones ▨ as shown in Figure 2.10. But if both zones contain at least one pin p_i with $i \in \{\ell+1, \ell+2, \ldots, m-1\}$, the bounding box of $\{p_1, \ldots, p_{m-1}\}$ contains T_{i_0} and thus p_m cannot respect the externality condition. Hence all pins p_i with $i \in \{\ell+1, \ell+2, \ldots, m-1\}$ are in the same zone.

Assume w.l.o.g. that $\{p_{\ell+1}, \ldots, p_{m-1}\}$ are in the upper right zone of Figure 2.10 (otherwise, in the proof that follows, cases $F1, \ldots, F4+$ of Figure 2.12 are replaced by cases $G1, \ldots, G4+$). If p_m respects the independence condition, it must lie in the lower left corner of the bounding box of $\{p_1, \ldots, p_\ell\}$ and every future pin of T_{i_0} lies in the same corner leading to a $\oplus$-decomposable child T_{i_0} which contradicts our hypothesis. Thus p_m must be a separating pin and $m = \ell + 2$.

As only one point can lie on the sides of a bounding box, there are only four possible positions for p_m as depicted in Figure 2.10. If there is no pin separating p_m from $\{p_1, \ldots, p_{m-1}\}$ then p_m is either in position ① (case $F1$ on Figure 2.12) or ② (case $F2$); moreover pins $\{p_1, p_2, \ldots, p_\ell, p_m\}$ form a block and thus represent T_{i_0} (because T_{i_0} is $\oplus$-indecomposable). Otherwise there is exactly one pin p_{m+1} separating p_m from the bounding box of $\{p_1, \ldots, p_\ell\}$, thus p_m is either in position ③ (cases $F3$ and $F3+$) or ④ (cases $F4$ and $F4+$). Suppose that it is in position ④ then p_{m+1} is a left pin separating p_m from the preceding ones. There are again two different cases: if p_{m+2} respects the independence condition (case $F4$), then p_{m+1} ends T_{i_0} (since T_{i_0} is $\oplus$-indecomposable). If p_{m+2} respects the separation condition then it can only separate p_{m+1} and $\{p_1, \ldots, p_m\}$ from below (case $F4+$). This process can be repeated alternating between left and down pins until the following pin p_{m+k+1} is an independent pin, ending the child T_{i_0}.

Thus we have proved that T_{i_0} is read in exactly two pieces, $p_1, \ldots p_\ell$ for the first part and $p_m, \ldots, p_{m+k}$ with $m = \ell + 2$ for the second part. And from Lemma 2.39 the pin $p_{\ell+1}$ is by itself a child T_i of the root. It is then straightforward to check from Figure 2.12 that π has a decomposition tree of one shape given in Figure 2.11 with $S = \{p_1, \ldots, p_\ell\}$ and $x = p_{\ell+1}$. ■

Note that in the proof of Lemma 2.40, the order in which the points corresponding to the leaves of $T_{i_0} \setminus S$ are read is uniquely determined, leading to the following remark:

Remark 2.41. If a child T_{i_0} is read in two pieces with the first part fixed, then the second part consists of all remaining points of T_{i_0} and the order in which they are read is uniquely determined.

We now start the description of the set of pin words encoding any pin-permutation, by case study on Equation ($\star$) (p.69).

2.4.3 Non-recursive case: size 1 and simple pin-permutations

Permutation of size 1. Notice first that the permutation $\pi = 1 = \bullet$ (whose decomposition tree is a leaf) has exactly four pin words – namely, $P(\pi) = \{1, 2, 3, 4\}$.

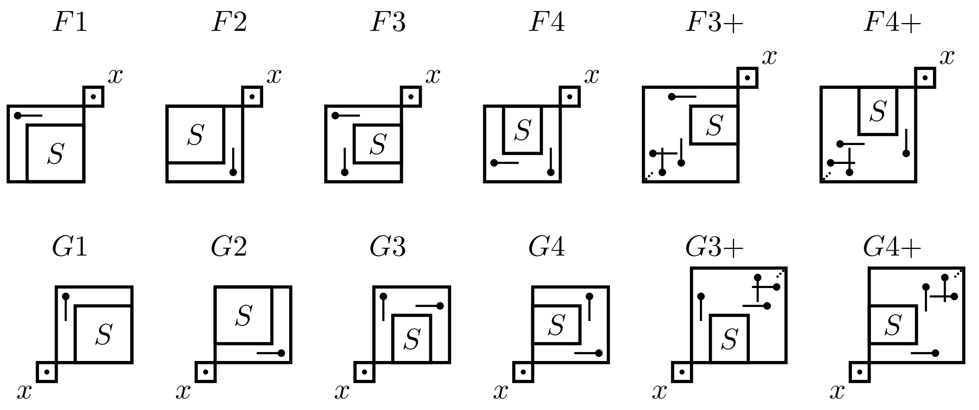
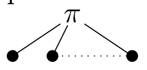


Figure 2.12: Diagram of T_{i_0} and x if T_{i_0} is read in two pieces, the first part being S.

Simple permutations. The only pin-permutations whose decomposition trees have a prime root and are non-recursive are those whose decomposition trees are of the form  , i.e., the simple pin-permutations. Theorem 2.48 describes properties of their pin words. We need a few lemmas and definitions to state it.

Lemma 2.42. *Let π be a simple permutation. There is at most one pin representation of π beginning with a given ordered pair of points. Moreover such a pin representation (if it exists) can be computed in linear time w.r.t. $|\pi|$.*

Proof. Because π is simple, its pin representations are always proper (see Remark 2.2 p.56). The pin representation starting with a given ordered pair of points (p_1, p_2) is then obtained as follows (if it exists). If $(p_1, \dots, p_i)$ has already been computed then, since the pin representation we look for is proper, p_{i+1} separates $p_i = \pi_k$ from previous points. Moreover by Lemma 2.1 it is the only point separating them. It means that either it separates them vertically, and then $p_{i+1} = \pi_{k+1}$ or $p_{i+1} = \pi_{k-1}$, or it separates them horizontally and then its value must be $\pi_k \pm 1$. Therefore, if we compute π^{-1} in advance (which is easily done by a linear-time precomputation) we can find the next point in a proper pin representation in constant time. If at one step of this procedure we have no possible next point or two possible next points, it means that there is no pin representation of π beginning with (p_1, p_2). ■

Definition 2.43. Consider a permutation π given by its diagram. We say that two points x and y of π are (or that the pair of points (x, y) is) in *knight position* when the distance between the points x and y is exactly 3 cells and the two points are neither on the same row nor on the same column (see Figure 2.13).

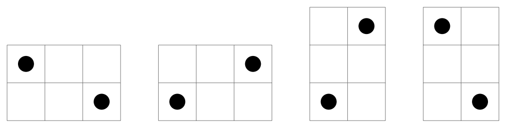


Figure 2.13: Two points in knight position.

Lemma 2.44. *[BBR11] Let $p = (p_1, p_2, \dots, p_n)$ denote a proper pin representation of some permutation π. If $|\pi| > 2$ the first two pins p_1, p_2 are in knight position.*

Lemma 2.45. *[BBR11] Let π be a simple pin-permutation and $p = (p_1, p_2, \ldots, p_n)$ be one of its pin representations. If two points p_i and p_j of π are in knight position then i or j is equal to $1, 2$ or n.*

Definition 2.46. An *active pair* in a pin-permutation π is an unordered pair of points (x, y) that can be the first two points of a pin representation of π. An *active point* is a point belonging to an active pair. An *active knight* is an active pair in knight position.

Notice that from Lemma 2.44, an active pair of a simple permutation is necessarily an active knight. As a consequence, using Lemma 2.42 the number of pin representations of a simple permutation is twice the number of active knights, which is studied in the next lemma.

Lemma 2.47. *[BBR11] In any simple pin-permutation π, there are at most two active knights except for the four permutations:* 3142, 2413, 25314 *and* 41352 *which have four active knights. The simple pin-permutations of size at most* 6 *and their active knights are represented on Table 2.2.*

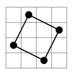
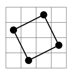
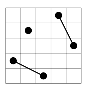
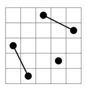
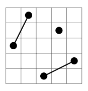
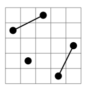
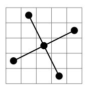
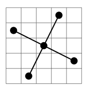
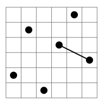
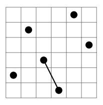
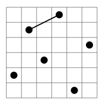
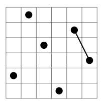
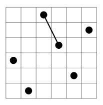
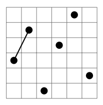
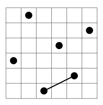
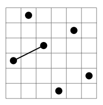
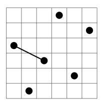
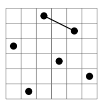
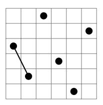
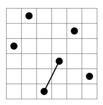
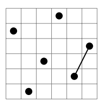
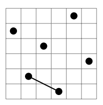
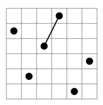
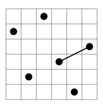
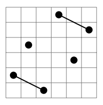
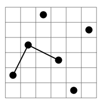
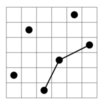
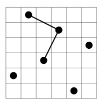
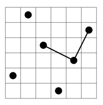
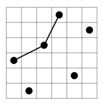
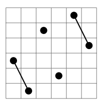
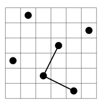
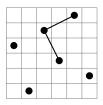
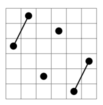
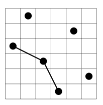
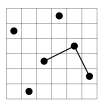
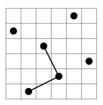
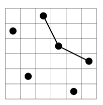
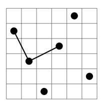
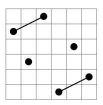
n	1 active knight	2 active knights	4 active knights
4			 2 4 1 3  3 1 4 2
5		 2 4 1 5 3  3 1 5 2 4  3 5 1 4 2  4 2 5 1 3	 2 5 3 1 4  4 1 3 5 2
6	 2 5 1 4 6 3  2 5 3 1 6 4  2 5 3 6 1 4  2 6 4 1 5 3  3 1 6 4 2 5  3 5 1 4 6 2  3 6 1 4 2 5  3 6 4 1 5 2  4 1 3 6 2 5  4 1 6 3 5 2  4 2 6 3 1 5  4 6 1 3 5 2  5 1 3 6 2 4  5 2 4 1 6 3  5 2 4 6 1 3  5 2 6 3 1 4	 2 4 1 6 3 5  2 4 6 3 1 5  2 5 1 3 6 4  2 6 3 5 1 4  2 6 4 1 3 5  3 1 4 6 2 5  3 1 5 2 6 4  3 6 2 4 1 5  4 1 5 3 6 2  4 6 2 5 1 3  4 6 3 1 5 2  5 1 3 6 4 2  5 1 4 2 6 3  5 2 6 4 1 3  5 3 1 4 6 2  5 3 6 1 4 2	

Table 2.2: The simple pin-permutations of size $n \leq 6$ and their active knights.

For each $n > 6$, all simple pin-permutations of size n have exactly only one active knight, except twelve of them that have two active knights, and that are:

- *the four oscillations of size n,*
- *the eight quasi-oscillations of size n.*

We are now able to state Theorem 2.48 about pin words of simple permutations:

Theorem 2.48. *A simple permutation has at most* 48 *pin words, which are all strict or quasi-strict.*

Proof. Let π be a simple permutation. Then $|\pi| \geq 3$ and any pin representation p of π is proper (see Remark 2.2 p.56), so p_3 is a separating pin and p is associated to 6 pin words by Remark 2.4 (p.57).

Moreover by Lemma 2.47 π has at most 4 active knights, thus by Lemma 2.44 there are at most 8 possible ordered pairs (p_1, p_2) beginning a pin representation of π. Furthermore from Lemma 2.42, each of these beginnings gives at most one pin representation p of π. So π has at most 8 pin representations and at most 48 pin words. Finally the first statement of Remark 2.8 (p.58) ensures that they are all strict or quasi-strict, since the pin representations they encode are proper. ■

Theorem 2.48 of course does not describe the set $P(\pi)$ of pin words encoding a simple pin-permutation π explicitly, but Algorithm 3 explains how to compute $P(\pi)$ in this case.

Algorithm 3: Pinwords function

```
input  : a simple permutation $\pi$
output: The set $P$ of pin words encoding $\pi$
// Count the number of ordered pairs of points in knight position
$E \leftarrow \varnothing$;
foreach $\pi_i$ do
    $E \leftarrow E \bigcup \{(\pi_i, \pi_j)$ in knight position$\}$
// If more than 48 pairs are found, $\pi$ is not a pin-permutation
if $|E| > 48$ then
    return $\varnothing$
// Otherwise each knight may be the beginning of a pin representation
   of $\pi$
$P \leftarrow \varnothing$;
foreach $(\pi_i, \pi_j) \in E$ do
    $P \leftarrow P \bigcup \{$ pin words of the pin representation beginning with $(\pi_i, \pi_j)\}$
return $P$
```

Lemma 2.49. *Algorithm 3 computes the set of pin words encoding a simple permutation π in linear time with respect to the length n of π.*

Proof. Algorithm 3 can be decomposed into two parts. First, we count the number of ordered pairs of points in knight position that should be smaller than 48. Indeed from Lemma 2.45, if π is a simple pin-permutation of length n, in any of its pin representations $(p_1, \ldots, p_n)$, every unordered pair of points $\{p_i, p_j\}$ that is a knight contains at least one of the points p_1, p_2 or p_n. As only 8 points can be in knight position with a given point (see Figure 2.14), the permutation π has at most 24 unordered pairs of points in knight position, hence at most 48 ordered pairs (p_i, p_j) that are knights.

Therefore given a simple permutation π, we count the number of ordered pairs of points in knight position. To do this, we take each point p of the permutation and we check if another point is in knight position with p. As at most 8 cells can contain a point in knight position with p, this counting part runs in time $8n$.

If this number is greater than 48, π is not a pin-permutation. Otherwise, the second part of the algorithm computes, for each ordered pair of points in knight position, the pin representation beginning with it (if it exists) and its associated pin words. This can also be done in linear time from Lemma 2.42.

Finally by Remark 2.5 (p.57), computing all pin words can easily be done in linear time from the pin representation. ■

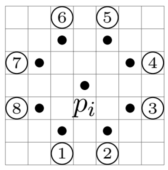


Figure 2.14: At most 8 points in knight position with a given point p_i.

2.4.4 Non-recursive case: decomposition trees with a linear root

W.l.o.g., since we focus on the non-recursive case, $\pi = \overset{\oplus}{\xi_1\ \xi_2 \cdots \xi_r}$ where ξ_i are increasing oscillations (see Remark 2.36 p.70). Lemma 2.50 is a direct consequence of Lemma 2.40.

Lemma 2.50. *Let $p = (p_1, p_2, \dots, p_n)$ be a pin representation of π. The only child ξ_i which may be read in several pieces is the child ξ_{i_0} to which p_1 belongs. Moreover if p reads ξ_{i_0} in several pieces, it is read in two pieces, the second child ξ_i read by p is either ξ_{i_0-1} or ξ_{i_0+1} and is a leaf, denoted x. Finally, the set $E = \xi_{i_0} \cup \{x\}$ is read in one piece by p.*

Lemma 2.50 together with Lemma 2.37 leads to the following.

Consequence 2.51. *Every pin representation p of π begins by entirely reading two consecutive children of the root, say ξ_i and ξ_{i+1}, then p reads in one piece each of the others ξ_j. Moreover the children ξ_j for $j < i$ are read in decreasing order $(\xi_{i-1}, \xi_{i-2}, \dots, \xi_1)$ and the children ξ_j for $j > i+1$ are read in increasing order $(\xi_{i+2}, \xi_{i+3}, \dots, \xi_r)$.*

Consequence 2.51 implies that the restriction of p to each child ξ_j where $j < i$ (resp. $j > i+1$) is a pin representation of ξ_j whose origin lies in quadrant 1 (resp. 3) with respect to the bounding box of the set of points of ξ_j. Indeed p_1 and p_2 are in ξ_i or ξ_{i+1} thus they lie in quadrant 1 (resp. 3) with respect to ξ_j. Since only p_2 may separate p_0 from p_1, p_0 is also in quadrant 1 (resp. 3). Therefore we introduce the following functions $P^{(\ell)}$: for any increasing oscillation ξ, we denote by $P^{(\ell)}(\xi)$ the set of pin words that encode ξ and whose origin lies in quadrant $h = 1$ or 3 with respect to the points of ξ.

To characterize the pin words that encode a permutation $\pi = \oplus[\xi_1, \xi_2, \dots, \xi_r]$ where every ξ_i is an increasing oscillation, Consequence 2.51 leads us naturally to introduce the shuffle product of sequences. From the above discussion, Theorem 2.53 then follows, providing the desired characterization.

Definition 2.52. Let $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2, \dots, \mathsf{A}_q)$ and $\mathsf{B} = (\mathsf{B}_1, \mathsf{B}_2, \dots, \mathsf{B}_s)$ be two sequences of sets of words. The *shuffle product* $\mathsf{A} \sqcup\!\sqcup \mathsf{B}$ of A and B is defined as

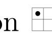
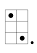
$$\begin{aligned}\mathsf{A} \sqcup\!\sqcup \mathsf{B} \quad &= \big\{c = c_1 \cdot \ldots \cdot c_{q+s} \mid \exists I = \{i_1, \dots, i_q\}, J = \{j_1, \dots, j_s\},\ I \cap J = \emptyset \\ &\quad \text{with } i_1 < \ldots < i_q, j_1 < \ldots < j_s,\ (c_{i_1}, \dots, c_{i_q}) \in \mathsf{A},\ (c_{j_1}, \dots, c_{j_s}) \in \mathsf{B}\big\}.\end{aligned}$$

Theorem 2.53. *The set $P(\pi)$ of pin words of a permutation $\pi = \oplus[\xi_1, \dots, \xi_j]$ where every ξ_i is an increasing oscillation is:*

$$P(\pi) = \bigcup_{1 \le i \le r-1} P(\oplus[\xi_i, \xi_{i+1}]) \cdot \Big((P^{(1)}(\xi_{i-1}), \dots, P^{(1)}(\xi_1)) \sqcup\!\sqcup (P^{(3)}(\xi_{i+2}), \dots, P^{(3)}(\xi_j)) \Big).$$

Lemmas 2.54 and 2.59 below give explicit expressions for $P^{(1)}(\xi)$, $P^{(3)}(\xi)$ and $P(\oplus[\xi_i, \xi_j])$ for every increasing oscillations ξ, ξ_i and ξ_j, hence with Theorem 2.53 an explicit expression for $P(\pi)$. Note that similar results can be obtained for permutations with root $\ominus$ and decreasing oscillations using Remark 2.36 (p.70).

We first recall a few definitions and facts about increasing oscillations.

Lemma 2.47 (p.74) describes the active pairs of simple pin-permutations (that are necessarily in knight position). In particular, an increasing oscillation of size at least 5 has exactly two active knights. They are located at both ends of the main diagonal and they consist of two points in relative order 21 (see Figure 2.15). These active knights are either in horizontal (H) position  or in vertical (V) position . Therefore there are four types of increasing oscillations: (x, y) with $x, y \in \{H, V\}$, where x is the type of the lower left active knight and y for the upper right. This definition can be extended to increasing oscillations of size 4, considering their two active knights in relative order 21 (see Figure 2.15). Note that an even size oscillation has type (H, H) or (V, V) and an odd size one (H, V) or (V, H). An comprehensive study of the different cases illustrated in Figure 2.15 leads to the following statement.

Lemma 2.54. *Let ξ be an increasing oscillation of size $n \ge 5$.*

If n is even, let $n = 2p + 2$, then

$$P^{(1)}(\xi) = \begin{cases} 3L(DL)^p & \text{if } \xi \text{ has type } (H,H) \\ 3D(LD)^p & \text{if } \xi \text{ has type } (V,V) \end{cases} \quad P^{(3)}(\xi) = \begin{cases} 1R(UR)^p & \text{if } \xi \text{ has type } (H,H) \\ 1U(RU)^p & \text{if } \xi \text{ has type } (V,V). \end{cases}$$

If n is odd, let $n = 2p + 1$, then

$$P^{(1)}(\xi) = \begin{cases} 3(DL)^p & \text{if } \xi \text{ has type } (H,V) \\ 3(LD)^p & \text{if } \xi \text{ has type } (V,H) \end{cases} \quad P^{(3)}(\xi) = \begin{cases} 1(RU)^p & \text{if } \xi \text{ has type } (H,V) \\ 1(UR)^p & \text{if } \xi \text{ has type } (V,H). \end{cases}$$

For the increasing oscillations of size less than 4, the values of $P^{(1)}$ and $P^{(3)}$ are:

$P^{(1)}(1) = 3$	$P^{(3)}(1) = 1$	$P^{(1)}(21) = \{3D, 3L\}$	$P^{(3)}(21) = \{1R, 1U\}$
$P^{(1)}(231) = 3DL$	$P^{(3)}(231) = 1RU$	$P^{(1)}(312) = 3LD$	$P^{(3)}(312) = 1UR$
$P^{(1)}(2413) = 3LDL$	$P^{(3)}(2413) = 1RUR$	$P^{(1)}(3142) = 3DLD$	$P^{(3)}(3142) = 1URU$

Remark 2.55. If the increasing oscillation ξ is of size 2 then $P^{(\ell)}(\xi)$ contains two words, otherwise it is a singleton. Moreover, for the map ϕ studied in Section 2.3 (see Definition 2.18 p.62), and for any increasing oscillation ξ, we have $\phi(P^{(3)}(\xi)) \subseteq \{U, R\}^\star$ and $\phi(P^{(1)}(\xi)) \subseteq \{L, D\}^\star$.

We are further interested in describing the set of pin words of $\oplus[\xi_i, \xi_j]$ for any increasing oscillations ξ_i and ξ_j. This is achieved in Lemma 2.59. For this purpose, we first describe the set $P(\xi)$ of pin words of any increasing oscillation ξ, and the set $P^{mix}(\xi_i, \xi_j)$ of pin words of $\oplus[\xi_i, \xi_j]$ such that one of the two oscillations is read in two pieces.

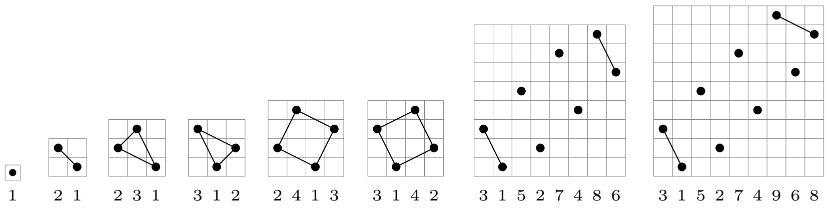


Figure 2.15: The increasing oscillations of size less than 5 and two increasing oscillations respectively of size 8 with type (V, V) and 9 with type (V, H). Active pairs are marked by edges between their two active points.

Lemma 2.56. *Let Q^- (resp. S_H^-, resp. S_V^-) be the set of pin words of the permutation* 21 *that are quasi-strict (resp. that are strict and end with R or L, resp. with U or D): $Q^- = \{12, 14, 22, 24, 32, 34, 42, 44\}$, $S_H^- = \{1R, 2R, 3L, 4L\}$ and $S_V^- = \{1U, 2D, 3D, 4U\}$. Define similarly Q^+ (resp. S_H^+, resp. S_V^+) for the permutation* 12.

Let ξ be an increasing oscillation of size $n \geq 5$.

If n is even, let $n = 2p + 2$, then

$$P(\xi) = \begin{cases} (Q^- + S_H^-) \cdot (DL)^p + (Q^- + S_H^-) \cdot (UR)^p & \text{if } \xi \text{ has type } (H, H) \\ (Q^- + S_V^-) \cdot (LD)^p + (Q^- + S_V^-) \cdot (RU)^p & \text{if } \xi \text{ has type } (V, V). \end{cases}$$

If n is odd, let $n = 2p + 1$, then

$$P(\xi) = \begin{cases} (Q^- + S_V^-) \cdot L(DL)^{p-1} + (Q^- + S_H^-) \cdot U(RU)^{p-1} & \text{if } \xi \text{ has type } (H, V) \\ (Q^- + S_H^-) \cdot D(LD)^{p-1} + (Q^- + S_V^-) \cdot R(UR)^{p-1} & \text{if } \xi \text{ has type } (V, H). \end{cases}$$

For the increasing oscillations of size less than 5, we have:

$P(1) = \{1, 2, 3, 4\} \qquad P(21) = Q^- + S_H^- + S_V^-$
$P(231) = (Q^- + S_H^-) \cdot U + (Q^- + S_V^-) \cdot L + (Q^+ + S_H^+ + S_V^+) \cdot 4$
$P(312) = (Q^- + S_H^-) \cdot D + (Q^- + S_V^-) \cdot R + (Q^+ + S_H^+ + S_V^+) \cdot 2$
$P(2413) = (Q^- + S_H^-) \cdot (UR + DL) + (Q^+ + S_V^+) \cdot (RD + LU)$
$P(3142) = (Q^+ + S_H^+) \cdot (UL + DR) + (Q^- + S_V^-) \cdot (RU + LD)$

In particular, $|P(\xi)| \leq 48$ for any increasing oscillation ξ and if $|\xi| \neq 3$, $P(\xi)$ contains only strict and quasi-strict pin words.

Proof. By comprehensive examination of the cases illustrated in Figure 2.15. ■

Definition 2.57. For any pair of increasing oscillations (ξ_i, ξ_j), we denote by $P^{mix}(\xi_i, \xi_j)$ the set of pin words encoding a pin representation of $\oplus[\xi_i, \xi_j]$ that reads one of the two oscillations in two pieces.

Lemma 2.58. *Let ξ_i and ξ_j be two increasing oscillations.*

If none of these two oscillations is of size 1, or if both of them are of size 1, then $P^{mix}(\xi_i, \xi_j)$ is empty.

Otherwise, assume w.l.o.g. that $\xi_i = 1$, and set $|\xi_j| = 2p + q + 1$ with $q \in \{0, 1\}$. If $|\xi_j| \geq 4$, then

$$P^{mix}(\xi_i, \xi_j) = \begin{cases} (13 + 23 + 33 + 43 + 1D + 4D) \cdot (RU)^p R^q & \text{if } \xi \text{ has type } (H, H) \text{ or } (H, V) \\ (13 + 23 + 33 + 43 + 1L + 2L) \cdot (UR)^p U^q & \text{if } \xi \text{ has type } (V, V) \text{ or } (V, H). \end{cases}$$

If $|\xi_j| = 3$, i.e., $\xi_j = 231$ or 312, we have

$$P^{mix}(1, 231) = P(12) \cdot 3R + (13 + 23 + 33 + 43 + 1D + 4D) \cdot RU,$$
$$P^{mix}(1, 312) = P(12) \cdot 3U + (13 + 23 + 33 + 43 + 1L + 2L) \cdot UR.$$

If $|\xi_j| = 2$, i.e., $\xi_j = 21$, we have
$P^{mix}(1, 21) = (13 + 23 + 33 + 43 + 1D + 4D) \cdot R + (13 + 23 + 33 + 43 + 1L + 2L) \cdot U.$
In particular, $|P^{mix}(\xi_i, \xi_j)| \leq 22$ for any increasing oscillations ξ_i and ξ_j.

Proof. From Lemma 2.50 (p.76) when one of the oscillations ξ_i or ξ_j is read in two pieces, then the other one has size 1. W.l.o.g. assume that $|\xi_i| = 1$. Then from Lemma 2.40 (p.71) the decomposition tree of $\oplus[\xi_i, \xi_j]$ has one of the shapes of Figure 2.11 (p.71), with T_{i_0} corresponding to ξ_j and x corresponding to ξ_i.

If ξ_j has size 2, then $\xi_j = 21$ and $\oplus[\xi_i, \xi_j]$ maps only to configurations $G1$ and $G2$ (see Figures 2.11 and 2.12). Therefore there are two pin representations of $\oplus[\xi_i, \xi_j]$ where ξ_j is read in two pieces. The twelve corresponding pin words (see Remark 2.4 p.57) are those given in Lemma 2.58.

If ξ_j has size 3, then $\xi_j = 231$ (resp. 312) and $\oplus[\xi_i, \xi_j]$ maps only to configurations $G2$ and $G4$ (resp. $G1$ and $G3$), and we conclude similarly.

Otherwise, $|\xi_j| \geq 4$. Since ξ_j is an increasing oscillation, $\oplus[\xi_i, \xi_j]$ maps only to configuration $G3+$ or to configuration $G4+$, with $|S| = 1$. These two cases are exclusive, and the configuration ($G3+$ or $G4+$) to which $\oplus[\xi_i, \xi_j]$ maps is determined by the type (V or H) of the lower left active knight of ξ_j. Lemma 2.40 and Remark 2.41 (p.72) ensure that there is exactly one pin representation for $\oplus[\xi_i, \xi_j]$ that reads ξ_j in two pieces. The six corresponding pin words are those given in Lemma 2.58. ■

From the expressions of $P, P^{mix}, P^{(1)}$ and $P^{(3)}$ we can deduce the explicit expression of $P(\oplus[\xi_i, \xi_j])$ making use of the following result:

Lemma 2.59. *For any pair of increasing oscillations (ξ_i, ξ_j):*

- *If $|\xi_i| > 1$ and $|\xi_j| > 1$, $P(\oplus[\xi_i, \xi_j]) = \left(P(\xi_j) \cdot P^{(1)}(\xi_i)\right) \bigcup \left(P(\xi_i) \cdot P^{(3)}(\xi_j)\right)$*
- *If $|\xi_i| = 1$ and $|\xi_j| = 1$, $P(\oplus[\xi_i, \xi_j]) = P(12)$*
- *Otherwise assume w.l.o.g. that $|\xi_i| = 1$ and $|\xi_j| = 2p + q$ with $q \in \{0, 1\}$:*
  $P(\oplus[\xi_i, \xi_j]) = P^{mix}(\xi_i, \xi_j) \bigcup \left(P(\xi_j) \cdot 3\right) \bigcup \left(\{1, 2, 3, 4\} \cdot P^{(3)}(\xi_j)\right) \bigcup P^{sep}(\xi_j)$
  $$\text{with } P^{sep}(\xi_j) = \begin{cases} (2 + 3) \cdot (UR)^p U^q & \text{if } \xi_j \text{ has type } (H, H) \text{ or } (H, V) \\ (3 + 4) \cdot (RU)^p R^q & \text{if } \xi_j \text{ has type } (V, V) \text{ or } (V, H). \end{cases}$$

In particular, $|P(\oplus[\xi_i, \xi_j])| \leq 192$ for any oscillations ξ_i and ξ_j.

Proof. For the first item, Lemma 2.58 ensures that the pin words of $\oplus[\xi_i, \xi_j]$ encode pin representations reading both ξ_i and ξ_j in one piece. The two terms of the union are obtained according to which oscillation (among ξ_i and ξ_j) is read first. The second statement follows directly from the fact that $\oplus[\xi_i, \xi_j] = 12$ in this case. From Lemma 2.58, the situation of the third statement is the one where there are pin words encoding pin representations reading ξ_j in two pieces. The four terms of the union account for four different kinds of pin words. Namely, $P^{mix}(\xi_i, \xi_j)$ is the set of pin words reading ξ_j in two pieces, $P(\xi_j) \cdot 3$ is the set of pin words reading first ξ_j and then ξ_i, $\{1, 2, 3, 4\} \cdot P^{(3)}(\xi_j)$ is the set of pin words reading first ξ_i and then ξ_j starting with an independent pin, and $P^{sep}(\xi_j)$ is the set of pin words reading first ξ_i and then ξ_j starting with a separating pin. Notice that in this last situation, the first pin of ξ_j may be separating only because $|\xi_i| = 1$, so that this case does not need to appear in the first item. ■

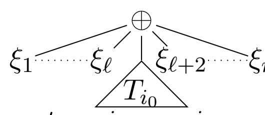
This completes the explicit description of all the sets of pin words appearing in Theorem 2.53.

2.4.5 Recursive case: decomposition trees with a linear root

In this section we focus on pin-permutations whose decomposition trees have a linear root $\oplus$ and a child T_{i_0} which is not an increasing oscillation. From Lemma 2.33, T_{i_0} is then the first child read by any pin representation. Lemma 2.40 (p.71) gives a characterization of permutations in which T_{i_0} may be read in several pieces. Moreover from Remark 2.41 if T_{i_0} is read in two pieces, the first part S being fixed, then the order of the points of the remaining part is uniquely determined. Nevertheless, since some permutations may satisfy several conditions $F1$ to $G4+$ of Lemma 2.40, the first part S to be read is not uniquely determined. For example every permutation satisfying $F3$ also satisfies $F1$, and some permutations satisfy both $F1$ and $G2$ (see Figure 2.12 p.73). In Figure 2.16 we classify the permutations according to the conditions they satisfy.

Let $\mathcal{H}$ be the set of permutations in which T_{i_0} may be read in several pieces. Then any permutation of $\mathcal{H}$ satisfies exactly one of the conditions (iHj) of Figure 2.16. We say that a permutation satisfies condition (iHj) when its diagram has the corresponding shape in Figure 2.16 (up to symmetry) and does not satisfy any condition that appears above (iHj) in Figure 2.16. For example a permutation in $(1H2)$ cannot be in $(2H2)$. One can check by a comprehensive verification that there is no other combination (up to symmetry) of the conditions $F1$ to $G4+$ of Lemma 2.40. Moreover as T_{i_0} is not an increasing oscillation, the sets S and T that appear on Figure 2.16 are such that $|S| \geq 2$ and $|T| \geq 1$.

Recall that $P(\pi)$ denotes the set of pin words encoding π. Let us also denote by $P(T)$ the set of pin words that encode the permutation whose decomposition tree is T.

Theorem 2.60. *Let $\pi =$*  *be a $\oplus$-decomposable permutation where T_{i_0} is the only child that is not an increasing oscillation.*

For every i such that $1 \leq i \leq \ell$ and j such that $\ell+2 \leq j \leq r$, set

$$\mathfrak{P}^{(1)}_{(i)} = \big(P^{(1)}(\xi_i), \dots, P^{(1)}(\xi_1)\big) \text{ and } \mathfrak{P}^{(3)}_{(j)} = \big(P^{(3)}(\xi_j), \dots, P^{(3)}(\xi_r)\big).$$

We describe below the set $P(\pi)$ of pin words encoding π. When $\pi \in \mathcal{H}$, these sets are given when the diagram of π is one of those shown in Figure 2.16. When the diagram of π is one of their symmetries, $P(\pi)$ is modified accordingly.

- *If $\pi \notin \mathcal{H}$ (i.e., if π does not satisfy any condition shown up to symmetry on Figure 2.16) then $P(\pi) = P_0 = P(T_{i_0}) \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+2)}$.*
- *If π satisfies condition $(1H1)$ then $P(\pi) = P_0 \cup P_1$, with*

$$P_1 = \underbrace{P(S)\cdot\underbrace{1}_{x}\cdot\underbrace{L}_{a}}_{x\bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)}.$$

- *If π satisfies condition $(1H1+)$ then $P(\pi) = P_0 \cup P_1$, with*

$$P_1 = \underbrace{P(S)\cdot\underbrace{1}_{x}\cdot w}_{x\bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)},$$

where w is the unique word encoding the unique reading of the remaining leaves of T_{i_0}. Notice that w is obtained from the unique word of $P^{(1)}(\xi)$ (see Remark 2.55 p.77) by deleting its first letter.

- *If π satisfies condition $(1H2\star)$ then $P(\pi) = P_0 \cup P_1 \cup P_2$, with*

$$P_1 = \underbrace{P(S)\cdot\underbrace{1}_{x}\cdot\underbrace{D}_{b}\cdot\underbrace{L}_{a}}_{x\bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)} \text{ and } P_2 = \underbrace{P(S')\cdot\underbrace{1}_{x}\cdot\underbrace{D}_{b}}_{x\bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)}.$$

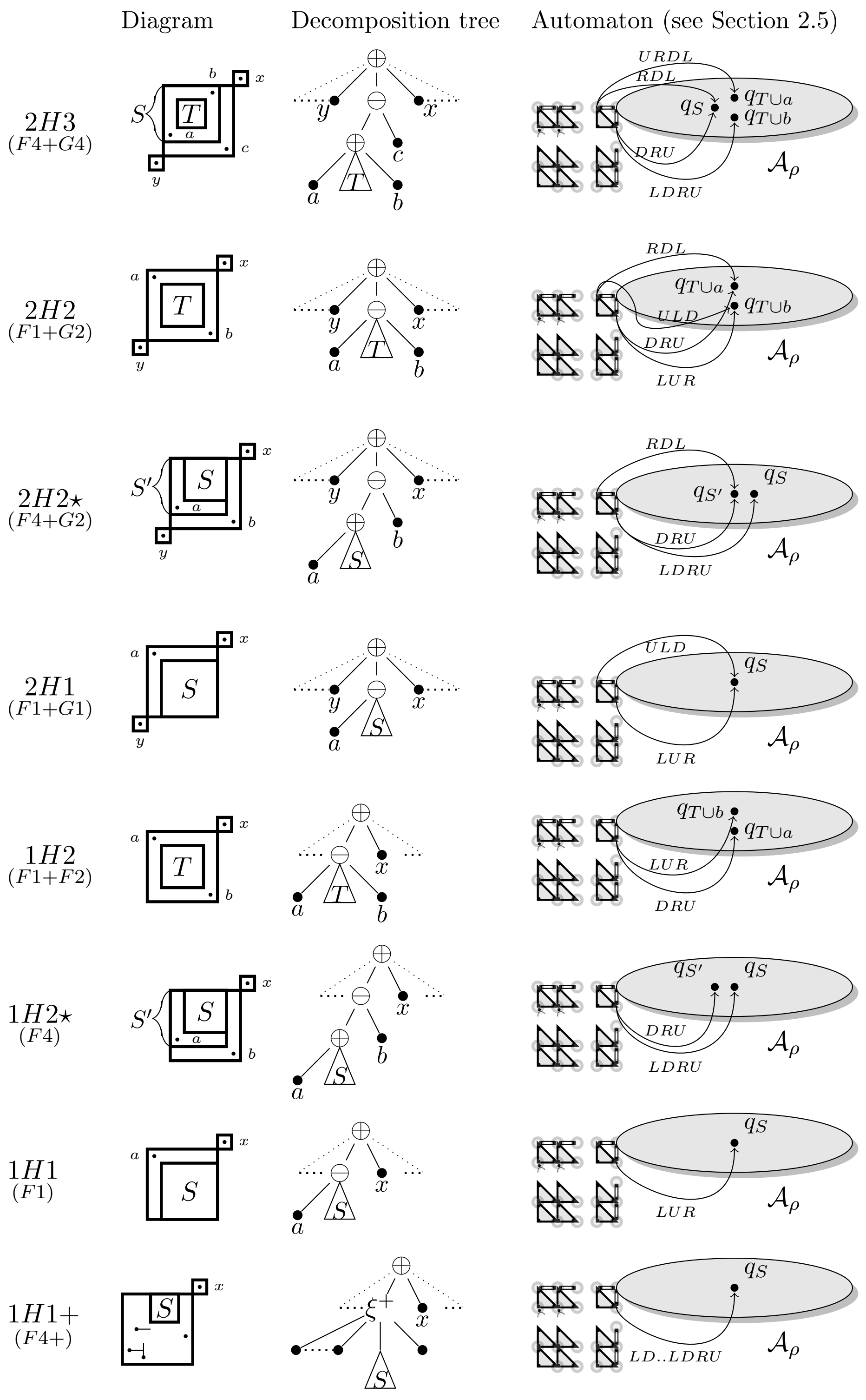


Figure 2.16: The set $\mathcal{H}$ and conditions (iHj): $\pi \in \mathcal{H}$ if and only if π satisfies one of the conditions (iHj) shown above up to symmetry, that form a partition of $\mathcal{H}$.

- *If π satisfies condition $(1H2)$ then $P(\pi) = P_0 \cup P_1 \cup P_2$, with*

$$P_1 = \underbrace{P(T\cup a)\cdot \underbrace{1}_{x} \cdot \underbrace{D}_{b}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)} \text{ and } P_2 = \underbrace{P(T\cup b)\cdot \underbrace{1}_{x} \cdot \underbrace{L}_{a}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)}.$$

- *If π satisfies condition $(2H1)$ then $P(\pi) = P_0 \cup P_1 \cup P_2$, with*

$$P_1 = \underbrace{P(S)\cdot \underbrace{1}_{x} \cdot \underbrace{L}_{a}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)} \text{ and } P_2 = \underbrace{P(S)\cdot \underbrace{3}_{y} \cdot \underbrace{U}_{a}}_{y \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell-1)} \shuffle \mathfrak{P}^{(3)}_{(\ell+2)}.$$

- *If π satisfies condition $(2H2\star)$ then $P(\pi) = P_0 \cup P_1 \cup P_2 \cup P_3$, with*

$$P_1 = \underbrace{P(S)\cdot \underbrace{1}_{x} \cdot \underbrace{D}_{b} \cdot \underbrace{L}_{a}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)},\ P_2 = \underbrace{P(S')\cdot \underbrace{1}_{x} \cdot \underbrace{D}_{b}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)}$$

$$\text{and } P_3 = \underbrace{P(S')\cdot \underbrace{3}_{y} \cdot \underbrace{R}_{b}}_{y \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell-1)} \shuffle \mathfrak{P}^{(3)}_{(\ell+2)}.$$

- *If π satisfies condition $(2H2)$ then $P(\pi) = P_0 \cup P_1 \cup P_2 \cup P_3 \cup P_4$, with*

$$P_1 = \underbrace{P(T\cup a)\cdot \underbrace{1}_{x} \cdot \underbrace{D}_{b}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)},\ P_2 = \underbrace{P(T\cup b)\cdot \underbrace{1}_{x} \cdot \underbrace{L}_{a}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)},$$

$$P_3 = \underbrace{P(T\cup a)\cdot \underbrace{3}_{y} \cdot \underbrace{R}_{b}}_{y \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell-1)} \shuffle \mathfrak{P}^{(3)}_{(\ell+2)},\ P_4 = \underbrace{P(T\cup b)\cdot \underbrace{3}_{y} \cdot \underbrace{U}_{a}}_{y \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell-1)} \shuffle \mathfrak{P}^{(3)}_{(\ell+2)}.$$

- *If π satisfies condition $(2H3)$ then $P(\pi) = P_0 \cup P_1 \cup P_2 \cup P_3 \cup P_4$, with*

$$P_1 = \underbrace{P(S)\cdot \underbrace{1}_{x} \cdot \underbrace{D}_{c}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)},\ P_2 = \underbrace{P(T\cup b)\cdot \underbrace{1}_{x} \cdot \underbrace{D}_{c} \cdot \underbrace{L}_{a}}_{x \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell)} \shuffle \mathfrak{P}^{(3)}_{(\ell+3)},$$

$$P_3 = \underbrace{P(T\cup a)\cdot \underbrace{3}_{y} \cdot \underbrace{R}_{c} \cdot \underbrace{U}_{b}}_{y \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell-1)} \shuffle \mathfrak{P}^{(3)}_{(\ell+2)},\ P_4 = \underbrace{P(S)\cdot \underbrace{3}_{y} \cdot \underbrace{R}_{c}}_{y \bigcup T_{i_0}} \cdot \mathfrak{P}^{(1)}_{(\ell-1)} \shuffle \mathfrak{P}^{(3)}_{(\ell+2)}.$$

Proof. For each item, it is easy to check that the given pin words are pin words encoding π. Conversely, we prove that a pin word encoding π is necessarily in the set claimed to be $P(\pi)$. First of all, by Lemma 2.33, every pin representation of π starts in the only child that is not an increasing oscillation, i.e., with T_{i_0}.

Let us start with the first point of Theorem 2.60. In this case, by definition of $\mathcal{H}$, we know that T_{i_0} is read in one piece. By Lemma 2.39 (p.71), the other children are also read in one piece, and Lemma 2.37 ensures that the children closest to T_{i_0} are read first. As there is no relative order between children $\xi_{\ell+2}$ to ξ_r and children ξ_ℓ to ξ_1, this leads to the shuffling operation between pin words corresponding to these children, with an external origin placed in quadrant 3 (resp. 1) with respect to the block.

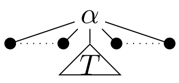
In the other cases of Theorem 2.60, by Lemma 2.40, every pin representation of π either reads T_{i_0} in one piece or in two pieces. In case T_{i_0} is read in one piece, the pin representation is as before encoded by pin words of P_0. If it is read in two pieces, Lemma 2.40 and its proof and Remark 2.41 ensure that the corresponding pin words are those described.

Consider for example π satisfying condition $(1H1)$. Then π satisfies condition $F1$ of Lemma 2.40, and only this one by definition of $(1H1)$. If T_{i_0} is read in two pieces, then Lemma 2.40 ensures that S is the first part of $T_{i_0} \cup \{x\}$ to be read, followed by x and finally a. The corresponding pin words are indeed those described in P_1.

Taking the other example of condition $(2H3)$, P_1 corresponds to condition $F2$ with $S = T \cup a \cup b$, P_2 corresponds to condition $F4$ with $S = T \cup b$, P_3 corresponds to condition $G4$ with $S = T \cup a$ and P_4 corresponds to condition $G2$ with $S = T \cup a \cup b$. ■

Remark 2.61. If π is a $\ominus$-decomposable permutation, similar description of $P(\pi)$ can be obtained from Remark 2.36 (p.70).

2.4.6 Recursive case: decomposition trees with a prime root

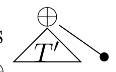
We now turn to the study of the recursive case where the decomposition tree has a root which is a simple permutation α. We start with the case where $\pi =$ 

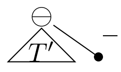
for a tree T that is not a leaf.

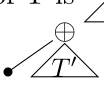
We begin with the characterization of the possible ways a pin representation of π may read T, introducing first a condition that will be useful in the sequel.

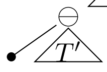
Definition 2.62. For a permutation $\pi = \alpha[1, \ldots, 1, T, 1, \ldots, 1]$ with $\alpha = \alpha_1 \ldots \alpha_k$, we define condition $(\mathcal{C})$ as follows:

$(\mathcal{C})$

- • α is an increasing – resp. decreasing – quasi-oscillation (see p.67);
- • T expands an auxiliary point of α;
- • the shape of T is 

– resp. 

– if the auxiliary point is α_k or α_{k-1} and 

– resp. 

– if the auxiliary point is α_1 or α_2.

Lemma 2.63. *Let $p = (p_1, \ldots, p_n)$ be a pin representation associated to $\pi = \alpha[1, \ldots, 1, T, 1, \ldots, 1]$. Then one of the following statements holds:*

(1) *$p_1 \in T$ and T is read in one piece by p;*

(2) *$T = \{p_1, \ldots, p_i\} \bigcup \{p_n\}$ with $i \neq n-1$, and π satisfies condition $(\mathcal{C})$;*

(3) *$p_1 \notin T$, $T = \{p_2, p_n\}$, and π satisfies condition $(\mathcal{C})$.*

Moreover if (3) is satisfied then p is a proper pin representation uniquely determined by α and its auxiliary point; it is up to symmetry the one depicted in the first diagram of Figure 2.17. If (2) is satisfied, defining T' as in condition $(\mathcal{C})$, then $T' = \{p_1, \ldots, p_i\}$ and $(p_{i+1}, \ldots, p_n)$ is uniquely determined by α and its auxiliary point, as shown in the second diagram of Figure 2.17 up to symmetry.

Proof. If $p_1 \notin T$, then by Lemma 2.35*(ii)*, $T = \{p_2, p_n\}$. Up to symmetry assume that $\{p_1, p_2\}$ is an increasing subsequence of π. As $\{p_2, p_n\}$ forms a block, p_n is in one of the 4 positions shown in Figure 2.18. But position ③ is forbidden because it is inside of the bounding box $\mathcal{B}$ of $\{p_1, p_2\}$. Positions ② and ④ lie on the side of the bounding box $\mathcal{B}$. Thus, if p_n lies in one of these positions, it must be read immediately after p_1 and p_2 and thus must be p_3 from Lemma 2.1 (p.56). But $n > 3$ (α is simple so $|\alpha| \geq 4$) so that these positions are forbidden. Hence p_n lies in position ① and $T = 12$.

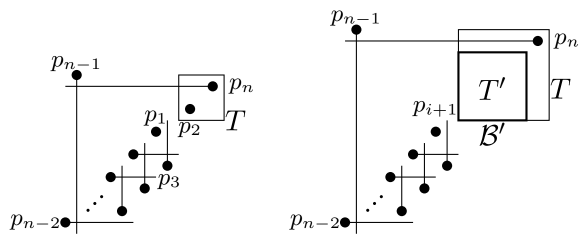


Figure 2.17: Diagram of π when one child T is not a leaf, is read in two pieces and $p_1 \notin T$ or $p_1 \in T$.

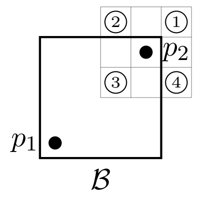
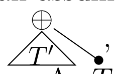


Figure 2.18: Possible positions for p_n.

As α is a simple pin-permutation, p_3 respects the separability condition. But if p_3 lies above or on the right of the bounding box $\mathcal{B}$ then p_n will be on the side of the bounding box of $\{p_1, p_2, p_3\}$, hence $p_n = p_4$. But in that case, α has only 3 children, hence $|\alpha| = 3$, contradicting the fact that α is simple.

By symmetry we can assume that p_3 lies below $\mathcal{B}$ (see the first diagram of Figure 2.17). The same argument goes for every pin p_i with $i = 3, \dots, n-2$ and these pins form an alternating sequence of left and down pins. As p_n separates p_{n-1} from all other pins, p_{n-1} must be an up or right pin (depending on the parity of n). Then α is a quasi-oscillation in which the point expanded by T is an auxiliary point and $T = 12$ or $T = 21$ depending on the nature of α – increasing or decreasing. Consequently, π satisfies condition $(\mathcal{C})$. Notice that given α and its auxiliary point, once we know that $p_1 \notin T$ then p is uniquely determined.

Suppose now that $p_1 \in T$ but T is not read in one piece. By Lemma 2.34*(i)*, it is read in two pieces, the second part being p_n. Then $T = \{p_1, \dots, p_i\} \bigcup \{p_n\}$ with $n \neq i+1$. Thus p_n does not lie on the sides of the bounding box $\mathcal{B}'$ of $\{p_1, \dots, p_i\}$. Up to symmetry, we can assume that p_n lies in quadrant 1 with respect to $\mathcal{B}'$ (see Figure 2.17). Therefore, $T =$ 

, T' being the sub-forest of T whose leaves are the points in $\mathcal{B}'$, i.e., are $p_1, \dots, p_i$. As T is a block, no pin must lie on the sides of the bounding box of $\{p_1, \dots, p_i, p_n\}$. Moreover p_{i+1} does not lie in quadrant 1 with respect to T, otherwise p_n would lie inside the bounding box of $\{p_1, \dots, p_{i+1}\}$. If p_{i+1} lies in quadrant 2 or 4, p_n would lie on the side of the bounding box of $\{p_1, \dots, p_{i+1}\}$ and thus must be p_{i+2}. This is in contradiction with α being simple. Thus p_{i+1} lies in quadrant 3 with respect to T. The same goes for p_j with $j \in \{i+1, \dots, n-2\}$. Because α is simple, we therefore deduce that all these pins form an alternating sequence of left and down pins until p_{n-1} which must be an up or right pin depending on the parity of n. Thus α is a quasi-oscillation in which the point expanded by T is an auxiliary point. Moreover, π satisfies condition $(\mathcal{C})$ with $T' = \{p_1, \dots, p_i\}$, as can be seen (up to symmetry) on the right part of Figure 2.17. Notice that given α and its auxiliary point, once we know that $T = \{p_1, \dots, p_i\} \bigcup \{p_n\}$ with $i \neq n-1$ then $(p_{i+1}, \dots, p_n)$ is uniquely determined.

Finally if we are not in one of the two cases discussed above, then $p_1 \in T$ and T is read in one piece by p, concluding the proof. ■

With the description of the pin representations of π in Lemma 2.63 and the definition that follows, we are able to give in Theorem 2.66 below an explicit description of the set $P(\pi)$ of pin words that encode π.

Definition 2.64. For every simple pin-permutation α, with an active point x (see p.74) marked, we define $Q_x(\alpha)$ as the set of strict pin words obtained by deleting the first letter of a quasi-strict pin word of α whose first point read in α is x.

Notice that $|u| = |\alpha| - 1$ for all $u \in Q_x(\alpha)$.

Remark 2.65. To each pin representation of α whose first point read is x corresponds exactly one word of $Q_x(\alpha)$. Indeed the quasi-strict pin words associated to a pin representation differ only in their first letter (see Figure 2.5 p.57 and Remark 2.4 p.57).

Theorem 2.66. *Let π be a pin-permutation, whose decomposition tree has a prime root α, with exactly one child T that is not a leaf. Then, denoting by x the point of α expanded by T, the following holds:*

- *If π does not satisfy condition $(\mathcal{C})$, then $P(\pi) = P(T) \cdot Q_x(\alpha)$.*
- *If π satisfies condition $(\mathcal{C})$, we define T' as in $(\mathcal{C})$, and we distinguish two sub-cases according to the number of leaves $|T|$ of T:*

  (a) *if $|T| \geq 3$, let w be the unique word encoding the unique reading of the remaining leaves in π after T' is read when T is read in two pieces. Then $P(\pi) = P(T) \cdot Q_x(\alpha) \cup P(T') \cdot w$.*

  (b) *if $|T| = 2$, let $P_{\{1,n\}}(\pi)$ be the set of pin words encoding the unique pin representation p of π such that $T = \{p_1, p_n\}$. Define similarly $P_{\{2,n\}}(\pi)$ for the case $T = \{p_2, p_n\}$. Then $P(\pi) = P(T) \cdot Q_x(\alpha) \cup P_{\{1,n\}}(\pi) \cup P_{\{2,n\}}(\pi)$.*

Proof. In each case, it is easy to check that the given pin words are pin words encoding π. Conversely, we prove that a pin word encoding π is necessarily in the set claimed to be $P(\pi)$. Let $u = u_1 \dots u_n \in P(\pi)$ and $p = (p_1, \dots, p_n)$ be the associated pin representation. Then p satisfies one statement of Lemma 2.63.

If p satisfies statement (1) of Lemma 2.63 then, setting $k = |T|$, $(p_k, \dots, p_n)$ is a pin representation of α beginning with x. Moreover as T is a block of π, p_{k+1} is an independent pin, so that u_{k+1} is a numeral. Thus for all h in $\{1, 2, 3, 4\}$, $h\, u_{k+1} \dots u_n$ is a pin word encoding α and starting with two numerals. As α is simple, its pin words are strict or quasi-strict, hence $h\, u_{k+1} \dots u_n$ is quasi-strict. Therefore $u_{k+1} \dots u_n \in Q_x(\alpha)$. Moreover $(p_1, \dots, p_k)$ is a pin representation of T. Hence $u \in P(T) \cdot Q_x(\alpha)$ which is included in the set claimed to be $P(\pi)$, regardless of whether π satisfies condition $(\mathcal{C})$ or not.

If p satisfies statement (2) of Lemma 2.63 then π satisfies condition $(\mathcal{C})$. If $|T| = 2$ then $T = \{p_1, p_n\}$ and $u \in P_{\{1,n\}}(\pi)$. Notice that the uniqueness of the pin representation such that $T = \{p_1, p_n\}$ follows from Lemma 2.63. Indeed in this case $(p_{i+1}, \dots, p_n)$ is uniquely determined, $i = 1$ and p_1 is the only remaining point. If $|T| \geq 3$ then from Lemma 2.63, $T' = \{p_1, \dots, p_{k-1}\}$ with $k = |T|$ thus the prefix of length $k - 1$ of u is in $P(T')$. From Lemma 2.63, $(p_k, \dots, p_n)$ is uniquely determined. Moreover, as $k \geq 3$, Remark 2.4 (p.57) ensures that the letters encoding these points are uniquely determined. This allows to define uniquely the word w encoding $(p_k, \dots, p_n)$, yielding $u \in P(T') \cdot w$.

If p satisfies statement (3) of Lemma 2.63 then π satisfies condition $(\mathcal{C})$, $|T| = 2$ and $u \in P_{\{2,n\}}(\pi)$. Notice that the uniqueness of the pin representation such that $T = \{p_2, p_n\}$ follows from Lemma 2.63. ■

To make the set $P(\pi)$ of pin words in the statement of Theorem 2.66 explicit (up to the recursive parts $P(T)$ and $P(T')$), we conclude the study of the case $\pi = \alpha[1, \dots, 1, T, 1, \dots, 1]$ by stating some properties of the (sets of) words $Q_x(\alpha)$, w, $P_{\{1,n\}}(\pi)$ and $P_{\{2,n\}}(\pi)$ that appear in Theorem 2.66.

Remark 2.67. The set $Q_x(\alpha) \subseteq P(\alpha)$ can be determined in linear time w.r.t. $|\alpha|$. Indeed as α is simple it is sufficient to examine the proper pin representations of α which start with an active knight containing x. By Lemma 2.42 (p.73), these are entirely determined by their first two points. Since α is simple these two points are in knight position. Consequently, there are at most 8 proper pin representations of α starting with x, and associated pin words are obtained in linear time using Remark 2.4 (p.57).

Lemma 2.68. *In Theorem 2.66, when π satisfies condition $(\mathcal{C})$ and $|T| \geq 3$, the word w is a strict pin word of size at least 4 encoding α. Denoting by w' the suffix of length 2 of w, then $P(T') \cdot \phi^{-1}(w') \subseteq P(T)$. Moreover there exist a word $\bar{w}$ of $Q_x(\alpha)$ and a letter Z such that $w = \bar{w} \cdot Z$ and no word of $\phi(Q_x(\alpha))$ contains Z. Finally when $|\alpha| \geq 5$ then $Q_x(\alpha)$ contains only $\bar{w}$. Otherwise $|\alpha| = 4$ and $Q_x(\alpha)$ contains two words.*

Proof. Assume that π satisfies condition $(\mathcal{C})$ and $|T| \geq 3$. Define T' as in condition $(\mathcal{C})$ and let $i = |T'|$. Notice that $i \geq 2$. By definition of w there exists a pin representation $p = (p_1, \ldots, p_n)$ of π such that $T' = \{p_1, \ldots, p_i\}$, T is read in two pieces and any corresponding pin word $u = u_1 \ldots u_n$ satisfies $u_{i+1} \ldots u_n = w$. Then p satisfies statement (2) of Lemma 2.63, thus $T = \{p_1, \ldots, p_i\} \cup \{p_n\}$ as in the second diagram of Figure 2.17 and $\{p_{i+2}, \ldots, p_n\}$ are separating pin. As $i \geq 2$ and T' is a block of π, p_{i+1} is an independent pin encoded with a numeral. So $w = u_{i+1} \ldots u_n$ is a strict pin word.

Moreover as $T = \{p_1, \ldots, p_i\} \cup \{p_n\}$, $(p_{i+1}, \ldots, p_n)$ is a pin representation of α ending with x thus w is a pin word encoding α and $|w| = |\alpha| \geq 4$. Likewise $(p_i, \ldots, p_{n-1})$ is a pin representation of α beginning with x.

Denoting by w' the suffix of length 2 of w, then from Lemma 2.19 (p.62) $\phi^{-1}(w')$ is a numeral indicating the quadrant in which p_n lies with respect to T'. And as $T = T' \cup \{p_n\}$, for all u' in $P(T')$, $u' \cdot \phi^{-1}(w')$ belongs to $P(T)$.

Moreover letting $\bar{w}$ be the prefix of w of length $|w| - 1$, for all h in $\{1, 2, 3, 4\}$, $h\,\bar{w}$ is a quasi-strict pin word of α. Therefore $\bar{w} \in Q_x(\alpha)$. Denoting by Z the last letter of w, Z encodes p_n. Moreover $\bar{w}$ encodes $p_{i+1}, \ldots, p_{n-1}$ and the position of $p_{i+1}, \ldots, p_n$ is the same (up to symmetry) as the one shown on Figure 2.17. On this figure, it is immediate to check that $\phi(\bar{w})$ does not contain Z. To prove that this holds not only for $\bar{w}$ but also for all words of $Q_x(\alpha)$, we first study the cardinality of $Q_x(\alpha)$.

From Remark 2.65 to each pin representation of α whose first point read is x corresponds exactly one word of $Q_x(\alpha)$. Recall from Remark 2.67 that a pin representation of α is determined by its first two points, which form an active knight. So we just have to compute the number of active knights of α to which x belongs, remembering that α is an increasing oscillation and x is an auxiliary point of α. By definition, every increasing oscillation has size at least 4, and we distinguish cases depending on whether it is of size greater than or at most 5.

Every increasing oscillation of size greater than 5 has exactly two active knights, with exactly one containing the auxiliary point (which is uniquely determined in this case). Indeed, from Lemma 2.47 (p.74), the main substitution point belongs to exactly two active knights – one formed with the auxiliary point and one formed with the point separating it from the auxiliary point (see the last diagram of Figure 2.6 p.67)– and there are no other active knights.

For increasing oscillations of size 4 or 5 (see Figure 2.7 p.68) we may also apply Lemma 2.47 to count their active knights that involve the auxiliary point x. An increasing oscillation of size 5 has exactly 4 active knights, all of them contain the main substitution point, and exactly one of them contains the auxiliary point x. An increasing oscillation of size 4 has exactly 4 active knights and each of its point (including the auxiliary point x) belongs to exactly two active knights.

Consequently if α is an increasing quasi-oscillation of size greater than 4 and x is an auxiliary point of α, then $|Q_x(\alpha)| = 1$ as x belongs to only one active knight; and when $|\alpha| = 4$, $|Q_x(\alpha)| = 2$ as x belongs to two active knights.

To conclude the proof, recall that the word $\bar{w}$ defined earlier belongs to $Q_x(\alpha)$ and is such that $\phi(\bar{w})$ does not contain Z. When $|\alpha| \neq 4$, we have $|Q_x(\alpha)| = 1$ so that $Q_x(\alpha) = \{\bar{w}\}$ and we conclude that no word of $\phi(Q_x(\alpha))$ contains Z. When $|\alpha| = 4$, $|Q_x(\alpha)| = 2$ and there is only one word $\bar{w}'$ different from $\bar{w}$ in $Q_x(\alpha)$, which may be computed from Figure 2.7 (p.68). We then check by comprehensive verification (of the four cases of size 4 on Figure 2.7) that $\phi(\bar{w}')$ does not contain Z. Details are left to the reader. ■

We are furthermore able to describe w explicitly in Remark 2.70 below, and we record its expression here for future use in our work.

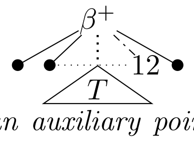
Definition 2.69. To each quasi-oscillation α of which an auxiliary point A is marked, we associate a word w_α^A defined below. Denoting by M the main substitution point of α corresponding to A and by $K_{A,M}$ the active knight formed by A and M then:
When α is increasing and $K_{A,M}$ is of type H (resp. V),
if A is in the top right corner of α, we set
$w_\alpha^A = (DL)^{p-2}DRU$ (resp. $w_\alpha^A = (LD)^{p-2}LUR$) if $|\alpha| = 2p$
$w_\alpha^A = (DL)^{p-2}UR$ (resp. $w_\alpha^A = (LD)^{p-2}RU$) if $|\alpha| = 2p-1$;
if A is in the bottom left corner of α, we set
$w_\alpha^A = (UR)^{p-2}ULD$ (resp. $w_\alpha^A = (RU)^{p-2}RDL$) if $|\alpha| = 2p$
$w_\alpha^A = (UR)^{p-2}DL$ (resp. $w_\alpha^A = (RU)^{p-2}LD$) if $|\alpha| = 2p-1$;
When α is decreasing, w_α^A is obtained by symmetry exchanging left and right.

Notice that for quasi-oscillations that are both increasing and decreasing the choice of A determines their nature, so that w_α^A is properly defined.

Remark 2.70. If A is in the top right corner of α (see Figure 2.17 p.84 or Figure 2.6 p.67), then $w = 3 \cdot w_\alpha^A$. If A is in the bottom left (resp. top left, bottom right) corner of α then $w = 1 \cdot w_\alpha^A$ (resp. $w = 4 \cdot w_\alpha^A$, $w = 2 \cdot w_\alpha^A$).

Remark 2.71. In Theorem 2.66, if π satisfies condition $(\mathcal{C})$ and $|T| = 2$, then $P_{\{1,n\}}(\pi) \cup P_{\{2,n\}}(\pi) = \{u \in P(\pi) \mid u \text{ strict or quasi-strict}\}$, denoted $P_{\text{SQS}}(\pi)$. This set corresponds to two proper pin representations, so it contains 12 pin words (see Remark 2.4 p.57). Moreover, with the notations of Lemma 2.56 (p.78), and $K_{A,M}$ as in Definition 2.69, we have an explicit description of $P_{\text{SQS}}(\pi)$:
When $K_{A,M}$ is of type H (resp. V),
if α is increasing (see Figure 2.17 p.84), then
$P_{\text{SQS}}(\pi) = (Q^+ + S_H^+) \cdot w_\alpha^A$ (resp. $P_{\text{SQS}}(\pi) = (Q^+ + S_V^+) \cdot w_\alpha^A$)
and if α is decreasing, then
$P_{\text{SQS}}(\pi) = (Q^- + S_H^-) \cdot w_\alpha^A$ (resp. $P_{\text{SQS}}(\pi) = (Q^- + S_V^-) \cdot w_\alpha^A$).

This concludes the study of the case $\pi = \alpha[1, \ldots, 1, T, 1, \ldots, 1]$. It now remains to deal with the case where more than one child of α is not a leaf. From Theorem 2.31, in this case α is an increasing (resp. decreasing) quasi-oscillation having exactly two children that are not leaves, and these are completely determined.

Theorem 2.72. *Let* $\pi =$ 

where β^+ *is an increasing quasi-oscillation, the permutation* 12 *expands an auxiliary point of* β^+ *and* T *(of size at least* 2*) expands the corresponding main substitution point of* β^+*. Then* $P(\pi) = P(T) \cdot w$ *where* w *is the unique word encoding the unique reading of the remaining leaves in* π *after* T *is read.*

Proof. Let $p = (p_1, \dots, p_n)$ be a pin representation associated to π. According to Section 3.4 of [BBR11], the configuration depicted on Figure 2.19 is the only possible configuration up to symmetry for a pin-permutation whose root is a simple permutation with two non-trivial children. Thus the sequence $(p_{k+1}, \dots, p_n)$ is uniquely determined in π. Moreover $k+1 \geq 3$, so that the suffix encoding $(p_{k+1}, \dots, p_n)$ in a pin word of p is a word w uniquely determined from Remark 2.4 (p.57). ■

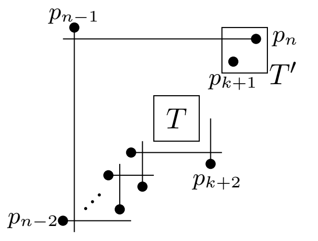


Figure 2.19: Diagram of π if two children are not leaves

Remark 2.73. The word w in the statement of Theorem 2.72 is a strict pin word uniquely determined by β^+ and the two points expanded in β^+.

More precisely, taking the notations of Definition 2.69 (with β^+ instead of α), $w = 1 \cdot w^A_{\beta^+}$ (resp. $w = 3 \cdot w^A_{\beta^+}$) when A is in the top right (resp. bottom left) corner of β^+ (see Figure 2.19).

For decomposition trees whose root is a decreasing quasi-oscillation, we obtain a description of $P(\pi)$ similar to the one of Theorem 2.72. This is derived in the fashion explained in Remark 2.36 (p.70).

2.5 Building deterministic automata accepting the languages $\mathcal{L}_\pi$

Recall that our goal is, using Theorem 2.27 (p.66) and given a finite basis B, to find an algorithm checking if the language $\mathcal{M} \setminus \cup_{\pi \in B} \mathcal{L}_\pi$ is finite, $\mathcal{M}$ being the set of words of length at least 2 over $\{U, D, L, R\}$ such that L, R is followed by U, D and conversely (see p.62). For algorithmic reasons to be detailed later, we rather consider the language $\overleftarrow{\mathcal{M} \setminus \cup_{\pi \in B} \mathcal{L}_\pi}$ where for any language $\mathcal{L}$ we denote by $\overleftarrow{\mathcal{L}}$ the language $\{\overleftarrow{v} \mid v \in \mathcal{L}\}$ and for any word $v = v_1 \dots v_p$ we denote by $\overleftarrow{v} = v_p \dots v_1$ the reverse of v. By definition of $\mathcal{M}$, we have $\overleftarrow{\mathcal{M} \setminus \cup_{\pi \in B} \mathcal{L}_\pi} = \overleftarrow{\mathcal{M}} \setminus \cup_{\pi \in B} \overleftarrow{\mathcal{L}_\pi} = \mathcal{M} \setminus \cup_{\pi \in B} \overleftarrow{\mathcal{L}_\pi}$. With usual constructions of automata theory (see [HU79] among other references), the construction of an automaton accepting the language $\mathcal{M} \setminus \cup_{\pi \in B} \overleftarrow{\mathcal{L}_\pi}$ is easily obtained from automata recognizing $\overleftarrow{\mathcal{L}_\pi}$ for $\pi \in B$.

From the definition of $\mathcal{L}_\pi$ given p.64 we have:

$$\overleftarrow{\mathcal{L}_\pi} = \bigcup_{\substack{u \in P(\pi) \\ u = u^{(1)} u^{(2)} \dots u^{(j)}}} A^\star \overleftarrow{\phi(u^{(j)})} A^\star \dots A^\star \overleftarrow{\phi(u^{(2)})} A^\star \overleftarrow{\phi(u^{(1)})} A^\star \tag{2.1}$$

where $A = \{U, D, L, R\}$, ϕ is the map introduced in Definition 2.18 (p.62) and for every pin word u, by $u = u^{(1)} u^{(2)} \dots u^{(j)}$ we mean (here and everywhere after) that $u^{(1)} u^{(2)} \dots u^{(j)}$ is the strong numeral-led factor decomposition of u.

In this section, we give for any pin-permutation π an explicit construction of a deterministic automaton $\mathcal{A}_\pi$ that recognizes the language $\overleftarrow{\mathcal{L}_\pi}$. We also present an alternative construction of $\mathcal{A}_\pi$ whose complexity is optimized; but instead of $\overleftarrow{\mathcal{L}_\pi}$, the automaton recognizes a language $\mathcal{L}$ such that $\mathcal{L} \cap \mathcal{M} = \overleftarrow{\mathcal{L}_\pi} \cap \mathcal{M}$. Moreover, in addition to being deterministic, both automata are complete, have a unique final state without outgoing transitions except for a loop labeled by all letters of A, and are *accessible* (i.e., for any state q there is a path from the initial state to q) and *co-accessible* (i.e., for any state q there is a path from q to the final state). These properties of $\mathcal{A}_\pi$ are inherited from the smaller automata used in its construction.

Our construction of automata accepting words $v \in \overleftarrow{\mathcal{L}_\pi}$ relies on a greedy principle: at each step we find the first occurrence of $\overleftarrow{\phi(u^{(\ell)})}$ that appears in the suffix of the word v that has not yet been read by the automaton. This is facilitated by the fact that in $\mathcal{L}_\pi$ the factors $\overleftarrow{\phi(u^{(\ell)})}$ are separated by $A^\star$.

The reason why we consider *reversed* words is in order to preserve determinism. Indeed intuitively the possible beginnings of pin words encoding a permutation may be numerous, whereas all these words end with very similar shuffle products as it appears in Theorems 2.53 and 2.60 (p.77 and 80).

To avoid exponential complexity it is important that the construction provides deterministic automata. From deterministic automata $\mathcal{A}_\pi$ recognizing the languages $\overleftarrow{\mathcal{L}_\pi}$, we can obtain a deterministic automaton accepting words whose reverses encode proper pin-permutations containing some pattern $\pi \in B$, for any finite set B of patterns. This is achieved using the Cartesian product of the automata $\mathcal{A}_\pi$ to compute a deterministic automaton accepting the union $\cup_{\pi \in B} \overleftarrow{\mathcal{L}_\pi}$. This deterministic automaton can then be complemented in linear time (in order to build an automaton recognizing $\mathcal{M} \setminus \cup_{\pi \in B} \overleftarrow{\mathcal{L}_\pi}$), while the same operation on non-deterministic automata is exponential in the worst case. This last step will be detailed in Section 2.6.

For now, we focus on the construction of the automata $\mathcal{A}_\pi$. In Subsection 2.5.1, we present generic constructions of automata that will be used several times. In Subsections 2.5.2 to 2.5.5, we construct recursively the automata $\mathcal{A}_\pi$ that recognize the languages $\overleftarrow{\mathcal{L}_\pi}$ for any pin-permutation π, distinguishing cases according to the decomposition tree of π – see Equation ($\star$) p.69. In these constructions, some states of the automata must be marked, and this is detailed in Subsection 2.5.6. We conclude with Subsection 2.5.7 analysing the complexity of building $\mathcal{A}_\pi$.

2.5.1 Generic constructions of deterministic automata

We present some generic constructions that are used in the next subsections. We refer the reader to [HU79] for more details about automata.

Aho-Corasick algorithm. Let X be a finite set of words over a finite alphabet A. The Aho-Corasick algorithm [AC75] builds a deterministic automaton that recognizes $A^\star X$ in linear time and space w.r.t. the sum $\|X\|$ of the lengths of the words of X. The first step of the algorithm consists in constructing a tree-automaton whose states are labeled by the prefixes of the words of X. The initial state is the empty word ε. For any word u and any letter a there is a transition labeled by a from state u to state ua if ua is a prefix of a word of X. At this step the final states are those labeled by the words of X. These states are the leaves of the tree, and some internal nodes if some word of X is a prefix of another one. The second step consists in adding transitions in the automaton according to a breadth-first traversal of the tree-automaton to obtain a complete automaton. For any state u and any letter a, the transition from u labeled by a goes to the state corresponding to the longest suffix of ua that is also a prefix of a word of X. The set of final states is the set of states corresponding to words having a suffix in X. These states correspond to a leaf or an internal node – when there is a factor relation between two words of X – of the original tree-automaton. The ones corresponding to words having a proper suffix in X are marked on the fly during the construction of the missing transitions.

Remark 2.74. Notice that all transitions labeled with a letter of A that does not appear in any word of X go to the initial state. Moreover the reading of any word u by the automaton leads to the state labeled with the longest suffix of u that is also a prefix of a word of X.

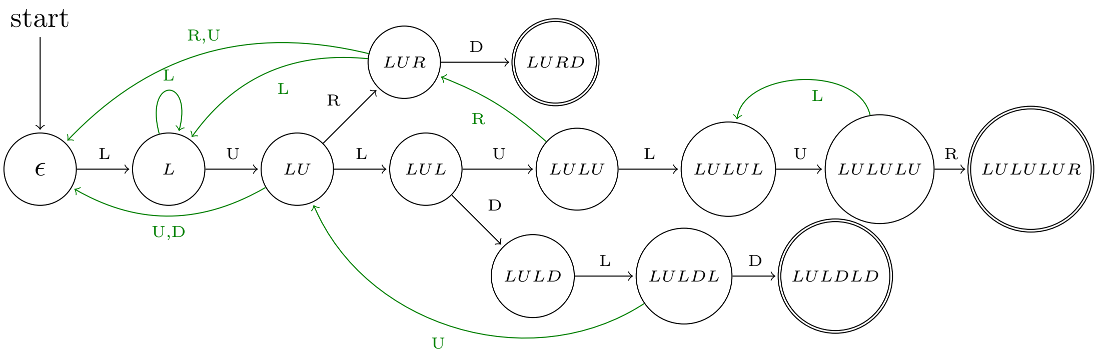


Figure 2.20: Aho-Corasick automaton for $X = \{LULULUR, LULDLD, LURD\}$ with the tree-automaton obtained after the first step drawn in black with straight arrows, and with transitions added in the second step drawn in green with bent arrows (for clarity reasons, we have drawn only a few of these latter transitions).

A variant for first occurrences. An adaptation of the Aho-Corasick algorithm allows us to build in linear time and space w.r.t. $\|X\|$ a deterministic automaton, denoted $\mathcal{AC}(X)$, recognizing the set of words ending with a *first* occurrence of a word of X (which is strictly included in $A^\star X$). First we perform the first step of the Aho-Corasick algorithm on X, obtaining a tree automaton. We modify the second step as follows: in the breadth-first traversal, we stop the exploration of a branch and delete its descendants as soon as a final state is reached. Moreover we do not build the outgoing transitions from the final states, nor the loops on the final states. This ensures that the language recognized is the set of words ending with a first occurrence of a word of X. Finally we merge the final states into a unique final state f to obtain $\mathcal{AC}(X)$. Moreover if we add a loop labeled by all letters of A on f we obtain an automaton $\mathcal{AC}^{\circlearrowleft}(X)$ that recognizes the set $A^\star X A^\star$ of words having a factor in X.

Remark 2.75. The main difference between our variant and the construction of Aho-Corasick is that we *stop* as soon as a *first* occurrence of a word of X is read. This ensures that $\mathcal{AC}(X)$ has a unique final state without any outgoing transition.

This variant for first occurrences satisfies properties analogous to Remark 2.74:

Lemma 2.76. *In $\mathcal{AC}(X)$, all transitions labeled with a letter that does not appear in any word of X go to the initial state. Moreover let u be a word without any factor in X except maybe as a suffix. Then the reading of u by $\mathcal{AC}(X)$ leads to the state labeled with the longest suffix of u that is also a prefix of a word of X.*

Proof. Let $\mathcal{A}$ be the usual Aho-Corasick automaton on X. Then $\mathcal{AC}(X)$ (before the merge of all final states in f) is a subautomaton of $\mathcal{A}$, and therefore the first assertion is a direct consequence of Remark 2.74. Let u be a word without any factor in X except maybe as a suffix. Then the path of the reading of u by $\mathcal{A}$ does not visit any final state, except maybe the last state reached. Thus all this path is included in $\mathcal{AC}(X)$ and we conclude using Remark 2.74. ■

A variant for a partition X_1, X_2. When the set X is partitioned into two subsets X_1 and X_2 such that no word of X_1 (resp. X_2) is a factor of a word of X_2 (resp. X_1) [3], we adapt the previous construction and build a deterministic automaton $\mathcal{AC}(X_1, X_2)$ which recognizes the same language as $\mathcal{AC}(X)$. But instead of merging all final states into a unique final state, we build two final states f_1 and f_2 corresponding to the first occurrence of a word of X_1 (resp. X_2). This construction is linear in time and space w.r.t. $\|X_1\|+\|X_2\|$. In what follows we will use this construction only when X_1 and X_2 are languages on disjoint alphabets, so that the factor independence condition is trivially satisfied.

Concatenation. Building an automaton $\mathcal{A}_1 \cdot \mathcal{A}_2$ recognizing the concatenation $\mathcal{L}_1 \cdot \mathcal{L}_2$ of two languages respectively recognized by the deterministic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ is easy when $\mathcal{A}_1$ has a unique final state without outgoing transitions. Indeed it is enough to merge the final state of $\mathcal{A}_1$ with the initial state of $\mathcal{A}_2$ into a unique state that is not initial (resp. not final), except when the initial state of $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) is final. Note that the resulting automaton is deterministic and of size at most $|\mathcal{A}_1| + |\mathcal{A}_2|$, where the *size* $|\mathcal{A}|$ is the number of states of any automaton $\mathcal{A}$. This construction is done in constant time.

When the final state of $\mathcal{A}_1$ has no outgoing transitions except for a loop labeled by all letters of A and when $\mathcal{L}_2$ is of type $A^\star \cdot \mathcal{L}$ for an arbitrary language $\mathcal{L}$, we can do the same construction to obtain an automaton recognizing the concatenation $\mathcal{L}_1 \cdot \mathcal{L}_2 = \mathcal{L}_1 \cdot A^\star \cdot \mathcal{L}$. We just have to delete the loop on the final state of $\mathcal{A}_1$ before merging states.

In particular, according to this construction, the automaton obtained concatenating $\mathcal{AC}^\circlearrowleft(X)$ and $\mathcal{A}$ is $\mathcal{AC}(X) \cdot \mathcal{A}$. Therefore, even though $\mathcal{AC}(X)$ recognizes a language strictly included in $A^\star X$, $\mathcal{AC}(X) \cdot \mathcal{A}$ recognizes $A^\star X A^\star \mathcal{L}$ when $\mathcal{A}$ recognizes a language $A^\star \mathcal{L}$. This construction is often used in the sequel.

Union. We say that an automaton is *almost complete* if for any letter a, all non-final states have an outgoing transition labeled by a (notice that the only difference with a complete automaton is that final states are allowed to miss some transitions). Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two deterministic automata that are accessible, co-accessible and almost complete[4]. We define the automaton $\mathcal{U}(\mathcal{A}_1, \mathcal{A}_2)$ as follows. We perform the Cartesian product of $\mathcal{A}_1$ and $\mathcal{A}_2$ beginning from the pair of initial states (see [HU79]). However we stop exploring a path when it enters a final state of $\mathcal{A}_1$ or $\mathcal{A}_2$. Therefore in $\mathcal{U}(\mathcal{A}_1, \mathcal{A}_2)$ there is no outgoing transition from any state (q_1, q_2) such that q_1 or q_2 is final. Moreover these states are merged into a unique final state of $\mathcal{U}(\mathcal{A}_1, \mathcal{A}_2)$. Let $\mathcal{L}_1$ (resp. $\mathcal{L}_2$, $\mathcal{L}$) be the language recognized by $\mathcal{A}_1$ (resp. $\mathcal{A}_2$, $\mathcal{U}(\mathcal{A}_1, \mathcal{A}_2)$). Then $\mathcal{L}$ is the set of words of $\mathcal{L}_1 \cup \mathcal{L}_2$ truncated after their first factor in $\mathcal{L}_1 \cup \mathcal{L}_2$. The language $(\mathcal{L}_1 \cup \mathcal{L}_2)A^\star = \mathcal{L}A^\star$ is recognized by the automaton $\mathcal{U}^\circlearrowleft(\mathcal{A}_1, \mathcal{A}_2)$ with an additional loop labeled by all letters of A on the final state. Notice that $\mathcal{U}(\mathcal{A}_1, \mathcal{A}_2)$ (resp. $\mathcal{U}^\circlearrowleft(\mathcal{A}_1, \mathcal{A}_2)$) is deterministic, accessible, co-accessible, almost complete (resp. complete) and has a unique final state without outgoing transitions (resp. with a loop labeled by all letters of A). The complexity in time and space of these constructions is in $\mathcal{O}(|\mathcal{A}_1| \cdot |\mathcal{A}_2|)$.

2.5.2 Pin-permutation of size 1 and simple pin-permutations

Pin-permutation of size 1. When $\pi = 1$, we have $P(\pi) = \{1, 2, 3, 4\}$. Then

$$\overleftarrow{\mathcal{L}_\pi} = \{A^\star \overleftarrow{\phi(w)} A^\star \mid w \in P(\pi)\} = A^\star \mathcal{M}_2 A^\star$$

[3]This is a simple condition that allows us to define without ambiguity words ending with a first occurrence either in X_1 or in X_2.

[4]Notice that the automata $\mathcal{AC}(X)$, $\mathcal{AC}^\circlearrowleft(X)$ and $\mathcal{AC}(X_1, X_2)$ satisfy these conditions.

where

$$\mathcal{M}_2 = \mathcal{M} \cap A^2 = \{UR, UL, DR, DL, RU, RD, LU, LD\}.$$

The language $\overleftarrow{\mathcal{L}_\pi}$ is recognized by the automaton $\mathcal{A}_\pi$ of Figure 2.21.

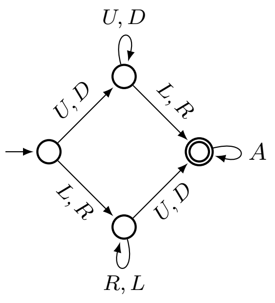


Figure 2.21: The automaton $\mathcal{A}_\pi$ when $\pi = 1$.

Simple pin-permutations. In this paragraph, for a simple permutation π whose set of pin words $P(\pi)$ is given, we build the automaton $\mathcal{A}_\pi$. The computation of $P(\pi)$ from π is done using Algorithm 3 (p.75). The study that follows is based on the upcoming lemma.

Lemma 2.77. *For every permutation π (not necessarily simple), we have*

$$\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict or quasi-strict}}} \mathcal{L}(u) = A^\star \cdot E_\pi^{\mathrm{S}} \cdot A^\star \ \cup \ A^\star \cdot \mathcal{M}_2 \cdot A^\star \cdot E_\pi^{\mathrm{QS}} \cdot A^\star$$

where $E_\pi^{\mathrm{S}} = \{\phi(u) \mid u \in P(\pi), u$ is strict$\}$ and $E_\pi^{\mathrm{QS}} = \{\phi(u^{(2)}) \mid u = u^{(1)}u^{(2)} \in P(\pi), u$ is quasi-strict$\}$.

Proof. By definition of $\mathcal{L}(u)$ (see Definition 2.23 p.64),

$$\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict or quasi-strict}}} \mathcal{L}(u) = \Big(\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict}}} A^\star \phi(u) A^\star \Big) \bigcup \Big(\bigcup_{\substack{u = u^{(1)}u^{(2)} \in P(\pi) \\ u \text{ quasi-strict}}} A^\star \phi(u^{(1)}) A^\star \phi(u^{(2)}) A^\star \Big).$$

Moreover, as can be seen on Figure 2.5 (p.57), for every quasi-strict pin word $u = u^{(1)}u^{(2)} \in P(\pi)$, the words $hu^{(2)}$ also belong to $P(\pi)$ for all $h \in \{1, 2, 3, 4\}$. This allows to write

$$\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict or quasi-strict}}} \mathcal{L}(u) = \Big(\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict}}} A^\star \phi(u) A^\star \Big) \bigcup \Big(\bigcup_{h \in \{1,2,3,4\}} A^\star \phi(h) \ \cdot \bigcup_{\substack{u = u^{(1)}u^{(2)} \in P(\pi) \\ u \text{ quasi-strict}}} A^\star \phi(u^{(2)}) A^\star \Big).$$

Hence

$$\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict or quasi-strict}}} \mathcal{L}(u) = A^\star \cdot E_\pi^{\mathrm{S}} \cdot A^\star \ \cup \ A^\star \cdot \mathcal{M}_2 \cdot A^\star \cdot E_\pi^{\mathrm{QS}} \cdot A^\star,$$

concluding the proof. ■

Lemma 2.78. *For every permutation π whose sets of strict and quasi-strict pin words (or equivalently E_π^{S} and E_π^{QS}) are given, one can build in time and space $\mathcal{O}\left(|E_\pi^{\mathrm{S}}| \cdot |E_\pi^{\mathrm{QS}}| \cdot |\pi|^2\right)$ an accessible and co-accessible deterministic complete automaton $\mathcal{A}_\pi^{\mathrm{SQS}}$ having a unique final state without outgoing transitions except for a loop labeled by all letters of A that recognizes the language* $\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict or quasi-strict}}} \overleftarrow{\mathcal{L}(u)}$.

Proof. From Lemma 2.77, we have

$$\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict or quasi-strict}}} \overleftarrow{\mathcal{L}(u)} = \overleftarrow{A^\star E_\pi^{\mathrm{S}} A^\star} \bigcup \overleftarrow{A^\star \mathcal{M}_2 A^\star E_\pi^{\mathrm{QS}} A^\star} = \Big(A^\star \overleftarrow{E_\pi^{\mathrm{S}}} \bigcup (A^\star \overleftarrow{E_\pi^{\mathrm{QS}}} \cdot A^\star \mathcal{M}_2) \Big) A^\star.$$

Recall that $\mathcal{AC}(\overleftarrow{E^{\mathrm{S}}_\pi})$, $\mathcal{AC}(\overleftarrow{E^{\mathrm{QS}}_\pi})$ and $\mathcal{AC}(\mathcal{M}_2)$ are automata recognizing respectively the set of words ending with a first occurrence of a word of $\overleftarrow{E^{\mathrm{S}}_\pi}$, $\overleftarrow{E^{\mathrm{QS}}_\pi}$ and $\mathcal{M}_2$ and obtained using the construction given in Section 2.5.1. The sizes of the first two automata are respectively $\mathcal{O}\left(|E^{\mathrm{S}}_\pi| \cdot |\pi|\right)$ and $\mathcal{O}\left(|E^{\mathrm{QS}}_\pi| \cdot |\pi|\right)$, and the size of the third one is constant. Indeed for all w in E^{S}_π, $|w| = |\pi| + 1$ and for all w in E^{QS}_π, $|w| = |\pi|$ so that $\|E^{\mathrm{S}}_\pi\| = |E^{\mathrm{S}}_\pi| \cdot (|\pi| + 1)$ and $\|E^{\mathrm{QS}}_\pi\| = |E^{\mathrm{S}}_\pi| \cdot |\pi|$.

Then the deterministic automaton $\mathcal{A}^{\mathrm{SQS}}_\pi$ is obtained as the union $\mathcal{U}^{\circlearrowleft}(\mathcal{AC}(\overleftarrow{E^{\mathrm{S}}_\pi})\ ,\ \mathcal{AC}(\overleftarrow{E^{\mathrm{QS}}_\pi}) \cdot \mathcal{AC}(\mathcal{M}_2))$ in time and space $\mathcal{O}\left(|E^{\mathrm{S}}_\pi| \cdot |E^{\mathrm{QS}}_\pi| \cdot |\pi|^2\right)$. ■

Lemma 2.78 is used mostly in two special cases where *all* the pin words of π are strict or quasi-strict, and that we identify explicitly in the following remark.

Remark 2.79. By Theorem 2.48 (p.75) the pin words encoding a simple permutation are either strict or quasi-strict and there are at most 48 of them. Therefore when π is a simple permutation, we take $\mathcal{A}_\pi = \mathcal{A}^{\mathrm{SQS}}_\pi$ (from Lemma 2.78) and the time and space complexity of the construction of $\mathcal{A}_\pi$ is quadratic w.r.t. $|\pi|$, as soon as the pin words of π are given.

Notice also that when $\pi = 12$, $\mathcal{A}_\pi = \mathcal{A}^{\mathrm{SQS}}_\pi$ and the time and space complexity of the construction is $\mathcal{O}(1)$.

The above construction follows the general scheme announced at the beginning of the section, but it is not optimized. We actually can provide a more specific construction of $\mathcal{A}_\pi$ whose complexity is *linear* when the permutation π is simple. This construction relies on the same idea as the one we give in [BBPR10].

Definition 2.80. For any permutation π, we define $E_\pi = E^{\mathrm{S}}_\pi \ \cup\ \mathcal{M}_2 \cdot E^{\mathrm{QS}}_\pi$.

Lemma 2.81. *For any permutation π, we have*

$$\bigcup_{\substack{u \in P(\pi)\\ u \text{ strict or quasi-strict}}} \mathcal{L}(u) \cap \mathcal{M} = \left(A^\star \cdot E_\pi \cdot A^\star\right) \cap \mathcal{M},$$

Proof. From Lemma 2.77, we have

$$\bigcup_{\substack{u \in P(\pi)\\ u \text{ strict or quasi-strict}}} \mathcal{L}(u) = A^\star \cdot E^{\mathrm{S}}_\pi \cdot A^\star \ \cup\ A^\star \cdot \mathcal{M}_2 \cdot A^\star \cdot E^{\mathrm{QS}}_\pi \cdot A^\star.$$

Since $(A^\star \cdot \mathcal{M}_2 \cdot A^\star \cdot E^{\mathrm{QS}}_\pi \cdot A^\star) \cap \mathcal{M} = (A^\star \cdot \mathcal{M}_2 \cdot E^{\mathrm{QS}}_\pi \cdot A^\star) \cap \mathcal{M}$ and $E_\pi = E^{\mathrm{S}}_\pi \ \cup\ \mathcal{M}_2 \cdot E^{\mathrm{QS}}_\pi$, we obtain

$$\bigcup_{\substack{u \in P(\pi)\\ u \text{ strict or quasi-strict}}} \mathcal{L}(u) \cap \mathcal{M} = \left(A^\star \cdot E_\pi \cdot A^\star\right) \cap \mathcal{M},$$

concluding the proof. ■

Lemma 2.82. *For every permutation π whose sets of strict and quasi-strict pin words (or equivalently E^{S}_π and E^{QS}_π) are given, one can build in time and space $\mathcal{O}\left((|E^{\mathrm{S}}_\pi| + |E^{\mathrm{QS}}_\pi|) \cdot |\pi|\right)$ an accessible and co-accessible deterministic complete automaton $\mathcal{A}^{\mathrm{SQS}}_\pi$ having a unique final state without outgoing transitions except for a loop labeled by all letters of A that recognizes a language $\mathcal{L}$ such that*

$$\mathcal{L} \cap \mathcal{M} = \bigcup_{\substack{u \in P(\pi)\\ u \text{ strict or quasi-strict}}} \overleftarrow{\mathcal{L}(u)} \cap \mathcal{M}.$$

Proof. Setting $\mathcal{A}^{\mathrm{SQS}}_\pi = \mathcal{AC}^{\circlearrowleft}(\overleftarrow{E_\pi})$, the language recognized by $\mathcal{A}$ is $\mathcal{L} = A^\star \cdot \overleftarrow{E_\pi} \cdot A^\star$. We conclude using Lemma 2.81. ■

Remark 2.83. When π is a simple permutation, the automaton $\mathcal{A}_\pi^{\text{SQS}}$ of Lemma 2.82 recognizes a language $\mathcal{L}$ such that $\mathcal{L} \cap \mathcal{M} = \overleftarrow{\mathcal{L}_\pi} \cap \mathcal{M}$. In the optimized alternative construction of $\mathcal{A}_\pi$ mentioned at the beginning of Section 2.5, for a simple permutation π we take $\mathcal{A}_\pi = \mathcal{A}_\pi^{\text{SQS}}$ from Lemma 2.82 and the time and space complexity of building the automaton $\mathcal{A}_\pi$ is *linear* w.r.t. $|\pi|$ as soon as $P(\pi)$ is given.

2.5.3 Pin-permutations with a linear root: non-recursive case

w.l.o.g. (see Remark 2.36 p.70), the only non-recursive case with a linear root is the one where $\pi = \oplus[\xi_1, \dots, \xi_r]$, every ξ_i being an increasing oscillation. Theorem 2.53 (p.77) gives an explicit description of the elements of $P(\pi)$. These words are the concatenation of a pin word belonging to some $P(\oplus[\xi_i, \xi_{i+1}])$ with a sequence of pin words belonging to the shuffle product

$$(P^{(1)}(\xi_{i-1}), \dots, P^{(1)}(\xi_1)) \sqcup\!\sqcup \, (P^{(3)}(\xi_{i+2}), \dots, P^{(3)}(\xi_r)).$$

To shorten the notations in the following, let us define

$$\chi_j^{(h)} = \overleftarrow{\phi(P^{(h)}(\xi_j))} \text{ for } h = 1 \text{ or } 3 \text{ and } 1 \le j \le r.$$

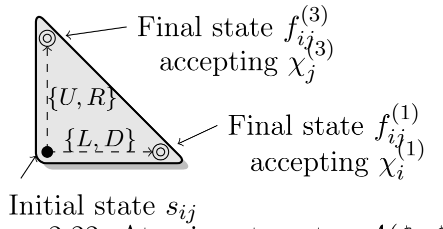
From Lemma 2.54 (p.77), for all j, the pin words of $P^{(1)}(\xi_j)$ and $P^{(3)}(\xi_j)$ respectively are strict. Hence, the decomposition of $u \in P(\pi)$ into strong numeral-led factors that is needed to describe $\overleftarrow{\mathcal{L}_\pi}$ (see p.88) is easily obtained and gives:

$$\overleftarrow{\mathcal{L}_\pi} = \bigcup_{1 \le i \le r-1} \left(\left(A^\star \chi_1^{(1)}, \dots, A^\star \chi_{i-1}^{(1)} \right) \sqcup\!\sqcup \left(A^\star \chi_r^{(3)}, \dots, A^\star \chi_{i+2}^{(3)} \right) \right) \cdot \overleftarrow{\mathcal{L}_{\oplus[\xi_i, \xi_{i+1}]}}$$

In the above equation, we have $\overleftarrow{\mathcal{L}_{\oplus[\xi_i, \xi_{i+1}]}}$ where we might have expected to rather find $A^\star \overleftarrow{\phi(P(\oplus[\xi_i, \xi_{i+1}]))} A^\star$. The reason is that the term $A^\star \overleftarrow{\phi(P(\oplus[\xi_i, \xi_{i+1}]))} A^\star$ is not properly defined, since $P(\oplus[\xi_i, \xi_{i+1}])$ contains pin words that are not strict.

The automaton $\mathcal{A}_\pi$ is then built by assembling smaller automata, whose construction is explained below.

Construction of $\mathcal{A}(\xi_i, \xi_j)$. Since for all i, j, the languages $\chi_i^{(1)}$ and $\chi_j^{(3)}$ – that contain at most two words – are defined on disjoint alphabets (see Remark 2.55 p.77), we can use the construction given in Section 2.5.1 to build the deterministic automata $\mathcal{A}(\xi_i, \xi_j) = \mathcal{AC}(\chi_i^{(1)}, \chi_j^{(3)})$. Figure 2.22 shows a diagram of $\mathcal{A}(\xi_i, \xi_j)$ and defines states s_{ij}, $f_{ij}^{(1)}$ and $f_{ij}^{(3)}$.


Final state $f_{ij}^{(3)}$ accepting $\chi_j^{(3)}$
$\{U, R\}$
$\{L, D\}$
Final state $f_{ij}^{(1)}$ accepting $\chi_i^{(1)}$
Initial state s_{ij}
Figure 2.22: Atomic automaton $\mathcal{A}(\xi_i, \xi_j)$ used in the construction of $\mathcal{A}_\pi$.

Lemma 2.84. *For all i, j, the complexity in time and space of the construction of $\mathcal{A}(\xi_i, \xi_j)$ is $\mathcal{O}(|\xi_i| + |\xi_j|)$.*

Construction of $\mathcal{A}_{\xi_i}$. By Lemma 2.56 (p.78), for any i, $|P(\xi_i)| \le 48$, the pin words in $P(\xi_i)$ are explicit and all of them are either strict or quasi-strict except when $|\xi_i| = 3$.

If $|\xi_i| \ne 3$, from Lemma 2.78 it is possible to build the deterministic automaton $\mathcal{A}_{\xi_i}$ in quadratic time and space w.r.t. $|\xi_i|$, and from Lemma 2.82 the construction can be optimized to be linear in time and space w.r.t. $|\xi_i|$.

If $|\xi_i| = 3$, $P(\xi_i)$ can be partitioned into two parts $P_{\text{SQS}} \uplus P(12) \cdot h$ where $h = 4$ (resp. 2) if $\xi_i = 231$ (resp. 312) and P_{SQS} is the set of strict and quasi-strict pin words of $P(\xi_i)$. With Lemma 2.78 or 2.82 we build the automaton $\mathcal{A}_{\xi_i}^{\text{SQS}}$ corresponding to P_{SQS}, and the automaton corresponding to $P(12) \cdot h$ is the concatenation of two basic automata, $\mathcal{AC}(\overleftarrow{\phi(h)})$

(where $\phi(h)$ for $h = 2$ or 4 is given p.62) and $\mathcal{A}_{12}$ (see Remark 2.79 p.93). Finally the automaton $\mathcal{A}_{\xi_i}$ is the union $\mathcal{U}^{\circlearrowleft}(\mathcal{A}^{\text{SQS}}_{\xi_i}, \mathcal{AC}(\overleftarrow{\phi(h)}) \cdot \mathcal{A}_{12})$. As $|\xi_i| = 3$, $\mathcal{A}_{\xi_i}$ is built in constant time.

Lemma 2.85. *For all i, building $\mathcal{A}_{\xi_i}$ is done in time and space $\mathcal{O}(|\xi_i|^2)$ with the classical construction and $\mathcal{O}(|\xi_i|)$ in the optimized version.*

Construction of $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1})$. We now explain how to build a deterministic automaton $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1})$ recognizing $\overleftarrow{\mathcal{L}_{\oplus[\xi_i,\xi_{i+1}]}}$. Lemma 2.59 (p.79) describes the pin words of $P(\oplus[\xi_i, \xi_{i+1}])$, proving the correctness of the following constructions:

If $|\xi_i| > 1$ and $|\xi_{i+1}| > 1$, we obtain $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1})$ by gluing together automata $\mathcal{A}(\xi_i, \xi_{i+1})$, $\mathcal{A}_{\xi_i}$ and $\mathcal{A}_{\xi_{i+1}}$ as shown in Figure 2.23. More precisely $f^{(1)}_{i(i+1)}$ (resp. $f^{(3)}_{i(i+1)}$) and the initial state of $\mathcal{A}_{\xi_{i+1}}$ (resp. $\mathcal{A}_{\xi_i}$) are merged into a unique state that is neither initial nor final. The final states of $\mathcal{A}_{\xi_i}$ and $\mathcal{A}_{\xi_{i+1}}$ are also merged into a unique final state f_i having a loop labeled by all letters of A.

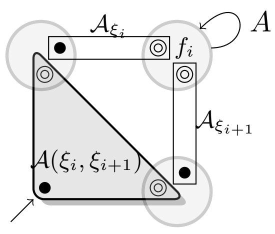


Figure 2.23: Automaton $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1})$.

If $|\xi_i| = 1$ and $|\xi_{i+1}| = 1$ then $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1}) = \mathcal{A}_{12}$ is built using Remark 2.79.

Otherwise assume w.l.o.g. that $|\xi_i| = 1$ and $|\xi_{i+1}| > 1$. The set of pin words $P(\oplus[\xi_i, \xi_{i+1}])$ can be partitioned into two parts P_{SQS} and P', P_{SQS} being the set of strict and quasi-strict pin words. If $|\xi_{i+1}| \neq 3$, then $P' = P(\xi_{i+1})\cdot 3$ and $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1}) = \mathcal{U}^{\circlearrowleft}\left(\mathcal{A}^{\text{SQS}}_{\oplus[\xi_i,\xi_{i+1}]}, \mathcal{AC}(\overleftarrow{\phi(3)}) \cdot \mathcal{A}_{\xi_{i+1}}\right)$. If $|\xi_{i+1}| = 3$, then $P' = P(\xi_{i+1}) \cdot 3 \bigcup P(12) \cdot 3X$ where X is a direction, and we use again concatenation and union but performed on automata of constant size.

Note that in all cases $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1})$ has a unique final state – that we denote f_i – without outgoing transitions except for the loop labeled by all letters of A.

Lemma 2.86. *For all i, building the automaton $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1})$ is done in time and space $\mathcal{O}(|\xi_i|^4 + |\xi_{i+1}|^4)$ with the classical construction and $\mathcal{O}(|\xi_i|^2 + |\xi_{i+1}|^2)$ in the optimized version.*

Proof. The complexity of the construction of $\mathcal{A}(\xi_i, \xi_{i+1})$ is linear w.r.t. $|\xi_i| + |\xi_{i+1}|$ from Lemma 2.84 and the one of $\mathcal{A}_{\xi_i}$ is quadratic – or linear in the optimized version – w.r.t. $|\xi_i|$ (see Lemma 2.85). From Lemma 2.59 (p.79), we have an explicit description of $P(\oplus[\xi_i, \xi_{i+1}])$ and $|P(\oplus[\xi_i, \xi_{i+1}])| \leq 192$. Hence, the complexity of the construction of $\mathcal{A}^{\text{SQS}}_{\oplus[\xi_i,\xi_{i+1}]}$ when $|\xi_i| = 1$ is quadratic – or linear in the optimized version – w.r.t. $|\xi_{i+1}|$, using Lemmas 2.78 and 2.82. Then the result follows from the complexity of the union of two automata, which is proportional to the product of the sizes of the automata. ■

Assembling $\mathcal{A}_\pi$. According to the description of $\overleftarrow{\mathcal{L}_\pi}$ given p.94, the automata $\mathcal{A}(\xi_i, \xi_j)$ and $\mathcal{A}^{\oplus}(\xi_i, \xi_{i+1})$ can be glued together to finish the construction of $\mathcal{A}_\pi$, as shown in Figure 2.24. More precisely for any i, j such that $1 \leq i < j \leq r$

- if $i + 1 \neq j$, then s_{ij}, $f^{(1)}_{(i-1)j}$ and $f^{(3)}_{i(j+1)}$ are merged into a unique state q_{ij} that is neither initial (except when $i = 1$ and $j = r$) nor final,
- if $i + 1 = j$, $f^{(1)}_{(i-1)j}$, $f^{(3)}_{i(j+1)}$ and the initial state of $\mathcal{A}^{\oplus}(\xi_i, \xi_j) = \mathcal{A}^{\oplus}(\xi_i, \xi_{i+1})$ are merged into a unique state q_{ij} that is neither initial nor final,

and the final states f_i of the automata $\mathcal{A}^{\oplus}(\xi_i,\xi_{i+1})$ are merged into a unique final state f having a loop labeled by all letters of A. The states q_{ij} defined above correspond to the shuffle product construction. To be more precise, taking the final state to be the merged state q_{ij} and adding a loop labeled by all letters of A on it, the accepted language would be $\left(A^\star\chi_1^{(1)},\dots,A^\star\chi_{i-1}^{(1)}\right) \sqcup \left(A^\star\chi_r^{(3)},\dots,A^\star\chi_{j+1}^{(3)}\right)\cdot A^\star$ as it will be proved in the following. The automata $\mathcal{A}^{\oplus}(\xi_i,\xi_{i+1})$ in the second item above correspond to the concatenation with $\overleftarrow{\mathcal{L}_{\oplus[\xi_i,\xi_{i+1}]}}$.

Note that if $r=2$, $\mathcal{A}_\pi$ is $\mathcal{A}^{\oplus}(\xi_1,\xi_2)$.

Lemma 2.87. *For any permutation π such that $\pi=\oplus[\xi_1,\dots,\xi_r]$, every ξ_i being an increasing oscillation, the construction of the automaton $\mathcal{A}_\pi$ is done in time and space $\mathcal{O}(|\pi|^4)$ with the classical construction and $\mathcal{O}(|\pi|^2)$ in the optimized version.*

Proof. Denote by n the size of π, then $n=\sum_{i=1}^r|\xi_i|$. By construction, taking into account the merging of states:

$$|\mathcal{A}_\pi| \;\le\; \sum_{i=1}^{r-2}\sum_{j=i+2}^{r}|\mathcal{A}(\xi_i,\xi_j)|+\sum_{i=1}^{r-1}|\mathcal{A}^{\oplus}(\xi_i,\xi_{i+1})|$$

and from Lemmas 2.84 and 2.86, in the classical construction

$$|\mathcal{A}_\pi| \;=\; \mathcal{O}\left(\sum_{i=1}^{r-2}\sum_{j=i+2}^{r}(|\xi_i|+|\xi_j|)+\sum_{i=1}^{r-1}(|\xi_i|^4+|\xi_{i+1}|^4)\right)=\mathcal{O}(n^4).$$

and in the optimized version

$$|\mathcal{A}_\pi| \;=\; \mathcal{O}\left(\sum_{i=1}^{r-2}\sum_{j=i+2}^{r}(|\xi_i|+|\xi_j|)+\sum_{i=1}^{r-1}(|\xi_i|^2+|\xi_{i+1}|^2)\right)=\mathcal{O}(n^2)$$

We conclude the proof by noticing that all these automata are built in linear time w.r.t. their size. ■

Correctness of the construction. We now prove that the automaton $\mathcal{A}_\pi$ given in Figure 2.24 recognizes the language $\overleftarrow{\mathcal{L}_\pi}$, for $\pi=\oplus[\xi_1,\dots,\xi_r]$, every ξ_i being an increasing oscillation.

Definition 2.88. Let $\mathcal{A}$ be a deterministic complete automaton over the alphabet A whose set of states is Q and let u be a word in $A^\star$. We define $trace_{\mathcal{A}}(u)$ as the word of $Q^{|u|+1}$ that consists in the states of the automaton that are visited when reading u from the initial state of $\mathcal{A}$, and for all $q\in Q$ we define $q\cdot u$ to be the state of $\mathcal{A}$ reached when reading u from q.

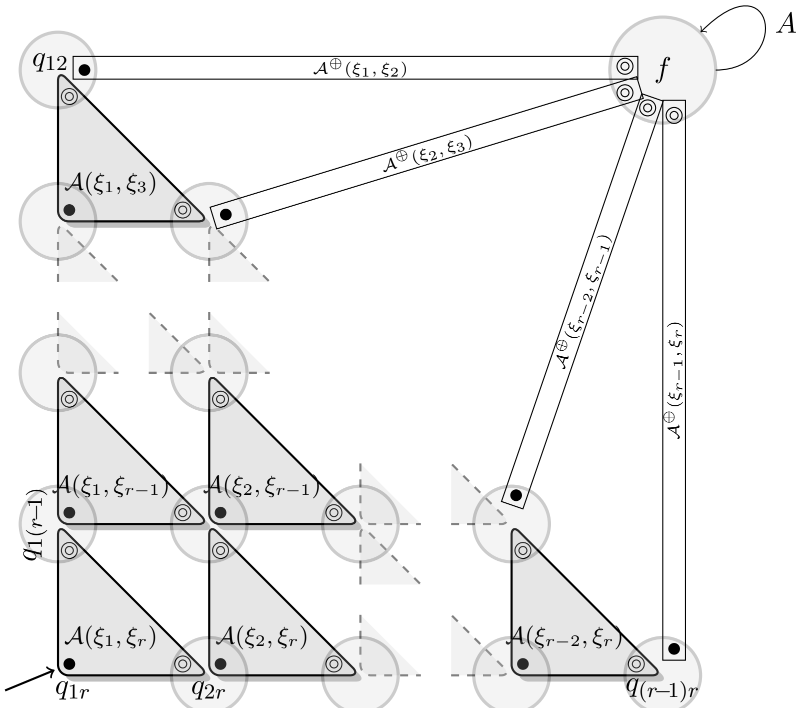
Let u be a word in $A^\star$. We define two parameters on u:

$i(u)=\max\{i\in\{0,r\}\mid u\in A^\star\cdot\chi_1^{(1)}\cdot A^\star\cdot\chi_2^{(1)}\dots A^\star\cdot\chi_i^{(1)}\cdot A^\star\}$, and

$j(u)=\min\{j\in\{1,r+1\}\mid u\in A^\star\cdot\chi_r^{(3)}\cdot A^\star\cdot\chi_{r-1}^{(3)}\dots A^\star\cdot\chi_j^{(3)}\cdot A^\star\}$.

Lemma 2.89. *Every word u such that $i(u)\ge i$ and $j(u)\le j$ belongs to* $\left((A^\star\chi_1^{(1)},A^\star\chi_2^{(1)},\dots,A^\star\chi_i^{(1)})\sqcup(A^\star\chi_r^{(3)},A^\star\chi_{r-1}^{(3)},\dots,A^\star\chi_j^{(3)})\right)\cdot A^\star$.

Proof. By definition, u belongs to $A^\star\cdot\chi_1^{(1)}\cdot A^\star\cdot\chi_2^{(1)}\dots A^\star\cdot\chi_i^{(1)}\cdot A^\star$ and to $A^\star\cdot\chi_r^{(3)}\cdot A^\star\cdot\chi_{r-1}^{(3)}\dots A^\star\cdot\chi_j^{(3)}\cdot A^\star$. Moreover, by Remark 2.55 (p.77), all $\chi_k^{(1)}$ are words on the alphabet $\{L,D\}$ while all $\chi_k^{(3)}$ are words on $\{U,R\}$. These alphabets being disjoint, we conclude that u belongs to $\left((A^\star\chi_1^{(1)},A^\star\chi_2^{(1)},\dots,A^\star\chi_i^{(1)})\sqcup(A^\star\chi_r^{(3)},A^\star\chi_{r-1}^{(3)},\dots,A^\star\chi_j^{(3)})\right)\cdot A^\star$. ■



A
f
q_{12}
$\mathcal{A}^{\oplus}(\xi_1,\xi_2)$
$\mathcal{A}^{\oplus}(\xi_2,\xi_3)$
$\mathcal{A}(\xi_1,\xi_3)$
$\mathcal{A}^{\oplus}(\xi_{r-2},\xi_{r-1})$
$\mathcal{A}^{\oplus}(\xi_{r-1},\xi_r)$
$\mathcal{A}(\xi_1,\xi_{r-1})$
$\mathcal{A}(\xi_2,\xi_{r-1})$
$q_{1(r-1)}$
$\mathcal{A}(\xi_1,\xi_r)$
$\mathcal{A}(\xi_2,\xi_r)$
$\mathcal{A}(\xi_{r-2},\xi_r)$
q_{1r}
q_{2r}
$q_{(r-1)r}$

Figure 2.24: Automaton $\mathcal{A}_\pi$ for $\pi = \oplus[\xi_1,\ldots,\xi_r]$ where every ξ_i is an increasing oscillation.

The following lemma characterizes the first visit of state q_{ij} in $\mathcal{A}_\pi$:

Lemma 2.90. *Let Q be the set of states of $\mathcal{A}_\pi$ (see Figure 2.24), $(i,j) \neq (1,r)$ and $u \in A^\star$. Then $trace_{\mathcal{A}_\pi}(u) \in (Q \setminus q_{ij})^\star \cdot q_{ij}$ if and only if $u = vw$ with either $w \in \chi^{(1)}_{i-1}$, $i(v) = i-2$ and $j(v) = j+1$; or $w \in \chi^{(3)}_{j+1}$, $i(v) = i-1$ and $j(v) = j+2$.*

Proof. By induction on $r-j+i-1$, using $\mathcal{A}(\xi_i,\xi_j) = \mathcal{AC}(\chi^{(1)}_i,\chi^{(3)}_j)$. Notice that $r-j+i-1$ is the number of automata $\mathcal{A}(\xi_k,\xi_\ell)$ that we need to go through before reaching q_{ij}. ■

Theorem 2.91. *If $\pi = \oplus[\xi_1,\ldots,\xi_r]$ where every ξ_i is an increasing oscillation then automaton $\mathcal{A}_\pi$ given in Figure 2.24 recognizes the language $\overleftarrow{\mathcal{L}_\pi}$.*

Proof. Assume that $r \geq 3$ (otherwise $r = 2$, $\mathcal{A}_\pi$ is $\mathcal{A}^{\oplus}(\xi_1,\xi_2)$ and the result trivially holds). We first prove that every word recognized by $\mathcal{A}_\pi$ is in $\overleftarrow{\mathcal{L}_\pi}$. Let u be a word recognized by $\mathcal{A}_\pi$. Then $trace_{\mathcal{A}_\pi}(u)$ ends with the final state f of $\mathcal{A}_\pi$. As f is accessible only from some $\mathcal{A}^{\oplus}(\xi_k,\xi_{k+1})$, $trace_{\mathcal{A}_\pi}(u)$ contains some $q_{k(k+1)}$. Moreover for all $(i,j) \neq (1,r)$, every path from the initial state q_{1r} to q_{ij} contains $q_{(i-1)j}$ or $q_{i(j+1)}$. Therefore $trace_{\mathcal{A}_\pi}(u) \in q_{i_1j_1}Q^\star q_{i_2j_2}Q^\star \ldots q_{i_{r-1}j_{r-1}}Q^\star f$ where $(i_1,j_1) = (1,r)$, $(i_{r-1},j_{r-1}) = (k,k+1)$ and for all ℓ, $(i_{\ell+1},j_{\ell+1}) \in \{(i_\ell+1,j_\ell),(i_\ell,j_\ell-1)\}$. Hence by definition of $\mathcal{A}(\xi_i,\xi_j)$ and $\mathcal{A}^{\oplus}(\xi_k,\xi_{k+1})$ and by the expression of $\overleftarrow{\mathcal{L}_\pi}$ given p.94, $u \in \overleftarrow{\mathcal{L}_\pi}$.

Conversely, let $u \in \overleftarrow{\mathcal{L}_\pi}$. We want to prove that $q_{1r} \cdot u = f$, q_{1r} being the initial state of $\mathcal{A}_\pi$ and f its final state. The expression of $\overleftarrow{\mathcal{L}_\pi}$ given p.94 ensures that there exists k such that $u = vw$ with $i(v) \geq k-1$, $j(v) \leq k+2$ and $w \in \overleftarrow{\mathcal{L}_{\oplus[\xi_k,\xi_{k+1}]}}$. Let $u = v'w'$ with $v' = v_1 \ldots v_s$ the shortest prefix of u such that $j(v') - i(v') \leq 3$, and set $i = i(v')$. Since

v' is of minimal length, $j(v') = i+3$ and there exists $v'' \in A^\star$ such that $v = v'v''$. So $w' = v''w$ belongs to $A^\star \cdot \overleftarrow{\mathcal{L}_{\oplus[\xi_k,\xi_{k+1}]}} = \overleftarrow{\mathcal{L}_{\oplus[\xi_k,\xi_{k+1}]}}$. Thus, using also Lemma 2.89, we have:

$$\begin{aligned} u &= \overbrace{\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\in (A^\star\chi_1^{(1)},\dots,A^\star\chi_{k-1}^{(1)}) \sqcup (A^\star\chi_r^{(3)},\dots,A^\star\chi_{k+2}^{(3)})}}^{v}\ \overbrace{\underbrace{\qquad\qquad}_{\in \overleftarrow{\mathcal{L}_{\oplus[\xi_k,\xi_{k+1}]}}}}^{w} \\ &= \overbrace{\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\in (A^\star\chi_1^{(1)},\dots,A^\star\chi_{i}^{(1)}) \sqcup (A^\star\chi_r^{(3)},\dots,A^\star\chi_{i+3}^{(3)})}}^{v'}\ \overbrace{\qquad\qquad\qquad\qquad}^{w'} \end{aligned}$$

Since v' is of minimal length, $i(v_1 \dots v_{s-1}) < i(v')$ or $j(v_1 \dots v_{s-1}) > j(v')$. Thus $v' = \bar{v}\bar{w}$ with either $\bar{w} \in \chi^{(1)}_{i(v')}$, $i(\bar{v}) = i(v') - 1$ and $j(\bar{v}) = j(v')$; or $\bar{w} \in \chi^{(3)}_{j(v')}$, $i(\bar{v}) = i(v')$ and $j(\bar{v}) = j(v') + 1$. By Lemma 2.90, $trace_{\mathcal{A}_\pi}(v')$ ends with $q_{i(v')+1,j(v')-1}$.

Therefore $u = v'w'$ with $q_{1r} \cdot v' = q_{i+1,i+2}$ and $w' \in \overleftarrow{\mathcal{L}_{\oplus[\xi_k,\xi_{k+1}]}}$.

If $i = k-1$ then $q_{1r} \cdot u = (q_{1r} \cdot v') \cdot w' = q_{k,k+1} \cdot w' = f$ as w' belongs to the language $\overleftarrow{\mathcal{L}_{\oplus[\xi_k,\xi_{k+1}]}}$ recognized by the automaton $\mathcal{A}^\oplus(\xi_k, \xi_{k+1})$.

If $i \le k-3$. By definition $i(u) \ge k-1$ and $i(v') = i$, and as $u = v'w'$, $w' \in A^\star \cdot \chi^{(1)}_{i+1} \cdot A^\star \cdot \chi^{(1)}_{i+2} \cdot A^\star \dots A^\star \cdot \chi^{(1)}_{k-1} \cdot A^\star$. Therefore as $i \le k-3$, $w' \in A^\star \cdot \chi^{(1)}_{i+1} \cdot A^\star \cdot \chi^{(1)}_{i+2} \cdot A^\star$ and w' belongs to the language $\overleftarrow{\mathcal{L}_{\oplus[\xi_{i+1},\xi_{i+2}]}}$ recognized by the automaton $\mathcal{A}^\oplus(\xi_{i+1}, \xi_{i+2})$. Finally as $u = v'w'$ and $trace_{\mathcal{A}_\pi}(v')$ ends with $q_{i+1,i+2}$, $trace_{\mathcal{A}_\pi}(u)$ ends with f.

If $i \ge k+1$ then $j(v') \ge k+4$ and by symmetry of $i(u)$ and $j(u)$ the proof is similar to the previous case.

If $i = k-2$. As $v = v'v''$ and $i(v) \ge k-1$ and $i(v') = i$ then $v'' \in A^\star \cdot \chi^{(1)}_{i+1} \cdot A^\star$. Moreover $w \in \overleftarrow{\mathcal{L}_{\oplus[\xi_{i+2},\xi_{i+3}]}}$ so $w' = v''w \in A^\star \cdot \chi^{(1)}_{i+1} \cdot \overleftarrow{\mathcal{L}_{\oplus[\xi_{i+2},\xi_{i+3}]}}$. Therefore $w' \in \overleftarrow{\mathcal{L}_{\oplus[\xi_{i+1},\xi_{i+2},\xi_{i+3}]}}$ and by Theorem 2.24(*ii*) (p.65), $w' \in \overleftarrow{\mathcal{L}_{\oplus[\xi_{i+1},\xi_{i+2}]}}$. Hence w' is recognized by $\mathcal{A}^\oplus(\xi_{i+1}, \xi_{i+2})$ so $q_{1r} \cdot u = f$ (since $q_{1r} \cdot v' = q_{i+1,i+2}$).

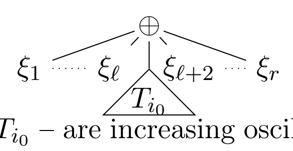
If $i = k$, by symmetry of $i(u)$ and $j(u)$ the proof is similar to the previous case, concluding the proof of Theorem 2.91. ■

Remark 2.92. With the optimized construction of $\mathcal{A}_\pi$, we prove similarly that $\mathcal{A}_\pi$ recognizes a language $\mathcal{L}$ such that $\mathcal{L} \cap \mathcal{M} = \overleftarrow{\mathcal{L}_\pi} \cap \mathcal{M}$.

We end this section with a remark which will be useful in Subsection 2.5.6.

Remark 2.93. Let $\pi^{(1)} = \oplus[\xi_2, \dots, \xi_r]$ be the pattern of π obtained by deletion of the elements of ξ_1. If $r \ge 3$ then $\mathcal{A}_{\pi^{(1)}}$ is obtained by taking q_{2r} (see Figure 2.24) as initial state and by considering only the states of $\mathcal{A}_\pi$ that are accessible from q_{2r}. Thus in $\mathcal{A}_\pi$ the language recognized from q_{2r} is $\overleftarrow{\mathcal{L}_{\pi^{(1)}}}$. If $r = 2$ then $\pi^{(1)} = \xi_2$, $\mathcal{A}_{\pi^{(1)}}$ is also a part of $\mathcal{A}^\oplus(\xi_1, \xi_2) = \mathcal{A}_\pi$ and $\overleftarrow{\mathcal{L}_{\pi^{(1)}}}$ is the language recognized from the bottom right state of Figure 2.23. The same property holds with the pattern $\pi^{(r)} = \oplus[\xi_1, \dots, \xi_{r-1}]$, the state $q_{1(r-1)}$ and the top left state of Figure 2.23.

2.5.4 Pin-permutations with a linear root: recursive case

Suppose w.l.o.g. that the decomposition tree of π is 

, i.e., the root has label $\oplus$ and all of its children but one – denoted T_{i_0} – are increasing oscillations. Then the automaton $\mathcal{A}(T_{i_0}) = \mathcal{A}_\rho$ associated to the permutation ρ whose decomposition tree is T_{i_0} is recursively obtained. We explain how to build $\mathcal{A}_\pi$ from $\mathcal{A}_\rho$.

If $\pi \notin \mathcal{H}$, i.e. if π does not satisfy any condition of Figure 2.16 (p.81). Then Theorem 2.60 (p.80) ensures that $P(\pi) = P(\rho) \cdot \mathfrak{P}^{(1)}_{(\ell)} ⧢ \mathfrak{P}^{(3)}_{(\ell+2)}$ with

$$\mathfrak{P}^{(1)}_{(\ell)} = \left(P^{(1)}(\xi_\ell), \dots, P^{(1)}(\xi_1)\right) \text{ and } \mathfrak{P}^{(3)}_{(\ell+2)} = \left(P^{(3)}(\xi_{\ell+2}), \dots, P^{(3)}(\xi_r)\right).$$

This characterization translates into the following expression for $\overleftarrow{\mathcal{L}_\pi}$.

$$\overleftarrow{\mathcal{L}_\pi} = \left(\left(A^\star\chi^{(1)}_1, \dots, A^\star\chi^{(1)}_\ell\right) ⧢ \left(A^\star\chi^{(3)}_r, \dots, A^\star\chi^{(3)}_{\ell+2}\right)\right) \cdot \overleftarrow{\mathcal{L}_\rho}$$

To deal with the shuffle product, we use again the automata $\mathcal{A}(\xi_i, \xi_j)$ whose initial and final states are s_{ij}, $f^{(1)}_{ij}$ and $f^{(3)}_{ij}$ (see Figure 2.22 p.94). We furthermore introduce the deterministic automata $\mathcal{A}^{(1)}(\xi_i) = \mathcal{AC}(\chi^{(1)}_i)$ for $1 \le i \le \ell$ and $\mathcal{A}^{(3)}(\xi_j) = \mathcal{AC}(\chi^{(3)}_j)$ for $\ell + 2 \le j \le r$ whose initial and final states are denoted respectively $s^{(1)}_i$, $f^{(1)}_i$, $s^{(3)}_j$ and $f^{(3)}_j$. The automaton $\mathcal{A}^{(1)}(\xi_i)$ (resp. $\mathcal{A}^{(3)}(\xi_j)$) corresponds to the reading of parts of $\mathfrak{P}^{(1)}_{(\ell)}$ $\left(\text{resp. } \mathfrak{P}^{(3)}_{(\ell+2)}\right)$ in the shuffle product $\mathfrak{P}^{(1)}_{(\ell)} ⧢ \mathfrak{P}^{(3)}_{(\ell+2)}$ after all the parts of $\mathfrak{P}^{(3)}_{(\ell+2)}$ $\left(\text{resp. } \mathfrak{P}^{(1)}_{(\ell)}\right)$ are read.

With these notations, the language $\overleftarrow{\mathcal{L}_\pi}$ associated to π is the one recognized by the automaton $\mathcal{A}_\pi$ given in Figure 2.25 where the following merges are performed:

- for any i, j such that $1 \le i \le \ell$ and $\ell + 2 \le j \le r$, s_{ij}, $f^{(1)}_{(i-1)j}$ and $f^{(3)}_{i(j+1)}$ are merged into a unique state q_{ij} that is neither initial (except when $i = 1$ and $j = r$) nor final,
- for $1 \le i \le \ell$, $s^{(1)}_i$, $f^{(1)}_{(i-1)}$ and $f^{(3)}_{i(\ell+2)}$ are merged into a unique state q_i that is neither initial nor final,
- for $\ell + 2 \le j \le r$, $s^{(3)}_j$, $f^{(3)}_{j+1}$ and $f^{(1)}_{\ell j}$ are merged into a unique state q_j that is neither initial nor final,
- $f^{(3)}_{\ell+2}$, $f^{(1)}_\ell$ and the initial state of $\mathcal{A}_\rho$ are merged into a unique state q_ρ that is neither initial nor final.

Note that if $\ell = 0$ (resp. $r = \ell + 1$) i.e., if T_{i_0} is the first (resp. last) child, then only the automaton $\mathcal{A}_\rho$ and the automata $\mathcal{A}^{(3)}(\xi_j)$ (resp. $\mathcal{A}^{(1)}(\xi_i)$) appear in $\mathcal{A}_\pi$ whose initial state is then q_r (resp. q_1).

The proof that the automaton $\mathcal{A}_\pi$ obtained by this construction recognizes $\overleftarrow{\mathcal{L}_\pi}$ is omitted. However this construction is very similar to the non-recursive case of the previous section where the proofs are detailed.

Lemma 2.94. *For any pin-permutation $\pi = \oplus[\xi_1, \dots, \xi_\ell, \rho, \xi_{\ell+2}, \dots, \xi_r]$ such that every ξ_i but ρ is an increasing oscillation and $\pi \notin \mathcal{H}$, the construction of the automaton $\mathcal{A}_\pi$ (see Figure 2.25) is done in time and space $\mathcal{O}\left((|\pi| - |\rho|)^2\right)$ plus the additional complexity due to the construction of $\mathcal{A}_\rho$, both in the classical and the optimized construction.*

Proof. First notice that $|\pi| - |\rho| = \sum_{i=1, i\neq\ell+1}^{r} |\xi_i|$. Moreover, taking into account the merge of states:

$$|\mathcal{A}_\pi| \le \sum_{i=1}^{\ell} \sum_{j=\ell+2}^{r} |\mathcal{A}(\xi_i, \xi_j)| + \sum_{i=1}^{\ell} |\mathcal{A}^{(1)}(\xi_i)| + \sum_{j=\ell+2}^{r} |\mathcal{A}^{(3)}(\xi_j)| + |\mathcal{A}_\rho|.$$

From Lemma 2.84 (p.94) and the fact that $|P^{(1)}(\xi_i)| \le 2$ and $|P^{(3)}(\xi_j)| \le 2$ (see Remark 2.55 p.77), it follows that

$$|\mathcal{A}_\pi| - |\mathcal{A}_\rho| = \mathcal{O}\left(\sum_{i=1}^{\ell} \sum_{j=\ell+2}^{r} (|\xi_i| + |\xi_j|) + \sum_{\substack{i=1 \\ i\neq\ell+1}}^{r} |\xi_i|\right) = \mathcal{O}((|\pi| - |\rho|)^2),$$

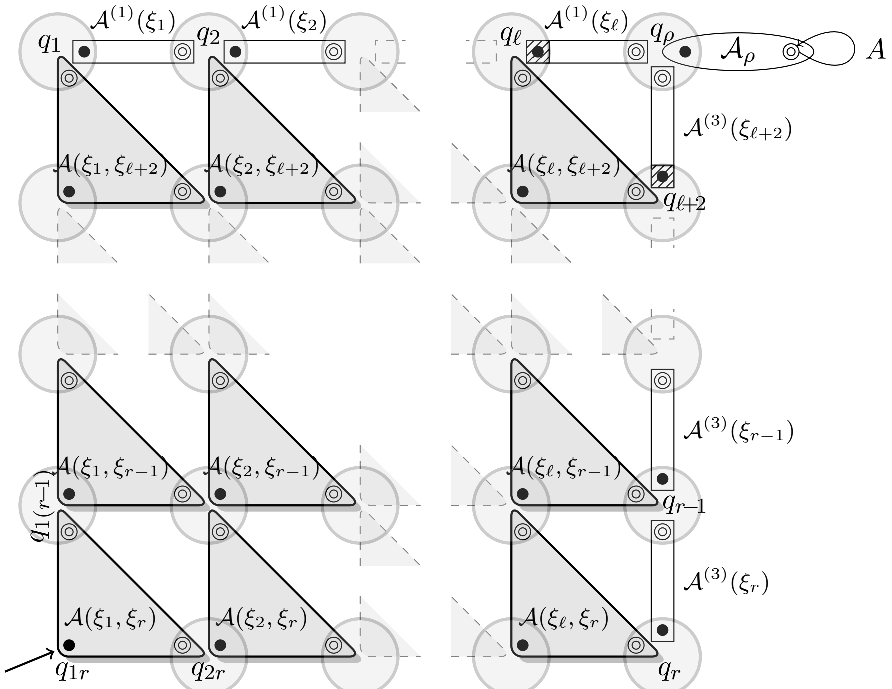

$\mathcal{A}^{(1)}(\xi_1)$
$\mathcal{A}^{(1)}(\xi_2)$
$\mathcal{A}^{(1)}(\xi_\ell)$
q_1
q_2
q_ℓ
q_ρ
$\mathcal{A}_\rho$
A
$\mathcal{A}^{(3)}(\xi_{\ell+2})$
$\mathcal{A}(\xi_1,\xi_{\ell+2})$
$\mathcal{A}(\xi_2,\xi_{\ell+2})$
$\mathcal{A}(\xi_\ell,\xi_{\ell+2})$
$q_{\ell+2}$
$\mathcal{A}^{(3)}(\xi_{r-1})$
$\mathcal{A}(\xi_1,\xi_{r-1})$
$\mathcal{A}(\xi_2,\xi_{r-1})$
$\mathcal{A}(\xi_\ell,\xi_{r-1})$
$q_{1(r-1)}$
q_{r-1}
$\mathcal{A}^{(3)}(\xi_r)$
$\mathcal{A}(\xi_1,\xi_r)$
$\mathcal{A}(\xi_2,\xi_r)$
$\mathcal{A}(\xi_\ell,\xi_r)$
q_{1r}
q_{2r}
q_r

Figure 2.25: The automaton $\mathcal{A}_\pi$ for $\pi = \oplus[\xi_1, \dots, \xi_\ell, \rho, \xi_{\ell+2}, \dots, \xi_r]$, where every ξ_i but ρ is an increasing oscillation (in the case $\pi \notin \mathcal{H}$).

concluding the proof, since the time of the construction is linear w.r.t. the size of the automaton. ■

We end this paragraph with a remark which will be useful in Subsection 2.5.6.

Remark 2.95. If $\ell \neq 0$, let $\pi^{(1)}$ be the pattern of π obtained by deletion of the elements of ξ_1. Then $\mathcal{A}_{\pi^{(1)}}$ is obtained by taking q_{2r} (see Figure 2.25) as initial state and by considering only the states of $\mathcal{A}_\pi$ that are accessible from q_{2r}. Thus in $\mathcal{A}_\pi$ the language recognized from q_{2r} is $\overleftarrow{\mathcal{L}_{\pi^{(1)}}}$. If $r \neq \ell + 1$ the same property holds with the pattern $\pi^{(r)}$ (obtained by deletion of the elements of ξ_r) and the state $q_{1(r-1)}$. We take the convention that $q_{1(\ell+1)} = q_1$, $q_{(\ell+1)r} = q_r$ and $q_{(\ell+1)(\ell+1)} = q_\rho$.

If $\pi \in \mathcal{H}$, i.e. if one of the conditions given in Figure 2.16 (p.81) holds for π. Then Theorem 2.60 (p.80) ensures that $P(\pi)$ is the union of the set $P_0 = P(\rho)\cdot\mathfrak{P}^{(1)}_{(\ell)} \sqcup \mathfrak{P}^{(3)}_{(\ell+2)}$ that we consider in the previous paragraph and some other sets that are very similar, all ending with the same kind of shuffle product. As the automaton $\mathcal{A}_\pi$ recognizes reversed words, these similar ends lead to similar beginnings in the automaton. So the automaton $\mathcal{A}_\pi$ has the same general structure as automaton $\mathcal{A}$ of Figure 2.25 but some transitions are added to account for the words in $P(\pi)$ not belonging to P_0.

More precisely $\mathcal{A}_\pi$ is obtained by applying the following modifications to the automaton $\mathcal{A}$ of Figure 2.25. First we add new paths as depicted in the last column of Figure 2.16 (p.81). These paths start in the shaded states q_ℓ or $q_{\ell+2}$ of Figure 2.25 and arrive in marked states of $\mathcal{A}_\rho$. If a path is labeled in Figure 2.16 by a word w with s letters we build $s - 1$ new states and s transitions such that the reading of w from the shaded state leads to the corresponding marked states of $\mathcal{A}_\rho$. These marked states may be seen as initial states of subautomata: in Figure 2.16, for all Y, q_Y is a state of $\mathcal{A}_\rho$ such that the language recognized from q_Y is $\overleftarrow{\mathcal{L}_\sigma}$, where σ is the permutation whose diagram is Y. The way such states of $\mathcal{A}_\rho$ are marked is explained in Section 2.5.6.

Moreover to keep the resulting automaton deterministic and complete when adding these new paths we have to make some other changes. Notice that state q_ℓ (resp. $q_{\ell+2}$) is the initial state of $\mathcal{A}^{(1)}(\xi_\ell) = \mathcal{AC}(\chi_\ell^{(1)})$ (resp. $\mathcal{A}^{(3)}(\xi_{\ell+2}) = \mathcal{AC}(\chi_{\ell+2}^{(3)})$). Remark 2.55 (p.77) ensures that $\chi_\ell^{(1)} = \overleftarrow{\phi(P^{(1)}(\xi_\ell))}$ (resp. $\chi_{\ell+2}^{(3)} = \overleftarrow{\phi(P^{(3)}(\xi_{\ell+2}))}$) is defined on the alphabet $\{L, D\}$ (resp. $\{U, R\}$). Therefore, from Lemma 2.76 (p.90), transitions labeled by U or R (resp. L or D) leaving q_ℓ (resp. $q_{\ell+2}$) are loops on the initial state q_ℓ (resp. $q_{\ell+2}$) of $\mathcal{A}^{(1)}(\xi_\ell)$ (resp. $\mathcal{A}^{(3)}(\xi_{\ell+2})$). Hence, as can be seen in Figure 2.16, the new transitions leaving shaded states are labeled by directions that correspond to loops in $\mathcal{A}$. So we just have to delete the loops and replace them by the new transitions in order to preserve the determinism of the automaton.

Now we make some other changes to preserve completeness and ensure that even though we have deleted loops on shaded states, all words that were recognized by the automaton $\mathcal{A}$ are still recognized by the modified automaton $\mathcal{A}_\pi$. As q_ℓ (resp. $q_{\ell+2}$) is not reachable from $q_{\ell+2}$ (resp. q_ℓ) we can handle separately new states reachable from q_ℓ and new states reachable from $q_{\ell+2}$. Let q_0 be q_ℓ (resp. $q_{\ell+2}$). As in the Aho-Corasick algorithm we label any new state q reachable from q_0 by the shortest word labeling a path from q_0 to q. So these labels begin with U or R (resp. L or D) (see Figure 2.16). Notice that the states in the part $\mathcal{A}^{(1)}(\xi_\ell)$ (resp. $\mathcal{A}^{(3)}(\xi_{\ell+2})$) of $\mathcal{A}$ are also labeled in such a way, but their labels differ from the ones of the new states since they contain only letters L or D (resp. U or R). By Lemma 2.76 (p.90), we know that in $\mathcal{A}^{(1)}(\xi_\ell)$ (resp. $\mathcal{A}^{(3)}(\xi_{\ell+2})$) all transitions labeled by U or R (resp. L or D) go to q_0, therefore we replace them by transitions going to the new state labeled by U or R (resp. L or D) if such a new state exists (otherwise we keep the transition going to q_0). We complete the construction by adding missing transitions from the states newly created: for any such state q, the transition from q labeled by Z goes to the longest suffix of $q \cdot Z$ that is a state of the automaton – either a new state or a state of $\mathcal{A}^{(1)}(\xi_\ell)$ (resp. $\mathcal{A}^{(3)}(\xi_{\ell+2})$).

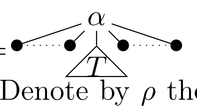
The proof that the automaton $\mathcal{A}_\pi$ obtained by this construction recognizes $\overleftarrow{\mathcal{L}_\pi}$ is omitted to avoid the examination of the eight cases of Figure 2.16. However, it is similar to the proof of Theorem 2.98 (p.103), with some of the difficulties released (since labels on the new paths are explicit in Figure 2.16, while they are not in the proof of Theorem 2.98).

Lemma 2.96. *The complexity of building $\mathcal{A}_\pi$ given in Lemma 2.94 (p.99) still holds if $\pi \in \mathcal{H}$.*

Proof. When $\pi \in \mathcal{H}$, the construction of $\mathcal{A}_\pi$ is the same as in the case $\pi \notin \mathcal{H}$, with some new paths added. There are at most four new paths, $\mathcal{O}(|\rho|)$ new states in each path, $\mathcal{O}(|\rho|)$ transitions from these new states, and the modification of transitions in $\mathcal{A}^{(1)}(\xi_\ell)$ (resp. $\mathcal{A}^{(3)}(\xi_{\ell+2})$) is done in $\mathcal{O}(|\xi_\ell|)$ (resp. $\mathcal{O}(|\xi_{\ell+2}|)$). So in the construction of $\mathcal{A}_\pi$ described above, we have to add a time and space complexity $\mathcal{O}(|\rho| + |\xi_\ell| + |\xi_{\ell+2}|)$ w.r.t. the case $\pi \notin \mathcal{H}$. As $|\xi_\ell| + |\xi_{\ell+2}| = \mathcal{O}\left((|\pi| - |\rho|)\right)$ and as the complexity of the construction of $\mathcal{A}_\rho$ is bigger than $\mathcal{O}(|\rho|)$, this does not change the overall estimation of the complexity of the construction of $\mathcal{A}_\pi$ given in Lemma 2.94. ■

2.5.5 Pin-permutations with a prime root: recursive case

Exactly one child of the root is not a leaf.

Let $\pi =$ 

where α is a simple permutation all of whose children but T are leaves. Denote by ρ the permutation whose decomposition tree is T, and by x the point of α expanded by T.

Recall that $Q_x(\alpha)$ (see Definition 2.64 p.84) denotes the set of strict pin words obtained by deleting the first letter of quasi-strict pin words of α whose first point read in α is x.

If π does not satisfy condition $(\mathcal{C})$ (see Definition 2.62 p.83). Then from Theorem 2.66 (p.85), $P(\pi) = P(\rho) \cdot Q_x(\alpha)$. So $\overleftarrow{\mathcal{L}_\pi} = A^\star \cdot \overleftarrow{\phi(Q_x(\alpha))} \cdot \overleftarrow{\mathcal{L}_\rho}$ and since $\overleftarrow{\mathcal{L}_\rho} = A^\star \cdot \overleftarrow{\mathcal{L}_\rho}$ the automaton $\mathcal{A}_\pi$ recognizing $\overleftarrow{\mathcal{L}_\pi}$ is obtained by the concatenation of $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$ with $\mathcal{A}_\rho$, which is recursively obtained.

If π satisfies condition $(\mathcal{C})$ and $|T| \geq 3$. Then by Theorem 2.66 (p.85) – and using the notations of this theorem, $P(\pi)$ contains $P(\rho) \cdot Q_x(\alpha)$ and some other words. Defining T' as in condition $(\mathcal{C})$, ρ' the permutation whose decomposition tree is T', and w the unique word encoding the unique reading of the remaining leaves in π after T' is read when T is read in two pieces, these other words are $P(\rho') \cdot w$. Note that from Lemma 2.68 (p.86) w is a strict pin word. So $\overleftarrow{\mathcal{L}_\pi} = A^\star \cdot \overleftarrow{\phi(Q_x(\alpha))} \cdot \overleftarrow{\mathcal{L}_\rho} \cup A^\star \cdot \overleftarrow{\phi(w)} \cdot \overleftarrow{\mathcal{L}_{\rho'}}$. The skeleton of $\mathcal{A}_\pi$ is the concatenation of the automaton $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$ with $\mathcal{A}_\rho$ and then as in the recursive case with a linear root, we add some new transitions to account for the words in $P(\rho') \cdot w$.

Denoting Z the last letter of w (i.e., the first letter of $\overleftarrow{\phi(w)}$), Lemma 2.68 ensures that no word of $\overleftarrow{\phi(Q_x(\alpha))}$ contains Z and therefore by Lemma 2.76 (p.90) all the transitions labeled by Z in $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$ go to q_0. We built an automaton $\mathcal{A}$ by performing the following modifications on $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$: remove the loop labeled by Z on q_0 and add a path reading $\overleftarrow{\phi(w)}$ from q_0 to a new final state f'. Label all states q of $\mathcal{A}$ by the shortest word labeling a path from the initial state q_0 to q. Replace any transition labeled by Z from a state q of $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$ to q_0 by a transition from q to the new state labeled by Z. Finally complete the automaton with transitions from the states of the added path: for all such states q but f', the transition from q labeled by a goes to the longest suffix of $q \cdot a$ that is a state of the automaton – either a new state or a pre-existing state. Notice that the automaton $\mathcal{A}$ we obtain is almost complete and has exactly two final states, without outgoing transitions: f – the unique final state of $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$ – and f'.

The automaton $\mathcal{A}_\pi$ is then obtained from $\mathcal{A}$ and $\mathcal{A}_\rho$ by merging f with the initial state q_T of $\mathcal{A}_\rho$ and f' with a marked state $q_{T'}$ (see Section 2.5.6) of $\mathcal{A}_\rho$ which is a state from which the recognized language is $\overleftarrow{\mathcal{L}_{\rho'}}$. This construction is shown in Figure 2.26.

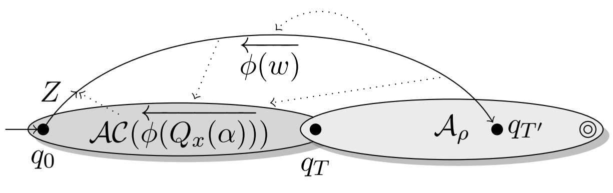

$\overleftarrow{\phi(w)}$
Z
$\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$
$\mathcal{A}_\rho$
$q_{T'}$
q_0
q_T

Figure 2.26: Automaton $\mathcal{A}_\pi$ for $\pi = \alpha[1, \dots, 1, \rho, 1, \dots, 1]$.

Notice that the automaton $\mathcal{A}$ obtained from $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$ is somehow very similar to $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))}, \{\overleftarrow{\phi(w)}\})$ but because $\overleftarrow{\phi(w)}$ has a suffix in $\overleftarrow{\phi(Q_x(\alpha))}$ (from Lemma 2.68), the sets of words $X_1 = \overleftarrow{\phi(Q_x(\alpha))}$ and $X_2 = \{\overleftarrow{\phi(w)}\}$ do not satisfy the independence condition required in our construction of $\mathcal{AC}(X_1, X_2)$.

Lemma 2.97. *The automaton $\mathcal{A}$ of the above construction recognizes the set of words ending with a first occurrence of a word of $\overleftarrow{\phi(Q_x(\alpha))}$. Moreover for any word u recognized by $\mathcal{A}$, $q_0 \cdot u = f'$ if $\overleftarrow{\phi(w)}$ is a suffix of u, and $q_0 \cdot u = f$ otherwise.*

Proof. From Lemma 2.68 (p.86), there exists a word $\bar{w} \in \overleftarrow{\phi(Q_x(\alpha))}$ and a letter $Z \in A$ such that $\overleftarrow{\phi(w)} = Z\bar{w}$. Moreover no word of $\overleftarrow{\phi(Q_x(\alpha))}$ contains Z.

Therefore by construction, merging states f and f' of $\mathcal{A}$ into a unique final state, we would obtain the automaton $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))} \cup \{\overleftarrow{\phi(w)}\})$. Consequently, since $\overleftarrow{\phi(w)}$ has a suffix in $\overleftarrow{\phi(Q_x(\alpha))}$, the automaton $\mathcal{A}$ recognizes the set of words ending with a first occurrence of a word of $\overleftarrow{\phi(Q_x(\alpha))}$.

Let u be a word ending with its first occurrence of a word of $\overleftarrow{\phi(Q_x(\alpha))}$, then u does not have any factor in $\overleftarrow{\phi(Q_x(\alpha))} \cup \{\overleftarrow{\phi(w)}\}$ except as a suffix. Lemma 2.76 (p.90) ensures that $q_0 \cdot u$ is the state labeled with longest suffix of u that is also a prefix of a word of $\overleftarrow{\phi(Q_x(\alpha))} \cup \{\overleftarrow{\phi(w)}\}$, concluding the proof. ■

Lemma 2.97 allows us to prove the correctness of the above construction of $\mathcal{A}_\pi$. The idea is the following: if $u \in A^\star \cdot \overleftarrow{\phi(Q_x(\alpha))} \cdot \overleftarrow{\mathcal{L}_\rho}$ (resp. $A^\star \cdot \overleftarrow{\phi(w)} \cdot \overleftarrow{\mathcal{L}_{\rho'}}$) and if $trace_{\mathcal{A}_\pi}(u)$ contains $q_{T'}$ (resp. q_T) and not q_T (resp. $q_{T'}$) before, then u is still accepted by $\mathcal{A}_\pi$ since $\overleftarrow{\mathcal{L}_\rho} \subseteq \overleftarrow{\mathcal{L}_{\rho'}}$ (resp. $\overleftarrow{\phi(w)} \cdot \overleftarrow{\mathcal{L}_{\rho'}} \subseteq \overleftarrow{\mathcal{L}_\rho}$). This is formalized in the following theorem.

Theorem 2.98. *The automaton $\mathcal{A}_\pi$ obtained by the above construction recognizes $\overleftarrow{\mathcal{L}_\pi}$.*

Proof. Recall that $\overleftarrow{\mathcal{L}_\pi} = A^\star \cdot \overleftarrow{\phi(Q_x(\alpha))} \cdot \overleftarrow{\mathcal{L}_\rho} \cup A^\star \cdot \overleftarrow{\phi(w)} \cdot \overleftarrow{\mathcal{L}_{\rho'}}$. The above construction ensures that every word accepted by $\mathcal{A}_\pi$ belongs to the language $\overleftarrow{\mathcal{L}_\pi}$. Conversely let us prove that every word of $\overleftarrow{\mathcal{L}_\pi}$ is accepted by $\mathcal{A}_\pi$.

Let u be a word of $\overleftarrow{\mathcal{L}_\pi}$. From Lemma 2.68 (p.86), there is a word $\bar{w} \in \overleftarrow{\phi(Q_x(\alpha))}$ and a letter $Z \in A$ such that $\overleftarrow{\phi(w)} = Z\bar{w}$. Therefore u has a factor in $\overleftarrow{\phi(Q_x(\alpha))}$. Hence we can decompose u uniquely as $u = u_1u_2$ where u_1 is the prefix of u ending with the first occurrence of a factor in $\overleftarrow{\phi(Q_x(\alpha))}$. Consequently from Lemma 2.97 $q_0 \cdot u_1$ is either q_T or $q_{T'}$, namely $q_0 \cdot u_1 = q_{T'}$ if $\overleftarrow{\phi(w)}$ is a suffix of u_1 and $q_0 \cdot u_1 = q_T$ otherwise.

Moreover, since u belongs to $\overleftarrow{\mathcal{L}_\pi}$, and because by definition $\overleftarrow{\mathcal{L}_\rho} = A^\star \cdot \overleftarrow{\mathcal{L}_\rho}$ (and similarly for ρ'), we deduce that u_2 belongs to $\overleftarrow{\mathcal{L}_\rho}$ or $\overleftarrow{\mathcal{L}_{\rho'}}$. Let us finally notice that, since $\rho' \leq \rho$, Theorem 2.24 (p.65) ensures that $\overleftarrow{\mathcal{L}_\rho} \subseteq \overleftarrow{\mathcal{L}_{\rho'}}$ thus $u_2 \in \overleftarrow{\mathcal{L}_{\rho'}}$.

If $q_0 \cdot u_1 = q_{T'}$ then as $u_2 \in \overleftarrow{\mathcal{L}_{\rho'}}$, u is recognized by $\mathcal{A}_\pi$. Assume on the contrary that $q_0 \cdot u_1 = q_T$. Then $q_0 \cdot u = q_T \cdot u_2$ and by definition of q_T it is enough to prove that $u_2 \in \overleftarrow{\mathcal{L}_\rho}$.

Assume first that $u \notin A^\star \cdot \overleftarrow{\phi(w)} \cdot \overleftarrow{\mathcal{L}_{\rho'}}$. Then since $u \in \overleftarrow{\mathcal{L}_\pi}$, we have $u \in A^\star \cdot \overleftarrow{\phi(Q_x(\alpha))} \cdot \overleftarrow{\mathcal{L}_\rho}$. Because u_1 ends with the first occurrence of a factor of $\overleftarrow{\phi(Q_x(\alpha))}$, we deduce that $u_2 \in A^\star \cdot \overleftarrow{\mathcal{L}_\rho} = \overleftarrow{\mathcal{L}_\rho}$.

Otherwise $u \in A^\star \cdot \overleftarrow{\phi(w)} \cdot \overleftarrow{\mathcal{L}_{\rho'}}$. Recall that u_1 is the prefix of u ending with the first occurrence of a factor of $\overleftarrow{\phi(Q_x(\alpha))}$. First (using also Lemma 2.97 and $q_0 \cdot u_1 = q_T$), this implies that $\overleftarrow{\phi(w)}$ is not a suffix of u_1. And second, this also implies that $\overleftarrow{\phi(w)}$ is not a factor of u_1. But by assumption $\overleftarrow{\phi(w)}$ is a factor of u. We claim that the first occurrence of $\overleftarrow{\phi(w)}$ in u starts after the end of u_1. We have just proved that $\overleftarrow{\phi(w)}$ is not a factor of u_1. Moreover, $\overleftarrow{\phi(w)} = Z\bar{w}$ starts with the letter Z, and from Lemma 2.68 (p.86) the $|\bar{w}|$ last letters of u_1 are different from Z (recall that all words of $\overleftarrow{\phi(Q_x(\alpha))}$ have the same length $|\alpha| = |\bar{w}|$). This proves our claim, and consequently, $u_2 \in A^\star \cdot \overleftarrow{\phi(w)} \cdot \overleftarrow{\mathcal{L}_{\rho'}}$. Let $v \in A^\star$ and $v' \in \overleftarrow{\mathcal{L}_{\rho'}}$ such that $u_2 = v \cdot \overleftarrow{\phi(w)} \cdot v'$. From Lemma 2.68 p.86, denoting by w' the suffix of length 2 of w, for all u' in $P(\rho')$, $u' \cdot \phi^{-1}(w')$ belongs to $P(\rho)$. Therefore $\overleftarrow{w'}\overleftarrow{\mathcal{L}_{\rho'}} \subseteq \overleftarrow{\mathcal{L}_{\rho}}$. But $v' \in \overleftarrow{\mathcal{L}_{\rho'}}$ and $\overleftarrow{w'}$ is a prefix of $\overleftarrow{\phi(w)}$, thus $u_2 = v \cdot \overleftarrow{\phi(w)} \cdot v' \in A^\star \cdot \overleftarrow{w'} \cdot A^\star \cdot \overleftarrow{\mathcal{L}_{\rho'}} \subseteq \overleftarrow{\mathcal{L}_{\rho}}$, concluding the proof. ■

Remark 2.99. With the optimized construction of $\mathcal{A}_\pi$, we prove similarly that $\mathcal{A}_\pi$ recognize a language $\mathcal{L}$ such that $\mathcal{L} \cap \mathcal{M} = \overleftarrow{\mathcal{L}_\pi} \cap \mathcal{M}$.

If π satisfies condition ($\mathcal{C}$) and $|T| = 2$. Then the construction is no longer recursive. Permutation π and its pin words are explicit. More precisely from Theorem 2.66 (p.85), $P(\pi) = P_{\{1,n\}}(\pi) \cup P_{\{2,n\}}(\pi) \cup P(T) \cdot Q_x(\alpha)$. Thus from Remark 2.71 (p.87),

$$\overleftarrow{\mathcal{L}_\pi} = \Big(\bigcup_{\substack{u \in P(\pi) \\ u \text{ strict or quasi-strict}}} \overleftarrow{\mathcal{L}(u)} \Big) \bigcup A^\star \cdot \overleftarrow{\phi(Q_x(\alpha))} \cdot \overleftarrow{\mathcal{L}_\rho}.$$

Therefore $\mathcal{A}_\pi$ is the automaton $\mathcal{U}^\circlearrowleft(\mathcal{A}_\pi^{\text{SQS}}, \mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))}) \cdot \mathcal{A}_\rho)$.

Lemma 2.100. *Let $\pi = \alpha[1, \dots, 1, \rho, 1, \dots, 1]$ where α is a simple permutation whose set $P(\alpha)$ of pin words is given. Then the construction of the automaton $\mathcal{A}_\pi$ is done in time and space $\mathcal{O}(|\pi| - |\rho|)$ plus the additional time and space due to the construction of $\mathcal{A}_\rho$, except when π satisfies condition ($\mathcal{C}$) and $|T| = 2$. In this latter case, the complexity is $\mathcal{O}(|\pi|^3)$ with the classical construction and $\mathcal{O}(|\pi|^2)$ in the optimized version.*

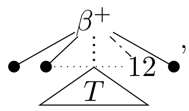
Proof. Recall that $Q_x(\alpha)$ contains words of size $|\alpha| - 1$. Its cardinality is smaller than the one of $P(\alpha)$, hence smaller than 48 (see Theorem 2.48 p.75). Moreover $Q_x(\alpha)$ can be determined in linear time w.r.t. $|\alpha|$ as described in Remark 2.67 (p.85). Consequently, $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$ is built in time and space $\mathcal{O}(|\alpha|) = \mathcal{O}(|\pi| - |\rho|)$.

If π does not satisfy condition ($\mathcal{C}$) then $\mathcal{A}_\pi = \mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))}) \cdot \mathcal{A}_\rho$, so that $|\mathcal{A}_\pi| = |\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})| + |\mathcal{A}_\rho|$ and the time complexity of this construction is $\mathcal{O}(|\pi| - |\rho|)$ plus the additional time to build $\mathcal{A}_\rho$.

If π satisfies condition ($\mathcal{C}$) and $|T| \geq 3$, then $|w| = |\alpha|$ and by Remark 2.70 (p.87), w is explicitly determined. Consequently, so is the additional path labeled by $\overleftarrow{\phi(w)}$ added to the automaton (see Figure 2.26). The modifications of the transitions between this path and $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})$ are performed in linear time w.r.t. the length of this path and $|\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))})|$, i.e., in $\mathcal{O}(|\phi(w)| + |\alpha|) = \mathcal{O}(|\pi| - |\rho|)$. We conclude that $\mathcal{A}_\pi$ is built in $\mathcal{O}(|\pi| - |\rho|)$ time and space plus the additional time and space to build $\mathcal{A}_\rho$.

If π satisfies condition ($\mathcal{C}$) and $|T| = 2$, then $\mathcal{A}_\pi = \mathcal{U}^\circlearrowleft(\mathcal{A}_\pi^{\text{SQS}}, \mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))}) \cdot \mathcal{A}_\rho)$. Recall that $P_{\text{SQS}}(\pi)$ is given in Remark 2.71 (p.87) and contains 12 pin words. Hence, with the classical construction (resp. in the optimized version), from Lemma 2.78 (p.92) (resp. Lemma 2.82 p.93) and Remark 2.71, we can build $\mathcal{A}_\pi^{\text{SQS}}$ in time and space $\mathcal{O}(|\pi|^2)$ (resp. $\mathcal{O}(|\pi|)$). Moreover since $|\rho| = 2$, $\mathcal{A}_\rho$ is obtained in constant time, so that $\mathcal{AC}(\overleftarrow{\phi(Q_x(\alpha))}) \cdot \mathcal{A}_\rho$ is obtained in time and space $\mathcal{O}(|\pi| - |\rho|) = \mathcal{O}(|\pi|)$. Finally, $\mathcal{A}_\pi$ is built in time and space $\mathcal{O}(|\pi|^3)$ (resp. $\mathcal{O}(|\pi|^2)$) with the classical (resp. optimized) construction. ■

Two children are not leaves.

Up to symmetry this means that $\pi =$ , where β^+ is an increasing quasi-oscillation, the permutation 12 expands an auxiliary point of β^+ and T expands the corresponding main substitution point of β^+.

Theorem 2.72 (p.87) ensures that the pin words encoding π are of the form $v.w$ where $v \in P(T)$ and w is a strict pin word of size $|\beta^+|$ uniquely determined by β^+ and the two points expanded in β^+, and known explicitly from Remark 2.73 (p.88).

Therefore $\overleftarrow{\mathcal{L}_\pi} = A^\star \overleftarrow{\phi(w)} \overleftarrow{\mathcal{L}_\rho}$ where ρ is the permutation whose decomposition tree is T. The automaton $\mathcal{A}_\pi$ recognizing $\overleftarrow{\mathcal{L}_\pi}$ is the concatenation of $\mathcal{AC}(\{\overleftarrow{\phi(w)}\})$ with $\mathcal{A}_\rho$, which is recursively obtained.

This construction is done in $\mathcal{O}\left(|\overleftarrow{\phi(w)}|\right) = \mathcal{O}\big(|\pi| - |\rho|\big)$ time and space in addition to the time and space complexity of the construction of $\mathcal{A}_\rho$.

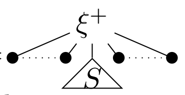
2.5.6 Marking states

In our constructions of Subsections 2.5.4 and 2.5.5 we need transitions going to initial states of subautomata. We could duplicate the corresponding subautomata. But when these are recursively obtained an exponential blow-up can occur. To keep a polynomial complexity we replace duplication by the marking of these special states. The marking is made on the fly during the construction and we explain how in this subsection.

The need of creating a transition going to a marked state (of a subautomaton) happens only when building the automaton $\mathcal{A}_\pi$ in Section 2.5.4 for a permutation π whose decomposition tree has a linear root and satisfies a condition (iHj) of Figure 2.16 (p.81), or in Section 2.5.5 for a permutation π whose decomposition tree has a prime root and satisfies condition $(\mathcal{C})$ (see Definition 2.62 p.83) with $|T| \geq 3$.

In both cases we need to mark in the subautomaton $\mathcal{A}_\rho$ with $\rho \leq \pi$ some states q_Y such that the language recognized taking q_Y as initial state is $\overleftarrow{\mathcal{L}_\sigma}$, where $\sigma \leq \rho$ is the permutation whose diagram (or decomposition tree) is Y.

As it appears in Figure 2.16 and in condition $(\mathcal{C})$, in almost all such situations, the marked state belongs to a subautomaton corresponding to a permutation ρ whose decomposition tree R has a linear root. There is only one situation where this root is prime: when π satisfies condition $(1H1+)$. We first focus on this case.

Prime root. Let θ be a permutation with decomposition tree $R =$  where ξ^+ is an increasing oscillation, and let γ be the permutation whose decomposition tree is S. In the case where π satisfies condition $(1H1+)$, we need to mark in the automaton $\mathcal{A}_\theta$ the state q such that when starting from q the language recognized is the one recognized by $\mathcal{A}_\gamma$ (notice that w.r.t. the previous paragraph, we have changed the notations ρ to θ and σ to γ to avoid confusions with the notations used in Subsection 2.5.5).

The automaton $\mathcal{A}_\theta$ is obtained as described in Subsection 2.5.5, when exactly one child of the root is not a leaf (indeed $|S| \geq 2$). The marking of state q depends on how the automaton $\mathcal{A}_\theta$ is built and in particular on whether θ satisfies condition $(\mathcal{C})$ or not.

Recall that ξ^+ is an increasing oscillation. If ξ^+ has a size at least 5, it is not a quasi-oscillation, and θ does not satisfy condition $(\mathcal{C})$. Therefore $\mathcal{A}_\theta$ is the concatenation of two automata, the second of which is $\mathcal{A}_\gamma$, whose initial state can be readily marked.

If ξ^+ has size 4, then $\xi^+ = 2\,4\,1\,3$ or $3\,1\,4\,2$ is a quasi-oscillation and θ may satisfy condition $(\mathcal{C})$. If it is not the case, $\mathcal{A}_\theta$ is obtained as above and so is the marking of state q. If on the contrary θ satisfies condition $(\mathcal{C})$, the construction of $\mathcal{A}_\theta$ depends on $|S|$. If $|S| \geq 3$, $\mathcal{A}_\theta$ is again the concatenation of two automata the second one being $\mathcal{A}_\gamma$, but with some states and transitions added. As these transitions are not reachable from the initial state of $\mathcal{A}_\gamma$, we mark it as above. If $|S| = 2$, then R has size 5 and the construction is not recursive anymore. We want to mark in $\mathcal{A}_\theta$ a state q corresponding to the initial state of $\mathcal{A}_\gamma$. But in the construction of $\mathcal{A}_\theta$ in Subsection 2.5.5, we have built an automaton $\mathcal{A}'$ such that $\mathcal{A}_\theta = \mathcal{U}^{\circlearrowleft}(\mathcal{A}_\theta^{\text{sqs}}, \mathcal{A}' \cdot \mathcal{A}_\gamma)$. Therefore $\mathcal{A}_\theta$ is a Cartesian product and the state q has been replicated several times. As $|S| = 2$, $\mathcal{A}_\gamma$ has a constant size, hence in this particular case we just duplicate it and mark its initial state instead of marking a state inside $\mathcal{A}_\theta$.

Linear root. Consider now the case where the decomposition tree R of the permutation ρ has a linear root. The need of a marked state in $\mathcal{A}_\rho$ happens only when the leftmost (resp. rightmost) child of R is a leaf z.

In almost all cases, the marked state q is such that the language accepted starting from q is the set of words encoding the readings of all nodes of R except the leaf z. There are at most two such leaves and from Remarks 2.93 and 2.95 (p.98 and 100), the corresponding marked states of $\mathcal{A}_\rho$ (which is built as described in Section 2.5.3 or 2.5.4) are $q_{1(r-1)}$ and q_{2r} in Figure 2.24 (p.97) or 2.25 (p.100) – with ρ instead of π. There is however one exception, corresponding to the special case described in Remark 2.93: when R has exactly two children, which are z and an increasing oscillation ξ. In this special case the construction of $\mathcal{A}_\rho$ is not recursive anymore. Instead of marking in $\mathcal{A}_\rho$ a state q corresponding to the initial state of $\mathcal{A}_\xi$, we just duplicate $\mathcal{A}_\xi$ and mark its initial state. If $|\xi| < 4$ then $|\mathcal{A}_\xi| = \mathcal{O}(1)$. Otherwise ξ is a simple permutation and $|\mathcal{A}_\xi|$ is quadratic (or linear in the optimized complexity) w.r.t. $|\xi|$. In both cases $|\mathcal{A}_\xi| + |\mathcal{A}_\rho|$ has the same order as $|\mathcal{A}_\rho|$ and since the construction is not recursive, this does not change the overall complexity of the construction of $\mathcal{A}_\pi$.

The few cases where the marked state q is not as above (i.e., is not such that the language accepted starting from q is the set of words encoding the readings of all nodes of R except z) correspond to state q_S of conditions $(2H2\star)$ and $(1H2\star)$ and states $q_{T\cup a}$ and $q_{T\cup b}$ of condition $(2H3)$. In these cases, R has exactly two children: z and a subtree R' whose root is linear. Then the leftmost (resp. rightmost) child of R' is a leaf z' and the marked state q is such that the language accepted starting from q is the set of words encoding the readings of all nodes of R' except the leaf z'. We are in the same situation as above except that we have to mark states in $\mathcal{A}_{\rho'}$ instead of $\mathcal{A}_\rho$, where ρ' is the permutation whose decomposition tree is R' and $\mathcal{A}_{\rho'}$ is a subautomaton of $\mathcal{A}_\rho$ built recursively.

Notice that we never create transitions going to marked states belonging to automata built more than two levels of recursion deeper. Indeed in all conditions above the created transitions go to the automaton built in the previous step of recursion, except for conditions $(2H3)$, $(2H2\star)$ and $(1H2\star)$ where two levels of recursion are involved.

2.5.7 Complexity analysis

Theorem 2.101. *For every pin-permutation π of size n, $\mathcal{A}_\pi$ is built in time and space $\mathcal{O}(n^2)$ in the optimized version and $\mathcal{O}(n^4)$ in the classical version.*

Proof. To build $\mathcal{A}_\pi$, we first need to decide which shape of Equation $(\star)$ is matched by the decomposition tree of π, and whether $\pi \in \mathcal{H}$ or whether π satisfies condition $\mathcal{C}$. The reader familiar with matching problems will be convinced that this can be done in $\mathcal{O}(n)$ time. In any case, a linear algorithm for this tree matching problem is detailed in Subsection 2.6.2 as a subprocedure of the global algorithm of Section 2.6.

Then Theorem 2.101 follows from the complexities of the previous constructions, which are summarized in Table 2.3 in which we denote by ρ the permutation whose decomposition tree is T.

pin-permutation of size n	Complexity	Optimized	Lemma
size 1	$\mathcal{O}(1)$	$\mathcal{O}(1)$	
simple	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	2.79, 2.83
root $\oplus$ non-recursive	$\mathcal{O}(n^4)$	$\mathcal{O}(n^2)$	2.87
root $\oplus$ recursive, one child T is not an increasing oscillation	$\mathcal{O}(\,(n-\lvert\rho\rvert)^2\,)$ + time for $\mathcal{A}_\rho$	$\mathcal{O}(\,(n-\lvert\rho\rvert)^2\,)$ + time for $\mathcal{A}_\rho$	2.94, 2.96
root is prime recursive, $\mathcal{C}$ is satisfied, and T has size 2	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	2.100
root is prime recursive (if not preceding case)	$\mathcal{O}(\,n-\lvert\rho\rvert\,)$ + time for $\mathcal{A}_\rho$	$\mathcal{O}(\,n-\lvert\rho\rvert\,)$ + time for $\mathcal{A}_\rho$	2.100, § 2.5.5

Table 2.3: Complexities of the automata constructions, in all possible cases.

In the optimized version (resp. in the classical version) the complexity is at most of $\mathcal{O}(n^2)$ (resp. $\mathcal{O}(n^4)$) in the non-recursive cases and at most of $\mathcal{O}((n-|\rho|)^2)$ plus the additional complexity of the construction of $\mathcal{A}_\rho$ in the recursive cases. Notice that no extra time is needed to mark the states of the automaton, as they are marked when they are built. Consequently in the optimized version (resp. in the classical version) the automaton $\mathcal{A}_\pi$ can be built in time $\mathcal{O}(n^2)$ (resp. $\mathcal{O}(n^4)$), n being the size of $|\pi|$.

Indeed let K be the number of levels of recursion needed in the construction of $\mathcal{A}_\pi$. Then we can set $\rho_1 = \pi$ and define recursively permutations ρ_i for $2 \leq i \leq K$, ρ_i being the permutation ρ that appears recursively when building $\mathcal{A}_{\rho_{i-1}}$. From Table 2.3, we deduce that, in the optimized version, the time and space complexity for building $\mathcal{A}_\pi$ is:

$$\mathcal{O}\left((|\rho_1| - |\rho_2|)^2\right) + |\mathcal{A}_{\rho_2}| = \ldots = \mathcal{O}\left(\sum_{i=1}^{K-1}(|\rho_i| - |\rho_{i+1}|)^2 + |\rho_K|^2\right).$$

Since every ρ_i is a pattern of π, we have $|\rho_i| - |\rho_{i+1}| \leq n$ and $|\rho_K| \leq n$. Hence, the time and space complexity for building $\mathcal{A}_\pi$ is:

$$\mathcal{O}\left(n \cdot \left(\sum_{i=1}^{K-1}(|\rho_i| - |\rho_{i+1}|) + |\rho_K|\right)\right) = \mathcal{O}\left(n \cdot |\rho_1|\right) = \mathcal{O}(n^2).$$

In the same way we get the complexity $\mathcal{O}(n^4)$ for the classical version. ■

2.6 A polynomial algorithm deciding whether a class contains a finite number of simple permutations

In this section, we put together the steps of the algorithm determining whether a permutation class $\mathcal{C} = Av(B)$ contains a finite number of simple permutations, under the condition that B is finite and given in input. Denoting k the number of pin-permutations in B, $s = \max\{|\pi| : \pi \text{ pin-permutation} \in B\}$, and $n = \sum_{\pi \in B} |\pi|$, the complexity of the algorithm is polynomial w.r.t. n and s and exponential w.r.t. k. The algorithm can be decomposed into several steps.

2.6.1 Finitely many parallel alternations and wedge simple permutations in $\mathcal{C}$?

Following [BRV08] (see Theorem 2.9 p.58) we first check if $\mathcal{C}$ contains finitely many parallel alternations and wedge simple permutations. From Lemmas 2.10, 2.11 and 2.12 (p.59) this problem is equivalent to testing if the permutations in B contain some patterns of size at most 4. Using a result of [AAAH01], this can be done in $\mathcal{O}(n \log n)$ time (see Lemma 2.13 p.59).

2.6.2 Finding pin-permutations in the basis.

The next step is to determine the subset $PB \subseteq B$ of pin-permutations of B. To do so we use the characterization of the class of pin-permutations by their decomposition trees established in [BBR11].

More precisely, for each $\pi \in B$, we proceed as follows.

• First we compute its decomposition tree T_π.

This is achieved in linear time w.r.t. $|\pi|$, computing first the skeleton of T_π following [BCMR08] or [BXHP05], and next the labels of linear and prime nodes as explained in [BCMR11, §2.2].

• Second we add some information on the decomposition tree.

This information will be useful in later steps of our algorithm to check whether π is a pin-permutation, and next (in the affirmative) to determine which construction of the automaton $\mathcal{A}_\pi$ (see Section 2.5) applies to π.

- For each prime node N, we record whether the simple permutation α labeling N is an increasing or decreasing oscillation or quasi-oscillation.

This may be recorded by performing a linear time depth-first traversal of T_π, and checking each node when it is reached. As there are 4 oscillations of each size that are explicitly described as 2 4 1 6 3 8 5 ... (see Figure 2.6 p.67) or one of its symmetries, checking if a simple permutation α is of this form can be done in linear time w.r.t. $|\alpha|$. The same kind of explicit description also holds for quasi-oscillations, and in addition we can record which children correspond to the auxiliary and main substitution points.

- For each node N, we record whether the subtree rooted at N encodes an increasing or decreasing oscillation.

This may be recorded easily, along the same depth-first traversal of T_π as above. Indeed oscillations of size greater than 3 are simple permutations, and increasing (resp. decreasing) oscillations of smaller sizes are 1, 21, 231 and 312 (resp. 1, 12, 132 and 213). So it is sufficient to check whether N is a leaf, or a prime node labeled by an increasing (resp. decreasing) oscillation all of whose children are leaves, or a linear node with exactly two children satisfying extra constraints: they are either both leaves, or one is a leaf and the second one is a linear node with exactly two children that are both leaves. In this later case the oscillation is increasing (resp. decreasing) if N is labeled $\ominus$ (resp. $\oplus$).

These operations are preformed in linear time w.r.t. $|\alpha|$ for any prime node labeled by $|\alpha|$, and in constant time for any linear node. Hence, the overall complexity of this step is linear w.r.t. $|\pi|$. Indeed $|\pi|$ is the number of leaves in the decomposition tree of π, and the overall complexity of this step is linear w.r.t. the sum of the sizes of the labels of all internal nodes. Moreover the sum of the sizes of the labels of all internal nodes is at most the number of leaves plus the number of internal nodes, and since each node is of arity at least two, the number of internal nodes is at most the number of leaves.

• Finally we determine whether π is a pin-permutation or not.

To do so, we recursively check starting with the root whether its decomposition tree is of the shape described in [BBR11] (see Equation ($\star$) p.69).

- If the root is linear, with the additional information stored we can check whether all its children are increasing (resp. decreasing) oscillations in linear time w.r.t. the number of children. If exactly one child is not an increasing (resp. decreasing) oscillation, we check recursively whether the subtree rooted at this child is the decomposition tree of a pin-permutation.

- If the root is prime, we first check whether its label α is a *pin*-permutation. More precisely, with Algorithm 3 (p.75) we compute the set of pin words of α and test its emptiness. By Lemma 2.49, this is done in linear time w.r.t. $|\alpha|$. Then we check whether all the children of the root are leaves.

- If exactly one child is not a leaf, we furthermore have to check whether the point x it expands is an active point of α. From Remark 2.65 (p.85) we just have to test the emptiness of $Q_x(\alpha)$, which is computed in linear time w.r.t. $|\alpha|$ (see Remark 2.67 p.85). Then we check recursively whether the subtree rooted at x is the decomposition tree of a pin-permutation.

- If exactly two children are not leaves, with the additional information stored we can check in constant time whether α is an increasing (resp. decreasing) quasi-oscillation, if the two children that are not leaves expand the auxiliary and main substitution points, and if the one expanding the auxiliary point is the permutation 12 (resp. 21). Then we check recursively whether the subtree rooted at the main substitution point is the decomposition tree of a pin-permutation.

As the complexity of each step is linear w.r.t. the number of children (which is also the size of the label for a prime node), deciding whether a permutation π is a pin-permutation or not can be done in linear time w.r.t. $|\pi|$. The overall determination of PB is therefore linear in $n = \sum_{\pi \in B} |\pi|$.

Moreover, in addition to computing PB, the above procedure produces additional results, that we also record as they are useful in the next step. Namely, for every permutation π of PB, we record its decomposition tree T_π, together with the additional information computed on its nodes; and we also record the set of pin words that encode each simple permutation α labeling a prime node N of T_π and the set $Q_x(\alpha)$ when N has exactly one non-trivial child. Notice that the knowledge of these is sufficient to characterize the set of pin words that encode π thanks to results of Section 2.4.

2.6.3 Finitely many proper pin-permutations in $\mathcal{C}$?

From Theorem 2.27 (p.66) it is enough to check whether $\mathcal{M} \setminus \cup_{\pi\in B}\mathcal{L}_\pi$ is finite. This can be easily decided with a deterministic automaton $\mathcal{A}_\mathcal{C}$ recognizing $\overleftarrow{\mathcal{M} \setminus \cup_{\pi\in B}\mathcal{L}_\pi}$. From the previous step of the procedure (which we assume has been performed), we know the set PB of pin-permutations of B and some additional results described above. First notice that $\cup_{\pi\in B}\mathcal{L}_\pi = \cup_{\pi\in PB}\mathcal{L}_\pi$ as $\mathcal{L}_\pi$ is empty when π is not a pin-permutation (see p.64). We build the automaton $\mathcal{A}_\mathcal{C}$ as follows.

• First for each pin-permutation $\pi \in PB$, we construct $\mathcal{A}_\pi$ – which is deterministic and complete – recognizing a language $\mathcal{L}'_\pi$ such that $\mathcal{L'}_\pi \cap \mathcal{M} = \overleftarrow{\mathcal{L}_\pi} \cap \mathcal{M}$. This construction is performed in time and space at most $\mathcal{O}(|\pi|^2)$ as described in Section 2.5, with the optimized construction (see Theorem 2.101 p.106). Notice that the construction of $\mathcal{A}_\pi$ depends on the shape of the decomposition tree T_π of π. But thanks to the additional information stored in T_π, we can determine which tree shape matches T_π in linear time w.r.t. the number of children of the root of T_π, and the same holds at each recursive step of the construction.

• Then we build a deterministic automaton $\mathcal{A}_1$ recognizing $\bigcup_{\pi \in PB} \mathcal{L}'_\pi$, where $\mathcal{L}'_\pi$ is defined as in the first item. The automaton $\mathcal{A}_1$ is obtained performing the deterministic union (as a Cartesian product on the fly, see [HU79] for details) of all the automata $\mathcal{A}_\pi$. This is done in time and space $\mathcal{O}(\prod_{\pi \in PB} |\mathcal{A}_\pi|) = \mathcal{O}(\prod_{\pi \in PB} |\pi|^2)$.

• Then we build the automaton $\mathcal{A}_2$ which is the deterministic intersection (again as a Cartesian product) between $\mathcal{A}_1$ and the automaton $\mathcal{A}(\mathcal{M})$ given in Figure 2.27 in time and space $\mathcal{O}(|\mathcal{A}_1|.|\mathcal{A}(\mathcal{M}))|) = \mathcal{O}(\prod_{\pi \in PB} |\pi|^2)$.

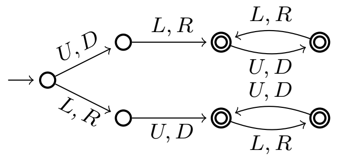


Figure 2.27: A deterministic automaton $\mathcal{A}(\mathcal{M})$ recognizing the set $\mathcal{M}$ of words of length at least 2 without any factor in $\{UU, UD, DU, DD, RR, RL, LR, LL\}$.

The automaton $\mathcal{A}_2$ recognizes $\left(\bigcup_{\pi \in PB} \mathcal{L}'_\pi\right) \cap \mathcal{M} = \left(\bigcup_{\pi \in PB} \overleftarrow{\mathcal{L}_\pi}\right) \cap \mathcal{M}$. By Lemma 2.26 (p.66) this is the language of words $\overleftarrow{\phi(w)}$ for all strict pin words w encoding permutations having a pattern in PB, i.e. that are not in $\mathcal{C}$. Notice that by Remark 2.8 (p.58) such permutations are necessarily proper pin-permutations.

• Next we complement $\mathcal{A}_2$ to build a deterministic automaton $\mathcal{A}_3$ recognizing $A^\star \setminus \left(\left(\bigcup_{\pi \in PB} \overleftarrow{\mathcal{L}_\pi}\right) \cap \mathcal{M}\right)$. As $\mathcal{A}_2$ is deterministic, its complement is obtained in linear time w.r.t. its size, by completing it and then turning every final (resp. non-final) state into a non-final (resp. final) state. Moreover the size of $\mathcal{A}_3$ is the one of $\mathcal{A}_2$, i.e., $\mathcal{O}(\prod_{\pi \in PB} |\pi|^2)$.

• Finally we compute the deterministic intersection between $\mathcal{A}_3$ and the automaton $\mathcal{A}(\mathcal{M})$ to obtain the automaton $\mathcal{A}_\mathcal{C}$. This is done in time and space $\mathcal{O}(|\mathcal{A}_3|.|\mathcal{A}(\mathcal{M}))|) = \mathcal{O}(\prod_{\pi \in PB} |\pi|^2)$. The automaton $\mathcal{A}_\mathcal{C}$ recognizes $\mathcal{M} \setminus \left(\bigcup_{\pi \in PB} \overleftarrow{\mathcal{L}_\pi}\right)$. This is the language of all words $\overleftarrow{\phi(w)}$ where w is a strict pin word encoding a permutation of $\mathcal{C}$ (that is necessarily a proper pin-permutation, as above).

Then, by Theorem 2.27 (p.66), checking whether the permutation class $\mathcal{C}$ contains a finite number of proper pin-permutations is equivalent to checking whether the language recognized by $\mathcal{A}_\mathcal{C}$ is finite i.e., whether $\mathcal{A}_\mathcal{C}$ does not contain any cycle that is accessible and co-accessible (i.e., a cycle that can be reached from an initial state and from which a final state can be reached). The automaton $\mathcal{A}_\mathcal{C}$ is not necessarily accessible and co-accessible. Its accessible part is made of all states that can be reached in a traversal of the automaton from the initial state; its co-accessible part is obtained similarly by a traversal from the set of final states taking the edges of the automaton backwards. Before looking for a cycle, we make $\mathcal{A}_\mathcal{C}$ accessible and co-accessible by keeping only its accessible and co-accessible part, yielding a smaller automaton $\mathcal{A}'_\mathcal{C}$. The complexity of this double reduction of the size of the automaton is linear in time w.r.t. the size of $\mathcal{A}_\mathcal{C}$. Moreover the size of $\mathcal{A}'_\mathcal{C}$ is smaller than or equal to the one of $\mathcal{A}_\mathcal{C}$, i.e., $\mathcal{O}(\prod_{\pi \in PB} |\pi|^2)$. Finally we test whether $\mathcal{A}'_\mathcal{C}$ does not contain any cycle. This can be done in $\mathcal{O}(|\mathcal{A}'_\mathcal{C}|)$ time with a depth-first traversal of $\mathcal{A}'_\mathcal{C}$.

Let s be the maximal size of a pin-permutation of B and k the number of pin-permutations in B, then $\mathcal{O}(\prod_{\pi \in PB} |\pi|^2) = \mathcal{O}(s^{2k})$. Hence putting all these steps together leads to an algorithm whose complexity is $\mathcal{O}(s^{2k})$ to check whether there are finitely many proper pin-permutations in $\mathcal{C}$, when the set PB of pin-permutations of B, their decomposition trees and the set of pin words of each simple permutation appearing in these trees are given.

2.6.4 Main result

We are now able to state the main theorem of this chapter:

Theorem 2.102. *Given a finite set of permutations B, we have described an algorithm that determines whether the permutation class $\mathcal{C} = Av(B)$ contains a finite number of simple permutations. Denoting $n = \sum_{\pi \in B} |\pi|$, $p = \prod |\pi|$ where the product is taken over all* pin*-permutations in B, k the number of* pin*-permutations in B and s the maximal size of a* pin*-permutation of B, the complexity of the algorithm is $\mathcal{O}(n \log n + s^{2k})$ or more precisely $\mathcal{O}(n \log n + p^2)$.*

Proof. We determine whether there are finitely many parallel alternations and wedge simple permutations in $\mathcal{C}$ in $\mathcal{O}(n \log n)$ time. Then we compute the set PB of pin-permutations of B and their decomposition trees with some additional information (like the set of pin words that encode each simple permutation labeling a node) in $\mathcal{O}(n)$ time. Finally we check whether there are finitely many proper pin-permutations in $\mathcal{C}$ as explained above (i.e., by looking for a cycle in the automaton $\mathcal{A}_{\mathcal{C}}$) in $\mathcal{O}(p^2)$ time which is $\mathcal{O}(s^{2k})$. From Theorem 2.9 (p.58) this concludes the proof. ∎

Moreover for the special case of a substitution-closed class of permutation, we have a much better complexity:

Theorem 2.103. *Given a finite set of simple permutations B, we have an algorithm that determines whether the permutation class $\mathcal{C} = Av(B)$ contains a finite number of simple permutations in time $\mathcal{O}(n \log n)$ where $n = \sum_{\pi \in B} |\pi|$.*

Proof. As in the general case, we determine whether there are finitely many parallel alternations and wedge simple permutations in $\mathcal{C}$ in $\mathcal{O}(n \log n)$ time. However in this special case, we can check whether there are finitely many proper pin-permutations in $\mathcal{C}$ in linear time w.r.t. n (see Lemma 2.105 below). ∎

Definition 2.104. For any set of permutations B, we set

$$E_B = \cup_{\pi \in B} E_\pi \cup \{UU, UD, DU, DD, RR, RL, LR, LL\}.$$

Lemma 2.105. *Algorithm 4 tests whether a substitution-closed permutation class given by its finite basis B contains a finite number of proper pin-permutations in linear time with respect to $n = \sum_{\pi \in B} |\pi|$.*

Algorithm 4: Deciding the finiteness of the number of proper pin-permutations

```
input  : a set $B$ of simple permutations
output: boolean : true if and only if $Av(B)$ contains only a finite number of proper
        pin-permutations

$\mathcal{P}_B$ ←PinWords($B$) // Determine the set of pin words associated to the
    elements of $B$ using Algorithm 3 (p.75)
$\mathcal{A} \leftarrow \mathcal{AC}^{\circlearrowleft}(E_B)$ // Build a complete deterministic automaton recognizing
    words having a factor in $E_B$
if $\mathcal{A}^c$ contains an accessible and co-accessible cycle then
    return false
else
    return true
```

Proof. From Theorem 2.27, we only have to prove that the language recognized by $\mathcal{A}^c$ is $\mathcal{M} \setminus \cup_{\pi\in B}\mathcal{L}_\pi$.
By definition, $\mathcal{A}^c$ recognizes $A^\star \setminus \left(A^\star \cdot E_B \cdot A^\star\right)$ with $E_B = \cup_{\pi\in B}E_\pi \cup \{UU, UD, \dots, LL\}$. From Lemma 2.81, for any simple permutation π, we have $\mathcal{L}_\pi \cap \mathcal{M} = \left(A^\star \cdot E_\pi \cdot A^\star\right) \cap \mathcal{M}$. Since $\mathcal{M} = A^\star \setminus A^\star\{UU, UD, \dots, LL\}A^\star$, we have

$$\begin{aligned}
\mathcal{M} \setminus \cup_{\pi\in B}\mathcal{L}_\pi &= \mathcal{M} \setminus \big(\cup_{\pi\in B}\, \mathcal{L}_\pi \cap \mathcal{M}\big)\\
&= \mathcal{M} \setminus \big(\cup_{\pi\in B}\, (A^\star \cdot E_\pi \cdot A^\star) \cap \mathcal{M}\big)\\
&= \mathcal{M} \setminus \big(\cup_{\pi\in B}\, (A^\star \cdot E_\pi \cdot A^\star)\big)\\
&= \mathcal{M}\ \cap\ A^\star \setminus \big(A^\star \cdot (\cup_{\pi\in B}E_\pi) \cdot A^\star\big)\\
&= \big(A^\star \setminus A^\star\{UU, UD, \dots, LL\}A^\star\big) \cap \big(A^\star \setminus (A^\star \cdot \cup_{\pi\in B}E_\pi \cdot A^\star)\big)\\
&= A^\star \setminus \big(A^\star \cdot E_B \cdot A^\star\big)
\end{aligned}$$

concluding the proof. ■

We have described in the above an efficient algorithm that determines whether a permutation class $\mathcal{C}$ contains a finite number of simple permutations.

As explained at the beginning of Part I, we may interpret our result as giving an algorithm testing a sufficient condition for a permutation class to be well-structured. In the next and last chapter of this part, we describe an algorithm that not only *tests* that there is an underlying structure in a permutation class, but also *computes* this structure.

Chapter 3

Combinatorial specification of permutation classes

This chapter presents a methodology that automatically derives a combinatorial specification for the permutation class $\mathcal{C} = Av(B)$, given its basis B of excluded patterns and the set of simple permutations in $\mathcal{C}$, when these sets are both finite. This is achieved considering both pattern avoidance and pattern containment constraints in permutations. The obtained specification yields a system of equations satisfied by the generating function of $\mathcal{C}$, this system being always positive and algebraic. It also yields a uniform random sampler of permutations in $\mathcal{C}$. The method presented is fully algorithmic.

3.1 Introduction

Despite the profusion of enumerative results on permutation classes, it is quite surprising to notice the lack of studies on random generation of permutations in a class $\mathcal{C}$. Let us mention though that some results exist on the *exhaustive* generation of permutations in a class [Bar09, DV, DFMV08]. A simple reason one may think of to explain this absence of results is that there exists a naive algorithm to generate uniformly at random a permutation in a class $\mathcal{C} = Av(B)$: generate a random permutation σ; test whether it avoids every pattern in B; in this case return σ otherwise reject σ and try again. This procedure is however very inefficient, and there are at least two reasons for it. First, there are $n!$ permutations of size n and only $\mathcal{O}(c^n)$ in class $\mathcal{C}$ [MT04], for some constant c, so the probability of rejection is close to 1 as n grows. Second, testing whether a permutation σ contains a pattern π of B is an NP-hard problem for general π [BBL98] (and except for some very special patterns π, the known algorithms are of exponential complexity w.r.t. $|\pi|$).

In this chapter we focus on combinatorial specifications. By *combinatorial specification* of a permutation class $\mathcal{C}$ (see [FS09]), we mean an unambiguous system of combinatorial equations that describe recursively the permutations of $\mathcal{C}$ using only combinatorial constructors (disjoint union, cartesian product, sequence, ...) and permutations of size 1. Obtaining algorithmically combinatorial specifications of permutation classes is of interest *per se* since they describe the structure of the class, but also because it then allows us to obtain by routine algorithms a system of equations satisfied by the generating function of $\mathcal{C}$ and a Boltzmann uniform random sampler of permutations in $\mathcal{C}$, using the methods of [FS09] and [DFLS04] respectively.

Our goal in this chapter is to provide a general algorithmic method to obtain a combinatorial specification for any permutation class $\mathcal{C}$ from its basis B and the set $\mathcal{S}_{\mathcal{C}}$ of simple permutations in $\mathcal{C}$, and assuming these two sets are finite. Notice that by previous chapters, it is enough to know the finite basis B of the class to decide whether the set $\mathcal{S}_{\mathcal{C}}$ is finite and (in the affirmative) to compute $\mathcal{S}_{\mathcal{C}}$.

In the following, we only consider classes whose basis B is given explicitly, and is finite. This does not cover the whole range of permutation classes, but it is a reasonable assumption when dealing with *algorithms* on permutation classes, that take a finite description of a permutation class in input. There are of course other ways of finitely describing some permutation classes, even one of infinite basis (by a recognition procedure for example). The assumption of the description by a finite basis has been preferred for two reasons: first, it encompasses most of the permutation classes that have been studied in the literature [KM03]; and second, it is a necessary condition for a permutation class to contain a finite number of simple permutations [AA05], which is an important condition in our work.

Moreover to avoid trivial cases, we assume in this chapter that the permutation class studied $\mathcal{C}$ contains 12 and 21.

The chapter is organized as follows. Section 3.2 proceeds with some background on combinatorial structures and random generation. Section 3.3 then explains how to obtain a system of combinatorial equations describing $\mathcal{C}$ from the set of simple permutations in $\mathcal{C}$, that we assume to be finite. The system so obtained may be ambiguous and Section 3.4 describes a disambiguation algorithm to obtain a combinatorial specification for $\mathcal{C}$. The most important idea of this disambiguation procedure is to transform ambiguous unions into disjoint unions of terms that involve both pattern avoidance and pattern containment constraints. This somehow allows us to interpret on the combinatorial objects themselves the result of applying the inclusion-exclusion on their generating functions. Section 3.5

provides a complete example of the whole process applied to a specific class, and examples of random permutations in this class. Finally, Section 3.6 concludes the whole algorithmic process by explaining how this specification can be plugged into the general methodologies of [FS09] and [DFLS04] to obtain a system of equations satisfied by the generating function of $\mathcal{C}$ and a Boltzmann uniform random sampler of permutations in $\mathcal{C}$. We also give a number of perspectives opened by our algorithm.

3.2 Combinatorial Structures and Random Generation

Let us leave aside permutations for now, and explain general ideas on enumeration and random generation of combinatorial objects, through their description as combinatorial structures, their generating functions, and with Boltzmann random samplers. We will see in Theorem 3.29 of Section 3.4 that the classes of permutations we are interested in fit in the general framework *constructible* structures, that is structures which admit a *combinatorial specification.*

3.2.1 Constructible combinatorial structures and generating functions

A class $\mathcal{C}$ of combinatorial structures is a set of discrete objects (also called *structures*) equipped with a notion of *size*: the size is a function of $\mathcal{C} \to \mathbb{N}$ denoted $|\cdot|$ such that for any n the number of objects of size n in $\mathcal{C}$ is finite. The subset of objects of size n in $\mathcal{C}$ is then denoted $\mathcal{C}_n$ and c_n denotes the cardinality of $\mathcal{C}_n$.

Among the combinatorial structures, we focus on *constructible* ones, from the framework introduced in [FS09]. Basically, a constructible combinatorial class is a set of structures that can be defined from atomic structures of size 1 (denoted by $\mathcal{Z}$) and assembled by means of admissible constructors. While a complete list of these combinatorial constructors is given in [FS09], we only use a subset of them:

- the disjoint union, denoted by $+$ (we may also use the notation $\sum$), to choose between structures,
- the cartesian product, denoted by $\times$, to form pairs of structures,
- the sequence, denoted by $\textsc{Seq}$; for any combinatorial class $\mathcal{A}$ that does not contain structures of size 0, a sequence of structures of $\mathcal{A}$ is defined by $\textsc{Seq}(\mathcal{A}) = \sum_{i=0}^{\infty} \underbrace{\mathcal{A} \times \cdots \times \mathcal{A}}_{i} = \mathcal{E} + \mathcal{A} + (\mathcal{A} \times \mathcal{A}) + (\mathcal{A} \times \mathcal{A} \times \mathcal{A}) + \cdots$, where $\mathcal{E}$ is an atom of size 0.

A cardinality constraint can be applied to the sequence constructor; the notation $\textsc{Seq}_{=k}$ stands for k-tuples and can be extended to sequences having at least k elements, which are defined by $\textsc{Seq}_{\geq k} = \sum_{i=k}^{\infty} \textsc{Seq}_{=i}$.

A *specification* for a combinatorial class is an equation or a set of equations of the form

$$\begin{cases} \mathcal{C}_1 &= \mathcal{H}_1(\mathcal{E}, \mathcal{Z}, \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m), \\ \mathcal{C}_2 &= \mathcal{H}_2(\mathcal{E}, \mathcal{Z}, \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m), \\ & \vdots \\ \mathcal{C}_m &= \mathcal{H}_m(\mathcal{E}, \mathcal{Z}, \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m), \end{cases}$$

where each $\mathcal{H}_i$ denotes a term built from $\mathcal{C}_1, \ldots, \mathcal{C}_m$ (and possibly $\mathcal{Z}$ and $\mathcal{E}$) using admissible constructors and atoms. For example, a sequence of atoms can be recursively defined by the equation $\textsc{Seq}(\mathcal{Z}) = \mathcal{E} + \mathcal{Z} \times \textsc{Seq}(\mathcal{Z})$. A class of combinatorial structures is said to be *constructible* if and only if it admits such a specification.

Constructor	Notation	$C(z)$ (o.g.f.)	Boltzmann sampler $\Gamma C(x)$
Atom	$\mathcal{Z}$	z	return an atom
Disjoint Union	$\mathcal{A}+\mathcal{B}$	$A(z)+B(z)$	call $\Gamma A(x)$ with proba. $A(x)/C(x)$, else $\Gamma B(x)$
Cartesian Product	$\mathcal{A}\times\mathcal{B}$	$A(z)B(z)$	call $\Gamma A(x)$ and $\Gamma B(x)$
Sequence	$\text{SEQ}(A)$	$\dfrac{1}{1-A(z)}$	choose k according to a geometric law of parameter $A(x)$ and call $\Gamma A(x)$ k times
Restricted Seq.	$\text{SEQ}_{=k}(A)$	$A(z)^k$	call the sampler $\Gamma A(x)$ k times

Table 3.1: Admissible constructors, the corresponding operators on ordinary generating functions and the associated Boltzmann samplers. Atoms are usually not considered as constructors, but they appear in this table for convenience.

In this framework, the *size* of a combinatorial structure is its number of atoms ($\mathcal{Z}$) and from there, combinatorial structures can be counted according to their size. The size information for a whole combinatorial class, say $\mathcal{C}$, is encoded by its *ordinary generating function*, which is a formal power series $C(z) = \sum_{n\geq 0} c_n z^n$ where the coefficient c_n, also denoted by $[z^n]C(z)$, is the number of structures of size n in $\mathcal{C}$.

There exist two sorts of generating functions: ordinary and exponential ($\sum \frac{c_n}{n!} z^n$). Ordinary generating functions are used for unlabelled enumeration while exponential ones deal with labelled structures. In the present work we only need to consider unlabelled structures, thus, in the sequel, generating functions are always ordinary ones.

To any admissible constructor, one can associate an operator on generating functions. The complete dictionary of these operators is given in [FS09], together with the proof that this translation from constructors of combinatorial classes to operators on their generating functions is correct. The ones we need are summarized in Table 3.1. Any specification for a combinatorial class can then turn automatically into a system of (possibly implicit) equations defining the generating functions of this class. A lot of information can be extracted from such functional systems; in particular, one can compute as many coefficients of the series as required and [FS09] provides many tools to get asymptotic equivalents for these same coefficients.

Example 3.1. Alternating unary-binary trees are planar rooted trees with internal nodes having either one or two subtrees and such that the unary and binary nodes alternate along each branch of the tree. They are defined by the following specification, which leads to the corresponding system of generating functions:

$$\begin{cases}\mathcal{T} &= \mathcal{U}+\mathcal{B}\\ \mathcal{U} &= \mathcal{Z}+\mathcal{Z}\times\mathcal{B}\\ \mathcal{B} &= \mathcal{Z}+\mathcal{Z}\times\text{SEQ}_{=2}(\mathcal{U}) = \mathcal{Z}+\mathcal{Z}\times\mathcal{U}^2\end{cases} \qquad \begin{cases}T(z) &= U(z)+B(z)\\ U(z) &= z+zB(z)\\ B(z) &= z+zU^2(z).\end{cases}$$

The first combinatorial structures of the class $\mathcal{T}$ are listed in Figure 3.1, up to size 7. White dots correspond to unary nodes while black ones stand for binary nodes.

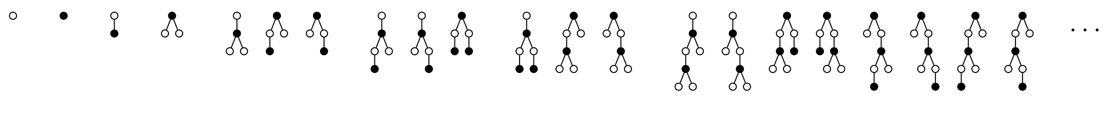


Figure 3.1: Alternating unary-binary trees up to size 7.

The corresponding ordinary generating functions are

$$\begin{aligned} T(z) &= 2\,z + z^2 + z^3 + 3\,z^4 + 3\,z^5 + 3\,z^6 + 8\,z^7 + 12\,z^8 + 13\,z^9 + O(z^{10}) \\ B(z) &= z + z^3 + 2\,z^4 + z^5 + 2\,z^6 + 6\,z^7 + 6\,z^8 + 7\,z^9 + O(z^{10}) \\ U(z) &= z + z^2 + z^4 + 2\,z^5 + z^6 + 2\,z^7 + 6\,z^8 + 6\,z^9 + O(z^{10}). \end{aligned}$$

In this particular case, there exists a closed form for the generating function:

$T(z) = \frac{(z+1)(1-\sqrt{1-4z^3-4z^4})}{2z^3} - 1.$

3.2.2 Automatic methods for uniform random generation

Since we are interested in automatic treatments for a wide family of combinatorial classes, we only discuss automatic methods for random generation in this section, leaving aside *ad hoc* methods. Dealing with constructible structures, two main methods allow the design of uniform random samplers: the recursive method [FZVC94] and Boltzmann method [DFLS04]. Both methods heavily rely on the recursive description given by combinatorial specifications to produce combinatorial structures uniformly at random. Indeed, a specification is in itself a procedure that allows us to derive combinatorial objects; randomizing the choices made during this derivation makes it a random sampler. The main difference between the two models lies in the way generating functions are used to reach uniformity. In the recursive method, a random sampler outputs structures of a given size, say n, and coefficients of the generating functions are used in the probabilistic choices to ensure that all structures of size n are assigned the same probability. In the Boltzmann model, the output size is not chosen *a priori* and samplers can draw structures in a whole combinatorial class; but still, uniformity for any size is guaranteed, i.e. for any fixed size n, the probability of outputting some object of size n is uniform among all objects of size n. Enumeration sequences $(c_n)_n$ are not involved in this process but values of generating functions at a given point (the Boltzmann parameter) are needed to ensure uniformity.

The Boltzmann method is particularly well suited to compute very large structures, as soon as a small tolerance (a few percents) is allowed on the size of the structures to be sampled. This occurs in particular when random sampler are used to conjecture asymptotic properties, which is why we choose to focus on Boltzmann samplers here. However, the recursive method can be slightly more efficient when dealing with large samples of small structures of exact size that can be used for statistical testing.

In the following, we recall some basic facts about the Boltzmann method; a complete overview is given in [DFLS04].

Boltzmann model This is a probabilistic model that, given a combinatorial class, associates to any of its structures a probability that is proportional to an exponential of its size.

Definition 3.2. The (ordinary) Boltzmann model of parameter x assigns to any object γ in the combinatorial class $\mathcal{C}$ the following probability:

$$\mathbb{P}_x(\gamma) = \frac{x^{|\gamma|}}{C(x)} \qquad \text{with} \quad C(x) = \sum_{\delta \in \mathcal{C}} x^{|\delta|}$$

A Boltzmann sampler $\Gamma\mathcal{C}(x)$ for $\mathcal{C}$ is a process that produces random objects according to the Boltzmann model.

In this model, two structures of $\mathcal{C}$ having the same size are equally likely to be drawn by $\Gamma\mathcal{C}(x)$, since their probability only depends on their size, for a fixed parameter x. Boltzmann samplers that operate freely under the sole effect of this parameter are called *free* samplers, to indicate that there is no restriction on the size of the output. However, the value of x influences the probability distribution of the size: the probability of drawing a structure of size n and the expected size of the structures drawn by $\Gamma\mathcal{C}(x)$ are respectively

$$\mathbb{P}_x(\text{size} = n) = \sum_{\gamma \in \mathcal{C}_n} \frac{x^{|\gamma|}}{C(x)} = \frac{c_n x^n}{C(x)} \qquad \text{and} \qquad \mathbb{E}_x(\text{size}) = \sum_{\gamma \in \mathcal{C}} \frac{|\gamma| \cdot x^{|\gamma|}}{C(x)} = x\frac{C'(x)}{C(x)}.$$

Design of Boltzmann samplers In [DFLS04], the authors describe a sampling algorithm for each admissible constructor. Hence, starting from a combinatorial specification, one can automatically derive a sampler that draws combinatorial structures according to the Boltzmann model. We only recall in this section the Boltzmann samplers for unions, products and sequences. This is summarized in the last column of Table 3.1. Assume that $\Gamma\mathcal{A}(x)$ and $\Gamma\mathcal{B}(x)$ are Boltzmann samplers for $\mathcal{A}$ and $\mathcal{B}$.

- The Boltzmann sampler $\Gamma\mathcal{C}(x)$ for the disjoint union $\mathcal{C} = \mathcal{A} + \mathcal{B}$ is an algorithm that chooses to return a structure produced by $\Gamma\mathcal{A}(x)$ with probability $A(x)/C(x)$ or a structure produced by $\Gamma\mathcal{B}(x)$ with probability $B(x)/C(x)$.
- The Boltzmann sampler for the cartesian product $\mathcal{A} \times \mathcal{B}$ returns a pair of structures in $\mathcal{A} \times \mathcal{B}$, performing independent calls to $\Gamma\mathcal{A}(x)$ and $\Gamma\mathcal{B}(x)$.
  Similarly, the Boltzmann sampler for $\textsc{Seq}_{=k}\,\mathcal{A}$ returns a k-tuple of structures of $\mathcal{A}$, performing k independent calls to $\Gamma\mathcal{A}(x)$, and for restricted sequences of the type $\textsc{Seq}_{\leq k}\,\mathcal{A}$, their Boltzmann samplers are obtained as finite unions of products.
- The Boltzmann sampler for the class $\mathcal{C} = \textsc{Seq}(\mathcal{A})$ chooses the cardinality k of the sequence according to a geometric law of parameter $p = A(x)$ (i.e. $\mathbb{P}(k = K) = p^K(1-p)$) and returns a sequence of k structures produced by k independent calls to $\Gamma\mathcal{A}(x)$. One can also get a Boltzmann sampler for $\mathcal{C}$ using its recursive specification $\mathcal{C} = \mathcal{E} + \mathcal{A} \times \mathcal{C}$ and combining the above algorithms together.
  For restricted sequences of the type $\textsc{Seq}_{\geq k}\,\mathcal{A}$, their Boltzmann samplers are obtained by writing $\textsc{Seq}_{\geq k}\,\mathcal{A} = \textsc{Seq}_{=k}\,\mathcal{A} \times \textsc{Seq}\,\mathcal{A}$ and combining the two corresponding Boltzmann samplers.

In order to implement effective Boltzmann samplers, two technical points still have to be considered. How to compute values of generating functions at a given point (the Boltzmann parameter) and how to control the size of the sampled structures? The first question is answered in [PSS08]. Any combinatorial specification only involving unions, products and sequences can be automatically translated into a numerical Newton iteration to compute the desired values, provided that the given parameter is chosen smaller than the radius of convergence of the generating function. This brings us to the second point, since the Boltzmann parameter influences the expected size of the structures to be drawn. In order to sample structures (in $\mathcal{C}$) of a target size n, the idea is to choose a parameter x such that the expected size is n, i.e. such that $xC'(x)/C(x) = n$ and then to reject any structure whose size does not fit in an interval $[n(1-\varepsilon), (1+\varepsilon)]$. This rejection process gives a uniform approximate-size sampler and choosing $\varepsilon = 0$ leads to an exact size sampler.

Proposition 3.3. *The time complexity of a Boltzmann sampler is always linear in the size of the structure produced. In the case of tree-like structures, the complexity of the approximate-size sampler remains in $O(n)$, where n is the target size and the exact size sampler has a quadratic complexity*[1].

[1]For tree-like structures, the Boltzmann distribution is concentrated on small structures. Thus, in order to get these complexities in practice, one may use special techniques such as pointing or singular samplers (see §6.3 and §7 of [DFLS04]).

Example 3.4. Coming back to our example of alternating unary-binary trees, we can compute the following Boltzmann samplers for $\mathcal{U}$ and $\mathcal{B}$ and combine them get a sampler for $\mathcal{T}$.

Algorithm 5: $\Gamma\mathcal{U}(x)$

```
Data: $x$, $U(x)$
Result: a random structure in $\mathcal{U}$
begin
    Choose $r$ uniformly at random on $[0,1]$
    If $x/U(x) < r$ then return $\circ$
    Else return $\overset{\circ}{\underset{\Gamma\mathcal{B}(x)}{|}}$
```

Algorithm 6: $\Gamma\mathcal{B}(x)$

```
Data: $x$, $B(x)$
Result: a random structure in $\mathcal{B}$
begin
    Choose $r$ uniformly at random on $[0,1]$
    If $x/B(x) < r$ then return $\bullet$
    Else return $\overset{\bullet}{\underset{\Gamma\mathcal{U}(x)\ \Gamma\mathcal{U}(x)}{/\backslash}}$
```

The values $U(x)$ and $B(x)$ are easily computed from the closed forms of the generating functions. As for the choice of the parameter, the equation for the expected size gives us $x_{1000} = 0.5449326566$, for instance.

The description of permutations in the framework of constructible structures that will be used in this chapter relies on substitution decomposition of permutations.

3.3 Combinatorial system describing $\mathcal{C}$

In this section, we follow the path used in [AA05] in the proof that a permutation class with finitely many simple permutations has an algebraic generating function.

Consider a class $\mathcal{C}$, which is assumed to have finite basis B and to contain a finite number of simple permutations, whose set is denoted $\mathcal{S}_\mathcal{C}$. In [AA05] the substitution decomposition of permutations (under the form of Theorem 0.29) is used to deduce from $\mathcal{S}_\mathcal{C}$ a specification that describes the permutations of $\mathcal{C}$ as constructible objects in the sense of Section 3.2. As explained earlier, this specification translates into a system of equations for the generating function $C(z)$ of $\mathcal{C}$. In this case, every equation is polynomial in C. This not only ensures that $C(z)$ is algebraic, but also, as is underlined in [AA05], it provides a method to *compute* C (or at least to render the system of equations satisfied by C explicit). Our purpose is to go through the details of this method, to complete its analysis, and to adapt it into an actual algorithm. Indeed unlike [AA05], we make the whole process fully algorithmic.

Let us now turn to explaining how to provide a recursive description of the permutations in $\mathcal{C} = Av(B)$ in terms of the set $\mathcal{S}_\mathcal{C}$ of simple permutations in $\mathcal{C}$, when they are finite in number. We begin with the case where $\mathcal{C}$ is substitution-closed.

3.3.1 The simple case of substitution-closed classes

Recall that we denote by $\hat{\mathcal{C}}$ the substitution closure of the permutation class $\mathcal{C}$, and that $\mathcal{C}$ is a substitution-closed class when $\mathcal{C} = \hat{\mathcal{C}}$, or equivalently when the permutations in $\mathcal{C}$ are exactly the ones whose decomposition trees have internal nodes labeled by $12, 21$ or any simple permutation of $\mathcal{C}$.[2]

For the purpose of this chapter, we additionally introduce the following notations:

[2]Recall that we assume in this chapter that $\mathcal{C}$ contains permutations 12 and 21.

Definition 3.5. For any set $\mathcal{A}$ of permutations, $\mathcal{A}^+$ (resp. $\mathcal{A}^-$) denotes the set of permutations of $\mathcal{A}$ that are $\oplus$-indecomposable (resp. $\ominus$-indecomposable) and $\mathcal{S}_\mathcal{A}$ denotes the set of simple permutations of $\mathcal{A}$.

Then Theorem 0.29 directly yields the following proposition:

Proposition 3.6 (Lemma 11 of [AA05]). *Let $\hat{\mathcal{C}}$ be a substitution-closed class (that contains* 12 *and* 21*).*[3] *Then $\hat{\mathcal{C}}$ satisfies the following system of equations, denoted $\mathcal{E}_{\hat{\mathcal{C}}}$:*

$$\hat{\mathcal{C}} = 1 \uplus 12[\hat{\mathcal{C}}^+,\hat{\mathcal{C}}] \uplus 21[\hat{\mathcal{C}}^-,\hat{\mathcal{C}}] \uplus \biguplus_{\pi\in\mathcal{S}_{\hat{\mathcal{C}}}} \pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}] \tag{3.1}$$
$$\hat{\mathcal{C}}^+ = 1 \uplus 21[\hat{\mathcal{C}}^-,\hat{\mathcal{C}}] \uplus \biguplus_{\pi\in\mathcal{S}_{\hat{\mathcal{C}}}} \pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}] \tag{3.2}$$
$$\hat{\mathcal{C}}^- = 1 \uplus 12[\hat{\mathcal{C}}^+,\hat{\mathcal{C}}] \uplus \biguplus_{\pi\in\mathcal{S}_{\hat{\mathcal{C}}}} \pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}]. \tag{3.3}$$

By uniqueness of substitution decomposition, unions are disjoint and so Equations (3.1) to (3.3) describe unambiguously the substitution-closed class $\hat{\mathcal{C}}$.

Hence, Proposition 3.10 can be transposed in the framework of constructible structures as follows:

Theorem 3.7. *Let $\mathcal{C}$ be a substitution-closed class. Then $\mathcal{C}$ can be described as a constructible combinatorial class in the sense of Section 3.2 with the specification:*

$$\begin{cases} \mathcal{C} = \mathcal{Z} + \mathcal{C}^+ \times \mathcal{C} + \mathcal{C}^- \times \mathcal{C} + \sum_{\pi\in\mathcal{S}_\mathcal{C}} \textsc{Seq}_{=|\pi|}(\mathcal{C}) \\ \mathcal{C}^+ = \mathcal{Z} + \mathcal{C}^- \times \mathcal{C} + \sum_{\pi\in\mathcal{S}_\mathcal{C}} \textsc{Seq}_{=|\pi|}(\mathcal{C}) \\ \mathcal{C}^- = \mathcal{Z} + \mathcal{C}^+ \times \mathcal{C} + \sum_{\pi\in\mathcal{S}_\mathcal{C}} \textsc{Seq}_{=|\pi|}(\mathcal{C}) \end{cases}$$

Moreover this system can be translated into an equation for the generating function C:

Proposition 3.8 (Theorem 12 of [AA05]). *Let $\mathcal{C}$ be a substitution-closed class, with generating function C. Then*

$$C^2 + (P(C) - 1 + z)C + P(C) + z = 0$$

with P denoting the generating function that enumerate simple permutations in $\mathcal{C}$, i.e. $P(z) = \sum_{\pi\in\mathcal{S}_\mathcal{C}} z^{|\pi|}$.

Hence, in the case of a substitution-closed class $\mathcal{C}$ (and for the substitution closure of any class), the system $\mathcal{E}_\mathcal{C}$ that recursively describes the permutations in $\mathcal{C}$ can be immediately deduced from the set $\mathcal{S}_\mathcal{C}$ of simple permutations in $\mathcal{C}$.

As soon as $\mathcal{S}_\mathcal{C}$ is finite, this system is explicit and gives a combinatorial specification. Hence, it provides an efficient way to compute the generating function of the class, and to generate uniformly at random a permutation of a given size in the class.

3.3.2 Adding constraints for classes that are not substitution-closed

The goal here is to describe an algorithm that computes a combinatorial system of equations for a general permutation class $\mathcal{C}$ from the simple permutations in $\mathcal{C}$, like for the case of substitution-closed classes. However, when the class is not substitution-closed, this is not as straightforward as in the previous case, and we provide details below on how to solve this general case.

We follow the guideline of [AA05], where the authors describe a method to derive the system of equations for a general class $\mathcal{C}$ with finite number of simple permutations from the specification of its substitution closure. The key idea is to compute the embeddings of non-simple permutations γ of the basis B of $\mathcal{C}$ into simple permutations π belonging

[3]To lighten the statements, we will not recall this in the remainder of this chapter.

to the class $\mathcal{C}$ (and into 12 and 21). These embeddings are block decompositions of the permutations γ, each block being translated into a new constraint in the decomposition tree, and creating a new equation in the specification for $\mathcal{C}$. The obtained combinatorial system may be ambiguous since the obtained sets are not always disjoint.

A first system of equations

When $\mathcal{C}$ is not substitution-closed, we compute a new system by adding constraints to the system obtained for $\hat{\mathcal{C}}$, as in [AA05].

Definition 3.9. For any set $\mathcal{A}$ of permutations and any set E of patterns, we denote by $\mathcal{A}\langle E\rangle$ the set of permutations of $\mathcal{A}$ that avoid every pattern in E.

Moreover we denote by $B^\star$ the subset of non-simple permutations of B.

Notice that given a set of permutations $\mathcal{A}$ and a set of patterns E, we have $(\mathcal{A}\langle E\rangle)^+ = \mathcal{A}^+\langle E\rangle$: both notations correspond to permutations of $\mathcal{A}$ that avoid E and that are $\oplus$-indecomposable. The same goes for $\mathcal{A}^-$.

Proposition 3.10. *Let $\mathcal{C}$ be a permutation class, that contains* 12 *and* 21*. We have that $\mathcal{C}^\varepsilon = \hat{\mathcal{C}}^\varepsilon\langle B^\star\rangle$ for $\varepsilon \in \{\emptyset, +, -\}$. Moreover,*

$$\hat{\mathcal{C}}\langle B^\star\rangle = 1 \uplus 12[\hat{\mathcal{C}}^+,\hat{\mathcal{C}}]\langle B^\star\rangle \uplus 21[\hat{\mathcal{C}}^-,\hat{\mathcal{C}}]\langle B^\star\rangle \uplus \biguplus_{\pi\in\mathcal{S}_\mathcal{C}} \pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}]\langle B^\star\rangle \quad (3.4)$$

$$\hat{\mathcal{C}}^+\langle B^\star\rangle = 1 \uplus 21[\hat{\mathcal{C}}^-,\hat{\mathcal{C}}]\langle B^\star\rangle \uplus \biguplus_{\pi\in\mathcal{S}_\mathcal{C}} \pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}]\langle B^\star\rangle \quad (3.5)$$

$$\hat{\mathcal{C}}^-\langle B^\star\rangle = 1 \uplus 12[\hat{\mathcal{C}}^+,\hat{\mathcal{C}}]\langle B^\star\rangle \uplus \biguplus_{\pi\in\mathcal{S}_\mathcal{C}} \pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}]\langle B^\star\rangle, \quad (3.6)$$

all these unions being disjoint.

Proof. Let $\sigma \in \mathcal{C}^\varepsilon$, then $\sigma \in \hat{\mathcal{C}}^\varepsilon$ and σ avoids $B^\star$, thus $\sigma \in \hat{\mathcal{C}}^\varepsilon\langle B^\star\rangle$. Conversely, let $\sigma \in \hat{\mathcal{C}}^\varepsilon\langle B^\star\rangle$ and let $\pi \in B$. If $\pi \in B^\star$ then σ avoids π. Otherwise, π is simple and $\pi \notin \mathcal{C}^\varepsilon$, hence $\pi \notin \hat{\mathcal{C}}^\varepsilon$. Indeed by definition of $\hat{\mathcal{C}}$, $\mathcal{S}_{\hat{\mathcal{C}}} = \mathcal{S}_\mathcal{C}$ (see Proposition 0.38). Since $\sigma \in \hat{\mathcal{C}}^\varepsilon$, σ avoids π. Hence $\sigma \in \mathcal{C}^\varepsilon$. Finally $\hat{\mathcal{C}}^\varepsilon(B^\star) = \mathcal{C}^\varepsilon$. Then the result follows from Proposition 3.6 recalling that $\mathcal{S}_{\hat{\mathcal{C}}} = \mathcal{S}_\mathcal{C}$. ∎

This description is not complete, since sets of the form $\pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}]\langle B^\star\rangle$ are not immediately described from $\hat{\mathcal{C}}\langle B^\star\rangle$. Lemma 18 of [AA05] states that sets such as $\pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}]\langle B^\star\rangle$ can be expressed as union of smaller sets:

$$\pi[\hat{\mathcal{C}},\dots,\hat{\mathcal{C}}]\langle B^\star\rangle = \bigcup_{i=1}^k \pi[\hat{\mathcal{C}}\langle E_{i,1}\rangle, \hat{\mathcal{C}}\langle E_{i,2}\rangle, \dots, \hat{\mathcal{C}}\langle E_{i,k}\rangle]$$

where the $E_{i,j}$ are sets of permutations which are patterns of some permutations of $B^\star$. This introduces sets of the form $\hat{\mathcal{C}}\langle E_{i,j}\rangle$ on the right-hand side of an equation of the system that do not appear on the left-hand side of any equation. We will call such sets *right-only* sets. Taking $E_{i,j}$ instead of $B^\star$ in Equations (3.4) to (3.6), we can recursively compute these right-only sets by introducing new equations in the system. This process terminates since there exists only a finite number of sets of patterns of elements of $B^\star$ (as B is finite). This will be explained in details in the following of this section. In the next subsection we describe formally the sets $E_{i,j}$.

Propagating constraints

Let us introduce some definitions useful to describe these sets $E_{i,j}$.

Definition 3.11. We denote 0 the empty permutation, which has size zero, and we take the convention $\mathcal{A}\langle 0\rangle = \emptyset$.

A *generalized substitution* $\sigma\{\pi^1, \pi^2, \dots, \pi^n\}$ is defined as a substitution (see Definition 0.25) with the particularity that any π^i may be the empty permutation. More formally:

Definition 3.12. Let σ be a permutation of size n and let $\pi^1, \dots, \pi^n$ be n permutations of size $p_1, \dots, p_n$ respectively, which may be empty, then the *generalized substitution* $\tau = \sigma\{\pi^1, \pi^2, \dots, \pi^n\}$ of $\pi^1, \pi^2, \dots, \pi^n$ in σ is

$$\sigma\{\pi^1, \pi^2, \dots, \pi^n\} = S^1 \dots S^n \text{ where } S^i = S^i_1 \dots S^i_{p_i} \text{ and}$$
$$S^i_x = (\pi^i_x + p_{\sigma^{-1}(1)} + \dots + p_{\sigma^{-1}(\sigma_i - 1)}) \text{ for any } x \text{ between 1 and } p_i.$$

Notice that $\sigma[\pi^1, \pi^2, \dots, \pi^n]$ necessarily contains σ whereas $\sigma\{\pi^1, \pi^2, \dots, \pi^n\}$ may avoid σ. For instance, the generalized substitution $1\,3\,2\{2\,1, 0, 1\}$ gives the permutation $2\,1\,3$ which avoids $1\,3\,2$.

Thanks to generalized substitutions, we define the notion of embedding, which express how a pattern γ can be involved in a permutation whose decomposition tree has a root π:

Definition 3.13. Let $\pi = \pi_1 \dots \pi_n \in S_n$ and $\gamma \in S_p$ be two permutations and P_γ the set of intervals of γ, including 0 and γ. An *embedding of* γ *in* $\pi = \pi_1 \dots \pi_n$ is a map α from $\{1, \dots, n\}$ to the set of (possibly empty) intervals[4] of γ such that:

- if the intervals $\alpha(i)$ and $\alpha(j)$ are not empty, and $i < j$, then $\alpha(i)$ consists of smaller indices than $\alpha(j)$;
- as a word, $\alpha(1) \dots \alpha(n)$ is a factorization of the word $1 \dots |\gamma|$ (which may include empty factors).
- denoting γ_I the pattern corresponding to $\gamma_{i_1} \dots \gamma_{i_\ell}$ for any interval I of indices from i_1 to i_ℓ in increasing order, we have $\pi\{\gamma_{\alpha(1)}, \dots, \gamma_{\alpha(n)}\} = \gamma$.

Example 3.14. For any permutations γ and π, $\alpha : \begin{cases} 1 & \mapsto [1..|\gamma|] \\ k > 1 & \mapsto \emptyset \end{cases}$ is an embedding of γ in π. Indeed $\gamma_{[1..|\gamma|]} = \gamma$ and $\pi\{\gamma, 0, \dots, 0\} = \gamma$.

Note that if we denote the non-empty images of α by $\alpha^1, \dots, \alpha^k$ and if we remove from π the π_i such that $\alpha(i) = 0$, we obtain a pattern σ of π such that $\gamma = \sigma[\gamma_{\alpha^1}, \dots, \gamma_{\alpha^k}]$. But this pattern σ may occur at several places in π so a block-decomposition $\gamma = \sigma[\gamma_{\alpha^1}, \dots, \gamma_{\alpha^k}]$ may correspond to several embeddings of γ in π.

Example 3.15. There are 11 embeddings of $\gamma = 5\,4\,6\,3\,1\,2$ into $\pi = 3\,1\,4\,2$, which correspond for instance to the generalized substitutions $\pi\{3241, 12, 0, 0\}$, $\pi\{3241, 0, 0, 12\}$ and $\pi\{0, 0, 3241, 12\}$ for the same expression of γ as the substitution $21[3241, 12]$, or $\pi\{3241, 1, 0, 1\}$ which is the only one corresponding to $312[3241, 1, 1]$.

These 11 embeddings are:

[4] Recall that here an interval of a permutation is a set of *indices* corresponding to a block of the permutation.

		$\gamma_{\alpha(1)}$	$\gamma_{\alpha(2)}$	$\gamma_{\alpha(3)}$	$\gamma_{\alpha(4)}$
$\gamma = 1[546312]$	$= \pi\{546312, 0, 0, 0\}$	546312	0	0	0
	$= \pi\{0, 546312, 0, 0\}$	0	546312	0	0
	$= \pi\{0, 0, 546312, 0\}$	0	0	546312	0
	$= \pi\{0, 0, 0, 546312\}$	0	0	0	546312
$\gamma = 21[213, 312]$	$= \pi\{213, 312, 0, 0\}$	213	312	0	0
	$= \pi\{213, 0, 0, 312\}$	213	0	0	312
	$= \pi\{0, 0, 213, 312\}$	0	0	213	312
$\gamma = 21[3241, 12]$	$= \pi\{3241, 12, 0, 0\}$	3241	12	0	0
	$= \pi\{3241, 0, 0, 12\}$	3241	0	0	12
	$= \pi\{0, 0, 3241, 12\}$	0	0	3241	12
$\gamma = 312[3241, 1, 1]$	$= \pi\{3241, 1, 0, 1\}$	3241	1	0	1

Notice that this definition of embeddings conveys the same notion as in [AA05], but it is formally different and it will turn out to be more adapted to the definition of the sets $E_{i,j}$.

Equations (3.4) to (3.6) can be viewed as Equations (3.1) to (3.3) "decorated" with pattern avoidance constraints. These constraints apply to every set $\pi[\hat{\mathcal{C}}_1, \dots, \hat{\mathcal{C}}_n]$ that appears in a disjoint union on the right-hand side of an equation. For each such set, the pattern avoidance constraints can be expressed by pushing constraints into the subtrees, using embeddings of excluded patterns in the root π. For instance, assume that $\gamma = 5\,4\,6\,3\,1\,2 \in B^\star$ and $\mathcal{S}_\mathcal{C} = \{3142\}$, and consider $3142[\hat{\mathcal{C}}, \hat{\mathcal{C}}, \hat{\mathcal{C}}, \hat{\mathcal{C}}]\langle\gamma\rangle$. The embeddings of γ in 3142 indicates how the pattern γ can be found in the subtrees in $3142[\hat{\mathcal{C}}, \hat{\mathcal{C}}, \hat{\mathcal{C}}, \hat{\mathcal{C}}]$. For instance, the first embedding of Example 3.15 indicates that the full pattern γ can appear all included in the first subtree. On the other hand, the last embedding of the same example tells us that γ can spread over all the subtrees of 3142 except the third. In order to avoid this particular embedding of γ, it is enough to avoid one of the induced pattern γ_I on one of the subtrees. However, in order to ensure that γ is avoided, the constraints resulting from all the embeddings must be considered and merged. This is formalized in Proposition 3.16.

Proposition 3.16. *Let π be a simple permutation of size n and $\mathcal{C}_1, \dots, \mathcal{C}_n$ be sets of permutations. For any permutation γ, the set $\pi[\mathcal{C}_1, \dots, \mathcal{C}_n]\langle\gamma\rangle$ rewrites as a union of sets of the form $\pi[\mathcal{D}_1, \dots, \mathcal{D}_n]$ where, for all i, $\mathcal{D}_i = \mathcal{C}_i\langle\gamma, \dots\rangle$ and the restrictions appearing after γ (if there are any) are patterns of γ corresponding to a block of γ.*

More precisely, let $\{\alpha_1, \dots, \alpha_\ell\}$ be the set of embeddings of γ in π, each α_i being associated to the generalized substitution $\gamma = \pi\{\gamma_{\alpha_i(1)}, \dots, \gamma_{\alpha_i(n)}\}$ where $\gamma_{\alpha_i(k)}$ is embedded in π_k. Then

$$\pi[\mathcal{C}_1, \dots, \mathcal{C}_n]\langle\gamma\rangle = \bigcup_{(k_1,\dots,k_\ell) \in K_\gamma^\pi} \pi[\mathcal{C}_1\langle E_{1,k_1\dots k_\ell}\rangle, \dots, \mathcal{C}_n\langle E_{n,k_1\dots k_\ell}\rangle] \qquad (3.7)$$

where $K_\gamma^\pi = \{(k_1, \dots, k_\ell) \in [1..n]^\ell \mid \forall i,\ \gamma_{\alpha_i(k_i)} \neq 0$ and $\gamma_{\alpha_i(k_i)} \neq 1\}$ and $E_{m,k_1\dots k_\ell} = \{\gamma_{\alpha_i(k_i)} \mid i \in [1..\ell]$ and $k_i = m\}$ is a set containing at least γ for $(k_1, \dots, k_\ell) \in K_\gamma^\pi$.

In a tuple $(k_1, \dots, k_\ell)$ of K_γ^π, k_i indicates a subtree of π where the pattern avoidance constraint ($\gamma_{\alpha_i(k_i)}$ excluded) forbids any occurrence of γ that could result from the embedding α_i. The set $E_{m,k_1\dots k_\ell}$ represents the pattern-avoidance constraints that have been pushed into the m-th subtree of π by embedding α_i of γ in π (for some values of i between 1 and ℓ) where the interval $\alpha_i(k_i)$ of γ is embedded into π_m (for some values of k_i between 1 and n).

Proof. Let $\sigma \in \pi[\mathcal{C}_1,\dots,\mathcal{C}_n]$; then $\sigma = \pi[\sigma^{(1)},\dots,\sigma^{(n)}]$ where each $\sigma^{(k)} \in \mathcal{C}_k$. Then σ contains γ if and only if there exists an embedding α_i of γ in π such that each $\gamma_{\alpha_i(k)}$ of the associated decomposition is a pattern of $\sigma^{(k)}$. Using contraposition, σ avoids γ if and only if for every embedding α_i, there exists a $\gamma_{\alpha_i(k)}$ which is not a pattern of $\sigma^{(k)}$, i.e. such that $\sigma^{(k)}$ avoids $\gamma_{\alpha_i(k)}$. Thus,

$$\pi[\mathcal{C}_1,\dots,\mathcal{C}_n]\langle\gamma\rangle = \bigcap_{i=1}^{\ell}\bigcup_{k=1}^{n}\pi[\mathcal{C}_1,\dots,\mathcal{C}_k\langle\gamma_{\alpha_i(k)}\rangle,\dots,\mathcal{C}_n].$$

Intersection being distributive over union, we have:

$$\pi[\mathcal{C}_1,\dots,\mathcal{C}_n]\langle\gamma\rangle = \bigcup_{(k_1,\dots,k_\ell)\in[1..n]^\ell}\ \bigcap_{i=1}^{\ell}\pi[\mathcal{C}_1,\dots,\mathcal{C}_{k_i}\langle\gamma_{\alpha_i(k_i)}\rangle,\dots,\mathcal{C}_n].$$

But, denoting 0 the empty permutation, for any set $\mathcal{A}$ of permutations we have $\mathcal{A}\langle 0\rangle = \emptyset$. So if for some $i \in [1..\ell]$, $\gamma_{\alpha_i(k_i)} = 0$, then $\bigcap_{i=1}^{\ell}\pi[\mathcal{C}_1,\dots,\mathcal{C}_{k_i}\langle\gamma_{\alpha_i(k_i)}\rangle,\dots,\mathcal{C}_n] = \emptyset$. The same goes for the trivial permutation 1 since any permutation contains 1.
Therefore with $K^\pi_\gamma = \{(k_1,\dots,k_\ell) \in [1..n]^\ell \mid \forall i,\ \gamma_{\alpha_i(k_i)} \neq 0 \text{ and } \gamma_{\alpha_i(k_i)} \neq 1\}$, we have:

$$\pi[\mathcal{C}_1,\dots,\mathcal{C}_n]\langle\gamma\rangle = \bigcup_{(k_1,\dots,k_\ell)\in K^\pi_\gamma}\ \bigcap_{i=1}^{\ell}\pi[\mathcal{C}_1,\dots,\mathcal{C}_{k_i}\langle\gamma_{\alpha_i(k_i)}\rangle,\dots,\mathcal{C}_n].$$

Moreover, as π is simple, by uniqueness of the substitution decomposition we have for any sets of permutations $E_1,\dots,E_n$, $F_1,\dots,F_n$:

$$\pi[\mathcal{C}_1\langle E_1\rangle,\dots,\mathcal{C}_n\langle E_n\rangle] \cap \pi[\mathcal{C}_1\langle F_1\rangle,\dots,\mathcal{C}_n\langle F_n\rangle] = \pi[\mathcal{C}_1\langle E_1\cup F_1\rangle,\dots,\mathcal{C}_n\langle E_n\cup F_n\rangle].$$

Thus,

$$\pi[\mathcal{C}_1,\dots,\mathcal{C}_n]\langle\gamma\rangle = \bigcup_{(k_1,\dots,k_\ell)\in[1..n]^\ell}\pi[\mathcal{C}_1\langle E_{1,k_1\dots k_\ell}\rangle,\dots,\mathcal{C}_n\langle E_{n,k_1\dots k_\ell}\rangle].$$

where $E_{m,k_1\dots k_\ell} = \{\gamma_{\alpha_i(k_i)} \mid i \in [1..\ell] \text{ and } k_i = m\}$. For a given m, there always exists an embedding α_{j_m} of γ in π that maps the whole permutation γ to π_m. Indeed take j_m the index such that $\gamma_{\alpha_{j_m}(m)} = \gamma$ and $\gamma_{\alpha_{j_m}(q)} = 0$ for $q \neq m$. If $k_{j_m} \neq m$ then $\gamma_{\alpha_{j_m}(k_{j_m})} = 0$ and $(k_1,\dots,k_\ell) \notin K^\pi_\gamma$. Therefore when $(k_1,\dots,k_\ell) \in K^\pi_\gamma$ then $k_{j_m} = m$ and the set $E_{m,k_1\dots k_\ell}$ contain at least γ, and by definition its other elements are patterns of γ corresponding to blocks of γ. ■

For an example, see section 3.5 (p.132).

Notice that Proposition 3.16 is directly inspired from the proof of Lemma 18 of [AA05]. We recall this lemma here:

Proposition 3.17 (Lemma 18 of [AA05]). *Suppose that $\pi \in \mathcal{S}_n$ is simple, $\mathcal{C}_1,\dots,\mathcal{C}_n$ are classes of permutations and $\gamma_1,\dots,\gamma_k$ is a sequence of permutations.*
Then $\pi[\mathcal{C}_1,\dots,\mathcal{C}_n]\langle\gamma_1,\dots,\gamma_k\rangle$ can be represented as a union of sets of the form $\pi[\mathcal{D}_1,\dots,\mathcal{D}_n]$ where for $1 \le i \le k$, $\mathcal{D}_i$ is $\mathcal{C}_i\langle\gamma_1,\dots,\gamma_k\rangle$ or a strong subclass of this class (a strong subclass D of C being a proper subclass of C which has the property that every basis element of D is involved in some basis element of C).

The main difference is that Proposition 3.16 is explicit and can be directly used for algorithmic purpose. Notice also that the cited lemma, as stated, is not correct, even if this does not affect the correctness of the result it is used for. Here is a counter example: take $\pi = 2413$, $\mathcal{C}_i = Av(12)$ for all i, $k = 1$ and $\gamma = 213$. Then, considering the embedding of γ in π that maps 21 in π_1 and 3 in π_2, we have $\mathcal{D}_1 = Av(12, 21)$, which is not a strong subclass of $\mathcal{C}_1\langle\gamma\rangle = Av(12)$.

We can extend Proposition 3.16 to the case of a set P of excluded patterns, instead of a single permutation γ:

Proposition 3.18. *For any simple permutation π of size n and for any set of permutations P, the set $\pi[\mathcal{C}_1, \dots, \mathcal{C}_n]\langle P\rangle$ rewrites as a union of sets $\pi[\mathcal{D}_1, \dots, \mathcal{D}_n]$ where for all i, $\mathcal{D}_i = \mathcal{C}_i\langle P \cup P_i\rangle$ with P_i containing only permutations corresponding to blocks of elements of P.*

This proposition is proved by induction on the size of P, using Proposition 3.16.

We show in the same way that $12[\mathcal{C}_1^+, \mathcal{C}_2]\langle P\rangle$ (resp. $21[\mathcal{C}_1^-, \mathcal{C}_2]\langle P\rangle$) rewrites as a union of sets $12[\mathcal{D}_1^+, \mathcal{D}_2]$ (resp. $21[\mathcal{D}_1^-, \mathcal{D}_2]$) where for $i = 1$ or 2, $\mathcal{D}_i = \mathcal{C}_i\langle P \cup P_i\rangle$ with P_i containing only permutations corresponding to blocks of elements of P. Indeed we need π to be simple in the proof of Proposition 3.16 only because we use the uniqueness of the substitution decomposition. In the case $\pi = 12$ (resp. 21), the uniqueness is ensured by taking the set $\mathcal{C}_1^+$ of $\oplus$-indecomposable permutations of $\mathcal{C}_1$ (resp. the set $\mathcal{C}_1^-$ of $\ominus$-indecomposable permutations).

Now we have all the results we need to describe an algorithm computing a (possibly ambiguous) combinatorial system describing $\mathcal{C}$.

An algorithm computing a combinatorial system describing $\mathcal{C}$

We describe here an algorithm that takes as input the set $\mathcal{S}_\mathcal{C}$ of simple permutations in a class $\mathcal{C}$ and the basis B of $\mathcal{C}$, and that produces in output a (possibly ambiguous) system of combinatorial equations describing the permutations of $\mathcal{C}$ through their decomposition trees. As a consequence of Proposition 3.10 and Proposition 3.18, using the fact that the set of blocks of elements of $B^\star$ is finite, we have the following proposition:

Proposition 3.19. *For any permutation class $\mathcal{C}$ with a finite number of simple permutations, we can derive a finite combinatorial system of equations for $\mathcal{C}$ of the form $\mathcal{D}_1 = 1 \cup \bigcup \pi[\mathcal{D}_2, \dots, \mathcal{D}_n]$, where $\mathcal{D}_i = \hat{\mathcal{C}}^\varepsilon\langle B^\star \cup B_i\rangle$ with $\varepsilon \in \{\emptyset, +, -\}$ and B_i contains only permutations corresponding to blocks of elements of $B^\star$, and every $\mathcal{D}_i$ that appears in the system is the left part of one equation of the system. This can be done using Algorithm 7.*

Starting from a finite basis of patterns B, Algorithm 7 describes the whole process to compute an ambiguous system defining the class $\mathcal{C} = Av(B)$ knowing its set of simple permutations $\mathcal{S}_\mathcal{C}$. The propagation of the constraints expressed by Equation (3.7) is performed by the procedure AddConstraints. It is applied to every set of the form $\pi[\mathcal{C}_1, \dots, \mathcal{C}_n]\langle B'\rangle$ that appears in the equation defining some $\hat{\mathcal{C}}^\varepsilon\langle B'\rangle$ by the procedure ComputeEqn. Finally, Algorithm 7 computes an ambiguous system for a permutation class $Av(B)$ containing a finite number of simple permutations: it starts from Equations (3.4) to (3.6), and adds new equations to this system calling procedure ComputeEqn, until every $\pi[\mathcal{C}_1, \dots, \mathcal{C}_n]\langle B'\rangle$ is replaced by some $\pi[\mathcal{C}'_1, \dots, \mathcal{C}'_n]$ and until every $\mathcal{C}'_i = \hat{\mathcal{C}}^\varepsilon\langle B'_i\rangle$ is defined by an equation of the system. All the sets B' are sets of patterns of some permutations in B. Since there is only a finite number of patterns of elements of B, there is a finite number of possible B', and Algorithm 7 terminates. This produces a system of equations, that may be ambiguous: the unions may not be disjoint. The disambiguation of this system will be examined in Section 3.4.

Consider for instance the class $\mathcal{C} = Av(B)$ for $B = \{1243, 2413, 531642, 41352\}$: $\mathcal{C}$ contains only one simple permutation (namely 3142), and $B^\star = \{1243\}$. Applying Algorithm 7

Algorithm 7: AMBIGUOUSSYSTEM(B)

```
Data: $B$ is a finite basis of patterns defining $\mathcal{C} = Av(B)$ such that $\mathcal{S}_{\mathcal{C}}$ is known
      and finite.
Result: A system of equations of the form $\mathcal{D} = \bigcup \pi[\mathcal{D}_1, \dots, \mathcal{D}_n]$ defining $\mathcal{C}$.
begin
    $\mathcal{E} \leftarrow$ ComputeEqn(($\hat{\mathcal{C}}, B^\star$)) $\cup$ ComputeEqn(($\hat{\mathcal{C}}^+, B^\star$)) $\cup$
    ComputeEqn(($\hat{\mathcal{C}}^-, B^\star$))
    while there is a right-only $\hat{\mathcal{C}}^\varepsilon\langle B'\rangle$ in some equation of $\mathcal{E}$ do
        $\mathcal{E} \leftarrow \mathcal{E} \cup$ ComputeEqn($\hat{\mathcal{C}}^\varepsilon$, $B'$)

/* Returns an equation defining $\hat{\mathcal{C}}^\varepsilon\langle B'\rangle$ as a union of $\pi[\mathcal{C}_1, \dots, \mathcal{C}_n]$ */
/* $B'$ is a set of permutations, $\hat{\mathcal{C}}^\varepsilon$ is given by $\mathcal{S}_{\hat{\mathcal{C}}}$ and $\varepsilon \in \{\emptyset, +, -\}$ */
ComputeEqn ($\hat{\mathcal{C}}^\varepsilon, B'$)
    $\mathcal{E} \leftarrow$ Equation (3.4) or (3.5) or (3.6) (depending on $\varepsilon$) written with $B'$ instead of
    $B^\star$
    foreach $t = \pi[\mathcal{C}_1, \dots, \mathcal{C}_n]\langle B'\rangle$ that appears in $\mathcal{E}$ do
        $t \leftarrow$ AddConstraints($\pi[\mathcal{C}_1, \dots, \mathcal{C}_n], B'$)
    return $\mathcal{E}$

/* Returns a rewriting of $\pi[\mathcal{C}_1 \dots \mathcal{C}_n]\langle E\rangle$ as a union $\bigcup \pi[\mathcal{D}_1, \dots \mathcal{D}_n]$ */
AddConstraints (($\pi[\mathcal{C}_1 \dots \mathcal{C}_n], E$))
    if $E = \emptyset$ then return $\pi[\mathcal{C}_1 \dots \mathcal{C}_n]$;
    ;
    else
        choose $\gamma \in E$ and compute all the embeddings of $\gamma$ in $\pi$
        compute $K^\pi_\gamma$ and sets $E_{m,k_1 \dots k_\ell}$ defined in Equation (3.7)
        return
        $\bigcup_{(k_1,\dots,k_\ell)\in K^\pi_\gamma}$ AddConstraints($\pi[\mathcal{C}_1\langle E_{1,k_1 \dots k_\ell}\rangle, \dots, \mathcal{C}_n\langle E_{n,k_1 \dots k_\ell}\rangle], E \setminus \gamma$).
```

to this class $\mathcal{C}$ gives the following system of equations:

$$\begin{aligned}
\hat{\mathcal{C}}\langle 1243\rangle &= 1 \cup 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 1243\rangle, \hat{\mathcal{C}}\langle 21\rangle] \cup 21[\hat{\mathcal{C}}^-\langle 1243\rangle, \hat{\mathcal{C}}\langle 1243\rangle] \\
&\cup 3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle, \hat{\mathcal{C}}\langle 132\rangle] \quad (3.8) \\
\hat{\mathcal{C}}\langle 12\rangle &= 1 \cup 21[\hat{\mathcal{C}}^-\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \quad (3.9) \\
\hat{\mathcal{C}}\langle 132\rangle &= 1 \cup 12[\hat{\mathcal{C}}^+\langle 132\rangle, \hat{\mathcal{C}}\langle 21\rangle] \cup 21[\hat{\mathcal{C}}^-\langle 132\rangle, \hat{\mathcal{C}}\langle 132\rangle] \quad (3.10) \\
\hat{\mathcal{C}}\langle 21\rangle &= 1 \cup 12[\hat{\mathcal{C}}^+\langle 21\rangle, \hat{\mathcal{C}}\langle 21\rangle]. \quad (3.11)
\end{aligned}$$

3.4 Disambiguation of the system

In the above, Equation (3.8) gives an ambiguous description of the class $\hat{\mathcal{C}}\langle 1243\rangle$. As noticed in [AA05], we can derive an unambiguous equation using the inclusion-exclusion principle: $\hat{\mathcal{C}}\langle 1243\rangle = 1 \cup 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 1243\rangle, \hat{\mathcal{C}}\langle 21\rangle] \setminus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle] \cup 21[\hat{\mathcal{C}}^-\langle 1243\rangle, \hat{\mathcal{C}}\langle 1243\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle] \setminus 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle]$. The system so obtained contains negative terms in general. This still gives a system of equations allowing the computation of the generating function of the class. However, this cannot easily be used for random generation, as the subtraction of combinatorial objects is not handled by random samplers. In this section we disambiguate this system to obtain a new positive one: the key idea is to replace the negative terms by *complement sets*, thereby transforming pattern avoidance constraints into pattern *containment* constraints.

3.4.1 General framework

The starting point of the disambiguation is to rewrite ambiguous terms like $A \cup B \cup C$ as a disjoint union $(A \cap B \cap C) \uplus (\bar{A} \cap B \cap C) \uplus (\bar{A} \cap \bar{B} \cap C) \uplus (\bar{A} \cap B \cap \bar{C}) \uplus (A \cap \bar{B} \cap C) \uplus (A \cap \bar{B} \cap \bar{C}) \uplus (A \cap B \cap \bar{C})$(see Figure 3.2). By disambiguating the union $A \cup B \cup C$ using complement sets instead of negative terms, we obtain an unambiguous description of the union with only positive terms. But when taking the complement of a set defined by pattern avoidance constraints, these are transformed into pattern *containment* constraints.

Therefore, for any set $\mathcal{P}$ of permutations, we define $\mathcal{P}\langle E\rangle(A)$ of $\mathcal{P}$ as the set of permutations that belong to $\mathcal{P}$ and that avoid every pattern of E and contain every pattern of A. More formally:

Definition 3.20. For any set $\mathcal{P}$ of permutations, we set
$$\mathcal{P}\langle E\rangle(A) = \{\sigma \in \mathcal{P} \mid \forall \pi \in E, \pi \not\leq \sigma \text{ and } \forall \pi \in A, \pi \leq \sigma\}.$$
When $\mathcal{P} = \hat{\mathcal{C}}^{\varepsilon}$ for $\varepsilon \in \{\emptyset, +, -\}$ and $\mathcal{C}$ a permutation class, such a set is called a *restriction.*

With this notation, notice also that for $A = \emptyset$, $\mathcal{C}\langle E\rangle = \mathcal{C}\langle E\rangle(\emptyset)$ is a standard permutation class. For the empty permutation 0, we take the convention $\mathcal{P}\langle 0\rangle(A) = \emptyset$ for any A, and $\mathcal{P}\langle E\rangle(0) = \mathcal{P}\langle E\rangle$ for any E. Moreover we assume that $A \cap E$ is empty, otherwise $\mathcal{P}\langle E\rangle(A) = \emptyset = \mathcal{P}\langle 0\rangle(A \setminus \{0\})$. Restrictions have the nice feature of being stable by intersection as $\mathcal{P}\langle E\rangle(A) \cap \mathcal{P}\langle E'\rangle(A') = \mathcal{P}\langle E \cup E'\rangle(A \cup A')$.

We also define a restriction term to be a set of permutations $\pi[\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_n]$ where π is a simple permutation or 12 or 21 and the $\mathcal{D}_i$ are restrictions. More precisely:

Definition 3.21. A *restriction term* is a set of permutations described as $\pi[\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_n]$ such that for $i \geq 2$ the $\mathcal{D}_i$ are restrictions of the form $\hat{\mathcal{C}}\langle E\rangle(A)$ and moreover, either π is a simple permutation and $\mathcal{D}_1$ is a restriction of the form $\hat{\mathcal{C}}\langle E\rangle(A)$, or π is 12 (resp. 21) and $\mathcal{D}_1$ is a restrictions of the form $\hat{\mathcal{C}}^{+}\langle E\rangle(A)$ (resp. $\hat{\mathcal{C}}^{-}\langle E\rangle(A)$).

By uniqueness of the substitution decomposition of a permutation (Theorem 0.29), restriction terms are stable by intersection as well and the intersection is performed componentwise for terms sharing the same root: $\pi[\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_n] \cap \pi[\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n] = \pi[\mathcal{D}_1 \cap \mathcal{T}_1, \mathcal{D}_2 \cap \mathcal{T}_2, \dots, \mathcal{D}_n \cap \mathcal{T}_n]$.

3.4.2 Disambiguation

The disambiguation of the system obtained by Algorithm 7 is performed by Algorithm 8. It consists in two main operations. One is the disambiguation of an equation according to the root of the terms that induce ambiguity, which may introduce right-only restrictions. This leads to the second procedure which computes new equations (that are added to the system) to describe these new restrictions (Algorithm 9).

Algorithm 8: DisambiguateSystem($\mathcal{E}$)

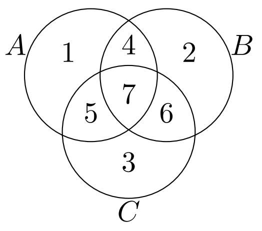
```
Data: A ambiguous system $\mathcal{E}$ of combinatorial equations /* obtained by Algo. 7 */
Result: An unambiguous system of combinatorial equations equivalent to $\mathcal{E}$
begin
    while there is an ambiguous equation $F$ in $\mathcal{E}$ do
        Take $\pi$ a root that appears several times in $F$ in an ambiguous way
        Replace the restriction terms of F whose root is $\pi$ by a disjoint union using
        Equations (3.12), (3.13) and (3.14)
        while there exists a right-only restriction $\hat{\mathcal{C}}^\varepsilon\langle E\rangle(A)$ in some equation of $\mathcal{E}$
        do
            $\mathcal{E} \longleftarrow \mathcal{E} \bigcup$ ComputeEqnForRestriction($\hat{\mathcal{C}}^\varepsilon$,$E$,$A$).   /* See Algo. 9 */
    return $\mathcal{E}$
```

As stated in Section 3.3, every equation F of our system can be written as $t = 1 \cup t_1 \cup t_2 \cup t_3 \ldots \cup t_k$ where the t_i are restriction terms (some $\pi[\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n]$) and t is a restriction (some $\hat{\mathcal{C}}^\varepsilon\langle E\rangle(A)$, with $A = \emptyset$ at the beginning of the disambiguation process). By uniqueness of the substitution decomposition of a permutation, restriction terms of this union which have different roots π are disjoint. Thus for an equation we only need to disambiguate unions of terms with same root.

For example in Equation (3.8), there are two pairs of ambiguous terms which are terms with root 3142 and terms with root 12. Every ambiguous union can be written in an unambiguous way:

Proposition 3.22. *A union of n sets $\bigcup_{i=1}^n A_i$ rewrites as the disjoint union of the $2^n - 1$ sets of the form $\bigcap_{i=1}^n X_i$ with $X_i \in \{A_i, \overline{A_i}\}$ where $\overline{A_i}$ is the complement of A_i in any set containing $\bigcup_{i=1}^n A_i$, and not every X_i is equal to $\overline{A_i}$:*

This proposition is the starting point of the disambiguation. See Figure 3.2 for an example.



$$\begin{aligned} A \cup B \cup C &= 1 \uplus 2 \uplus 3 \uplus 4 \uplus 5 \uplus 6 \uplus 7 \\ &= (A \cap \overline{B} \cap \overline{C}) \uplus (\overline{A} \cap B \cap \overline{C}) \uplus (\overline{A} \cap \overline{B} \cap C) \\ &\uplus (A \cap B \cap \overline{C}) \uplus (A \cap \overline{B} \cap C) \uplus (\overline{A} \cap B \cap C) \\ &\uplus (A \cap B \cap C) \end{aligned}$$

Figure 3.2: Rewriting unions as disjoint unions.

In order to use Proposition 3.22, we have to choose in which set we take complements.

Definition 3.23. For any restriction term $t = \pi[\mathcal{D}_1, \ldots, \mathcal{D}_n]$ with π simple and such that for all i, $\mathcal{D}_i \subset \hat{\mathcal{C}}$, we set $\overline{t} = \pi[\hat{\mathcal{C}}, \ldots, \hat{\mathcal{C}}] \setminus t$.
If $t = 12[\mathcal{D}_1^+, \mathcal{D}_2]$ with $\mathcal{D}_i \subset \hat{\mathcal{C}}$, we set $\overline{t} = \pi[\hat{\mathcal{C}}^+, \hat{\mathcal{C}}] \setminus t$.
If $t = 21[\mathcal{D}_1^-, \mathcal{D}_2]$ with $\mathcal{D}_i \subset \hat{\mathcal{C}}$, we set $\overline{t} = \pi[\hat{\mathcal{C}}^-, \hat{\mathcal{C}}] \setminus t$.
Moreover for any restriction $\mathcal{D}$ of $\hat{\mathcal{C}}^\varepsilon$ with $\varepsilon \in \{\emptyset, +, -\}$, we set $\overline{\mathcal{D}} = \hat{\mathcal{C}}^\varepsilon \setminus \mathcal{D}$.

Thus every ambiguous union of restriction terms sharing the same root in an equation of our system can be written in the following unambiguous way:

$$\bigcup_{i=1}^k t_i = \biguplus_{X \subseteq [1 \ldots k], X \neq \emptyset} \bigcap_{i \in X} t_i \cap \bigcap_{i \in \overline{X}} \overline{t_i}, \tag{3.12}$$

where the complement $\overline{t_i}$ of a restriction term t_i is the set of permutations of $\hat{\mathcal{C}}$ whose decomposition tree has the same root than t_i but that do not belong to t_i.

For instance, consider terms with root 3142 in Equation (3.8):
$t_1 = 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle, \hat{\mathcal{C}}\langle 132\rangle]$ and $t_2 = 3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle]$. Equation (3.12) applied to t_1 and t_2 gives an expression of the form $\hat{\mathcal{C}}\langle 1243\rangle = 1 \cup 12[\ldots] \cup 12[\ldots] \cup 21[\ldots] \cup (t_1 \cap t_2) \uplus (t_1 \cap \overline{t_2}) \uplus (\overline{t_1} \cap t_2)$.

We now explain how to compute the complement $\bar{t}$ of a restriction term t.

Proposition 3.24. *Let $t = \pi[\mathcal{D}_1, \ldots, \mathcal{D}_n]$ be a restriction term. Then $\bar{t}$ is the disjoint union of the $2^n - 1$ sets of the form $\pi[X_1, \ldots, X_n]$ with $X_i \in \{\mathcal{D}_i, \overline{\mathcal{D}_i}\}$, and not all X_i are equal to $\mathcal{D}_i$. More formally:*

$$\bar{t} = \biguplus_{X \subseteq \{1,\ldots,n\}, X \neq \emptyset} \pi[\mathcal{D}'_1, \ldots, \mathcal{D}'_n] \text{ where } \mathcal{D}'_i = \overline{\mathcal{D}_i} \text{ if } i \in X \text{ and } \mathcal{D}'_i = \mathcal{D}_i \text{ otherwise}, \quad (3.13)$$

For example, $\overline{21[\mathcal{D}_1, \mathcal{D}_2]} = 21[\mathcal{D}_1, \overline{\mathcal{D}_2}] \uplus 21[\overline{\mathcal{D}_1}, \mathcal{D}_2] \uplus 21[\overline{\mathcal{D}_1}, \overline{\mathcal{D}_2}]$.

Proof. By uniqueness of substitution decomposition, the set of permutations of $\hat{\mathcal{C}}$ that do not belong to t but whose decomposition tree has root π can be written as the union of terms $u = \pi[\mathcal{D}'_1, \mathcal{D}'_2, \ldots, \mathcal{D}'_n]$ where $\mathcal{D}'_i = \mathcal{D}_i$ or $\mathcal{D}'_i = \overline{\mathcal{D}_i}$ and at least one restriction $\mathcal{D}_i$ must be complemented.

More precisely, suppose that π is simple, then $\bar{t} = \pi[\hat{\mathcal{C}}, \ldots, \hat{\mathcal{C}}] \setminus t$.

Let $\sigma \in \bar{t} = \pi[\hat{\mathcal{C}}, \ldots, \hat{\mathcal{C}}] \setminus t$; then $\sigma = \pi[\sigma_1, \ldots, \sigma_n]$ where each $\sigma_i \in \hat{\mathcal{C}}$. So $\sigma \in \pi[X_1, \ldots, X_n]$ with $X_i = \mathcal{D}_i$ if $\sigma_i \in \mathcal{D}_i$ and $X_i = \overline{\mathcal{D}_i} = \hat{\mathcal{C}} \setminus \mathcal{D}_i$ otherwise. But $\sigma \notin t$ so there is at least one index i such that $\sigma_i \notin \mathcal{D}_i$ and $X_i \neq \mathcal{D}_i$.

Conversely if $\sigma \in \pi[X_1, \ldots, X_n]$ with $X_i \in \{\mathcal{D}_i, \overline{\mathcal{D}_i}\}$ and at least one X_i is equal to $\overline{\mathcal{D}_i}$, then $\sigma \in \pi[\hat{\mathcal{C}}, \ldots, \hat{\mathcal{C}}]$ and $\sigma \notin t$.

Finally for such sets X_i that are distinct, the sets $\pi[X_1, \ldots, X_n]$ are disjoints. Indeed $\mathcal{D}_i \cap \overline{\mathcal{D}_i}$ is empty and the writing as $\pi[\sigma_1, \ldots, \sigma_n]$ is unique. So the union describing $\bar{t}$ is disjoint. The proof is similar when $\pi = 12$ or $\pi = 21$. ■

Proposition 3.24 shows that $\overline{t_i}$ is not a restriction term in general. However it can be expressed as a disjoint union of some $\pi[\mathcal{D}'_1, \ldots, \mathcal{D}'_n]$, where the $\mathcal{D}'i$ are either restrictions or complements of restrictions. The complement operation being pushed from restriction terms down to restrictions, we now compute $\overline{\mathcal{D}}$, for a given restriction $\mathcal{D} = \hat{\mathcal{C}}^{\varepsilon}\langle E\rangle(A)$, $\overline{\mathcal{D}}$ denoting the set of permutations of $\hat{\mathcal{C}}^{\varepsilon}$ that are not in $\mathcal{D}$. Notice that given a permutation σ of A, then any permutation τ of $\hat{\mathcal{C}}^{\varepsilon}\langle\sigma\rangle$ is in $\overline{\mathcal{D}}$ because τ avoids σ whereas permutations of $\mathcal{D}$ must contain σ. Symmetrically, if a permutation σ is in E then permutations of $\hat{\mathcal{C}}^{\varepsilon}\langle\emptyset\rangle(\sigma)$ are in $\overline{\mathcal{D}}$. It is straightforward to check that $\overline{\hat{\mathcal{C}}^{\varepsilon}\langle E\rangle(A)} = \left[\bigcup_{\sigma \in E} \hat{\mathcal{C}}^{\varepsilon}\langle\emptyset\rangle(\sigma)\right] \bigcup \left[\bigcup_{\sigma \in A} \hat{\mathcal{C}}^{\varepsilon}\langle\sigma\rangle(\emptyset)\right]$. Unfortunately this expression is ambiguous. As before, we can rewrite it as an unambiguous union:

Proposition 3.25. *Let $\mathcal{D} = \hat{\mathcal{C}}^{\varepsilon}\langle E\rangle(A)$ be a restriction with $\varepsilon \in \{\emptyset, +, -\}$, $k = |E|$ and $\ell = |A|$. Then $\overline{\mathcal{D}}$ is the disjoint union of the $2^{k+\ell} - 1$ restrictions $\hat{\mathcal{C}}^{\varepsilon}\langle E'\rangle(A')$ with (E', A') a partition of $E \uplus A$ such that $(E', A') \neq (E, A)$. More formally:*

$$\overline{\hat{\mathcal{C}}^{\varepsilon}\langle E\rangle(A)} = \biguplus_{\substack{X \subseteq A, Y \subseteq E \\ X \times Y \neq \emptyset \times \emptyset}} \hat{\mathcal{C}}^{\varepsilon}\langle X \cup \overline{Y}\rangle(Y \cup \overline{X}), \text{ where } \overline{X} = A \setminus X \text{ and } \overline{Y} = E \setminus Y. \quad (3.14)$$

Proof. Let $\tau \in \overline{\mathcal{D}} = \hat{\mathcal{C}}^{\varepsilon} \setminus \mathcal{D}$. Then $\tau \in \hat{\mathcal{C}}^{\varepsilon}\langle E'\rangle(A')$ with $E' = \{\pi \in E \cup A \mid \pi \not\leq \tau\}$ and $A' = \{\pi \in E \cup A \mid \pi \leq \tau\}$. Moreover $E' \cap A' = \emptyset$, $E' \cup A' = E \cup A$ and $(E', A') \neq (E, A)$ otherwise τ would be in $\mathcal{D}$.

Conversely let $\tau \in \hat{\mathcal{C}}^{\varepsilon}\langle E'\rangle(A')$ with (E', A') a partition of $E \uplus A$ such that $(E', A') \neq (E, A)$, then $\tau \in \hat{\mathcal{C}}^{\varepsilon}$. As $(E', A') \neq (E, A)$, either there is some σ in E that τ contains, or there is some σ in A that τ contains. In both cases, $\tau \notin \mathcal{D}$ thus $\tau \in \hat{\mathcal{C}}^{\varepsilon} \setminus \mathcal{D} = \overline{\mathcal{D}}$.

Finally for distinct partitions (E', A') of $E \cup A$, the sets $\hat{\mathcal{C}}^{\varepsilon}\langle E'\rangle(A')$ are disjoints. Indeed a permutation in two sets of this form would have to both avoid and contain some permutation of $E \cup A$, which is impossible. ∎

In our example (Equations (3.8) to (3.11)), only trivial complements appear, since every restriction is of the form $\hat{\mathcal{C}}\langle\sigma\rangle(\emptyset)$ or $\hat{\mathcal{C}}\langle\emptyset\rangle(\sigma)$ for which complements are respectively $\hat{\mathcal{C}}\langle\emptyset\rangle(\sigma)$ and $\hat{\mathcal{C}}\langle\sigma\rangle(\emptyset)$.

Proposition 3.25 shows that $\overline{\mathcal{D}}$ is not a restriction in general but can be expressed as a disjoint union of restrictions. Moreover by uniqueness of the substitution decomposition, $\pi[\mathcal{D}_1, \dots, \mathcal{D}_k \uplus \mathcal{D}'_k, \dots, \mathcal{D}_n] = \pi[\mathcal{D}_1, \dots, \mathcal{D}_k, \dots, \mathcal{D}_n] \uplus \pi[\mathcal{D}_1, \dots, \mathcal{D}'_k, \dots, \mathcal{D}_n]$. Therefore using Equation (3.13) we have that for any restriction term t_i, its complement $\overline{t_i}$ can be expressed as a disjoint union of terms:

Proposition 3.26. *For any restriction term t, its complement $\bar{t}$ can be written as a disjoint union of restriction terms. More precisely if $t = \pi[\mathcal{D}_1, \dots, \mathcal{D}_n]$ with $D_i = \hat{\mathcal{C}}^{\varepsilon_i}\langle E_i\rangle(A_i)$ and $m = \sum_{i=1}^{n} |E_i| + |A_i|$, then $\bar{t}$ is the disjoint union of the $2^m - 1$ restriction terms $t = \pi[\mathcal{D}'_1, \dots, \mathcal{D}'_n]$ such that for all i, $D'_i = \hat{\mathcal{C}}^{\varepsilon_i}\langle E'_i\rangle(A'_i)$ with (E'_i, A'_i) a partition of $E_i \uplus A_i$, and there exists i such that $(E'_i, A'_i) \neq (E_i, A_i)$.*

By distributivity of $\cap$ over $\uplus$, Equation (3.12) above can therefore be rewritten as a disjoint union of intersection of terms. Because terms are stable by intersection, the right-hand side of Equation (3.12) is hereby written as a disjoint union of terms. This leads to the following result:

Proposition 3.27. *Any union of restriction terms can be written as a disjoint union of restriction terms, and this can be done algorithmically using Equations* (3.12), (3.13) *and* (3.14).

All together, for any equation of our system, we are able to rewrite it unambiguously as a disjoint union of restriction terms. As noted previously, some new right-only restrictions may appear during this process, for example as the result of the intersection of several restrictions or when complementing restrictions. To obtain a complete system we must compute iteratively equations defining these new restrictions using Algorithm 9 described below.

Finally, the termination of Algorithm 8 is easily proved. Indeed, for all the restrictions $\hat{\mathcal{C}}^{\varepsilon}\langle E\rangle(A)$ that are considered in the inner loop of Algorithm 8, every permutation in the sets E and A is a pattern of some element of the basis B of $\mathcal{C}$. And since B is finite, there is a finite number of such restrictions.

3.4.3 Compute an equation for a restriction

Let $\hat{\mathcal{C}}^{\varepsilon}\langle E\rangle(A)$ be a restriction. Our goal here is to find a combinatorial specification of this restriction in terms of smaller restriction terms (smaller w.r.t. inclusion).

If $A = \emptyset$, this is exactly the problem addressed in Section 3.3.2 and solved by pushing down the pattern avoidance constraints in the procedure AddConstraints of Algorithm 7. Algorithm 9 below shows how to propagate also the pattern *containment* constraints induced by $A \neq \emptyset$.

Algorithm 9: ComputeEqnForRestriction($\hat{\mathcal{C}}^\varepsilon, E, A$)

```
Data: $\hat{\mathcal{C}}^\varepsilon, E, A$ with $E, A$ sets of permutations, $\hat{\mathcal{C}}^\varepsilon$ given by $\mathcal{S}_{\hat{\mathcal{C}}}$ and $\varepsilon \in \{\emptyset, +, -\}$.
Result: An equation defining $\hat{\mathcal{C}}^\varepsilon\langle E\rangle(A)$ as a union of restriction terms.
begin
    $F \leftarrow$ Equation (3.1) or (3.2) or (3.3) (depending on $\varepsilon$)
    foreach $\sigma \in E$ do
        /* This step modifies $F$! */
        Replace any restriction term $t$ in $F$ by AddConstraints($t, \{\sigma\}$)        /* See
        Algo. 7 */
    foreach $\sigma \in A$ do
        /* This step modifies $F$! */
        Replace any restriction term $t$ in $F$ by AddMandatory($t, \sigma$)
    return $F$

AddMandatory ($\pi[\mathcal{D}_1, \dots, \mathcal{D}_n], \gamma$)
    return a rewriting of $\pi[\mathcal{D}_1, \dots, \mathcal{D}_n](\gamma)$ as a union of restriction terms using
    Equation (3.15).
```

The pattern *containment* constraints are propagated by AddMandatory, in a very similar fashion to the pattern *avoidance* constraints propagated by AddConstraints. To compute $t(\gamma)$ for γ a permutation and $t = \pi[\mathcal{D}_1, \dots, \mathcal{D}_n]$ a restriction term, we first compute all embeddings of γ into π. In this case, a permutation belongs to $t(\gamma)$ if and only if at least one embedding is satisfied. Then $\pi[\mathcal{D}_1, \dots, \mathcal{D}_n](\gamma)$ rewrites as a union of sets of the form $\pi[\mathcal{D}_1(\gamma_1), \dots, \mathcal{D}_n(\gamma_n)]$ where, for all i, γ_i is a block of γ which may be empty or γ itself (recall that if γ_i is empty, then $\mathcal{D}_j(\gamma_i) = \mathcal{D}_j$). More precisely:

Proposition 3.28. *Let π be a permutation of size n and $\mathcal{D}_1, \dots, \mathcal{D}_n$ be sets of permutations. For any permutation γ, let $\alpha_1, \dots, \alpha_\ell$ be the set of embeddings of γ in π, then*

$$\pi[\mathcal{D}_1, \dots, \mathcal{D}_n](\gamma) = \bigcup_{i=1}^{\ell} \pi[\mathcal{D}_1(\gamma_{\alpha_i(1)}), \mathcal{D}_2(\gamma_{\alpha_i(2)}), \dots, \mathcal{D}_n(\gamma_{\alpha_i(n)})]. \tag{3.15}$$

For instance, for $t = 2413[\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4]$ and $\gamma = 3214$, there are 9 embeddings of γ into 2413, and the embedding $2413\{321, 1, 0, 0\}$ contributes to the above union with the term $2413[\mathcal{D}_1(321), \mathcal{D}_2(1), \mathcal{D}_3, \mathcal{D}_4]$.

Proof. Let $\sigma \in \pi[\mathcal{D}_1, \dots, \mathcal{D}_n]$; then $\sigma = \pi[\sigma_1, \dots, \sigma_n]$ where each $\sigma_k \in \mathcal{D}_k$. Then σ contains γ, if and only if there exists an embedding α_i of $\gamma = \pi[\alpha_i(1) \dots \alpha_i(n)]$ such that $\alpha_i(j)$ is a pattern of σ_j for all j. ■

Hence, any restriction term $t = \pi[\mathcal{D}_1, \dots, \mathcal{D}_n](\gamma)$ rewrites as a (possibly ambiguous) union of restriction terms.

Notice that although the unions of Equation (3.15) may be ambiguous, they will be transformed into disjoint unions by the outer loop of Algorithm 8. Finally, the algorithm produces an unambiguous system which is the result of a finite number of iterations of computing equations followed by their disambiguation.

We are now able to state the main theorem of this chapter:

Theorem 3.29. *For any permutation class $\mathcal{C}$ whose set $\mathcal{S}_\mathcal{C}$ of simple permutations is finite, given its basis B and the set $\mathcal{S}_\mathcal{C}$, Algorithm 7 followed by Algorithm 8 derive a finite* *combinatorial system of equations of the form* $\mathcal{D}_1 = 1 \uplus \biguplus \pi[\mathcal{D}_2, \dots, \mathcal{D}_n]$*, where each* $\mathcal{D}_i$ *is a restriction* $\hat{\mathcal{C}}^{\varepsilon}\langle E\rangle(A)$ *with* $\varepsilon \in \{\emptyset, +, -\}$*,* E *and* A *contains only blocks of elements of* B*, and every* $\mathcal{D}_i$ *that appears in the system is the left part of one equation of the system. In particular this provides a combinatorial specification of* $\mathcal{C}$*.*

3.5 Case study of $\mathcal{C} = Av(2413, 1243, 2341, 531642, 41352)$

In this section, we apply the method described in this chapter to find a combinatorial specification for the permutation class $\mathcal{C} = Av(B)$ where $B = \{2413, 1243, 2341, 531642, 41352\}$. From this description we derive its generating function and furthermore generate at random large permutations of the class. We follow the different steps described in the diagram of Figure 3.4.

First, notice that $Av(B)$ is not substitution-closed as 1243 and 2341 are non-simple permutations and belong to the basis of the class. Then we test whether the class contains a finite number of simple permutations. In our case, the only simple permutation of the class is 3142.

3.5.1 Ambiguous system for $\mathcal{C}$

The first step of our algorithm deals with the substitution-closure of our class, that is, the substitution-closed class $\hat{\mathcal{C}}$ which contains only 3142 as simple permutation. Recall that a substitution-closed class can be either given by its basis or by the simple permutations it contains.

Substitution-closure $\hat{\mathcal{C}}$ of $\mathcal{C}$

From the simple permutations contained in a substitution-closed class, it is easy to give a combinatorial description of the class. Equations (3.1), (3.2) and (3.3) of Proposition 3.6 (p.120) translate into the following system:

$$\begin{aligned}
\hat{\mathcal{C}} &= 1 \uplus 12[\hat{\mathcal{C}}^+,\hat{\mathcal{C}}] \uplus 21[\hat{\mathcal{C}}^-,\hat{\mathcal{C}}] \uplus 3142[\hat{\mathcal{C}},\hat{\mathcal{C}},\hat{\mathcal{C}},\hat{\mathcal{C}}] && (3.16)\\
\hat{\mathcal{C}}^+ &= 1 \uplus 21[\hat{\mathcal{C}}^-,\hat{\mathcal{C}}] \uplus 3142[\hat{\mathcal{C}},\hat{\mathcal{C}},\hat{\mathcal{C}},\hat{\mathcal{C}}] && (3.17)\\
\hat{\mathcal{C}}^- &= 1 \uplus 12[\hat{\mathcal{C}}^+,\hat{\mathcal{C}}] \uplus 3142[\hat{\mathcal{C}},\hat{\mathcal{C}},\hat{\mathcal{C}},\hat{\mathcal{C}}] && (3.18)
\end{aligned}$$

From $\hat{\mathcal{C}}$ to $\mathcal{C} = Av(B)$

From the combinatorial specification for the substitution closure of $Av(B)$, using Algorithm 7 described in Section 3.3.2, it is possible to derive an ambiguous system describing $Av(B)$. Note that the non-simple permutations in B are 1243 and 2341. So we compute this system for $\mathcal{C} = \hat{\mathcal{C}}\langle 1243, 2341\rangle$ by adding these two constraints one by one.

Constraint 1243. Three different terms appear in the equations, $12[\hat{\mathcal{C}}^+,\hat{\mathcal{C}}]$, $21[\hat{\mathcal{C}}^-,\hat{\mathcal{C}}]$ and $3142[\hat{\mathcal{C}},\hat{\mathcal{C}},\hat{\mathcal{C}},\hat{\mathcal{C}}]$. To compute $\hat{\mathcal{C}}\langle 1243\rangle$, we propagate the constraint $\langle 1243\rangle$ into each of these 3 different terms using embeddings.

- First consider $21[\hat{\mathcal{C}}^-,\hat{\mathcal{C}}]\langle 1243\rangle$. The different embeddings of $\gamma = 1243$ into $\pi = 21$ are:

		$\gamma_{\alpha(1)}$	$\gamma_{\alpha(2)}$
$\gamma = 1[1243]$	$= \pi\{1243, 0\}$	1243	0
	$= \pi\{0, 1243\}$	0	1243

Thus, using Equation (3.7) of Proposition 3.16 (p.123) we have:

$$21[\hat{\mathcal{C}}^-, \hat{\mathcal{C}}]\langle 1243\rangle = 21[\hat{\mathcal{C}}^-\langle 1243\rangle, \hat{\mathcal{C}}\langle 1243\rangle]$$

Here $|K^\pi_\gamma| = 1$ because $\gamma_{\alpha_i(k_i)} \neq 0$ implies $k_i = i$ for all $i \in \{1, \ell\}$ (here $\ell = 2$).

- Second consider $3142[\hat{\mathcal{C}}, \hat{\mathcal{C}}, \hat{\mathcal{C}}, \hat{\mathcal{C}}]\langle 1243\rangle$. The different embeddings of $\gamma = 1243$ into $\pi = 3142$ are:

		$\gamma_{\alpha(1)}$	$\gamma_{\alpha(2)}$	$\gamma_{\alpha(3)}$	$\gamma_{\alpha(4)}$
$\gamma = 1[1243]$	$= \pi\{1243, 0, 0, 0\}$	1243	0	0	0
	$= \pi\{0, 1243, 0, 0\}$	0	1243	0	0
	$= \pi\{0, 0, 1243, 0\}$	0	0	1243	0
	$= \pi\{0, 0, 0, 1243\}$	0	0	0	1243
$\gamma = 12[12, 21]$	$= \pi\{12, 0, 21, 0\}$	12	0	21	0
	$= \pi\{0, 12, 21, 0\}$	0	12	21	0
	$= \pi\{0, 12, 0, 21\}$	0	12	0	21
$\gamma = 12[1, 132]$	$= \pi\{1, 0, 132, 0\}$	1	0	132	0
	$= \pi\{0, 1, 132, 0\}$	0	1	132	0
	$= \pi\{0, 1, 0, 132\}$	0	1	0	132
$\gamma = 132[12, 1, 1]$	$= \pi\{0, 12, 1, 1\}$	0	12	1	1

In order to avoid pattern 1243, a permutation that decomposes as $3142[\mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}]$ must invalidate each embedding, i.e. each line of the above array. Thus following Equation (3.7), we obtain (after simplifications such as $\hat{\mathcal{C}}\langle 1243, 12\rangle = \hat{\mathcal{C}}\langle 12\rangle$):

$$\begin{aligned}3142[\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C}]\langle 1243\rangle = {}& 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle, \hat{\mathcal{C}}\langle 21\rangle]\\ &\cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 21\rangle]\\ &\cup 3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle, \hat{\mathcal{C}}\langle 21\rangle]\\ &\cup 3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 21\rangle]\end{aligned}$$

Here $|K^\pi_\gamma| = 8$: there are 2 choices for k_i for $i = 5, 6, 7$ and no choice for the other k_i since we want $\gamma_{\alpha_i(k_i)} \neq 0$ and $\gamma_{\alpha_i(k_i)} \neq 1$.
After deleting redundant constraints, i.e. those that describe a set strictly included in another one, the remaining terms are:

$$3142[\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C}]\langle 1243\rangle = 3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle, \hat{\mathcal{C}}\langle 132\rangle]$$

- Finally consider $12[\hat{\mathcal{C}}^+, \hat{\mathcal{C}}]\langle 1243\rangle$. The embeddings of $\gamma = 1243$ into $\pi = 12$ are:

		$\gamma_{\alpha(1)}$	$\gamma_{\alpha(2)}$
$\gamma = 1[1243]$	$= \pi\{1243, 0\}$	1243	0
	$= \pi\{0, 1243\}$	0	1243
$\gamma = 12[12, 21]$	$= \pi\{12, 21\}$	12	21
$\gamma = 12[1, 132]$	$= \pi\{1, 132\}$	1	132

Hence, Equation (3.7) gives after simplifications:

$$12[\hat{\mathcal{C}}^+,\hat{\mathcal{C}}]\langle 1243\rangle = 12[\hat{\mathcal{C}}^+\langle 12\rangle,\hat{\mathcal{C}}\langle 132\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 1243\rangle,\hat{\mathcal{C}}\langle 21\rangle]$$

At last the system of Equations (3.16), (3.17) and (3.18) becomes:

$$\begin{aligned}
\hat{\mathcal{C}}\langle 1243\rangle &= 1 \;\cup\; 12[\hat{\mathcal{C}}^+\langle 12\rangle,\hat{\mathcal{C}}\langle 132\rangle] \;\cup\; 12[\hat{\mathcal{C}}^+\langle 1243\rangle,\hat{\mathcal{C}}\langle 21\rangle] \;\cup\; 21[\hat{\mathcal{C}}^-\langle 1243\rangle,\hat{\mathcal{C}}\langle 1243\rangle] \\
&\cup 3142[\hat{\mathcal{C}}\langle 1243\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 21\rangle,\hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 132\rangle,\hat{\mathcal{C}}\langle 132\rangle] \quad (3.19)\\
\hat{\mathcal{C}}^+\langle 1243\rangle &= 1 \;\cup\; 21[\hat{\mathcal{C}}^-\langle 1243\rangle,\hat{\mathcal{C}}\langle 1243\rangle] \;\cup\; 3142[\hat{\mathcal{C}}\langle 1243\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 21\rangle,\hat{\mathcal{C}}\langle 132\rangle] \\
&\cup 3142[\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 132\rangle,\hat{\mathcal{C}}\langle 132\rangle] \quad (3.20)\\
\hat{\mathcal{C}}^-\langle 1243\rangle &= 1 \;\cup\; 12[\hat{\mathcal{C}}^+\langle 12\rangle,\hat{\mathcal{C}}\langle 132\rangle] \;\cup\; 12[\hat{\mathcal{C}}^+\langle 1243\rangle,\hat{\mathcal{C}}\langle 21\rangle] \\
&\cup 3142[\hat{\mathcal{C}}\langle 1243\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 21\rangle,\hat{\mathcal{C}}\langle 132\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 132\rangle,\hat{\mathcal{C}}\langle 132\rangle] \quad (3.21)
\end{aligned}$$

Constraint 2341**.** Then we add the next constraint, that is 2341, in the above system as $\hat{\mathcal{C}}\langle 1243\rangle\langle 2341\rangle = \hat{\mathcal{C}}\langle 1243, 2341\rangle$. We propagate the constraint 2341 into the 5 different terms that appear in Equations (3.19) to (3.21).

- Adding constraint 2341 to term $21[\hat{\mathcal{C}}^-\langle 1243\rangle,\hat{\mathcal{C}}\langle 1243\rangle]$.

  The different embeddings of $\gamma = 2341$ into $\pi = 21$ are:

		$\gamma_{\alpha(1)}$	$\gamma_{\alpha(2)}$
$\gamma = 1[2341]$	$= \pi\{2341, 0\}$	2341	0
	$= \pi\{0, 2341\}$	0	2341
$\gamma = 21[123, 1]$	$= \pi\{123, 1\}$	123	1

  Then Equation (3.7) gives after simplifications:

$$21[\hat{\mathcal{C}}^-\langle 1243\rangle,\hat{\mathcal{C}}\langle 1243\rangle]\langle 2341\rangle = 21[\hat{\mathcal{C}}^-\langle 123\rangle,\hat{\mathcal{C}}\langle 1243, 2341\rangle]$$

- Adding constraint 2341 to term $3142[\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 132\rangle,\hat{\mathcal{C}}\langle 132\rangle]$.

  The different embeddings of $\gamma = 2341$ into $\pi = 3142$ are:

		$\gamma_{\alpha(1)}$	$\gamma_{\alpha(2)}$	$\gamma_{\alpha(3)}$	$\gamma_{\alpha(4)}$
$\gamma = 1[2341]$	$= \pi\{2341, 0, 0, 0\}$	2341	0	0	0
	$= \pi\{0, 2341, 0, 0\}$	0	2341	0	0
	$= \pi\{0, 0, 2341, 0\}$	0	0	2341	0
	$= \pi\{0, 0, 0, 2341\}$	0	0	0	2341
$\gamma = 21[123, 1]$	$= \pi\{123, 1, 0, 0\}$	123	1	0	0
	$= \pi\{123, 0, 0, 1\}$	123	0	0	1
	$= \pi\{0, 0, 123, 1\}$	0	0	123	1
$\gamma = 231[1, 12, 1]$	$= \pi\{1, 0, 12, 1\}$	1	0	12	1
$\gamma = 231[12, 1, 1]$	$= \pi\{12, 0, 1, 1\}$	12	0	1	1

  Equation (3.7) gives after simplifications:

$$3142[\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 132\rangle,\hat{\mathcal{C}}\langle 132\rangle]\langle 2341\rangle = 3142[\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 12\rangle,\hat{\mathcal{C}}\langle 132, 2341\rangle]$$

- Adding constraint 2341 to term $3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle]$.

  The possible embeddings of 2341 into 3142 have already been computed in the previous item and Equation (3.7) gives:
  $3142[\hat{\mathcal{C}}\langle 1243\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle, \hat{\mathcal{C}}\langle 132\rangle]\langle 2341\rangle = 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 21, 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle]$

- Adding constraint 2341 to term $12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle]$.

  The different embeddings of $\gamma = 2341$ into $\pi = 12$ are:

		$\gamma_{\alpha(1)}$	$\gamma_{\alpha(2)}$
$\gamma = 1[2341]$	$= \pi\{2341, 0\}$	2341	0
	$= \pi\{0, 2341\}$	0	2341

  Equation (3.7) gives after simplifications:
  $$12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle]\langle 2341\rangle = 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle]$$

- Adding constraint 2341 to term $12[\hat{\mathcal{C}}^+\langle 1243\rangle, \hat{\mathcal{C}}\langle 21\rangle]$.

  The different embeddings have already been computed and Equation (3.7) gives:
  $$12[\hat{\mathcal{C}}^+\langle 1243\rangle, \hat{\mathcal{C}}\langle 21\rangle]\langle 2341\rangle = 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle]$$

At last after deleting redundant constraints, Equations (3.19) becomes:

$$\begin{aligned}\hat{\mathcal{C}}\langle 1243, 2341\rangle &= 1 \cup 21[\hat{\mathcal{C}}^-\langle 123\rangle, \hat{\mathcal{C}}\langle 1243, 2341\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\ &\quad \cup 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle]\end{aligned}$$

Iterating this process for new classes $\hat{\mathcal{C}}\langle E\rangle$ of the right side of the above equation leads to the following ambiguous system:

$$\begin{aligned}\hat{\mathcal{C}}\langle 1243, 2341\rangle &= 1 \cup 21[\hat{\mathcal{C}}^-\langle 123\rangle, \hat{\mathcal{C}}\langle 1243, 2341\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\ &\quad \cup 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\ \hat{\mathcal{C}}\langle 123\rangle &= 1 \cup 21[\hat{\mathcal{C}}^-\langle 123\rangle, \hat{\mathcal{C}}\langle 123\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \\ \hat{\mathcal{C}}\langle 12\rangle &= 1 \cup 21[\hat{\mathcal{C}}^-\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \\ \hat{\mathcal{C}}\langle 132, 2341\rangle &= 1 \cup 21[\hat{\mathcal{C}}^-\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 132, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] \\ \hat{\mathcal{C}}\langle 132, 123\rangle &= 1 \cup 21[\hat{\mathcal{C}}^-\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 123\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 21, 12\rangle] \\ \hat{\mathcal{C}}\langle 21\rangle &= 1 \cup 12[\hat{\mathcal{C}}^+\langle 21\rangle, \hat{\mathcal{C}}\langle 21\rangle] \\ \hat{\mathcal{C}}\langle 21, 12\rangle &= 1\end{aligned}$$

The corresponding equations for $\hat{\mathcal{C}}^+\langle E\rangle$ (resp. $\hat{\mathcal{C}}^-\langle E\rangle$) follows from the one of $\hat{\mathcal{C}}\langle E\rangle$ by deleting terms of the form $12[\dots]$ (resp. $21[\dots]$).

3.5.2 Disambiguation

The preceding equations are unambiguous except equation

$$\begin{aligned}\hat{\mathcal{C}}\langle 1243, 2341\rangle &= 1 \cup 21[\hat{\mathcal{C}}^-\langle 123\rangle, \hat{\mathcal{C}}\langle 1243, 2341\rangle] \cup 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\ &\quad \cup 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle]\end{aligned}$$

for which there exists an ambiguity for the terms whose root is 12, i.e. for the terms $12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle]$ and $12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle]$. Indeed $12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \cap 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] = 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle]$ which is not empty.

To disambiguate those terms we use the techniques explained in Section 3.4.2 and more precisely in Algorithm DisambiguateSystem (Algorithm 8). As we only have to deal with 2 terms, using Equations (3.12) this algorithm rewrite our terms $t_1 = 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle]$ and $t_2 = 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle]$ as $(t_1 \cap \bar{t_2}) \uplus (\bar{t_1} \cap t_2) \uplus (t_1 \cap t_2)$.

Thus we compute the complement of the involved terms using Equation (3.13).

- For $12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle]$ Equation (3.13) gives:
  $12[\hat{\mathcal{C}}^+\langle 2341\rangle(1243), \hat{\mathcal{C}}(21)] \uplus 12[\hat{\mathcal{C}}^+(1243, 2341), \hat{\mathcal{C}}(21)] \uplus 12[\hat{\mathcal{C}}^+\langle 1243\rangle(2341), \hat{\mathcal{C}}(21)] \uplus 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}(21)] \uplus 12[\hat{\mathcal{C}}^+\langle 2341\rangle(1243), \hat{\mathcal{C}}\langle 21\rangle] \uplus 12[\hat{\mathcal{C}}^+(1243, 2341), \hat{\mathcal{C}}\langle 21\rangle] \uplus 12[\hat{\mathcal{C}}^+\langle 1243\rangle(2341), \hat{\mathcal{C}}\langle 21\rangle]$

- For the other term $12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle]$ Equation (3.13) gives:
  $12[\hat{\mathcal{C}}^+(12), \hat{\mathcal{C}}\langle 2341\rangle(132)] \uplus 12[\hat{\mathcal{C}}^+(12), \hat{\mathcal{C}}\langle 132\rangle(2341)] \uplus 12[\hat{\mathcal{C}}^+(12), \hat{\mathcal{C}}(132, 2341)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 2341\rangle(132)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132\rangle(2341)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}(132, 2341)] \uplus 12[\hat{\mathcal{C}}^+(12), \hat{\mathcal{C}}\langle 132, 2341\rangle]$

Thus we can write $12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] \cup 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle]$ as a disjoint union of terms:
$12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle] \uplus 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle(12), \hat{\mathcal{C}}\langle 21\rangle(132)] \uplus 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle(12), \hat{\mathcal{C}}\langle 21\rangle(2341)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle(132)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle(2341)] \uplus 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle(12), \hat{\mathcal{C}}\langle 21\rangle] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle(1243), \hat{\mathcal{C}}\langle 132, 2341\rangle(21)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle(2341), \hat{\mathcal{C}}\langle 132, 2341\rangle(21)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle(21)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle(1243), \hat{\mathcal{C}}\langle 21\rangle] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle(2341), \hat{\mathcal{C}}\langle 21\rangle]$

After eliminating terms that are empty or strictly included in another one, this leads to the unambiguous following system:

$$\begin{aligned}
\hat{\mathcal{C}}\langle 1243, 2341\rangle &= 1 \uplus 21[\hat{\mathcal{C}}^-\langle 123\rangle, \hat{\mathcal{C}}\langle 1243, 2341\rangle] \uplus 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\
&\quad \uplus 12[\hat{\mathcal{C}}^+\langle 1243, 2341\rangle(12), \hat{\mathcal{C}}\langle 21\rangle] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle(21)] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle] \\
\hat{\mathcal{C}}\langle 123\rangle &= 1 \uplus 21[\hat{\mathcal{C}}^-\langle 123\rangle, \hat{\mathcal{C}}\langle 123\rangle] \uplus 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \\
\hat{\mathcal{C}}\langle 12\rangle &= 1 \uplus 21[\hat{\mathcal{C}}^-\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \\
\hat{\mathcal{C}}\langle 132, 2341\rangle &= 1 \uplus 21[\hat{\mathcal{C}}^-\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \uplus 12[\hat{\mathcal{C}}^+\langle 132, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] \\
\hat{\mathcal{C}}\langle 132, 123\rangle &= 1 \uplus 21[\hat{\mathcal{C}}^-\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 123\rangle] \uplus 12[\hat{\mathcal{C}}^+\langle 12\rangle, \hat{\mathcal{C}}\langle 21, 12\rangle] \\
\hat{\mathcal{C}}\langle 21\rangle &= 1 \uplus 12[\hat{\mathcal{C}}^+\langle 21\rangle, \hat{\mathcal{C}}\langle 21\rangle] \\
\hat{\mathcal{C}}\langle 21, 12\rangle &= 1
\end{aligned}$$

As noticed in Section 3.4.2, new terms appear on the right side of these equations like $\hat{\mathcal{C}}\langle 132, 2341\rangle(21)$. But these terms are not defined by our system, so we have to compute a description for these new terms. This is done by iterating Algorithm ComputeEqnFor-Restriction (Algorithm 9) which returns a combinatorial equation.

This algorithm is similar to previous examples except that we have to deal with pattern containment constraints that is we must ensure that permutations contain given patterns. In $\hat{\mathcal{C}}\langle 132, 2341\rangle(21)$, the permutations must contain 21 as a pattern. We explain how the algorithm ComputeEquationForRestriction computes an equation for $\hat{\mathcal{C}}\langle 132, 2341\rangle(21)$:

First, we add constraints 132 and 2341 and compute an equation for $\hat{\mathcal{C}}\langle 132, 2341\rangle$ like before. We obtain the following equation:

$$\hat{\mathcal{C}}\langle 132, 2341\rangle = 1 \;\cup\; 21[\hat{\mathcal{C}}^-\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \;\cup\; 12[\hat{\mathcal{C}}^+\langle 132, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] \tag{3.22}$$

We add the constraint 21 as a mandatory pattern to this equation to compute a recursive description of $\hat{\mathcal{C}}\langle 132, 2341\rangle(21)$. As usual we compute the embeddings of 21 into the different roots. Note that if a tree has root 21 then all permutations $21[X, Y]$ contains pattern 21. So we only have to deal with term $12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle](21)$ and we compute the embeddings of $\gamma = 21$ into $\pi = 12$.

		$\gamma_{\alpha(1)}$	$\gamma_{\alpha(2)}$
$\gamma = 1[21]$	$= \pi\{21, 0\}$	21	0
	$= \pi\{0, 21\}$	0	21

Then, in order to contain 21, a term must satisfy at least one line in the above array. Thus using Equation (3.15) we obtain:

$$12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle](21) = 12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle(21), \hat{\mathcal{C}}\langle 21\rangle] \cup 12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle(21)]$$

Since $\hat{\mathcal{C}}\langle 21\rangle(21) = \emptyset$ we have $12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle](21) = 12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle(21), \hat{\mathcal{C}}\langle 21\rangle]$.

Then Equation (3.22) gives

$$\hat{\mathcal{C}}\langle 132, 2341\rangle(21) = 21[\hat{\mathcal{C}}^{-}\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \;\cup\; 12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle(21), \hat{\mathcal{C}}\langle 21\rangle].$$

We iterate the previous method until each term that appear in the equations is defined by an unambiguous equation, which finally leads to the following unambiguous system:

$$\begin{array}{rcl}
\hat{\mathcal{C}}\langle 1243, 2341\rangle & = & 1 \uplus 12[\hat{\mathcal{C}}^{+}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle(21)] \uplus 12[\hat{\mathcal{C}}^{+}\langle 1243, 2341\rangle(12), \hat{\mathcal{C}}\langle 21\rangle] \\
 & & \uplus 12[\hat{\mathcal{C}}^{+}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle] \uplus 21[\hat{\mathcal{C}}^{-}\langle 123\rangle, \hat{\mathcal{C}}\langle 1243, 2341\rangle] \uplus 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\
\hat{\mathcal{C}}\langle 1243, 2341\rangle(12) & = & 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \uplus 12[\hat{\mathcal{C}}^{+}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle(21)] \\
 & & \uplus 12[\hat{\mathcal{C}}^{+}\langle 12\rangle, \hat{\mathcal{C}}\langle 21\rangle] \uplus 12[\hat{\mathcal{C}}^{+}\langle 1243, 2341\rangle(12), \hat{\mathcal{C}}\langle 21\rangle] \uplus 21[\hat{\mathcal{C}}^{-}\langle 123\rangle(12), \hat{\mathcal{C}}\langle 12\rangle] \\
 & & \uplus 21[\hat{\mathcal{C}}^{-}\langle 12\rangle, \hat{\mathcal{C}}\langle 1243, 2341\rangle(12)] \uplus 21[\hat{\mathcal{C}}^{-}\langle 123\rangle(12), \hat{\mathcal{C}}\langle 1243, 2341\rangle(12)] \\
\hat{\mathcal{C}}^{+}\langle 1243, 2341\rangle(12) & = & 21[\hat{\mathcal{C}}^{-}\langle 123\rangle(12), \hat{\mathcal{C}}\langle 1243, 2341\rangle(12)] \uplus 21[\hat{\mathcal{C}}^{-}\langle 12\rangle, \hat{\mathcal{C}}\langle 1243, 2341\rangle(12)] \\
 & & \uplus 21[\hat{\mathcal{C}}^{-}\langle 123\rangle(12), \hat{\mathcal{C}}\langle 12\rangle] \uplus 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\
\hat{\mathcal{C}}^{-}\langle 123\rangle & = & 1 \uplus 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \uplus 12[\hat{\mathcal{C}}^{+}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \\
\hat{\mathcal{C}}^{-}\langle 123\rangle(12) & = & 3142[\hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \uplus 12[\hat{\mathcal{C}}^{+}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \\
\hat{\mathcal{C}}\langle 12\rangle & = & 1 \uplus 21[\hat{\mathcal{C}}^{-}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \\
\hat{\mathcal{C}}^{+}\langle 12\rangle & = & 1 \uplus 21[\hat{\mathcal{C}}^{-}\langle 12\rangle, \hat{\mathcal{C}}\langle 12\rangle] \\
\hat{\mathcal{C}}^{-}\langle 12\rangle & = & 1 \\
\hat{\mathcal{C}}\langle 132, 2341\rangle & = & 1 \uplus 21[\hat{\mathcal{C}}^{-}\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \uplus 12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle, \hat{\mathcal{C}}\langle 21\rangle] \\
\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle & = & 1 \uplus 21[\hat{\mathcal{C}}^{-}\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\
\hat{\mathcal{C}}\langle 132, 2341\rangle(21) & = & 21[\hat{\mathcal{C}}^{-}\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \uplus 12[\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle(21), \hat{\mathcal{C}}\langle 21\rangle] \\
\hat{\mathcal{C}}^{+}\langle 132, 2341\rangle(21) & = & 21[\hat{\mathcal{C}}^{-}\langle 132, 123\rangle, \hat{\mathcal{C}}\langle 132, 2341\rangle] \\
\hat{\mathcal{C}}^{-}\langle 132, 123\rangle & = & 1 \uplus 12[\hat{\mathcal{C}}^{+}\langle 12\rangle, \hat{\mathcal{C}}\langle 21, 12\rangle] \\
\hat{\mathcal{C}}\langle 21\rangle & = & 1 \uplus 12[\hat{\mathcal{C}}^{+}\langle 21\rangle, \hat{\mathcal{C}}\langle 21\rangle] \\
\hat{\mathcal{C}}^{+}\langle 21\rangle & = & 1 \\
\hat{\mathcal{C}}\langle 21, 12\rangle & = & 1
\end{array}$$

This system can be solved and provides the generating function for $\hat{\mathcal{C}}\langle 1243, 2341\rangle$ which is $Av(2413, 1243, 2341, 531642, 41352)$:

$$C(z) = -\frac{(z^6 - 7z^5 + 20z^4 - 28z^3 + 20z^2 - 7z + 1)z}{(2z^7 - 13z^6 + 37z^5 - 62z^4 + 59z^3 - 32z^2 + 9z - 1)}$$

3.5.3 Experiments

The above system can be used to generate permutations at random using the recursive method or a Boltzmann sampler. For generating random permutations of Figure 3.3, we used the recursive method which can generate in linear time a random permutation of a given size n modulo a preprocessing which roughly corresponds to the calculation of the first n coefficients of the involved generating functions.

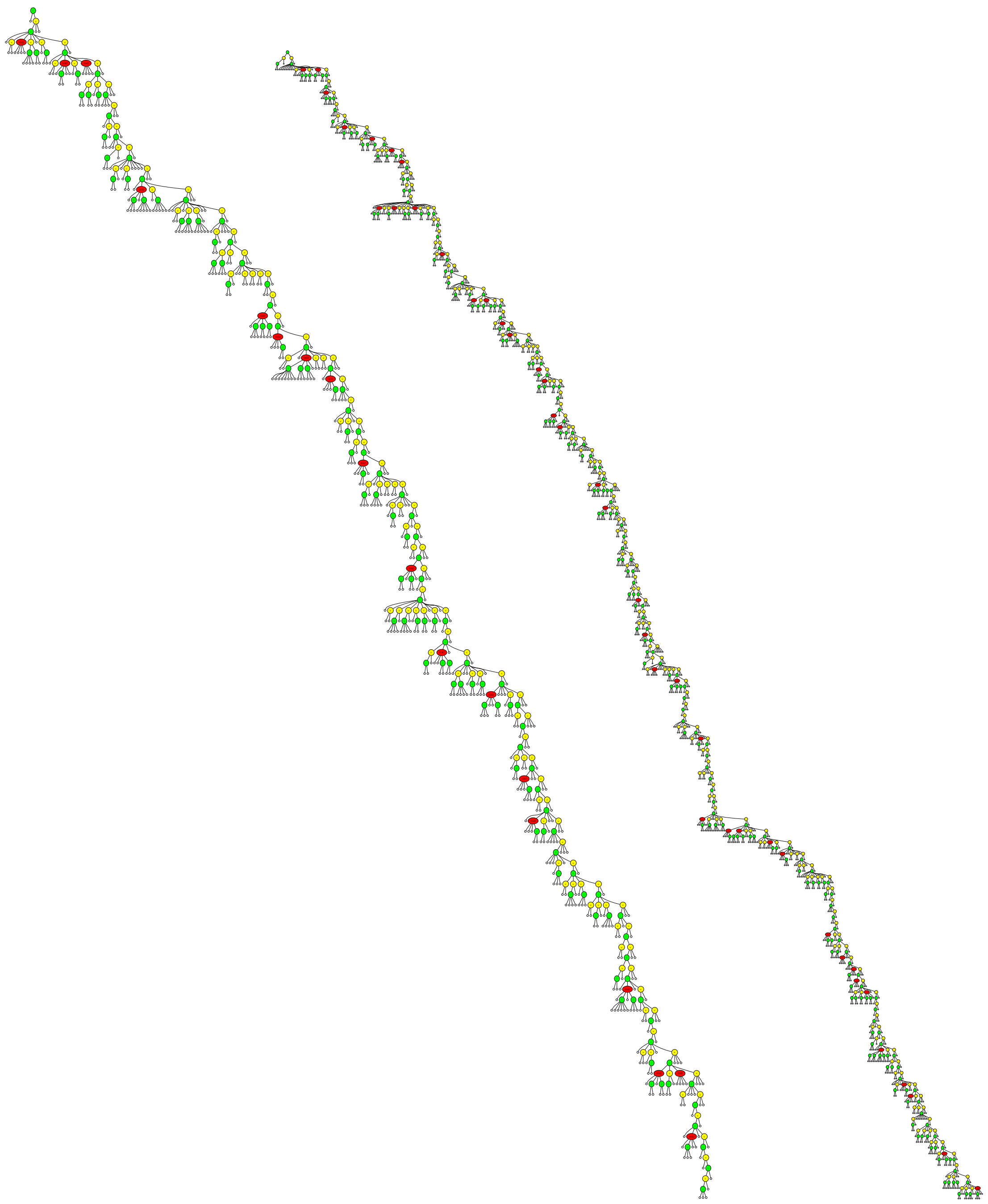


Figure 3.3: Decomposition trees of random permutations of $\mathcal{C}$ of size 500 and 900

3.6 Conclusion and perspectives

We provide an algorithm to compute a combinatorial specification for a permutation class $\mathcal{C} = Av(B)$, when its basis B and the set of its simple permutations are finite and given as input. The complexity of this algorithm is however still to analyze. In particular, we observe a combinatorial explosion of the number of equations in the system obtained, that needs to be quantified.

Combined with the results of the previous chapters, our procedure provides a full algorithmic chain from the basis (when finite) of a permutation class $\mathcal{C}$ to a specification for $\mathcal{C}$ (see Figure 3.4), as explained at the beginning of Part I. This procedure may fail to compute its result, when $\mathcal{C}$ contains an infinite number of simple permutations, this condition being tested algorithmically.

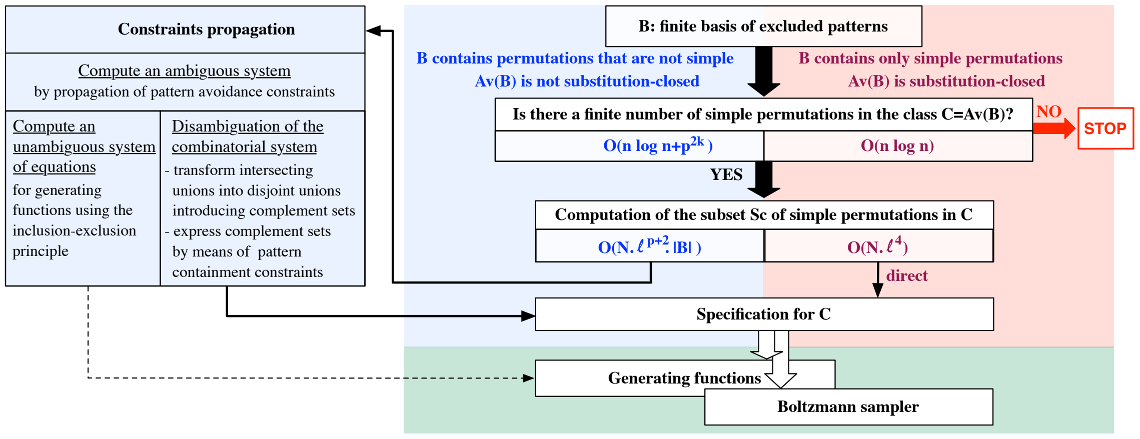

Constraints propagation
Compute an ambiguous system
by propagation of pattern avoidance constraints
Compute an unambiguous system of equations
for generating functions using the inclusion-exclusion principle
Disambiguation of the combinatorial system
- transform intersecting unions into disjoint unions introducing complement sets
- express complement sets by means of pattern containment constraints
B: finite basis of excluded patterns
B contains permutations that are not simple
Av(B) is not substitution-closed
B contains only simple permutations
Av(B) is substitution-closed
Is there a finite number of simple permutations in the class C=Av(B)?
NO
STOP
O(n log n+p^{2k})
O(n log n)
YES
Computation of the subset Sc of simple permutations in C
O(N. ℓ^{p+2}. |B|)
O(N. ℓ^4)
direct
Specification for C
Generating functions
Boltzmann sampler

Figure 3.4: The full algorithmic chain starting from the basis B of a permutation class $\mathcal{C}$, with complexities given w.r.t. $n = \sum_{\beta \in B} |\beta|$, $k = \sharp B$, $p = \max\{|\beta| : \beta \in B\}$, $N = \sharp \mathcal{S}_{\mathcal{C}}$ and $\ell = \max\{|\pi| : \pi \in \mathcal{S}_{\mathcal{C}}\}$ where $\mathcal{S}_{\mathcal{C}}$ is the set of simple permutations of $\mathcal{C}$.

This procedure has two natural algorithmic continuations. First, with the *dictionary* of [FS09], the constructors in the specification of $\mathcal{C}$ can be directly translated into operators on the generating function $C(z)$ of $\mathcal{C}$, turning the specification into a system of (possibly implicit) equations defining $C(z)$. Notice that, using the inclusion-exclusion principle as in [AA05], a system defining $C(z)$ could also be obtained from an *ambiguous* system describing $\mathcal{C}$. Second, the specification can be translated directly into a Boltzmann uniform random sampler of permutations in $\mathcal{C}$, in the same fashion as the above dictionary (see [DFLS04]). This second translation is possible only from an unambiguous system: indeed, whereas adapted when considering enumeration sequences, the inclusion-exclusion principle does not apply when working on the combinatorial objects themselves.

When generating permutations with a Boltzmann sampler, complexity is measured w.r.t. the size of the permutation produced (and is linear if we allow a small variation on the size of the output permutation; quadratic otherwise) and not at all w.r.t. the number of equations in the specification. In our context, this dependency is of course relevant, and opens a new direction in the study of Boltzmann random samplers.

With a complete implementation of the algorithmic chain from B to the specification and of the Boltzmann sampler, one should be able to test conjectures on and study permutation classes. One direction would be to somehow measure the randomness of permutations in a given class, by comparing random permutations with random permutations in a class, or random permutations in two different classes, w.r.t. well-known statistics on permutations. Another perspective would be to use the specifications obtained to compute or estimate the growth rates of permutation classes, to provide improvements on the known bounds on these growth rates. We could also explore the possible use of the computed specifications to provide more efficient algorithms to test membership of a permutation to a class.

However, a weakness of our procedure that we must acknowledge is that it fails to be completely general. Although the method is generic and algorithmic, the classes that are fully handled by the algorithmic process are those containing a finite number of simple permutations. By [AA05], such classes have finite basis (which is a restriction we imposed already), but they also have an *algebraic* generating function. Of course, this is not the case for every permutation class. We may wonder how restrictive this framework is, depending on which problems are studied. First, does it often happen that a permutation class contains finitely many simple permutations? From the work of [Vat10] it seems that a majority of permutations classes are not algebraic, thus do not contain finitely many simple permutations. But to properly express what *often* means, a probability distribution on permutation classes should be defined, which is a direction of research yet to be explored. Second, we may want to describe some problems (maybe like the distribution of some statistics) for which algebraic permutation classes are representative of all permutation classes.

To enlarge the framework of application of our algorithm, we could explore the possibility of extending it to permutation classes that contain an infinite number of simple permutations, but that are finitely described (like the family of oscillations of [BRV08] for instance). With such an improvement, more classes would enter our framework, but it would be hard to leave the algebraic case. This is however a promising direction for the construction of Boltzmann random samplers for such permutation classes.

Part II

Sorting with two stacks in series

Foreword: Stack sorting

Sorting has a central place in computer science theory. It is known that to sort a sequence of n elements in the comparison model, there is no general algorithm running in time less than $\mathcal{O}(n \log n)$. However, we can sort a sequence of n elements with a linear number of comparison w.r.t. n with a single pass through a stack when it avoids the pattern 231.

A stack is a last-in first-out sorting device with push and pop operations (denoted by X and S in Figure 3.5), introduced by Donald Knuth in the volume 1 of *The Art of Computer Programming* [Knu68] in 1968. More precisely in Section 2.2.1 of [Knu68] entitled *Stacks, Queues and Deques* we can find the following definitions:

> A *stack* is a linear list for which all insertions and deletions (and usually all accesses) are made at one end of the list.
>
> A *queue* is a linear list for which all insertions are made at one end of the list; all deletions (and usually all accesses) are made at the other end.
>
> A *deque* ("double-ended queue") is a linear list for which all insertions and deletions (and usually all accesses) are made at the ends of the list.
>
> We also distinguish output-restricted or input-restricted deques, in which deletions or insertions, respectively, are allowed to take place at only one end.

Then in the exercises of this section, Knuth characterized and enumerated the permutations that a stack could generate, taking the identity in input. By composition with the inverse permutation, this is equivalent to characterizing and enumerating the permutations that can be sorted with a stack: we say that a permutation σ is 1-stack-sortable if there is a way to pass σ through a stack to produce the identity permutation $12 \dots n$. More precisely, we consider σ as the sequence of integers $\sigma_1\sigma_2 \dots \sigma_n$ (with $\sigma_i = \sigma(i)$) that we take as input, and at each step two different operations are allowed, as depicted in Figure 3.5:

- either move X: take the next term of the input and push it on top of the stack
- or move S: pop the topmost element of the stack and write it in the output.

Then in volume 3 [Knu73a] Knuth introduced the problem of generating permutations through serial compositions of stacks.

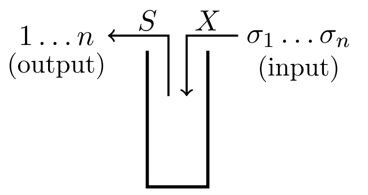


Figure 3.5: Sorting with one stack

Following Knuth, Pratt [Pra73] studied the case of two stacks in parallel and general deques and Tarjan generalized stack-sorting to sorting networks in the article *Sorting Using Networks of Queues and Stacks* [Tar72] published in 1972. Tarjan's model consists of an acyclic directed graph with a designated input node s and output node t and additional nodes representing sorting devices of type Q (queue) or S (stack).

For such a general model there are few results, but he studied particularly serial and parallel compositions. For queues his results were quite complete but for stacks he ended by commenting on the difficulty of these problems.

Since the 60's, numerous variants appeared by either considering other types of combinatorial structures or by changing rules, and many cases were studied in the literature (see [Bón02] for a survey).

In general a given sorting network does not allow one to sort all permutations. A natural question is then which permutations can be sorted. This line of research has led to the study of pattern involvement and permutation classes. Indeed Knuth [Knu68], Tarjan [Tar72] and Pratt [Pra73] noted that the permutations sortable by the various configurations could be described by forbidding certain patterns in the permutations.

For any sorting device, three natural questions among others arise:

1. Decision: what is the complexity of the problem consisting of deciding whether a given permutation is sortable or not?

2. Characterization: can one characterize permutations that are sortable?

3. Counting: how many sortable permutations of size n?

For the one-stack case these three problems were solved by Knuth in [Knu68].

By encoding the push operation by the letter S and the pop operation by the letter X, he showed that sorting processes are in bijection with well-parenthesized words. Then he established that a given permutation cannot be sorted using two different words, showing that the number of stack-sortable permutations of size n is the Catalan number $\frac{1}{n+1}\binom{2n}{n}$.

Finally, he characterized sortable permutations as being permutations avoiding the pattern 231, which is the earliest non-trivial example of a permutation class.

Notice that a greedy algorithm allows one to answer the decision problem in linear time. Indeed given a permutation σ in input, if we want to sort σ we have to perform move S if and only if the top of the stack is the next element to be output (i.e. the smallest element not output yet). Otherwise we do move X. We stop once each element of σ is passed through the stack. Then σ is 1-stack sortable if and only if the output obtained is the identity.

There is another greedy algorithm allowing to answer the decision problem in linear time. Indeed notice that if we want the output to be the identity, it is necessary that at each step elements in the stack are decreasing from bottom to top (this condition is often called Hanoi condition). Thus given a permutation σ in input, if we want to sort σ we have to perform move X if and only if the top of the stack is greater than the next element in the input. Otherwise we do move S. We stop once each element of σ is passed through the stack. Then σ is 1-stack sortable if and only if the output obtained is the identity. This algorithm is called *right-greedy algorithm.*

For a 1-stack sortable permutation σ, both greedy algorithms give the same result; indeed they perform the only way to sort σ. For a permutation which is not 1-stack sortable however these algorithms give different outputs. The result of the right-greedy algorithm applied to σ is denoted $S(\sigma)$ in the literature.

Since these seminal results for a single stack, the more general problem of sorting with multiple stacks in series or in parallel has been widely studied.

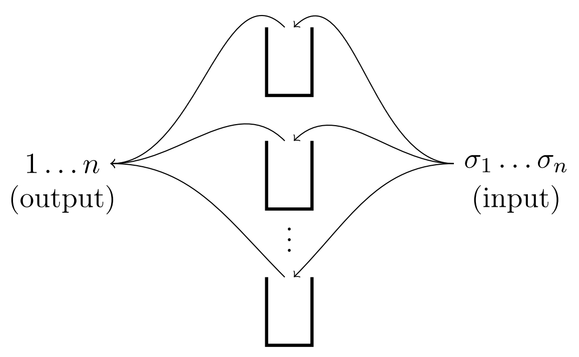


Figure 3.6: Sorting with stacks in parallel.

Regarding t parallel stacks, the decision problem can be answered in time $\mathcal{O}(n \log n)$ for $t = 1, 2, 3$, while for $t > 3$ it is NP-complete (this is proved by a reduction in [EI71] to a problem solved in [Ung92]). The characterization problem is studied in [Pra73]: for $t > 1$, the basis of the class of permutations sortable with t stacks in parallel is infinite. Finally, about the counting problem, when $t = 2$ the generating function is described in [ABM] by a system of equations with an extra variable.

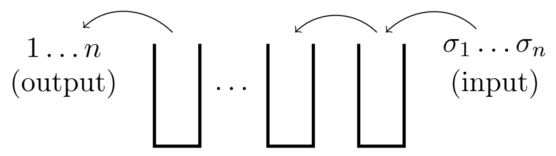


Figure 3.7: Sorting with stacks in series.

For stacks in series, it has been shown in Exercise 5.2.4.19 of [Knu73a] that every permutation of size n can be sorted by $\log_2(n)$ stacks in series. But none of the above three questions has been answered for more than one stack in series. For two stacks, Murphy [Mur02] proved that the basis of the class of sortable permutations is infinite, but the actual set is unknown. In his PhD thesis, he also studies the problem of deciding whether a given permutation is sortable with 2 stacks in series. He reduced this problem to a 3-SAT problem; at the same time he reduced a 2-SAT instance to the decision problem, and hoped than one of both reductions was actually an equivalence. But none of those results has been proved or disproved. In [Bón02], Bóna gives an overview of advances in sorting networks and mentions this problem as possibly NP-complete. More recently, this problem is also mentioned as possibly NP-complete in [AAL10]. More surprising, both conjectures exist: in [AMR02], the authors conjectured that the decision problem is NP-complete, while Murphy in his PhD ([Mur02] Conjecture 260) conjectured that it is in the complexity class P.

To make further progress on stacks in series, several weaker variants have been studied. First, West considered permutations sortable with two consecutive right greedy passes through a stack in [Wes90, Wes93]. Those permutations, called West-2-stack sortable, are a strict subset of general 2-stack sortable permutations. However in the literature they are often referred to simply as 2-stack sortable permutations. Be careful that this is not the case in this thesis: we use 2-stack sortable only for the unrestricted case. West-2-stack sortable permutations do not form a permutation class, since 35241 is West-2-stack sortable whereas its pattern 3241 is not. However these permutations can be described using an extended notion of pattern. West conjectured the enumeration formula which was proved after by Zeilberger [Zei92]: there are $\frac{2(3n)!}{(n+1)!(2n+1)!}$ West-2-stack sortable permutations of size n.

More generally for any integer t a permutation is West-t-stack sortable if $S^t(\sigma)$ is the identity, i.e. if σ is sorted after t right greedy passes through a stack. For $t > 2$, few results are known [Úlf11].

Another variant studied in [AMR02] is to consider decreasing stacks instead of general stacks: in this model the stacks have to respect the Hanoi condition (i.e. elements in the stacks must be decreasing from bottom to top). This variant is less restricted than the one of West but there are still some permutations sortable with general stacks in series that cannot be sorted with this model.

For two decreasing stacks in series, the three above problems are solved in [AMR02]: a left greedy algorithm allows one to decide in linear time whether a permutation is sortable with two decreasing stacks in series. Moreover permutations sortable with this restriction form a permutation class whose basis is infinite but explicit. Finally the generating function of these permutations is known.

In this part of the thesis, we define a new restriction of 2-stack sorting, namely 2-stacks pushall sorting, and prove that the decision problem in this case is polynomial. Moreover we give a quadratic algorithm computing an encoding of all pushall sortings of a given permutation taken in input. Finally we show that this variant is closely linked with general 2-stack sorting and we deduce a polynomial algorithm deciding whether a permutation given in input is sortable with two general stacks in series, settling a long-standing open problem.

Chapter 4

Pushall sorting: a new notion closely linked with general sorting

In this chapter we introduce 2-stack pushall permutations which form a subclass of 2-stack sortable permutations and we show that these two classes are closely related. We study the two corresponding bases, which are infinite, and characterize permutations of the bases whose decomposition tree has a linear root. Moreover, we characterize every possible pushall sorting of a permutation by means of a bicoloring of the permutation. Finally, we give an optimal $\mathcal{O}(n^2)$ algorithm which decides whether a given permutation of size n is 2-stack pushall sortable and which describes all its pushall sortings.

4.1 Introduction

While the case of a single stack is fairly simple, once one uses many stacks, be they in parallel or in series, studying stack sorting becomes very difficult.

Here we focus on sorting with two stacks in series. There are few general results: We know that permutations sortable with two stacks in series form a class, that we call 2-stack sortable permutations. Murphy [Mur02] proved that its basis is infinite. This basis is still unknown. Albert, Atkinson and Linton worked on the growth rate of 2-stack sortable permutations and bounded it between 8.156 and 13.374 (see [AAL10]).

Because general stack sorting has proven to be so complex to work with, a classical strategy emerged over the last 40 years: define simplifications and restrictions of the problem, in the hope that they will evolve into insight of their own, or into attacks of the main problem.

Here, we define a new restriction called pushall sorting which allows to prove that deciding whether a permutation is 2-stack sortable is polynomial.

A permutation is 2-stack *pushall* sortable if it is sortable with two stacks in series by a sorting process in two parts, the first part consisting only in putting all the elements in the stacks (only operations ρ and λ defined p.149 are allowed: operation μ is forbidden), and the second part consisting only in popping out the elements in increasing order (only operations λ and μ are allowed: operation ρ is forbidden).

The origin of this idea and the reason why it is closely linked with general sorting with two stacks in series is that a permutation that ends with 1 is 2-stack sortable if and only if it is 2-stack pushall sortable. More precisely, a permutation σ is 2-stack pushall sortable if and only if $\ominus[\sigma, 1]$ is 2-stack sortable. Solving pushall sorting is thus a prerequisite for solving general sorting.

The key idea to study this new kind of sorting is that a pushall sorting process can be viewed as two steps of sorting with a single stack. Indeed the first part of the procedure consists in doing only two operations (ρ and λ) thus it can be viewed as sorting with one stack (the right one H) considering the left stack V as the output, and the second part of the process consists in doing only two operations (λ and μ) to pop out elements in increasing order thus it can be viewed as sorting with one stack (the left one V) considering the right stack H as the input.

When sorting with a single stack with an input and an output fixed, all operations are determined. Thus in a pushall sorting process of a given permutation, the stack configuration at the end of the first part of the process encodes the whole process. We characterize such stack configurations, which in particular contain all the elements of the input permutation, and we show that they are in one-to-one correspondence with some bicolorings of the diagram of the permutation that we call valid colorings and that are defined thanks to forbidden colored patterns.

Then to obtain a $\mathcal{O}(n^2)$ algorithm computing all the valid colorings for a permutation of size n, we first show that we can test in linear time whether a given bicoloring is valid, and then that it is sufficient to check a linear number of bicolorings among the 2^n possibilities. To restrict the number of bicolorings to test, we perform a detailed study of valid colorings, showing that the color of some particular increasing sequences fixes the color of every other point.

This chapter is organized as follows. In Section 4.2 we define 2-stack pushall sorting and show the close correlation with general 2-stack sorting by studying stack words and stack configurations in these two models. This combinatorial study concludes on some partial characterization of both classes in terms of permutations they contain or permutations

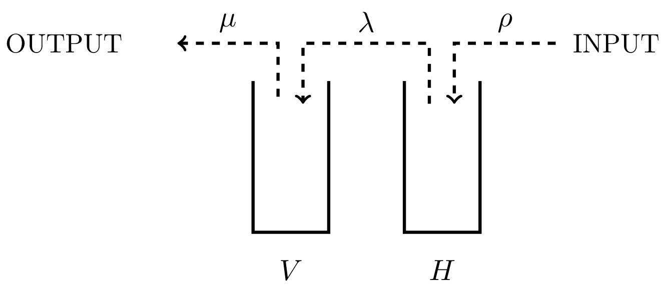


Figure 4.1: Sorting with two stacks in series.

in the basis. The key idea is to use the block-decomposition of permutations given in Theorem 0.27.

Then in Section 4.3 we prove that 2-stack pushall sorting can be expressed as a 2-color problem on the diagram of permutations. Moreover we characterize valid colorings of diagrams of permutations by studying the color of increasing sequences. This characterization leads to a polynomial algorithm to check whether a permutation is 2-stack pushall sortable by finding all valid colorings for its diagram.

Section 4.4 refines the results of Section 4.3 by limiting the number of colorings to test. This leads to an optimal algorithm computing in quadratic time an encoding of all pushall sortings of a given permutation, which thus decides whether a permutation is 2-stack pushall sortable.

4.2 Pushall sorting vs. 2-stack sorting

In this section we define pushall sorting and point out the close link between 2-stack sorting and 2-stack pushall sorting. Moreover, for each of these sorting problems we exhibit some recursive necessary and sufficient conditions for a permutation σ to be sortable when σ is $\ominus$-decomposable and when σ is $\oplus$-decomposable (see Definition 0.26 p.25). We conclude the section by studying the basis of 2-stack sortable permutations and the basis of pushall sortable permutations.

4.2.1 A word approach

First recall formally the problem of sorting with two stacks in series. Given two stacks H and V in series –as shown in Figure 4.1– and a permutation σ, we want to sort elements of σ using the stacks. We consider σ as the sequence of integers $\sigma = \sigma_1\sigma_2\dots\sigma_n$ with $\sigma_i = \sigma(i)$ and take it as input.

Then at each step we have three possible operations as described in Figure 4.1:

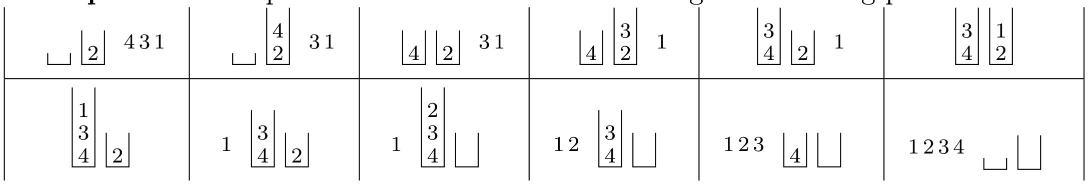
ρ: Take the next element of σ still in the input and push its value on top of the first stack denoted H.

λ: Pop the topmost element of stack H and push it on top of the second stack V.

μ: Pop the topmost element of stack V and write it to the output.

We iterate over these three possibilities until all elements have been output. If there exists a sequence of operations that leads to the identity on the output, then we say that the permutation is 2-stack sortable.

Example 4.1. The permutation 2431 is sortable using the following process:



Not all permutations are 2-stack sortable: the smallest non-sortable ones are of size 7, for instance $\sigma = 2435761$.

In a sorting process with two stacks in series, each operation performed can be encoded with a letter. For example, whenever an element is popped from stack H and pushed in stack V, we write λ (see Figure 4.1). Thus a sequence of operations is encoded by a word on the alphabet $\{\rho, \lambda, \mu\}$ whose length is the number of operations performed. However not all words on the alphabet $\{\rho, \lambda, \mu\}$ describe sorting processes. For example a word encoding a sorting process has to begin with letter ρ. Moreover a word encoding a sorting process of a permutation of size n has $3n$ letters: n times each letter ρ, λ, μ. Words on the alphabet $\{\rho, \lambda, \mu\}$ encoding a sorting process are called stack words and are formally defined below:

Definition 4.2. Let $\alpha \in \{\rho, \lambda, \mu\}$ and w a word on the alphabet $\{\rho, \lambda, \mu\}$. Then $|w|_\alpha$ denotes the number of letters α in w.

A *stack word* w is a word over the alphabet $\{\rho, \lambda, \mu\}$ such that $|w|_\rho = |w|_\lambda = |w|_\mu$ and for all prefix v of w, $|w|_\rho \geq |w|_\lambda \geq |w|_\mu$.

Intuitively a stack word is a word which describes a sequence of appropriate stack operations which take a permutation through two stacks in series (without necessarily sorting it).

Definition 4.3. A permutation σ is 2-stack sortable if there exists a stack word encoding a sorting process which leads to the identity in the output with σ as input. Such a word is called a *sorting word* for σ.

Example 4.4. The permutation 2431 is 2-stack sortable and $\rho\rho\lambda\rho\lambda\rho\lambda\mu\lambda\mu\mu\mu$ is a sorting word of 2431 encoding the sorting process given in Example 4.1.

In general there are several sorting words for a given permutation. For example, the permutation 2431 admits either $\rho\rho\lambda\rho\lambda\rho\lambda\mu\lambda\mu\mu\mu$ or $\rho\lambda\rho\rho\rho\lambda\mu\mu\lambda\mu\lambda\mu$ as sorting words. Note also that ρ and μ commute: if w is a sorting word for σ and w' is obtained from w by exchanging adjacent letters ρ and μ, then w' is a sorting word for σ. In his PhD [Mur02], Murphy studied 2-stack sorting by studying stack words. This presentation of 2-stack sorting allows us to define formally 2-stack pushall sorting.

Definition 4.5. A *pushall* stack word is a stack word such that the first occurrence of μ is after the last occurrence of ρ. A permutation σ of size n is 2-stack pushall sortable if and only it admits a pushall sorting word.

More informally, 2-stack pushall sortable permutations are those which can be sorted by pushing all elements in the stacks before writing any element to the output.

For example 2431 is 2-stack pushall sortable as the word $\rho\rho\lambda\rho\lambda\rho\lambda\mu\lambda\mu\mu\mu$ respects the required condition (as does $\rho\lambda\rho\rho\rho\lambda\mu\mu\lambda\mu\lambda\mu$).

Remark 4.6. A stack word w is a pushall stack word if and only if it can be written as $w = uv$ with $u \in \{\rho, \lambda\}^*$ and $v \in \{\lambda, \mu\}^*$. This decomposition is not unique. In the preceding example, the word $w = \rho\rho\lambda\rho\lambda\rho\lambda\mu\lambda\mu\mu\mu$ admits two decompositions: $w = (\rho\rho\lambda\rho\lambda\rho)(\lambda\mu\lambda\mu\mu\mu)$ and $w = (\rho\rho\lambda\rho\lambda\rho\lambda)(\mu\lambda\mu\mu\mu)$.

The previous definition of 2-stack pushall sortable permutations implies that they form a subset of 2-stack sortable permutations. Moreover it is easy to check that 2-stack pushall sorting is stable by pattern relation: if σ is 2-stack pushall sortable then every pattern π of σ is 2-stack pushall sortable. Indeed choose an occurrence of π in σ and a sorting word w of σ. To obtain a pushall sorting word of π, delete the letters of w that correspond to elements of σ not involved in the occurrence of π. The same reasoning holds for general 2-stack sorting.

Proposition 4.7. *The set of* 2*-stack pushall permutations forms a subclass of the* 2*-stack sortable permutation class.*

We do not know how big this subclass is, nor whether or not it is negligible with respect to the whole class. However there exists a close correlation between these two classes and solving 2-stack pushall sorting is a prerequisite for the more general case. We first study the possible configurations of the stacks during a sorting process. This will help us to obtain properties of stack sortable permutations thanks to their decomposition. In a last subsection, we study the basis of the 2-stack sortable permutation class and show how it is correlated to the 2-stack pushall one.

4.2.2 Stack configurations

At each step of a sorting process, some elements of the permutation lie in the stacks (or maybe none). We call a *stack configuration* the position of these elements in stacks H and V. In this section, we exhibit in Theorem 4.13 a necessary condition on stack configurations to be part of a sorting process. Then we give a one-to-one correspondence between pushall sorting processes and some particular stack configurations in Theorem 4.19. First we define formally stack configurations:

Definition 4.8. A *stack configuration* is a pair of two vectors of positive integers $(\overrightarrow{V}, \overrightarrow{H})$ of arbitrary (and maybe different) sizes, such that all coordinates are distinct. A stack configuration may be empty (if both vectors are of size zero). Vector $\overrightarrow{V}$ (resp. $\overrightarrow{H}$) represents elements that are in stack V (resp. H) given from bottom to top, so we can apply to stack configurations operations λ and μ, and also operation ρ if we know what is the next integer in the input.

A stack configuration is *poppable* if all elements in stacks H and V can be popped out in increasing order (using operations λ and μ).

Let σ be a permutation, a stack configuration of σ is a stack configuration in which coordinates are bounded by $|\sigma|$.

Definition 4.9. To any stack word w of size $3n$ and permutation σ of size n we associate the sequence of $3n+1$ stack configurations $\big(c_k(w,\sigma)\big)_{k\geq 0}$ describing how the sequence of operations $w = w_1 \dots w_{3n}$ takes σ through the stacks: $c_0(w,\sigma)$ is empty and we obtain $c_{k+1}(w,\sigma)$ from $c_k(w,\sigma)$ by doing operation w_k with σ as input at the beginning.

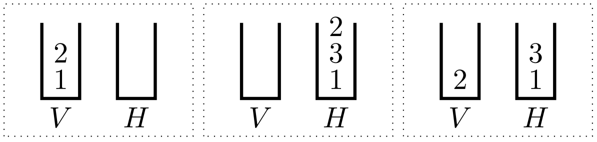
Definition 4.10. Let σ be a permutation. A stack configuration c is *reachable* for σ if there exists a stack word w and an integer k such that $c = c_k(w,\sigma)$. A stack configuration c is *total* for σ if all integers from 1 to $|\sigma|$ appear in c (this notion depends only on $|\sigma|$, we don't need c to be reachable for σ).

Intuitively, a stack configuration is reachable for σ if taking σ as input, there exists a sequence of operations ρ, λ, μ leading to this configuration. Note that in any reachable configuration, elements of H are in increasing order of indices from bottom to top. Another important remark is that:

Remark 4.11. Let w be a stack word of size $3n$ and σ a permutation of size n. Then w is a pushall stack word if and only if at least one of the stack configurations $\big(c_k(w,\sigma)\big)_{k\geq 0}$ is total.

During a sorting process, stack configurations must be poppable, which implies some constraints. Recall that in one-stack sorting, the stack must be in decreasing order (from bottom to top). For two-stack sorting, we have the same decreasing constraint on stack V but other constraints appear that can be represented as stack patterns.

Definition 4.12. The *unsortable stack-patterns* are the following three patterns, denoted respectively $|12|\,|$, $|\,|132|$ and $|2|13|$:



More precisely pattern $|12|\,|$ means that there is in stack V one element which has a smaller element below it. Pattern $|\,|132|$ means that there is in stack H one element which has a greater element below it and a smaller element further below. Pattern $|2|13|$ is somehow special as the pattern is divided in both stacks. It means that there are elements a, b, c such that $b \in V$, $a, c \in H$, $a < b < c$ and c is above a in stack H.

Theorem 4.13. *A stack configuration is poppable if and only if it avoids each unsortable stack-pattern.*

Proof. Notice that if a stack configuration contains any of the 3 unsortable stack-patterns, then elements involved in the pattern cannot be popped out in increasing order.

Conversely, we prove by induction on the number of elements in the stacks that a configuration which avoids the 3 unsortable stack-patterns is poppable. Suppose that the result has been proved for all stack configurations with k elements. Note that the result is trivially true for $k \leq 2$. Let c be a stack configuration with $k+1$ elements avoiding the 3 unsortable stack-patterns and m the smallest element of this configuration. We show that m can be popped out so that the stack configuration of the k remaining elements still avoids the 3 unsortable stack-patterns. Without loss of generality assume $m = 1$.

Suppose that 1 lies in stack V. As c avoids pattern $|12|\,|$, V is in decreasing order so 1 is at the top of it. It can be popped out and there remain k elements still avoiding the 3 unsortable stack-patterns. Thus they can be all popped out in increasing order by induction.

Suppose now that 1 lies in stack H. As c avoids pattern $|\,|132|$ and 1 is the smallest element, all elements above 1 are in increasing order (from 1 to top). All these elements can be pushed onto stack V so that stack V remains in decreasing order. Indeed as c avoids pattern $|2|13|$, the top of stack V is greater than the top of stack H. When all elements above 1 in stack H are transferred onto stack V, then 1 can be popped out both stacks H and V and the remaining configuration still avoids the 3 unsortable stack-patterns (as c avoids pattern $|\,|132|$, no pattern $|2|13|$ has been created) and we can apply the induction. ■

Remark 4.14. There is at most one way to pop out in increasing order elements from a stack configuration. Indeed to pop out we only use operations μ and λ, and if we want to pop out in increasing order we have to perform operation μ if and only if the smallest element lies in the top of V.

Algorithm 10: Pop out in increasing order

```
Data: $\sigma$ a permutation and $c$ a total stack configuration of $\sigma$.
Result: True if $c$ is poppable.
$i \longleftarrow 1$;
while $i \leq |\sigma|$ do
    if $top(V) = i$ then
        pop out $top(V)$ from stack $V$ and let $i \longleftarrow i + 1$
    else
        if $H$ is non-empty then
            pop $top(H)$ from stack $H$ and push it into $V$;
        else
            Return false;
        end
    end
end
Return true;
```

Proposition 4.15. *Let c be a total stack configuration of a permutation σ. Then Algorithm 10 applied to c returns true if and only if c is poppable. Moreover Algorithm 10 runs in linear time w.r.t. $|\sigma|$.*

Proof: At each step, Algorithm 10 performs either an operation μ or an operation λ. As at most $|\sigma|$ operations μ and $|\sigma|$ operations λ can be done, it runs in linear time w.r.t. $|\sigma|$. We conclude using Remark 4.14. ■

Theorem 4.13 gives conditions for a stack configuration to be poppable. Conditions of this theorem must be verified at each step of a sorting process. This is formalized in the following proposition:

Proposition 4.16. *If w is a sorting word for the permutation σ, then each stack configuration of $\big(c_k(w,\sigma)\big)_{k\geq 0}$ avoids the 3 unsortable stack-patterns.*

The converse is not true: let $w = (\rho\lambda\mu)^n$ then for any permutation σ of size n, each stack configuration of $\big(c_k(w,\sigma)\big)_{k\geq 0}$ avoids the 3 unsortable stack-patterns (as there is at most one element in the stacks); but if σ is not the identity, w is not a sorting word for σ.

For 2-stack pushall sorting, however, it is sufficient to check whether the stack configuration obtained just after the last element of σ has been pushed onto H is total and avoids the 3 unsortable stack-patterns.

Theorem 4.17. *A permutation σ is 2-stack pushall sortable if and only if there exists a poppable reachable total stack configuration of σ, i.e. if and only if there is a way to put all its elements in the stacks so that the total stack configuration obtained avoids the three unsortable stack-patterns.*

Proof: If σ is 2-stack pushall sortable, consider a pushall sorting process of σ. Then the stack configuration obtained just after the last element of σ has been pushed onto H is a poppable reachable total stack configuration of σ.

Conversely let c be a poppable reachable total stack configuration of σ. Then by definition there is a way to reach c starting with σ as input, and then elements of c can be popped out in increasing order, leading to a pushall sorting process of σ.

We conclude using Theorem 4.13. ■

Poppable reachable total stack configurations are not only a witness for a permutation to be 2-stack pushall sortable, but they encode pushall sorting processes:

Proposition 4.18. *Let c be a poppable reachable total stack configuration of a permutation σ. Then there exists a unique pushall stack word w and a unique integer k such that $c = c_k(w, \sigma)$. Moreover c characterizes w : knowing only c and σ, we can compute w in linear time.*

Proof. Since c is poppable, from Remark 4.14, there is a unique way to pop out elements of c in increasing order, which is computed in linear time by Algorithm 10. Similarly, starting with σ as input there is a unique way to obtain configuration c, which can be computed in linear time. Indeed as c is total we only use operations ρ and λ, and if we want to obtain c we have to perform operation λ if and only if the the top of H is the next element that is in V in configuration c. ■

More informally, a pushall sorting process can be viewed as two steps of sorting with only one stack. Indeed the first part of the procedure consists in doing only operations ρ and λ thus it can be viewed as sorting with one stack (the right one H) considering the left stack V as the output and the second part of the process consists in doing only operations λ and μ to pop out elements in increasing order thus it can be viewed as sorting with one stack (the left one V) considering the right stack H as the input.

Since when there is only one stack, knowing only the input and the output all operations are determined, then in pushall sorting, knowing the input and only one total stack configuration of the pushall sorting process, all operations are determined.

This encoding of pushall sorting processes by poppable reachable total stack configurations allows to build some bijections:

Theorem 4.19. *Let σ be a permutation. Then there is a one-to-one correspondence between:*

- *Pushall sorting processes of σ*
- *Pushall stack words of σ*
- *Poppable reachable total stack configurations of σ where σ_n is at the top of stack H*
- *Poppable reachable total stack configurations of σ where 1 is at the top of stack V*

Proof: The bijection between sorting processes and stack words is clear. We establish the two bijections between sorting processes and stack configurations.

During a pushall sorting process of σ, there is a unique total stack configuration such that σ_n is at the top of stack H: the one just after σ_n is pushed into H. Indeed if there are two stack configurations at times t and t' such that σ_n is at the top of stack H, then since these configurations are different at least one operation is performed between time t and time t', but only operation μ can be done so that σ_n stays at the top of stack H. Thus at least one operation μ is performed and one of these configurations is not total.

We show similarly with operation ρ instead of operation μ that during a pushall sorting process of σ, there is a unique total stack configuration such that 1 is at the top of stack V: the one just before 1 is popped out of V.

Conversely from Proposition 4.18, to any poppable reachable total stack configuration of σ corresponds exactly one pushall sorting process of σ. ■

4.2.3 Decomposition and stack sorting

In this subsection, we exhibit necessary and sufficient conditions for a permutation which is $\ominus$ or $\oplus$-decomposable to be 2-stack sortable (resp. 2-stack pushall sortable).

$\ominus$-decomposable permutations:

Proposition 4.20. *A permutation $\sigma = \ominus[\pi^{(1)}, \pi^{(2)}, \dots, \pi^{(k)}]$ is 2-stack sortable if and only if every $\pi^{(i)}$ for $i \in \{1 \dots k-1\}$ is 2-stack pushall sortable and $\pi^{(k)}$ is 2-stack sortable.*

Proof. Suppose that σ is 2-stack sortable. Let w_σ be a sorting word of σ. For $i \in \{1 \dots k\}$, consider the subword $w_{\pi^{(i)}}$ of w_σ by taking letters corresponding to an element of $\pi^{(i)}$. This word is of size $3|\pi^{(i)}|$ and has equal number of occurrences of the letters ρ, λ, μ. Moreover, it is a sorting word for $\pi^{(i)}$ as the relative order of elements of $\pi^{(i)}$ under the action of $w_{\pi^{(i)}}$ will be the same as the action of w_σ on σ. Furthermore, as the element 1 in σ belongs to the last block $\pi^{(k)}$, all elements of $\pi^{(i)}$ are pushed into the stacks before the first pop. Hence $\pi^{(i)}$ is 2-stack pushall sortable. Finally, 2-stack sortable permutations form a permutation class, so that $\pi^{(k)}$ must be 2-stack sortable.

Conversely, if every $\pi^{(i)}$ for $i \in \{1 \dots k-1\}$ is 2-stack pushall sortable and $\pi^{(k)}$ is 2-stack sortable, let w_i $(1 \le i \le k-1)$ be a pushall stack word for $\pi^{(i)}$ and w_k be a stack word for $\pi^{(k)}$. Then each w_i $(1 \le i \le k-1)$ can be written as $w'_i w''_i$ where w'_i contains no occurrence of μ and w''_i no occurrence of ρ. It is easy to check that the word $w'_1 w'_2 \dots w'_{k-1} w_k w''_{k-1} w''_{k-2} \dots w''_1$ is a sorting word for σ, hence σ is 2-stack sortable. ■

With a similar proof, we have the following result when restricting to 2-stack pushall sortable permutations:

Proposition 4.21. *A permutation $\sigma = \ominus[\pi^{(1)}, \pi^{(2)}, \dots, \pi^{(k)}]$ is 2-stack pushall sortable if and only if every $\pi^{(i)}$ for $i \in \{1 \dots k\}$ is 2-stack pushall sortable.*

$\oplus$-decomposable permutations: For general sorting, the case where σ is $\oplus$-decomposable is easy to deal with as each block of the decomposition can be popped out as soon as it is pushed into the stacks. So the only condition is given in the following proposition:

Proposition 4.22. *If $\sigma = \oplus[\pi^{(1)}, \dots, \pi^{(k)}]$ then σ is 2-stack sortable if and only if each $\pi^{(i)}$ is 2-stack sortable.*

For pushall sorting, $\oplus$-decomposable permutations are harder to handle. As no element can be popped out before all elements have been pushed, the element 1 which belongs to the first block must remain in the stacks until all elements are pushed. This induces several constraints which are proved in the following propositions. All these propositions aim at proving Theorem 4.23 which fully characterizes $\oplus$-decomposable 2-stack pushall sortable permutations.

Theorem 4.23. *Let σ be a $\oplus$-decomposable permutation. Then σ is 2-stack pushall sortable if and only if σ avoids the set B_+ of 37 patterns of size 6, 7 or 8, where*

$$\begin{aligned} B_+ = \{&132465, 135246, 142536, 142635, 143625, 153624, 213546, 214365, 214635, 215364, \\ &241365, 314265, 315246, 315426, 351426, 1354627, 1365724, 1436527, 1473526, 1546273, \\ &1573246, 1624357, 1627354, 1632547, 1632574, 1642573, 1657243, 2465137, 2631547, \\ &2635147, 3541627, 4621357, 4652137, 5136427, 5162437, 21687435, 54613287\}. \end{aligned}$$

The proof proceeds step by step in Propositions 4.24 to 4.31.

Proposition 4.24. *Let σ be a permutation such that either:*

- $\sigma \in Av(132)$
- $\sigma \in Av(213)$

- $\sigma \in \oplus[Av(132), Av(213)]$
- $\sigma \in \oplus[Av(213), Av(132)]$

Then σ is 2-stack pushall sortable.

Proof. Using Theorem 4.17, we only have to prove that we can put all elements of σ in the stacks so that they avoid the three unsortable patterns (see Definition 4.12 (p.152). In the first case, just push every element in stack H using only operations ρ. For the second case, we know from Knuth [Knu73a] that each permutation avoiding 231 can be sorted in increasing order with one stack. So each permutation avoiding 213 can be sorted in decreasing order with one stack. Hence we can use stack H to push all elements of σ in decreasing order onto stack V. For the last two cases, we push the elements of σ corresponding to $Av(132)$ in stack H and those corresponding to $Av(213)$ in stack V. In each case, the stack configuration avoids the three unsortable patterns. ■

Note that Proposition 4.24 gives a sufficient condition which is not necessary: the permutation 143652 is 2-stack pushall sortable using the word $\rho\rho\lambda\rho\lambda\lambda\rho\rho\rho\mu\lambda\mu\mu\mu\lambda\mu\lambda\mu$, and 143652 is $\oplus$-decomposable but does not belong to one of the preceding cases. In this proposition, an important role is given to classes $Av(213)$ and $Av(132)$. These indeed are exactly the classes of permutations that can be pushall sorted with a stack configuration where all elements lie in one *single* stack (V for $Av(213)$ and H for $Av(132)$). Thus the only difficult case is whenever a permutation contains both patterns 132 and 213. This is characterized by the following proposition:

Proposition 4.25. *A permutation σ contains both patterns 213 and 132 if and only if it contains one of the following patterns: 1324, 2143, 2413, 3142, 465213 and 546132.*

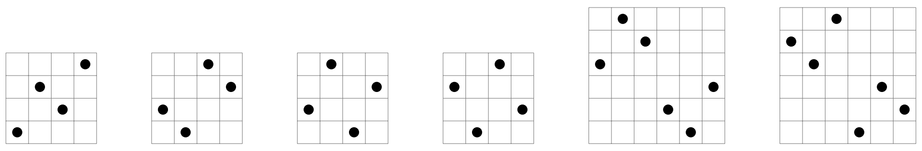


Figure 4.2: Minimal permutations containing patterns 132 and 213.

Proof. Minimal permutations that contain both 132 and 213 are exactly permutations of the basis of $Av(132) \bigcup Av(213)$. By minimality of the elements of the basis those permutations are at most of size 6 and a comprehensive study ends the proof. ■

To prove a complete characterization of $\oplus$-decomposable 2-stack pushall sortable permutations, we first deal with the case where the first and the last block of the $\oplus$-decomposition are non-trivial –i.e. not reduced to a singleton.

Proposition 4.26. *Suppose $\sigma = \oplus[\alpha_1 \dots \alpha_r]$ with $r \geq 2$, each α_i $\oplus$-indecomposable and blocks α_1 and α_r are non-trivial. Then σ is 2-stack pushall sortable if and only if σ avoids every pattern of $B_1 = \{132465, 213546, 214365, 214635, 215364, 241365, 314265, 1657243, 4652137, 21687435, 54613287\}$.*

Proof. By checking each pushall stack word of the appropriate size we determine that the permutations in B_1 are not 2-stack 2-stack pushall sortable. Hence if σ is 2-stack pushall sortable it avoids B_1. Conversely, let σ be a permutation avoiding every pattern of B_1. As α_1 and α_r are non-trivial and $\oplus$-indecomposable, they contain 21 as a pattern. But σ

avoids 214365 so that blocks α_i with $2 \leq i \leq r-1$ are trivial. Let $I = \{i \mid \alpha_i$ contains $132\}$ and $J = \{j \mid \alpha_j$ contains $213\}$. These sets are both included in $\{1, r\}$ and no one is equal to $\{1, r\}$ as σ avoids 132465 and 213546.

- If $I = J = \emptyset$, then $\alpha_1 \in Av(132)$ and $\oplus[\alpha_2 \dots \alpha_r] \in Av(213)$ so $\sigma \in \oplus[Av(132), Av(213)]$ and σ is 2-stack pushall sortable by Proposition 4.24.
- If $I = \emptyset$ and $J = \{j_0\}$, then $j_0 \in \{1, r\}$. If $j_0 = 1$ then $\alpha_1 \in Av(132)$ and $\oplus[\alpha_2 \dots \alpha_r] \in Av(213)$ hence $\sigma \in \oplus[Av(132), Av(213)]$ and σ is 2-stack pushall sortable by Proposition 4.24. If $j_0 = r$, as σ avoids 213546 then $r = 2$, but $\alpha_1 \in Av(213)$ and $\alpha_r \in Av(132)$ hence $\sigma \in \oplus[Av(213), Av(132)]$. Thus σ is 2-stack pushall sortable by Proposition 4.24.
- If $I = \{i_0\}$ and $J = \emptyset$, then $i_0 \in \{1, r\}$. If $i_0 = 1$, as σ avoids 132465 then $r = 2$, but $\alpha_1 \in Av(213)$ and $\alpha_r \in Av(132)$ hence $\sigma \in \oplus[Av(213), Av(132)]$. So σ is 2-stack pushall sortable by Proposition 4.24. If $i_0 = r$ then $\alpha_1 \in Av(132)$ and $\oplus[\alpha_2 \dots \alpha_r] \in Av(213)$ hence $\sigma \in \oplus[Av(132), Av(213)]$ and σ is 2-stack pushall sortable by Proposition 4.24.
- If $I = \{i_0\} \neq J = \{j_0\}$. If $i_0 = 1$ then $j_0 = r$ and $r = 2$ as σ avoids 132465. But $\alpha_1 \in Av(213)$ and $\alpha_r \in Av(132)$ hence $\sigma \in \oplus[Av(213), Av(132)]$ and σ is 2-stack pushall sortable by Proposition 4.24. If $i_0 = r$ then $j_0 = 1$, $\alpha_1 \in Av(132)$ and $\oplus[\alpha_2 \dots \alpha_r] \in Av(213)$ hence $\sigma \in \oplus[Av(132), Av(213)]$. So σ is 2-stack pushall sortable by Proposition 4.24.
- If $I = J = \{i_0\}$, then by Proposition 4.25, α_{i_0} contains either $1324, 2143, 2413, 3142,$ 465213 or 546132. Then we prove that σ contains a pattern of B_1, proving that this case is excluded. Indeed if α_{i_0} contains 1324, either $i_0 = 1$ and σ contains 132465, or $i_0 = r$ and σ contains 213546. Similarly if α_{i_0} contains 2143, then σ contains 214365. With the same argument (noticing that either $i_0 = 1$ or $i_0 = r$),

if α_{i_0} contains:	2413	3142	465213	546132
then σ contains:	241365	314265	46521387	54613287
			(thus 4652137)	
	or 214635	or 215364	or 21687435	or 21768354
				(thus 1657243)

Hence the case $I = J = \{i_0\}$ cannot occur. ■

Now we deal with the case where the first block of the $\oplus$-decomposition is trivial. We first need a few definitions.

Given two permutation classes $\mathcal{C}$ and $\mathcal{C}'$, their *horizontal juxtaposition* $[\mathcal{C}\ \mathcal{C}']$ consists of all permutations σ that can be written as a concatenation $[\pi, \tau]$ where π is order isomorphic to a permutation in $\mathcal{C}$ and τ is order-isomorphic to a permutation in $\mathcal{C}'$. In other words, a diagram of a permutation $\sigma \in [\mathcal{C}\ \mathcal{C}']$ can be divided by a vertical line into two parts, such that the left one is order-isomorphic to a permutation of $\mathcal{C}$ and the right one to a permutation of $\mathcal{C}'$. We can similarly define the *vertical juxtaposition* $\begin{bmatrix} \mathcal{C} \\ \mathcal{C}' \end{bmatrix}$ consisting of permutations having a diagram cut by a horizontal line.

Proposition 4.27. *A permutation $\oplus[1, \sigma]$ is 2-stack pushall sortable if and only if $\sigma \in \big[Av(213)\ Av(132)\big]$ and there exists an associated decomposition $\sigma = [\pi, \tau]$ such that there is no pattern 213 in σ where 2 is in π and 13 is in τ.*

Proof. If $\sigma = [\pi, \tau]$ with this decomposition satisfying the hypothesis of the proposition, then $\oplus[1, \sigma]$ is 2-stack pushall sortable using the following algorithm. Put 1 in H. Then push elements of π in stack V in decreasing order. Then put 1 at top of V and finally push every element of τ onto H. As there is no pattern 213 in σ with 2 in π and 13 in τ, the stack configuration obtained avoids the 3 unsortable stack-patterns hence can be popped out from Theorem 4.13.

Conversely, suppose that $\oplus[1, \sigma]$ is 2-stack pushall sortable and consider a pushall sorting process for this permutation. As 1 is the first element, the first operation is to push 1 to the bottom of H. Then maybe some elements are pushed onto 1 and into V before 1 is popped out from stack H to stack V. The remaining elements are pushed into H as they are greater than 1. We consider the stack configuration just before 1 is written in the output. At this moment, all elements have been pushed and 1 is at the top of V. This separates into two parts the elements of σ taking τ as the elements in H and π the elements in V apart from 1. From Theorem 4.13 decomposition $\sigma = [\pi, \tau]$ satisfies the conditions of the statement. ■

Proposition 4.28. *Let $E = \{\sigma \mid \oplus[1, \sigma]$ is 2-stack pushall sortable $\}$. Then E is a finitely based permutation class whose basis is $B_2 = \{21354, 24135, 31425, 31524, 32514, 42513, 243516, 254613, 325416, 362415, 435162, 462135, 513246, 516243, 521436, 521463, 531462, 546132, 4652137\}$.*

Proof. As 2-stack pushall sortable permutations form a permutation class, so does E. Let B_2 be the basis of E. To prove that B_2 is finite, we prove that every permutation in B_2 has size less than 9. Then a comprehensive computation gives the permutations in B_2.

By Proposition 4.27, $E = \{\sigma = [\pi, \tau] \mid \pi \in Av(213), \tau \in Av(132)$ and there is no pattern 213 in σ where 2 is in π and 13 is in $\tau\}$. Let $\sigma \in B_2$.By definition $\sigma \notin E$ so $\sigma \notin Av(213)$ and $\sigma \notin Av(132)$. Let $\sigma_i\sigma_j\sigma_k$ be a pattern 132 such that i is maximal and $\sigma_r\sigma_s\sigma_t$ be a pattern 213 such that t is minimal, then r minimal (for t fixed) and finally s maximal (for t and r fixed).

- If $t < i$ then $\theta = \sigma_r\sigma_s\sigma_t\sigma_i\sigma_j\sigma_k \notin E$, hence by minimality of the basis $\sigma = \theta$ so $|\sigma| = 6$.
- If $t = i$ then $\theta = \sigma_r\sigma_s\sigma_i\sigma_j\sigma_k \notin E$ and by minimality $\sigma = \theta$ so $|\sigma| = 5$.
- If $t > i$, consider the pattern $\sigma_r\sigma_s\sigma_t$ (shown in Figure 4.3). Minimality conditions for t and r and the maximality condition for s imply that the gray zones in the diagram of σ are empty. So $s = t-1$. As $\sigma \notin E$, there is no possible cut $\sigma = \pi\tau$ such that $\pi \in Av(213)$, $\tau \in Av(132)$ and there is no pattern 213 in σ where 2 is in π and 13 is in τ. Hence, all cuts in σ are forbidden, either because they are to the left of a 132 pattern or to the right of a 213 pattern or between element 2 and 1 of a pattern 213. More specially the cut between $t-1$ and t is forbidden. This cut cannot be to the left of a pattern 132 by maximality of i ($t > i$) and cannot be to the right of a pattern 213 by minimality of t. So this cut is between elements 2 and 1 of a pattern 213. We consider a pattern 213 denoted by $\sigma_x\sigma_y\sigma_z$ such that x is minimal and y is minimal for x fixed among patterns 213 such that $x \leq s = t-1$ and $y \geq t$.

  - If $r \leq i$, then E does not contains the pattern θ of σ obtained by keeping elements of $\{\sigma_r\sigma_s\sigma_t\sigma_i\sigma_j\sigma_k\sigma_x\sigma_y\sigma_z\}$. Indeed all cuts are forbidden: those before r by $\sigma_i\sigma_j\sigma_k$, between r and s by $\sigma_r\sigma_s\sigma_t$, between s and t by $\sigma_x\sigma_y\sigma_z$ and after t by $\sigma_r\sigma_s\sigma_t$. So by minimality of the basis $|\sigma| \leq 9$.

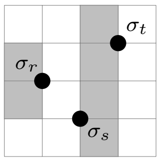


Figure 4.3: $\sigma_r\sigma_s\sigma_t$

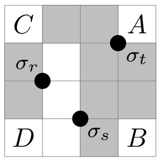


Figure 4.4: Case $r > i$

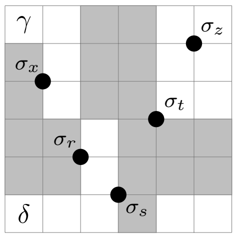


Figure 4.5: $\sigma_x\sigma_r\sigma_s\sigma_t\sigma_z$

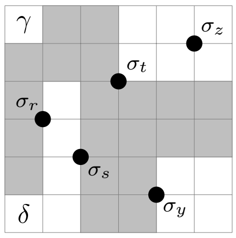


Figure 4.6: $\sigma_r\sigma_s\sigma_t\sigma_y\sigma_z$

- If $r > i$, we want to prove that $x \leq i$. As before, $\theta = \{\sigma_r\sigma_s\sigma_t\sigma_i\sigma_j\sigma_k\sigma_x\sigma_y\sigma_z\} \notin E$ since all cuts are forbidden (those before x by $\sigma_i\sigma_j\sigma_k$, between x and t by $\sigma_x\sigma_y\sigma_z$ and after t by $\sigma_r\sigma_s\sigma_t$) thus $|\sigma| \leq 9$.
  As $r > i$ and i maximal, the gray zones added in Figure 4.4 are empty. As $y \geq t$, σ_y and σ_z lie either both in A, or both in B, or σ_y lies in B and σ_z in A.
  - If σ_y and σ_z lie both in B, then σ_x lies in D. Thus $\sigma_x\sigma_t\sigma_z$ form the permutation 132 and as i is maximal, $x \leq i$.
  - If σ_y and σ_z lie both in A, then σ_x lies in C and by minimality of y we have $y = t$. Since x is minimal, the gray zones added in Figure 4.5 are empty. Suppose that $x > i$. The cut between i and $i + 1$ is forbidden as $\sigma \notin E$. As i is maximal the cut cannot be to the left of a pattern 132, neither to the right of a pattern 213 by minimality of t. Hence the cut lies between element 2 and 1 of a pattern 213. Let $\sigma_a\sigma_b\sigma_c$ be such a pattern 213 such that $a \leq i$ and $b > i$. Then $a < x$ and σ_a lies in area γ or δ and $c \geq t$ by minimality of t. If σ_a lies in γ then $\sigma_a\sigma_t\sigma_c$ form a pattern 213, which is forbidden by minimality of x. Hence σ_a lies in δ and $b \geq t$ otherwise $\sigma_a\sigma_b\sigma_t$ is a pattern 213 with $a \leq i < r$, which is also forbidden by minimality of r. Hence $\sigma_a\sigma_b\sigma_c$ is a pattern 213 with $a \leq i < x \leq s$ and $b \geq t$ which is impossible by minimality of x.
  - If σ_y lies in B and σ_z in A, by minimality of x, either $x = r$, or σ_x lies in C, or σ_x lies in D. If σ_x lies in C then $\sigma_x\sigma_t\sigma_z$ is a pattern 213 which contradicts the minimality of y. If σ_x lies in D, $\sigma_x\sigma_r\sigma_s$ is a pattern 132 hence $x \leq i$. If $x = r$, by minimality of x then y, the gray zones added in Figure 4.6 are empty. The cut between i and $i+1$ is forbidden as $\sigma \notin E$. As before the cut lies between elements 2 and 1 of a pattern 213. Let $\sigma_a\sigma_b\sigma_c$ such a pattern 213 such that $a \leq i$ and $b > i$. Then $a < r$ and σ_a lies in γ or δ and $c \geq t$ by minimality of t. If σ_a lies in γ then $\sigma_a\sigma_t\sigma_c$ is a pattern 213 and by minimality of x, $x \leq a \leq i$. If σ_a lies in δ then $b \geq t$ otherwise $\sigma_b\sigma_t\sigma_y$ is a pattern 132 with $b > i$, which is forbidden by maximality of i. But $\sigma_a\sigma_b\sigma_c$ is a pattern 213 with $a \leq i < r$ and $b \geq t$, so by minimality of x, $x \leq i$. ■

Now we turn to the study of the case where the last block of the $\oplus$-decomposition is trivial.

Proposition 4.29. *A permutation $\oplus[\sigma, 1]$ is 2-stack pushall sortable if and only if $\sigma \in \begin{bmatrix} Av(132) \\ Av(213) \end{bmatrix}$ and there exists an associated decomposition $\sigma = \begin{bmatrix} \pi \\ \tau \end{bmatrix}$ such that there is no pattern 132 in σ where element 3 is in π and elements 1 and 2 are in τ.*

Proof. Let $n = |\sigma| + 1$. Assume that $\oplus[\sigma, 1]$ is 2-stack pushall sortable. Consider a pushall sorting of $\oplus[\sigma, 1]$. This permutation has n as last element. We consider the stack configuration just after n enters stack H. By Theorem 4.13, this configuration avoids the pattern $|2|13|$, so that all elements in H –below n– are greater than those of V. Hence we can write $\sigma = \begin{bmatrix} \pi \\ \tau \end{bmatrix}$ where τ contains the elements of V and π those of H –except n. Then from Theorem 4.13, $\pi \in Av(132)$ and $\tau \in Av(213)$ since the elements of τ are sorted in decreasing order in V using only the stack H. Moreover there is no pattern 132 in σ where element 3 is in π and elements 1 and 2 are in τ otherwise element 3 by staying in stack H would prevent elements 1 and 2 to be in the right order in V.

Conversely, suppose that there exists a decomposition $\sigma = \begin{bmatrix} \pi \\ \tau \end{bmatrix}$ respecting the previous conditions; then we have a pushall sorting of the permutation $\oplus[\sigma, 1]$ using the following algorithm. While the input is not empty, if stack H is empty or if the top of H belongs to π, we push the next element of the input onto H. If σ_i, the top of H, belongs to τ, and if the next element of the input σ_j belongs to τ and is greater than σ_i, we push σ_j onto H, otherwise we pop σ_i from H and push it onto V. At each step we verify the conditions of Theorem 4.13 so that all elements can be popped out in increasing order at the end. ■

Proposition 4.30. *Let $F = \{\sigma \mid \oplus[\sigma, 1]$ is 2-stack pushall sortable $\}$. Then F is a finitely based permutation class whose basis is $B_3 = \{13524, 14253, 21354, 31524, 31542, 35142,$ 135462, 143652, 162435, 163254, 246513, 263154, 263514, 354162, 462135, 465213, 513642, $516243, 1657243\}$.*

Proof. As the set of 2-stack pushall sortable permutations is a permutation class, so is F. By Proposition 4.29, $F = \{\sigma \in \begin{bmatrix} Av(132) \\ Av(213) \end{bmatrix}$ such that there exists an associated decomposition $\sigma = \begin{bmatrix} \pi \\ \tau \end{bmatrix}$ without pattern 132 in σ where element 3 is in π and elements 1 and 2 are in $\tau\}$. Hence E and F are in one-to-one correspondence by the map taking an element of E, rotating its diagram by $-\pi/2$ and applying the symmetry with respect to axis (Oy) (see Figure 4.7). When elements are in one-to-one correspondence by rotation and symmetry so are the bases, which proves the result. ■

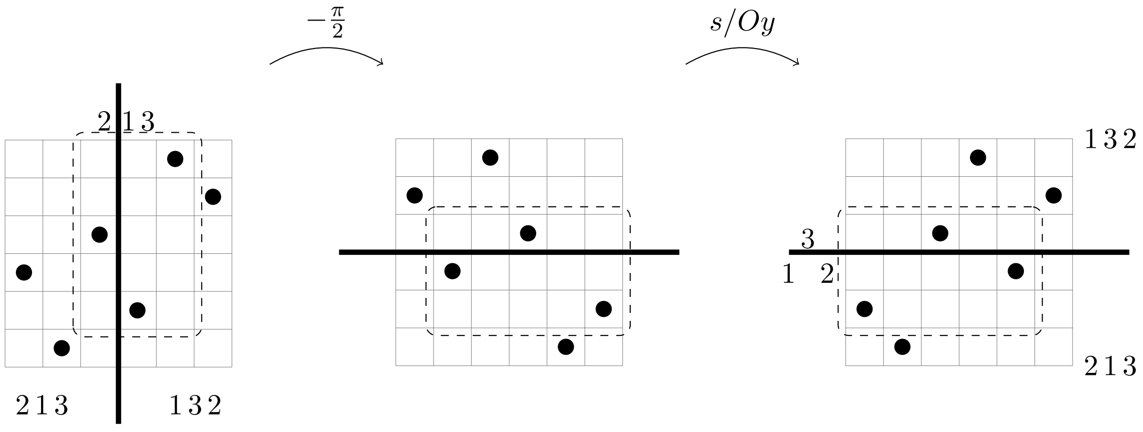


Figure 4.7: E and F are in one-to-one correspondence by symmetry.

Proposition 4.31. *A permutation $\oplus[1, \sigma, 1]$ is 2-stack pushall sortable if and only if $\sigma \in \oplus[Av(213), Av(132)]$.*

Proof. By Proposition 4.27, $\oplus[1, \sigma, 1]$ is 2-stack pushall sortable if and only if $\oplus[\sigma, 1] \in \big[Av(213), Av(132)\big]$ and there exists a corresponding decomposition $\sigma = [\pi, \tau]$ such that there is no pattern 213 in σ where element 2 is in π and 13 are in τ, which is equivalent to $\sigma \in \big[Av(213), Av(132)\big]$ and there exists a corresponding decomposition $\sigma = [\pi, \tau]$ such that there is no pattern 21 in σ where element 2 is in π and element 1 is in τ, i.e. $\sigma \in \oplus[Av(213), Av(132)]$. ■

We are now able to prove Theorem 4.23 (p.155).

Proof. Permutations of B_+ are not 2-stack pushall sortable (check each pushall stack word of the right size), hence if σ is 2-stack pushall sortable it avoids B_+. Conversely suppose that σ avoids B_+. Let $\sigma = \oplus[\alpha_1 \dots \alpha_r]$ be the $\oplus$-decomposition of σ with $r \geq 2$ and α_i $\oplus$-indecomposable for all i.

- If α_1 and α_r are non-trivial then σ is 2-stack pushall sortable thanks to Proposition 4.26. Indeed σ avoids $B_1 = \{132465, 213546, 214365, 214635, 215364, 241365, 314265, 1657243,$
  $4652137, 21687435, 54613287\}$ as $B_1 \subset B_+$.
- If α_1 is trivial then $\sigma = \oplus[1, \pi]$ and π avoids $B_2 = \{21354, 24135, 31425, 31524, 32514, 42513, 243516, 254613, 325416, 362415, 435162, 462135, 513246, 516243, 521436, 521463, 531462, 546132, 4652137\}$ so that σ is 2-stack pushall sortable by Proposition 4.28.
- If α_r is trivial then $\sigma = \oplus[\pi, 1]$ and π avoids $B_3 = \{13524, 14253, 21354, 31524, 31542, 35142, 135462, 143652, 162435, 163254, 246513, 263154, 263514, 354162, 462135, 465213, 513642, 516243, 1657243\}$ hence σ is 2-stack pushall sortable by Proposition 4.30. ■

Permutations in the class $Av(2413, 3142)$ are called *separable* permutations. There is no simple permutation in this class (see Proposition 1.17 p.39; recall that we do not consider 12 and 21 as simple). For separable permutations we have:

Theorem 4.32. *Let σ be a separable permutation. Then σ is 2-stack pushall sortable if and only if σ avoids $B = \{132465, 213546, 214365, 1354627, 1436527, 1624357, 1632547, 1657243, 4652137, 21687435, 54613287\}$.*

Proof. Notice that B is included in the set B_+ defined in Theorem 4.23. As permutations of B are not 2-stack pushall sortable, every 2-stack pushall sortable permutation avoids B. Conversely, assume that σ avoids B. As σ is separable, σ has no simple pattern thus σ is either $\oplus$-decomposable or $\ominus$-decomposable or trivial (i.e. of size 1), and σ avoids 2413 and 3142 which added to constraints of B gives that σ avoids B_+. If σ is $\oplus$-decomposable, then σ is 2-stack pushall sortable by Theorem 4.23. If σ is $\ominus$-decomposable, then $\sigma = \ominus[\pi^{(1)}, \pi^{(2)}, \dots, \pi^{(k)}]$ where each $\pi^{(i)}$ is either trivial or $\oplus$-decomposable. So σ is 2-stack pushall sortable by Proposition 4.21 and Theorem 4.23. ■

The results proved in this subsection allow to decide in polynomial time whether a separable permutation is 2-stack pushall sortable, using the above theorem, and to decide in polynomial time whether a separable permutation is 2-stack sortable, using also Propositions 4.20 and 4.22.

They also have consequences on the basis of 2-stack pushall sortable permutations and on the basis of 2-stack sortable permutations that are studied in the next subsection.

4.2.4 Basis of stack sorting classes

In the previous section, we showed that 2-stack pushall sortable separable permutations form a finitely based permutation class. This property does not hold for 2-stack pushall sortable permutations and we exhibit an infinite antichain of the basis in the following proposition:

Proposition 4.33. *The basis of* 2*-stack pushall sortable permutation class is infinite. More precisely the set of permutations depicted in Figure 4.8 is an infinite antichain of simple permutations belonging to the basis of* 2*-stack pushall sortable permutations.*

Proof. Consider permutations $2n{-}3\ 2n{-}1\ 2n{-}5\ 2n \ldots p\ p{+}5 \ldots 1\ 6\ 2\ 4$ for $n \geq 3$. The first ones are depicted in Figure 4.8. These permutations are simple and incomparable. To complete the proof, straightforward though technical, just check that those permutations are not 2-stack pushall sortable and that every pattern of these permutations are 2-stack pushall sortable. ■

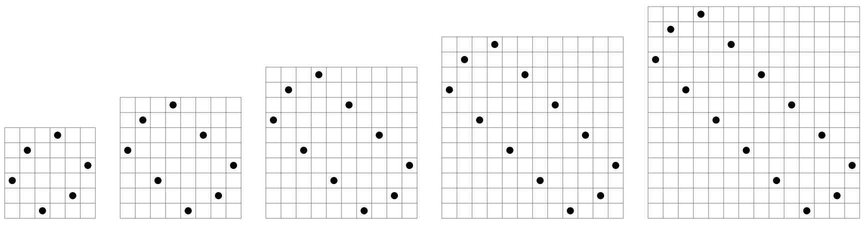


Figure 4.8: An antichain of the basis of 2-stack pushall sortable permutation class $2n{-}3\ 2n{-}1\ 2n{-}5\ 2n \ldots p\ p{+}5 \ldots 1\ 6\ 2\ 4$ for $n \geq 3$.

Note that the basis of 2-stack pushall sortable permutations is infinite and contains an infinite number of simple permutations, and the 2-stack pushall sortable class also contains an infinite number of simple permutations (otherwise its basis would be finite).

Recall that for the set of 2-stack sortable permutations, it has been proved by Murphy in [Mur02] that basis is also infinite, by exhibiting two different sets of basis elements, one of permutations that are essentially decreasing (see Figure 4.9), and one of permutations that are essentially increasing (see Figure 4.10).

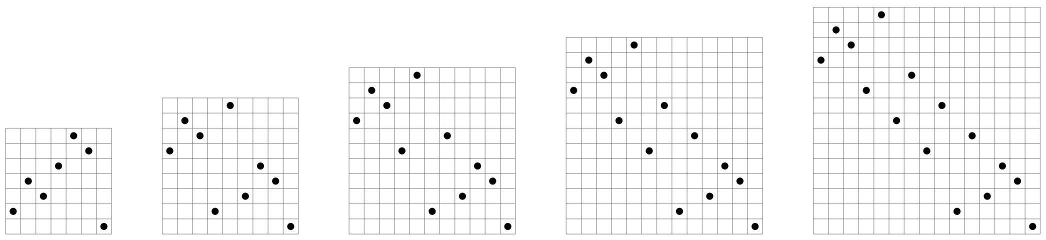


Figure 4.9: An antichain of the basis of 2-stack sortable permutation class 2 4 3 5 7 6 1, 6 8 7 2 9 3 5 4 1, 8 10 9 6 11 2 7 3 5 4 1, 10 12 11 8 13 6 9 2 7 3 5 4 1 and $2n{+}4\ \ 2n{+}6\ \ 2n{+}5\ \ 2n{+}2\ \ 2n{+}7\ \ 2n\ \ 2n{+}3 \ldots p\ \ p{+}3 \ldots 8\ 11\ 6\ 9\ 2\ 7\ 3\ 5\ 4\ 1$ for $n \geq 4$.

In the following, we study the elements of the basis of 2-stack sortable permutations and of 2-stack pushall sortable permutations depending on the root of their decomposition tree. We also study the link between these two bases.

Proposition 4.34. *If σ is in the basis of* 2*-stack pushall sortable permutations, then σ is* 2*-stack sortable.*

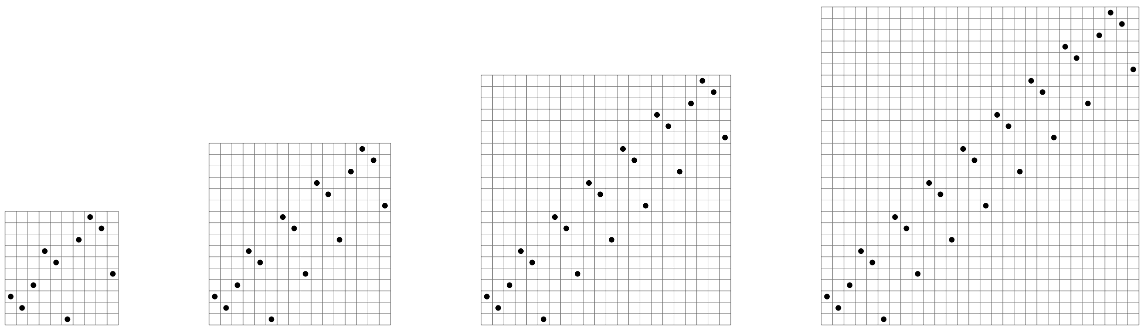


Figure 4.10: An antichain of the basis of 2-stack sortable permutation class
$3\ 2\ 4\ 7\ 6\ 1\ 10\ 9\ 5 \ldots p{+}1\ \ p\ \ p{-}4 \ldots 6n{+}1\ \ 6n\ \ 6n{-}4\ \ 6n{+}2\ \ 6n{+}4\ \ 6n{+}3\ \ 6n{-}1$ for $n \geq 1$.

Proof. Let $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ be in the basis of 2-stack pushall sortable permutations. By definition, $\sigma_1\sigma_2 \ldots \sigma_{n-1}$ is 2-stack pushall sortable. We can sort σ (not pushall sort σ) using the following algorithm. Push all elements σ_1 to σ_{n-1} in the stacks following the 2-stack pushall sortable operations of $\sigma_1 \ldots \sigma_{n-1}$. Then pop elements $1, 2, \ldots, \sigma_n - 1$, then push σ_n and pop it to the output and pop the remaining elements. It is easy to check that these operations are allowed. ■

Proposition 4.35. *Let π be a $\ominus$-decomposable permutation. Then π belongs to the basis of the 2-stack sortable permutations if and only if $\pi = \ominus[\sigma, 1]$ where σ belongs to the basis of the 2-stack pushall sortable permutations.*

Proof. Let $\pi = \ominus[\sigma, 1]$ with σ a permutation of the basis of 2-stack pushall sortable permutations. Since σ is not 2-stack pushall sortable, Proposition 4.20 ensures that π is not 2-stack sortable. Moreover every proper pattern of π is 2-stack sortable. Indeed if we delete element 1 then the obtained permutation is σ, hence it is 2-stack sortable by Proposition 4.34. Otherwise we delete an element of σ leading to σ' which is 2-stack pushall sortable by the definition of a permutation class basis. Then, $\ominus[\sigma', 1]$ is 2-stack pushall sortable using Proposition 4.20.

Conversely, if $\sigma = \ominus[\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(k)}]$ with $k \geq 2$ belongs to the basis of 2-stack sortable permutations, then by Proposition 4.20, either $\pi^{(k)}$ is not 2-stack sortable which contradicts the minimality of σ (σ is an element of the basis so that every pattern of σ must belong to the class) or there exists $1 \leq i \leq k-1$ such that $\pi^{(i)}$ is not 2-stack pushall sortable. But in that case, $\ominus[\pi^{(i)}, 1]$ is not 2-stack sortable by Proposition 4.20; hence $\sigma = \ominus[\pi^{(i)}, 1]$ by minimality of basis elements. If $\pi^{(i)}$ has a proper pattern τ which is not 2-stack pushall sortable then $\ominus[\tau, 1]$ is a proper pattern of σ which is not 2-stack sortable. This is impossible as σ belongs to the basis of 2-stack sortable permutations. So $\pi^{(i)}$ belongs to the basis of 2-stack pushall sortable permutations, which concludes the proof. ■

By putting together the results of this section we have the following theorem:

Theorem 4.36. *For 2-stack sortable permutations we have:*

- *There are no $\oplus$-decomposable permutations in the basis of 2-stack sortable permutations.*
- *The $\ominus$-decomposable permutations in the basis of 2-stack sortable permutations are $\ominus[\sigma, 1]$ where σ is in the basis of pushall sortable permutations.*

- *In the basis of* 2*-stack sortable permutations, there are an infinite number of permutations whose root of the decomposition tree is prime.*

And for 2*-stack pushall sortable permutations we have:*

- *There are an infinite number of simple permutations in the basis of* 2*-stack pushall sortable permutations.*
- *There are no* $\ominus$*-decomposable permutations in the basis of* 2*-stack pushall sortable permutations.*
- $\oplus$*-decomposable permutations in the basis of* 2*-stack pushall sortable permutations are* 132465, 135246, 142536, 142635, 143625, 153624, 213546, 214365, 214635, 215364, 241365, 314265, 315246, 315426, 351426, 1354627, 1365724, 1436527, 1473526, 1546273, 1573246, 1624357, 1627354, 1632547, 1632574, 1642573, 1657243, 2465137, 2631547, 2635147, 3541627, 4621357, 4652137, 5136427, 5162437, 21687435 *and* 54613287*.*

Proof. The first item follows from Proposition 4.22 and the second one from Proposition 4.35. The third item is a consequence of the proof by Murphy that the antichain of Figure 4.10 belongs to the basis of 2-stack sortable permutations. The three last items are consequences of Propositions 4.33 and 4.21 and Theorem 4.23. ■

We conclude this section by giving numerical results about 2-stack sortable permutations and 2-stack pushall sortable permutations:

	pushall sorting			general sorting		
n	sortable	unsortable	basis	sortable	unsortable	basis
5	120	0	0	120	0	0
6	698	22	22	720	0	0
7	4393	647	38	5018	22	22
8	28551	11769	25	39374	946	51
9	187403	175477	22	336870	26010	146
10	1231517	2397283	29	3066695	562105	604
11	8080058	31836742	34			

It seems that the basis of 2-stack sortable permutations is quickly growing, whereas the one of 2-stack pushall sortable permutations seems to have about a constant number of elements of each size.

4.3 Pushall sorting and bicoloring

4.3.1 A simple characterization

There is a natural relation between 2-stack pushall sorting and the coloring of permutation diagrams into two colors. The key idea is to look at the stack configuration once all elements of the permutation are pushed into the stacks. Then all elements of the permutation belong either to stack H or to stack V. We assign a color to them depending in which stack they lie at this particular step of the sorting. In this chapter we color like 🟥 points that lie in stack H and like 🟩 points in stack V.

However by Remark 4.6, such a stack configuration is not unique, and neither is the coloring.

Nevertheless properties of the stack configuration such as being poppable and reachable can be checked on the corresponding coloring. This is explained in details in the following.

Definition 4.37. A *bicoloring* of a permutation σ is a coloring of the points of the diagram of σ with two colors G and R.

A *valid coloring* is a bicoloring which avoids each of the four following colored patterns:

- pattern 132: there is a pattern 132 in R
- pattern 213: there is a pattern 213 in G
- pattern $1X2$: there is a point of R lying vertically between a pattern 12 of G
- pattern $2/13$: there is a point of G lying horizontally between a pattern 12 of R.

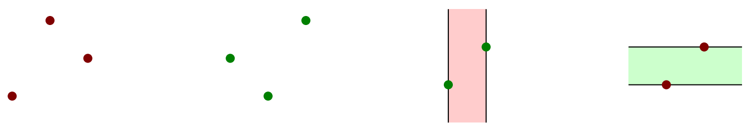


Figure 4.11: The four forbidden colored patterns for valid colorings.

Intuitively, the patterns avoided by a valid coloring correspond to the 3 unsortable stack-patterns. More precisely the pattern 132 corresponds to $|\ |132|$, the pattern $2/13$ corresponds to $|2|13|$, and the patterns 213 and $1X2$ ensure that it is possible to put the elements of G in stack V such that $|12|\ |$ is avoided.

In the definition below, we explain how to associate a stack configuration to a bicoloring.

Definition 4.38. Let σ be a permutation. To each total stack configuration of σ the map $Bicol$ assigns the bicoloring of σ such that elements of H are in R and elements of V are in G. To every bicoloring of a permutation σ the map $Conf$ associates the total stack configuration of σ such that elements of G lie in V in decreasing order of value from bottom to top and elements of R lie in H in increasing order of indices from bottom to top.

Remark 4.39. For any bicoloring b, $Bicol(Conf(b)) = b$. For any stack configuration c such that elements of V are in decreasing order of value from bottom to top and elements of H are in increasing order of indices from bottom to top, $Conf(Bicol(c)) = c$.

Proposition 4.40. *Let b be a bicoloring of a permutation σ. Then Algorithm 11 applied to b terminates in linear time w.r.t $|\sigma|$ and returns true if and only if $Conf(b)$ is reachable for σ. In this case the stack configuration to which Algorithm 11 leads is $Conf(b)$.*

To prove this proposition we need the following lemma:

Lemma 4.41. *At each step of Algorithm 11, the current stack configuration is reachable for σ, elements of H are in increasing order of indices from bottom to top, elements of V are in decreasing order of value from bottom to top, there is no element of R in V, there is no element of R above an element of G in H and elements of G that lie in H are in increasing order of value from bottom to top.*

Moreover if the condition of line 2 ($i \leq |\sigma|$) is verified then σ_i is the next element of the input, otherwise there is no more element in the input.

Algorithm 11: Algorithm deciding in linear time whether the configuration corresponding to a bicoloring is reachable

```
   Data: $\sigma$ a permutation and $b$ a bicoloring of $\sigma$.
   Result: True if the stack configuration corresponding to $b$ is reachable from $\sigma$.
 1 Begin with the empty stack configuration and $\sigma$ as input and $i = 1$;
 2 while $i \leq |\sigma|$ do
 3     if $H$ is empty or $top(H) \in R$ then
 4         push $\sigma_i$ into $H$;
 5         $i \longleftarrow i + 1$;
 6     else /* $top(H) \in G$ */
 7         if $\sigma_i \in R$ or $\sigma_i < top(H)$ then
 8             if $V$ is empty or $top(H) < top(V)$ then
 9                 pop $top(H)$ from stack $H$ and push it into $V$;
10             else
11                 Return false;
12             end
13         else /* $top(H) \in G$, $\sigma_i \in G$ and $\sigma_i > top(H)$ */
14             push $\sigma_i$ into $H$;
15             $i \longleftarrow i + 1$;
16         end
17     end
18 end
19 while $H$ is nonempty and $top(H) \in G$ do
20     if $top(H) < top(V)$ then
21         pop $top(H)$ from stack $H$ and push it into $V$;
22     else
23         Return false;
24     end
25 end
26 Return true;
```

Proof: The proof is by induction on the number of stack operations performed by the algorithm. Algorithm 11 begins with the empty stack configuration and σ as input and $i = 1$ so the properties are true at the beginning. Algorithm 11 performs only appropriate stack operations so at each step the configuration obtained is reachable for σ. Moreover in a reachable configuration, elements of H are in increasing order of indices. When an element is put in V (this happens at line 9 or 21) then this element is in G (checked at line 6 or 19) and is smaller than the top of V (checked at line 8 or 20) so that elements of V remain in decreasing order of value from bottom to top and V contains no element of R. When we put an element in H, it can be at line 4 or 14. In the first case, H is empty or its top is in R (checked at line 3) so all its elements are in R by the induction hypothesis. In the second case, the top of H is in G and the element we put in H is in G and greater than the top of H. This ensures that there is no element of R above an element of G in H and that elements of G that lie in H are in increasing order from bottom to top (using induction hypothesis). Finally i is increased exactly when σ_i is put into H so the last property remains true. ■

We are now able to prove Proposition 4.40:

Proof: At each step, Algorithm 11 performs either a legal operation ρ, or a legal operation λ, or returns false or true (and stops). As at most $|\sigma|$ legal operations ρ and $|\sigma|$ legal operations λ can be done, Algorithm 11 terminates after at most $2|\sigma| + 1$ steps. As each step is done in constant time, Algorithm 11 terminates in linear time w.r.t $|\sigma|$.

Moreover if Algorithm 11 applied to b returns true, then it reaches line 26. In particular the loop of line 19 stops so the top of H is not in G. Thus by Lemma 4.41 there is no element of G in H. In addition by the same lemma elements of H are in increasing order of indices from bottom to top, elements of V are in decreasing order of value from bottom to top and there is no element of R in V. Therefore the stack configuration we have is $Conf(b)$. Moreover Lemma 4.41 states that the stack configuration we have is reachable for σ, so $Conf(b)$ is reachable for σ.

Conversely if $Conf(b)$ is reachable for σ, then there is a sequence w of appropriate stack operations so that the configuration obtained with σ as input is $Conf(b)$. Let us prove that the sequence of operations w' performed by Algorithm 11 applied to b is w. We prove by induction on $k \le |w|$ $(k \ge 0)$ that w and w' have the same prefix of length k (obvious for $k = 0$). First notice that as $Conf(b)$ is a total stack configuration, thus w has no letter μ, and that Algorithm 11 performs only operations λ and ρ, so w' has no letter μ. Suppose that w and w' have the same prefix v of length k with $k < |w|$, let c' be the stack configuration obtained after performing operations of v with σ as input. We want to prove that w'_{k+1} exists and $w'_{k+1} = w_{k+1}$. By definition of w', w'_{k+1} is the operation performed by Algorithm 11 in configuration c' (setting by extension $w'_{k+1} = \alpha$ if Algorithm 11 terminates in configuration c', i.e. if $|w'| = k$), and by definition of w, w_{k+1} is an operation which allows to go from configuration c' to configuration $Conf(b)$ (maybe with some additional operations).

We check the value of i after Algorithm 11 has performed operations v. We know that at this step the stacks are in configuration c'.

If $i > |\sigma|$, then from Lemma 4.41 in configuration c' all elements of σ lie already in the stacks. As w is a sequence of appropriate stack operations, then $w_{k+1} \ne \rho$ so $w_{k+1} = \lambda$ (w has no letter μ). As w_{k+1} is an operation which allows moving from configuration c' to configuration $Conf(b)$ in which there is no elements of R in V and V is decreasing, then the top of H in c' is in G and smaller than the top of V (or V is empty). As $i > |\sigma|$ and as the top of H in c' is in G and smaller than the top of V (or V is empty) then Algorithm 11 performs line 21 so $w'_{k+1} = \lambda = w_{k+1}$.

If $i \le |\sigma|$ then we are in the loop beginning at line 2 of the algorithm and from Lemma 4.41 σ_i is the next element of the input. Suppose that $w_{k+1} = \lambda$. As w_{k+1} is a legal operation which allows moving from configuration c' to configuration $Conf(b)$ in which there is no element of R in V and V is decreasing, then H is non-empty, and the top of H is in G and smaller than the top of V (or V is empty). Suppose in addition that $\sigma_i \in G$. As σ_i is still on the input after w_{k+1} and w_{k+1} is an operation which allows to go to configuration $Conf(b)$ in which V is decreasing, then σ_i is smaller than the top of H in c'. So either $\sigma_i <$ top(H) or $\sigma_i \in R$. Thus from c' Algorithm 11 performs line 9 and $w'_{k+1} = \lambda = w_{k+1}$.

Suppose that $w_{k+1} = \rho$. If in configuration c' stack H is empty or top(H) $\in R$ then Algorithm 11 performs line 4 so $w'_{k+1} = \rho = w_{k+1}$. Otherwise let σ_h be the top of H in c', then $\sigma_h \in G$. So $\sigma_h \in V$ in $Conf(b)$. But once $w_{k+1} = \rho$ is performed σ_i is above σ_h in H. As w_{k+1} is an operation which allows to go from configuration c' to configuration $Conf(b)$ then σ_i is below σ_h in V in $Conf(b)$ (indeed it is impossible that σ_h goes to stack V and σ_i remains in stack H). So $\sigma_i \in G$ and as in $Conf(b)$ elements of V are in decreasing order, $\sigma_i > \sigma_h$. So the test of line 7 of the algorithm is false and Algorithm 11 performs line 14 so $w'_{k+1} = \rho = w_{k+1}$.

This ends the induction. We have proved that w is a prefix of w', so Algorithm 11 reaches configuration $Conf(b)$. We have now to prove that Algorithm 11 stops in this configuration and returns true.

When $Conf(b)$ is reached then there is no element in the input anymore, so from Lemma 4.41 $i > |\sigma|$, and $top(H) \notin G$ in $Conf(b)$. So both loops while of Algorithm 11 are finished and the algorithm reaches line 27, returns true and terminates in configuration $Conf(b)$. ■

Lemma 4.42. *Let b be a bicoloring of a permutation σ. If Algorithm 11 applied to b returns $false$ then b has a pattern $1X2$ or a pattern 213.*

Proof: We consider the stack configuration reached when Algorithm 11 returns $false$. We set $\sigma_h = top(H)$ and $\sigma_v = top(V)$. By Lemma 4.41, $\sigma_v \in G$. Algorithm 11 returns $false$ by reaching either line 11 or line 23. In both cases, $\sigma_h \in G$ and $\sigma_h > \sigma_v$. Now we consider the step of the algorithm where σ_v was put in V, the index i at this step of the algorithm, and the corresponding configuration c just before the operation putting σ_v into V. At this step σ_v is on the top of H, and $i > v$. If σ_h is in H in c, then it is below σ_v, contradicting Lemma 4.41 ($\sigma_h > \sigma_v$ and both are in G). As σ_h is in H when the algorithm ends, it cannot be in V in c. So σ_h is still in the input and $i \leq h \leq |\sigma|$. Recall that we consider the step of the algorithm where σ_v is put in V. This can happen at line 9 or 21 but $i \leq |\sigma|$ so it is at line 9. So the test of line 7 is true, thus either $\sigma_i \in R$ and then $\sigma_v, \sigma_i, \sigma_h$ is a pattern $1X2$ of b, or $\sigma_i \in G$ but $\sigma_i < \sigma_v$ (since σ_v is $top(H)$ of c) and then $\sigma_v, \sigma_i, \sigma_h$ is a pattern 213 of b. ■

Theorem 4.43. *The map Bicol is a bijection from the set of poppable reachable total stack configurations of σ to the set of valid colorings of σ. Moreover the inverse of Bicol is the map $Conf$.*

Proof: Let c be a poppable reachable total stack configuration of σ. Then from Theorem 4.13, σ avoids the three unsortable patterns. Set $b = Bicol(c)$, we have to prove that b is valid, i.e. avoids every forbidden colored pattern of Definition 4.37.

If b has a pattern 132 in R then there are three elements σ_i, σ_j and σ_k of R such that $i < j < k$ and $\sigma_i < \sigma_k < \sigma_j$. By definition of $Bicol$, in c elements σ_i, σ_j and σ_k lie in H. As c is reachable and $i < j < k$, σ_i is below σ_j which is below σ_k. So we have a stack-pattern $| \, |132|$ in c which contradicts our hypothesis. So b has no pattern 132.

If b has a pattern 213 in G then there are three element σ_i, σ_j and σ_k of G such that $i < j < k$ and $\sigma_j < \sigma_i < \sigma_k$. By definition of $Bicol$, in c elements σ_i, σ_j and σ_k lie in V. As c avoids stack-pattern $|12| \, |$, σ_k is below σ_i which is below σ_j. But then c is not reachable: as σ_k is below σ_i and σ_j in V, σ_i and σ_j have to stay in stack H until σ_k enters stack H. But as $i < j$, σ_i is below σ_j in stack H and cannot be below σ_j in stack V as going from stack H to stack V reverse the order. So b has no pattern 213.

If b has a point of R lying vertically between the elements of a pattern 12 of G then there are elements σ_i and σ_j of G and σ_k of R such that $i < k < j$ and $\sigma_i < \sigma_j$. By definition of $Bicol$, in c elements σ_i and σ_j lie in V and σ_k lies in H. Moreover configuration c is reachable. We consider a sequence of stack operations leading to c. As $i < k$, σ_i is already in the stacks when σ_k enters H. As σ_k remains in H in c but σ_i is in V in c, σ_i has to be already in V when σ_k enters stack H. As $k < j$, at this moment σ_j is not already in stack V, so σ_j will be above σ_i in V and they form a pattern $|12| \, |$ in c, which is excluded. So b has no pattern $1X2$.

If b has a point of G lying horizontally between the elements of a pattern 12 of R then in c these points form a pattern $|2|13|$ which is excluded. So b has no pattern $2/13$. Therefore b is a valid coloring.

Conversely let b be a valid coloring of σ. By definition $Conf(b)$ is a total stack configuration of σ. We have to prove that $Conf(b)$ is reachable for σ and avoids the three unsortable stack patterns. As b is a valid coloring, it avoids patterns $1X2$ and 213. So from Lemma 4.42, Algorithm 11 started with input b returns true. Thus from Proposition 4.40, c is reachable for σ. Moreover by definition of $Conf$, $Conf(b)$ avoids pattern $|12|\ |$. Furthermore we know that in $Conf(b)$, elements of H are in increasing order of indices from bottom to top. So if $Conf(b)$ has a pattern $|\ |132|$, then b has a pattern 132, and if $Conf(b)$ has a pattern $|2|13|$ then b has a pattern $2/13$. A b is a valid coloring, we conclude that $Conf(b)$ avoids the three unsortable stack patterns. Therefore $Conf(b)$ is a poppable reachable total stack configuration of σ.

Finally, using Remark 4.39 it's clear that $Conf$ is the inverse of $Bicol$. ■

Theorem 4.43 has many consequences:

Theorem 4.44. *Let σ be a permutation. Then there is a one-to-one correspondence between:*

- *Pushall sorting processes of σ*
- *Pushall stack words of σ*
- *Valid colorings of σ such that $\sigma_n \in R$*
- *Valid colorings of σ such that $1 \in G$*

Proof: This is a direct consequence of Theorems 4.19 and 4.43. ■

Theorem 4.45. *A permutation σ is 2-stack pushall sortable if and only if its diagram admits a valid coloring.*

Proof: This is a direct consequence of Theorems 4.17 and 4.43. ■

Now thanks to Theorem 4.45 we have a naive algorithm to check if a permutation σ is 2-stack pushall sortable: for all bicolorings b of σ, we can test whether b is valid by checking if b avoids patterns 213, $1X2$, $2/13$ and 132 of Definition 4.37. But first notice that we have a more efficient way to test if a bicoloring is valid:

Proposition 4.46. *Let b be a bicoloring of a permutation σ. We can check in linear time w.r.t. $|\sigma|$ whether b is a valid coloring. More precisely, b is a valid coloring if and only if Algorithm 11 applied to b returns true and Algorithm 10 applied to $Conf(b)$ returns true.*

Proof: From Theorem 4.43, b is valid if and only if $Conf(b)$ is poppable and reachable for σ. We conclude using Proposition 4.40 (which ensures that Algorithm 11 runs in linear time and returns true if and only if $Conf(b)$ is reachable) and Proposition 4.15 (which ensures that Algorithm 10 runs in linear time and returns true if and only if $Conf(b)$ is poppable). ■

However, even using this efficient way to test whether a bicoloring is valid, the naive algorithm described above is inefficient. Indeed there are $2^{|\sigma|}$ bicolorings of σ, leading to a exponential algorithm. Yet we will find a way to restrict the number of colorings to test to a polynomial number. The key idea is to look at increasing sequences in the permutation.

4.3.2 Increasing sequences in a valid coloring

First we reformulate the notion of valid coloring thanks to increasing and decreasing sequences.

Proposition 4.47. *Let c be a bicoloring of a permutation σ. Then c is a valid coloring if and only if c respects the following set of rules denoted $\mathcal{R}_8$:*

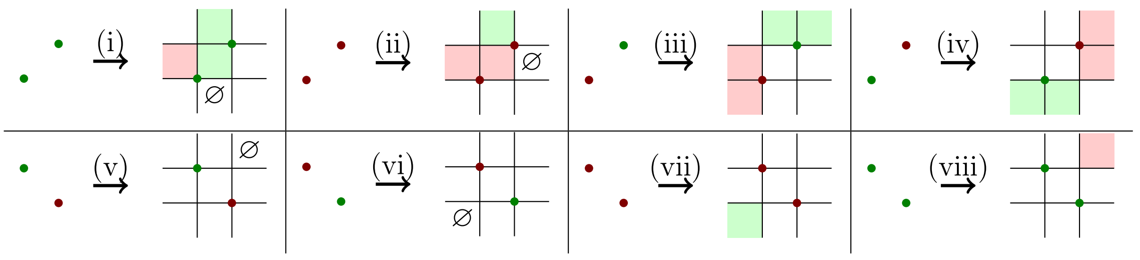


Figure 4.12: Coloring rules $\mathcal{R}_8$

Remark 4.48. Here and in all the following, when a zone of a diagram is colored with R (resp. G), it means than if there are some points lying in this zone, they are in R (resp. G). And when a zone of a diagram has an empty sign, it means than this zone is empty.

For example rule (i) means that if two points (i, σ_i) and (j, σ_j) are in increasing order $i < j$ and $\sigma_i < \sigma_j$ and belong to G then every point (k, σ_k) of the permutation must satisfy:

- If $i < k < j$ then $\sigma_k > \sigma_i$ and (k, σ_k) belongs to G.
- If $k < i$ and $\sigma_i < \sigma_k < \sigma_j$ then (k, σ_k) belongs to R.

Proof. We prove that c is not valid if and only if c violates a rule of $\mathcal{R}_8$. Suppose that c is not valid then c has one of the four colored patterns of Definition 4.37. If c has a pattern 132 then c violates rule (ii) applied to elements 1 and 3 of the pattern 132, as element 2 of the pattern lies in a zone that should be empty. If c has a pattern 213 then c violates rule (i) applied to elements 2 and 3 of the pattern 213, as element 1 of the pattern lies in a zone that should be empty. If c has a pattern $1X2$ then c violates rule (i) applied to elements of G of the pattern $1X2$. If c has a pattern $2/13$ then c violates rule (ii) applied to elements of R of the pattern $2/13$. Conversely if c violates a rule of $\mathcal{R}_8$ then a comprehensive study shows that c has one of the four colored patterns of Definition 4.37 and is not valid. ■

We can use the implication rules of $\mathcal{R}_8$ given in Figure 4.12 to limit the number of bicolorings to test, using the following idea: knowing the coloring of some points in the permutation (either in R or in G), the deduction rules of $\mathcal{R}_8$ can be applied until we obtain either a contradiction or no more rules can be applied. We can try the following algorithm: Set the color of two increasing points of σ, use the implication rules to deduce the color of the other points and test whether the coloring obtained is right. Unfortunately, the implication rules are not sufficient to ensure that given the color of two points, the color of all other points is set. We may have to choose arbitrarily the color of many points. To ensure that the number of bicoloring to test is polynomial, we have to study more precisely the properties of increasing sequences in a valid bicoloring.

Definition 4.49. Let c be a bicoloring of a permutation σ. We call *increasing sequence* RG a pair of points (σ_i, σ_j) such that $i < j$, $\sigma_i < \sigma_j$, $\sigma_i \in R$ and $\sigma_j \in G$. We define in the same way increasing sequences GR, RR or GG.

Rule (iii) of $\mathcal{R}_8$ implies that every increasing sequence RG fixes the color of all points to the left of σ_i below σ_j (which are in R) and to the right of σ_i above σ_j (which are in G). Proposition 4.51 shows that when σ is $\ominus$-indecomposable, the color of the points to the left of σ_i above σ_j is also fixed.

Definition 4.50. Let σ be a permutation. For any indices i and j we call zone Z_{RG} (resp. zone Z_{GR}) the part of the diagram of σ to the left of σ_i or above σ_j (resp. below σ_i or to the right of σ_j). More formally, $Z_{RG}(i,j) = \{\sigma_\ell \mid \ell \leq i \text{ or } \sigma_\ell \geq \sigma_j\}$ and $Z_{GR}(i,j) = \{\sigma_\ell \mid \sigma_\ell \leq \sigma_i \text{ or } \ell \geq j\}$.

Proposition 4.51. *Consider a valid coloring of a $\ominus$-indecomposable permutation σ. If there exists an increasing sequence RG (i.e. two points σ_i, σ_j such that $\sigma_i < \sigma_j$, $i < j$, $\sigma_i \in R$ and $\sigma_j \in G$), then the color of every point in $Z_{RG}(i,j)$ is determined and can be represented as in Figure 4.13, the second diagram being a short representation of this coloring which will be used in the sequel. Moreover, knowing σ_i and σ_j, Algorithm 12 decides the color of the points of zone $Z_{RG}(i,j)$ in linear time w.r.t. $|\sigma|$.*

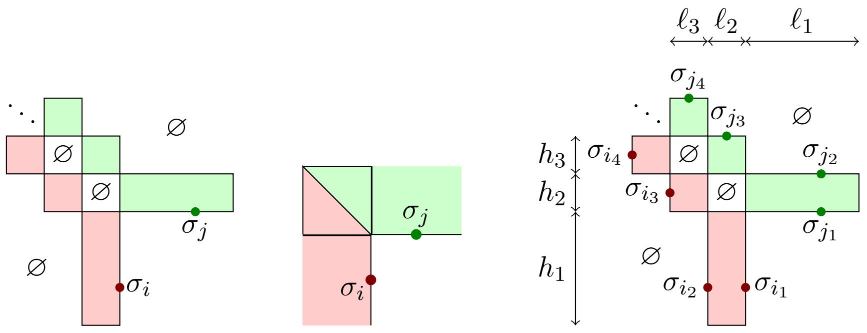


Figure 4.13: An increasing sequence RG $\sigma_i\sigma_j$ fixes the color of all points in zone $Z_{RG}(i,j)$.

Algorithm 12: Color zone Z_{RG}.

```
Data: $\sigma$ a permutation and $i, j$ indices of an increasing sequence $RG$.
Result: The only coloring of zone $Z_{RG}(i,j)$ which may respect rules $\mathcal{R}_8$.
$p \longleftarrow i$;
$q \longleftarrow j$;
while $p \neq 1$ or $\sigma_q \neq n$ do
    Color in $R$ each point of $H = \{\sigma_\ell \mid \ell \leq p \text{ and } \sigma_\ell \leq \sigma_q\}$;
    Color in $G$ each point of $V = \{\sigma_\ell \mid \ell \geq p \text{ and } \sigma_\ell \geq \sigma_q\}$;
    If $H \neq \emptyset$ then $p \longleftarrow \min\{\ell \mid \sigma_\ell \in H\}$;
    If $V \neq \emptyset$ then $q \longleftarrow k$ such that $\sigma_k = \max\{\sigma_\ell \mid \sigma_\ell \in V\}$;
end
```

Proof: Let i_k and j_k be the indices p and q considered by Algorithm 12 at step k and H_k and V_k be the sets H and V considered by the algorithm at step k. We have $i_1 = i$ and $j_1 = j$. We prove that Algorithm 12 builds sequences (σ_{i_k}) and (σ_{j_k}) such that $\sigma_{i_k}\sigma_{j_k}$ is an increasing sequence RG and the color of all points lying in the set $C_k = \{\sigma_\ell \mid i_k \leq \ell \leq i \text{ and } \sigma_j \leq \sigma_\ell \leq \sigma_{j_k}\}$ is uniquely determined by rules $\mathcal{R}_8$ and respects Figure 4.13.

We prove that if $(i_k \neq 1$ or $\sigma_{j_k} \neq n)$ then we can build $\sigma_{i_{k+1}}$ and $\sigma_{j_{k+1}}$ such that $\sigma_{i_{k+1}} < \sigma_{i_k}$ or $\sigma_{j_{k+1}} > \sigma_{j_k}$.

We have $H_k = \{\sigma_\ell \mid \ell \le i_k \text{ and } \sigma_\ell \le \sigma_{j_k}\}$ and $V_k = \{\sigma_\ell \mid \ell \ge i_k \text{ and } \sigma_\ell \ge \sigma_{j_k}\}$ (see Figure 4.14). By rule (iii) applied to σ_{i_k} and σ_{j_k}, $H_k \subset R$ and $V_k \subset G$. Then, different situations may happen depending on whether areas H_k and V_k are empty:

H_k and V_k empty: Then σ is $\ominus$-decomposable which is in contradiction with our hypothesis.

H_k and V_k both non-empty: If both of the colored zones H_k or V_k are non-empty, we have $i_{k+1} = \min\{\ell \mid \sigma_\ell \in H_k\}$ and $\sigma_{j_{k+1}} = \max V_k$ (see Figure 4.14). Then $C_{k+1} = C_k \cup H_k \cup V_k \cup Z_k$ is a partition of C_{k+1}, where $Z_k = \{\sigma_\ell \mid i_{k+1} \le \ell \le i_k \text{ and } \sigma_{j_k} \le \sigma_\ell \le \sigma_{j_{k+1}}\}$ (see Figure 4.14). The only points of C_{k+1} whose color is not determined yet are those of Z_k. If Z_k is not empty consider a point σ_ℓ of Z_k. If $\sigma_\ell \in V$ then rule (i) applied to σ_ℓ and $\sigma_{j_{k+1}}$ is in contradiction with the existence of σ_{i_k}. Hence $\sigma_\ell \in H$ but then rule (ii) applied to $\sigma_{i_{k+1}}$ and σ_ℓ is in contradiction with the existence of σ_{j_k}. So Z_k is empty and the color of all points of C_{k+1} is determined and respects Figure 4.13.

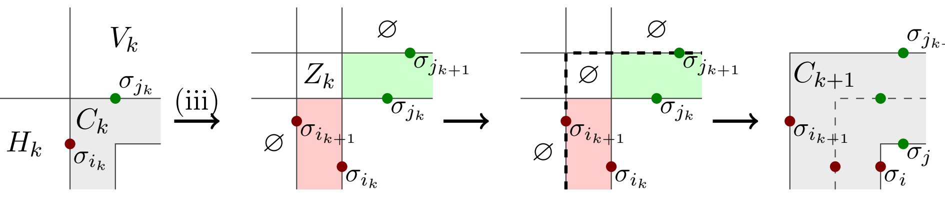


Figure 4.14: Step k of Algorithm 12 coloring zone Z_{RG} when H_k and V_k are both nonempty.

Only one area in H_k and V_k is empty: With the same argument as in the preceding case, we have a new point $\sigma_{j_{k+1}}$ or $\sigma_{i_{k+1}}$ depending on which area is empty and the coloring is computed as shown Figure 4.15 below.

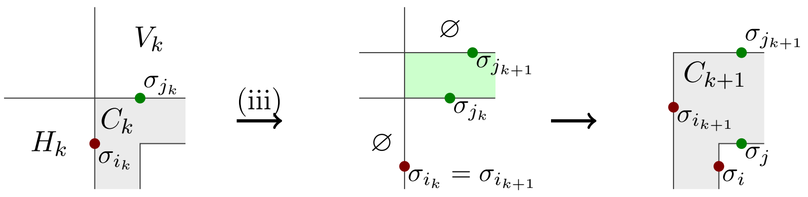


Figure 4.15: Step k of Algorithm 12 coloring zone Z_{RG} when H_k is empty.

We conclude the proof by induction on $n - \sigma_j + \sigma_i - 1$. If $n - \sigma_j + \sigma_i - 1 = 0$, then $\sigma_j = n$ and $i = 1$, thus zone $Z_{RG}(i, j)$ is reduced to σ_i and σ_j, whose color is determined, and Algorithm 12 never performs the while loop.

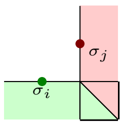
If $n-\sigma_j+\sigma_i-1 > 0$, then from the previous discussion we know that the colors assigned at each step of Algorithm 12 are determined by rules $\mathcal{R}_8$ and respects Figure 4.13. Moreover $i_{k+1} < i_k$ or $\sigma_{j_{k+1}} > \sigma_{j_k}$ so that Algorithm 12 terminates.

Finally Algorithm 12 can be implemented to run in linear time w.r.t. $|\sigma|$, as shown in its detailed version Algorithm 13 below. Indeed Algorithm 13 runs in time $\mathcal{O}(\sum h_k + \sum \ell_k)$ where $h_k = \sigma_{j_k} - \sigma_{j_{k-1}}$ and $\ell_k = i_{k-1} - i_k$ (see Figure 4.13) so that $\sum h_k \le n$ and $\sum \ell_k \le n$. ■

Algorithm 13: Color zone Z_{RG}: detailed version of Algorithm 12

```
**Data**: $\sigma$ a permutation and $i, j$ indices of an increasing sequence $RG$.
**Result**: The only coloring of zone $Z_{RG}(i,j)$ which may respect rules $\mathcal{R}_8$.
Compute $\sigma^{-1}$;
$i_{k-1} \longleftarrow n$;
$j_{k-1} \longleftarrow \sigma^{-1}(1)$;
$i_k \longleftarrow i$;
$j_k \longleftarrow j$;
**while** $i_k \neq 1$ *or* $\sigma_{j_k} \neq n$ **do**
    $M \longleftarrow \sigma_{j_k}$;
    **for** $\ell$ *from* $i_k$ *to* $i_{k-1}$ **do**
        If $\sigma_\ell > \sigma_{j_k}$ then color $\sigma_\ell$ in $G$;
        If $\sigma_\ell > M$ then $M \longleftarrow \sigma_\ell$;
    **end**
    $m \longleftarrow i_k$;
    **for** $\ell$ *from* $\sigma_{j_{k-1}}$ *to* $\sigma_{j_k}$ **do**
        If $\sigma^{-1}(\ell) < i_k$ then color $\sigma_\ell$ in $R$;
        If $\sigma^{-1}(\ell) < m$ then $m \longleftarrow \sigma^{-1}(\ell)$;
    **end**
    $i_k \longleftarrow m$;
    $j_k \longleftarrow \sigma^{-1}(M)$;
**end**
```

We also have a similar result for increasing sequence GR using rule (iv) instead of rule (iii):

Proposition 4.52. *Consider a valid coloring of a $\ominus$-indecomposable permutation σ. If there exists an increasing sequence GR (i.e. two points $\sigma_i < \sigma_j, i < j$ such that $\sigma_i \in G$ and $\sigma_j \in R$), then the color of every point in zone $Z_{GR}(i,j)$ is determined and can be computed in linear time w.r.t. $|\sigma|$. Such a zone will be represented as* 

in the sequel.

Proof: By symmetry using a rotation of 180 degrees, the same proof as for Proposition 4.51 holds. ■

Knowing Proposition 4.51 and Proposition 4.52, to set the color of as many points as possible, we are best of choosing the lower right increasing sequence RG or the upper left increasing sequence GR. Let us now define properly these particular increasing sequences.

We consider a valid bicoloring c of a permutation σ. We define A_{RG} as the set of increasing sequences RG of c.

Lemma 4.53. *Among increasing sequences RG of a valid coloring c such that $A_{RG} \neq \varnothing$, the pair (σ_i, σ_j) which maximizes i first then minimizes σ_j (for i fixed) is the same as the pair that minimizes σ_j first then maximizes i (for σ_j fixed).*

Proof. Let $(\sigma_{i_0}, \sigma_{j_0})$ be the pair that maximizes i_0 first then minimizes σ_{j_0} and $(\sigma_{i_1}, \sigma_{j_1})$ be the pair that minimizes σ_{j_1} first then maximizes i_1. Then by definition $i_0 \geq i_1$ and $\sigma_{j_1} \leq \sigma_{j_0}$.

If $j_1 < i_0$ then $(\sigma_{j_1}, \sigma_{j_0})$ is an increasing sequence GG and rule (i) is in contradiction with $\sigma_{i_0} \in H$ as $j_1 < i_0 < j_0$. If $\sigma_{j_1} < \sigma_{i_0}$ then $(\sigma_{i_1}, \sigma_{i_0})$ is an increasing sequence RR

and rule (ii) is in contradiction with $\sigma_{j_1} \in V$ as $\sigma_{i_1} < \sigma_{j_1} < \sigma_{i_0}$. Hence $(\sigma_{i_0}, \sigma_{j_1})$ is an increasing sequence RG. Then by definition of j_0, $\sigma_{j_0} \leq \sigma_{j_1}$ and by definition of i_1, $i_1 \geq i_0$. So $(\sigma_{i_0}, \sigma_{j_0}) = (\sigma_{i_1}, \sigma_{j_1})$. ■

By the preceding lemma, when $A_{RG} \neq \varnothing$ we can define i_{RG}, j_{RG} as the lower right increasing sequence RG. By symmetry, we can also define i_{GR}, j_{GR} the upper left increasing sequence GR when $A_{GR} \neq \varnothing$, where A_{GR} is the set of increasing sequences GR.

Now we have all the tools to prove that there are only a polynomial number of bicolorings to test. We just have to do a case study depending on whether A_{RG} or A_{GR} is empty.

4.3.3 Case study

Recall that from Proposition 4.21 (p.155), if σ is $\ominus$-decomposable then σ is 2-stack pushall sortable if and only if each $\ominus$-indecomposable block of σ is 2-stack pushall sortable. Thus, we can assume that σ is $\ominus$-indecomposable to decide whether σ is sortable since it is trivial to find the $\ominus$-indecomposable blocks of σ in polynomial time.

In this section, we consider a valid coloring c of a $\ominus$-indecomposable permutation σ. We prove that knowing if there are increasing sequences RG or GR in c and knowing i_{RG}, j_{RG}, i_{GR} and j_{GR} (when they exist), we can deduce the color of every point of σ.

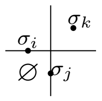
We prove this considering 4 cases depending on whether there are increasing sequences RG or GR in c.

There is no bicolored increasing sequence

If A_{RG} and A_{GR} are both empty, then the coloring is monochromatic:

Proposition 4.54. *Let σ be a $\ominus$-indecomposable permutation and c a valid coloring of σ such that every pattern 12 of σ is monochromatic. Then all points of σ have the same color.*

Proof: Let σ_i and σ_j be two consecutive left-to-right minima of σ. By definition there is no point below σ_i and to the left of σ_j as shown by the empty sign in the following figure

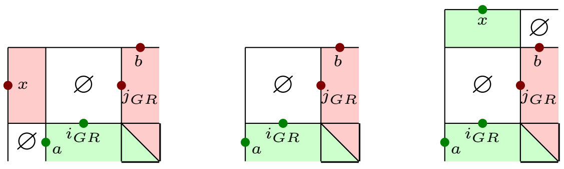


. As σ is $\ominus$-indecomposable, there exists a point σ_k above σ_i and to the right of σ_j. As increasing subsequences are monochromatic, σ_i and σ_k have the same color. The same goes for σ_j and σ_k. Thus σ_i and σ_j have the same color. So all left-to-right minima of σ have the same color. By definition of left-to-right minima, for every non-minimal point σ_ℓ there exists a left-to-right minimum σ_m such that (σ_m, σ_ℓ) is a pattern 12 of σ. Thus σ_ℓ has the same color as σ_m. Hence all points of σ have the same color. ■

There is no increasing sequence RG but some increasing sequences GR

We suppose in this section that there exists at least one increasing sequence GR but no increasing sequence RG. As A_{GR} is non-empty, i_{GR} and j_{GR} are defined. We prove that once i_{GR} and j_{GR} are determined, then the color of every other point of the permutation is fixed.

Proposition 4.55. *Let σ be a $\ominus$-indecomposable permutation and c a valid coloring of σ such that there is no increasing subsequence RG in c and there is an increasing sequence GR in c. Then c has one of the following shapes (where maybe $a = i_{GR}$ or $b = j_{GR}$):*



Proof: The color of every point σ_k such that $k > j_{GR}$ or $\sigma_k < \sigma_{i_{GR}}$ is determined by Proposition 4.52, as shown in the first diagram of Figure 4.16 below. Note that we denote by $*$ the zone where the color of the points is unknown. By maximality of $\sigma_{i_{GR}}$, any point above $\sigma_{i_{GR}}$ and lower left with respect to $\sigma_{j_{GR}}$ is in R. By minimality of j_{GR}, any point to the left of $\sigma_{j_{GR}}$ and top right with respect to $\sigma_{i_{GR}}$ is in G. As no point can be both in R and in G, we know that the zone between $\sigma_{i_{GR}}$ and $\sigma_{j_{GR}}$ is empty, as shown in the second diagram of Figure 4.16.

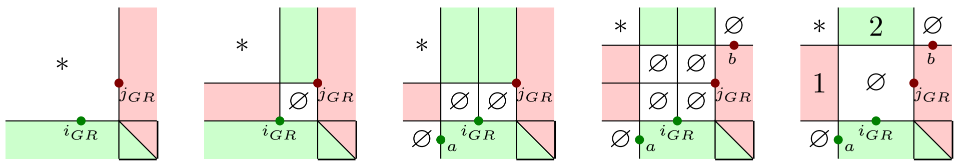


Figure 4.16: Only bicolored increasing subsequences GR exist

Let a be the leftmost point among points below i_{GR} (notice that a may be equal to i_{GR}). Applying rule (i) to a and i_{GR}, we obtain the third diagram (note that if $a = i_{GR}$ the column between i_{GR} and a does not exist). Let b be the topmost point to the right of j_{GR} (b may be equal to j_{GR}). Applying rule (ii) to j_{GR} and b, we obtain the fourth diagram of Figure 4.16 (if $b = j_{GR}$ the column between j_{GR} and b does not exist).

At last, we number two different areas and discuss about the different cases whether these zones are empty or not. These zones are pictured in the fifth diagram of Figure 4.16.

Zone 1 is not empty In this case, let x be the leftmost point inside zone 1. Note that x may be above or below j_{GR}. First diagram of Figure 4.17 illustrates the position of point x. Applying rule (ii) to x and b we obtain the second diagram of Figure 4.17. By hypothesis, there is no increasing sequence RG, thus there is no point in G in the up-right quadrant of x. This leads to the third diagram. At last, if the zone $*$ is not empty, then σ is $\ominus$-decomposable by cutting along the row of b and the column of x. Thus $*$ is empty and all points have a determined color, as in the first diagram of Proposition 4.55.

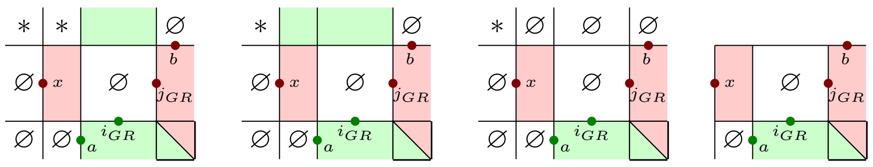


Figure 4.17: Zone 1 is not empty

Zone 1 **is empty** Assume that zone 1 is empty. If zone 2 is also empty then as σ is $\ominus$-indecomposable, zone $*$ is also empty and all points have a determined color, as in the second diagram of Proposition 4.55.

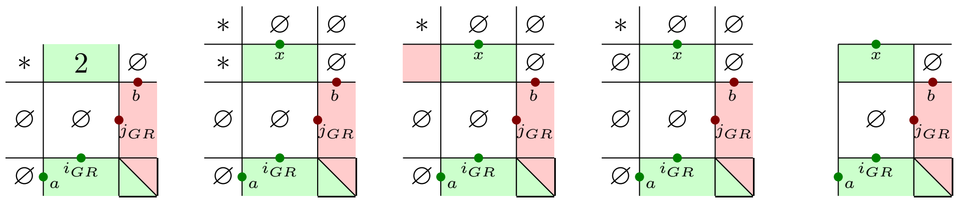
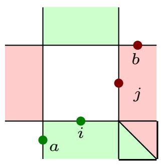


Figure 4.18: Zone 1 is empty

Otherwise, zone 2 is not empty and let x be the topmost point inside zone 2 (x may be to the left or to the right of i_{GR}). This is depicted in the second diagram of Figure 4.18. We apply rule (i) to a and x to obtain the third diagram. As there is no increasing subsequence RG, there is no point of R in the lower left quadrant of x as depicted in the fourth diagram. Moreover, σ is $\ominus$-indecomposable, thus zone $*$ is empty and each point has a determined color, as in the last diagram of Proposition 4.55. ■

Definition 4.56. Let σ be a permutation and i and j two indices of σ such that $\sigma_i\sigma_j$ forms a pattern 12. Set $a = \min\{k \mid \sigma_k \leq \sigma_i\}$ and b such that $\sigma_b = \max\{\sigma_k \mid k \geq j\}$. We define $C_{GR}(\sigma, i, j)$ as the partial bicoloring of σ having the following shape:



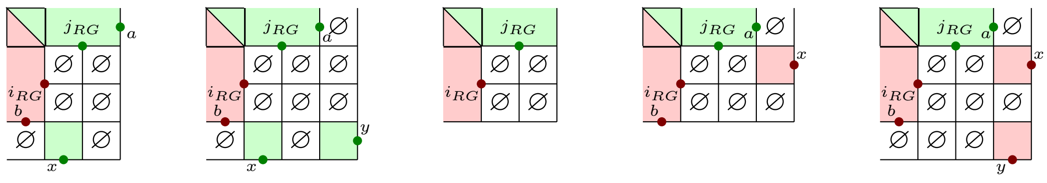
Proposition 4.57. *Let σ be a $\ominus$-indecomposable permutation and c a valid coloring of σ such that there is no increasing subsequence RG in c and there is at least an increasing sequence GR in c. Then $c = C_{GR}(\sigma, i_{GR}, j_{GR})$.*

Proof: This is a direct consequence of Proposition 4.55 and Definition 4.56. ■

All bicolored increasing sequences are labeled RG

We suppose in this section that there exists at least one increasing sequence RG but no increasing sequence GR. As A_{RG} is non-empty, i_{RG} and j_{RG} are defined. We prove that once i_{RG} and j_{RG} are determined, then the color of every other point of the permutation is fixed.

Proposition 4.58. *Let σ be a $\ominus$-indecomposable permutation and c a valid coloring of σ such that there is no increasing subsequence GR in c and there is at least an increasing sequence RG in c. Then c has one of the following shapes (where maybe $a = j_{RG}$ or $b = i_{RG}$):*



Proof: The color of every point σ_k such that $k < i_{RG}$ or $\sigma_k > \sigma_{j_{RG}}$ is determined by Proposition 4.51, as shown in the first diagram of Figure 4.19 below. We denote by $*$ the zone where the color of the points is unknown. By maximality of i_{RG} and minimality of $\sigma_{j_{GR}}$ we know the color of some other points, and as no point can be both in R and in G, we know that the zone between $\sigma_{i_{RG}}$ and $\sigma_{j_{RG}}$ must be empty, as shown in the second diagram of Figure 4.19.

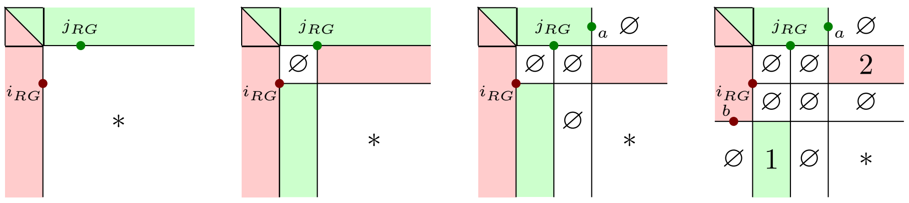


Figure 4.19: All bicolored increasing sequences are labeled RG

Let a be the rightmost point among points above j_{RG} (maybe $a = j_{RG}$). Rule (i) applied to points j_{RG} and a gives the third diagram of Figure 4.19 (note that if $a = j_{RG}$ the column between j_{RG} and a does not exist). Similarly let b be the lowest point among points to the left of i_{RG} (b may be equal to i_{RG}). Rule (ii) applied to b and i_{RG} leads to the fourth diagram of Figure 4.19. Note also that we numbered two specific zones in this diagram and we study now the different cases where they are empty or not.

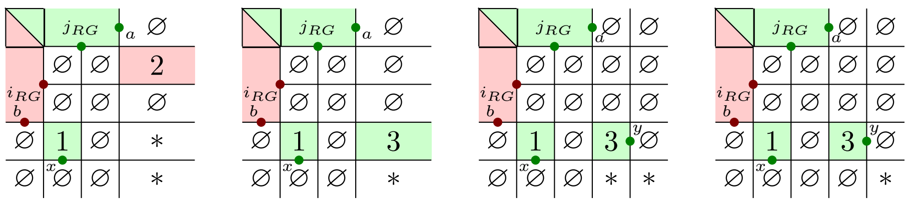


Figure 4.20: Zone 1 is non-empty

Zone 1 is non-empty If zone 1 is non-empty, let x be the lowest point inside this zone (see Figure 4.20). As there do not exist an increasing sequence GR, every point to the top-right of x is in G as shown in the second diagram of Figure 4.20, where we define a zone 3. If zone 3 is empty then zone $*$ is empty as σ is $\ominus$-indecomposable, hence every point has a assigned color as in the first diagram of Proposition 4.58. If zone 3 is non-empty, let y be the rightmost point inside this zone as shown in the third diagram. Applying rule (i) to x and y add another empty zone, leading to the last diagram. As σ is $\ominus$-indecomposable, zone $*$ is empty and all points have an assigned color as in the second diagram of Proposition 4.58.

Zone 1 is empty Assume that zone 1 is empty. If zone 2 is also empty then as σ is $\ominus$-indecomposable, zone $*$ is also empty and all points have a determined color, as in the third diagram of Proposition 4.58.

If zone 2 is non-empty, let x be the rightmost point of zone 2 (see Figure 4.21 below).

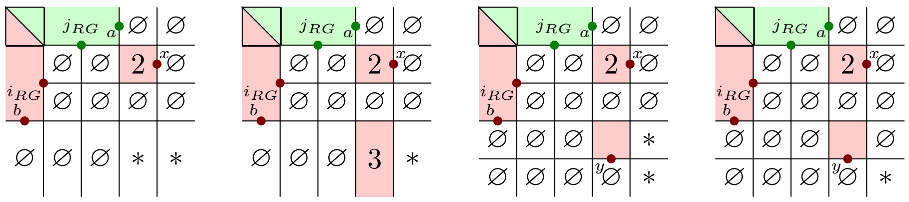
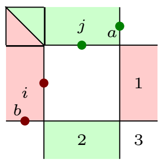


Figure 4.21: Zone 1 is empty

As there is no increasing subsequence GR, all points in the lower left quadrant of x lie in R as shown in the second diagram of Figure 4.21 where we define a zone 3. If zone 3 is empty then as σ is $\ominus$-indecomposable zone $*$ is also empty and all points have a determined color, as in the fourth diagram of Proposition 4.58. Otherwise zone 3 is non-empty and let y be the lowest point in zone 3 as depicted in the third diagram. We apply rule (ii) to x and y leading to the fourth diagram. As σ is $\ominus$-indecomposable, zone $*$ is empty and all points have a determined color, as in the last diagram of Proposition 4.58. ■

Definition 4.59. Let σ be a permutation and i and j two indices of σ such that $\sigma_i\sigma_j$ forms a pattern 12. Set $a = \max\{k \mid \sigma_k \geq \sigma_j\}$ and b such that $\sigma_b = \min\{\sigma_k \mid k \leq i\}$. We define $C_{RG}(\sigma, i, j)$ as the partial bicoloring of σ having the following shape:



where points of zone 3 are in G if zone 1 is empty and zone 2 is nonempty, in R if zone 1 is nonempty and zone 2 is empty, and have no color otherwise.

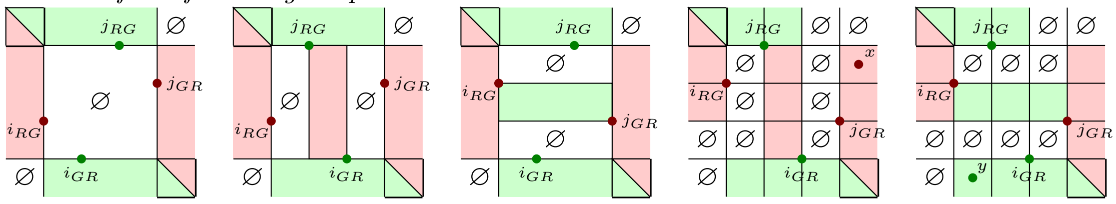
Proposition 4.60. *Let σ be a $\ominus$-indecomposable permutation and c a valid coloring of σ such that there is no increasing subsequence GR in c and there is at least an increasing sequence RG in c. Then $c = C_{RG}(\sigma, i_{RG}, j_{RG})$.*

Proof: This is a direct consequence of Proposition 4.58 and Definition 4.59. ■

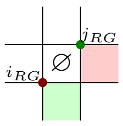
There exist increasing sequences labeled GR and RG

In this section we study the last case that remains to be dealt with, i.e. there is at least one increasing sequence colored RG and at least one colored GR. As A_{GR} and A_{RG} are non-empty, i_{GR}, j_{GR}, i_{RG} and j_{RG} are defined. We prove that once i_{GR}, j_{GR}, i_{RG} and j_{RG} are determined, then the color of every other point of the permutation is fixed.

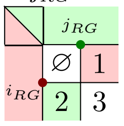
Proposition 4.61. *Let σ be a permutation and c a valid coloring of σ such that there exists an increasing sequence colored GR and an increasing sequence colored RG. Then c has one of the following shapes:*



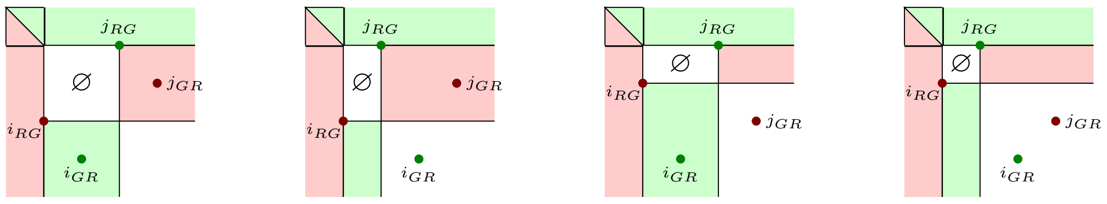
Proof: By maximality of i_{RG} and minimality of $\sigma_{j_{RG}}$ we have:



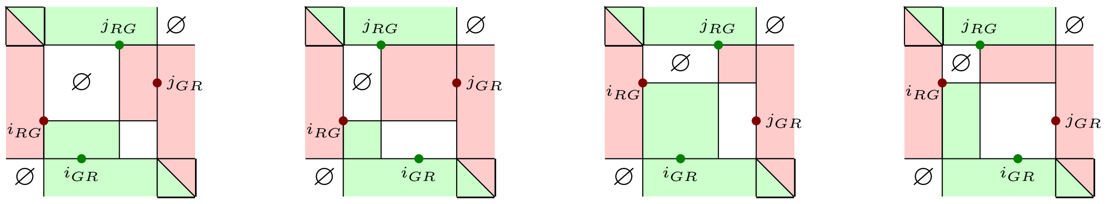
By Proposition 4.51 we obtain:



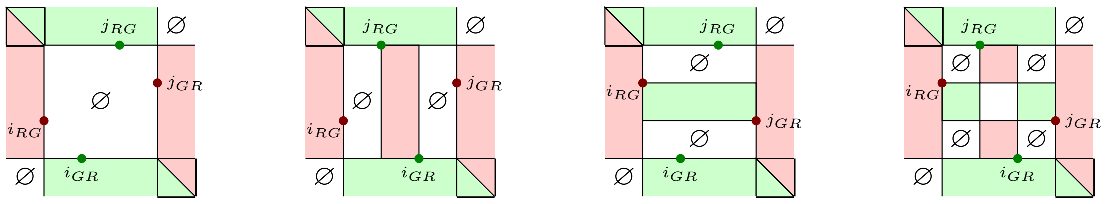
Recall that there exists an increasing sequence GR. We know that $i_{GR} \in G$. If i_{GR} is above j_{RG}, then $j_{GR} \in G$ since j_{GR} is on the top right of i_{GR}, but by definition $j_{GR} \in R$. Thus i_{GR} lies in quadrant 2 or 3 and j_{GR} in quadrant 1 or 3. Hence the coloring c has one of the 4 following shapes:



Applying Proposition 4.52 to i_{GR} and j_{GR} we obtain these new diagrams:



Finally, using maximality of $\sigma_{i_{GR}}$ and minimality of j_{GR}, we obtain:



In the first 3 diagrams, the color of each point is determined as in the first 3 diagrams of the statement of Proposition 4.61.

This leaves us with the last diagram reproduced in Figure 4.22 below for which we have again to consider several cases. Note that in this diagram we named several zones whose emptiness is relevant and we denote once more the unknown zone by $*$.

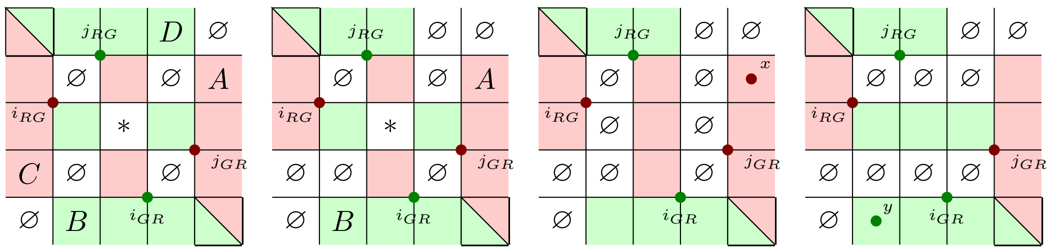


Figure 4.22: There exist increasing sequences labeled RG and GR

Applying rule (vii) to i_{RG} and j_{GR} implies that zone C is empty. Similarly, rule (viii) applied to j_{RG} and i_{GR} proves that zone D is empty. If there exists a point x in zone A, then applying rule (ii) to j_{GR} and x, all points in $*$ are determined –they lie in R– as shown in the third diagram. Symmetrically, if there exists a point y in B then applying rule (i) to y and i_{GR}, all points in $*$ should be in G–see the last diagram. These cases correspond to the 2 last diagrams of Proposition 4.61.

Thus this leaves us with the case where both A and B are empty. We show that this case is not possible.

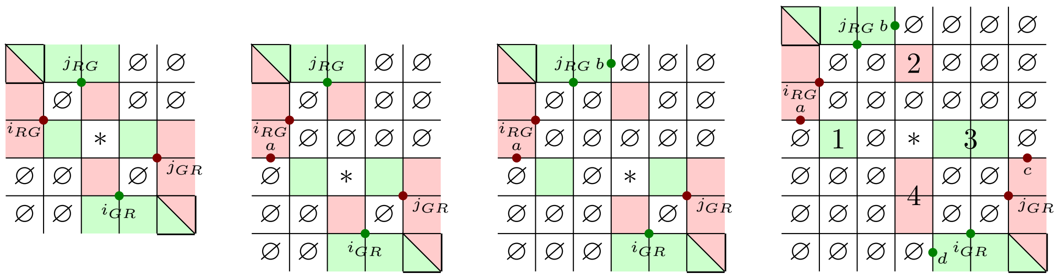


Figure 4.23: A and B are empty

A and B are empty Then the permutation is colored as shown in the first diagram of Figure 4.23. Let a be the lowest point among points to the left of i_{RG} (a may be equal to i_{RG}). Rule (ii) applied to a and i_{RG} implies the coloring shown in the second diagram –notice that if $a = i_{RG}$, the line between a and i_{GR} does not exist. Similarly, define b as the rightmost point among points above j_{RG} (b may be equal to j_{RG}). Rule (i) applied to b and j_{RG} leads to the third diagram. At last we consider the topmost point c among points to the right of j_{GR} (maybe $c = j_{GR}$) and we apply rule (ii) to c and j_{GR}. We also introduce d as the leftmost point among points below i_{GR} (maybe $d = i_{GR}$). Rule (i) applied to d and i_{GR} leads to the last diagram where different zones are numbered. We now study different cases according whether zone 1 is empty or not, and we prove that both are excluded.

Zone 1 is empty Suppose that zone 1 is empty. As σ is $\ominus$-indecomposable then zone 2 must contain at least one point. Denote by x the rightmost point of this zone. Figure 4.24 illustrates the proof.

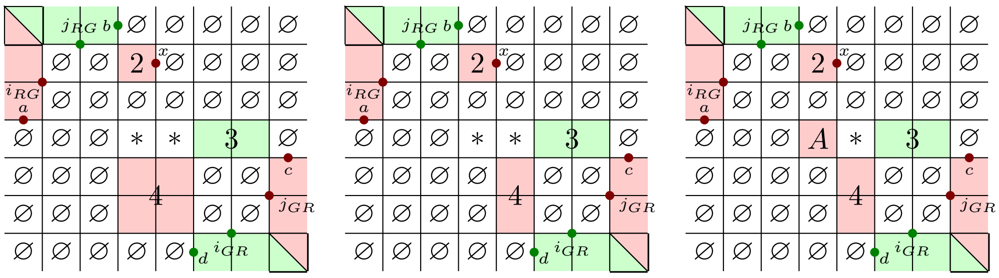


Figure 4.24: Zone 1 is empty.

Rule (vii) applied to x and c leads to the second diagram. Moreover as (i_{GR}, j_{GR}) is the topmost and leftmost increasing sequence GR, all points to the lower left quadrant of x lie in R, leading to the third diagram where we define a zone A.

Zone 4 is not empty We prove that this case is not possible. If zone 4 is not empty, let y be its leftmost point (above or below j_{GR}) as illustrated in the first diagram of Figure 4.25.

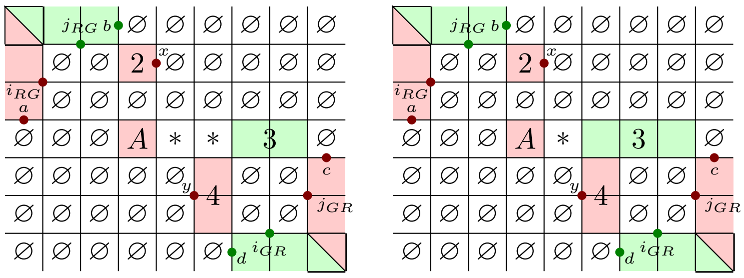


Figure 4.25: Zone 1 is empty and zone 4 is not empty

We apply rule (ii) to y and c and obtain the second diagram. But (i_{RG}, j_{RG}) is lowest-right increasing sequence RG, thus there is no point labeled G in the above-right quadrant of y. Hence zone 3 is empty which is forbidden as σ is $\ominus$-indecomposable.

Zone 4 is empty We prove that this case is also not possible. Suppose that zone 4 is empty as illustrated in the first diagram of Figure 4.26.

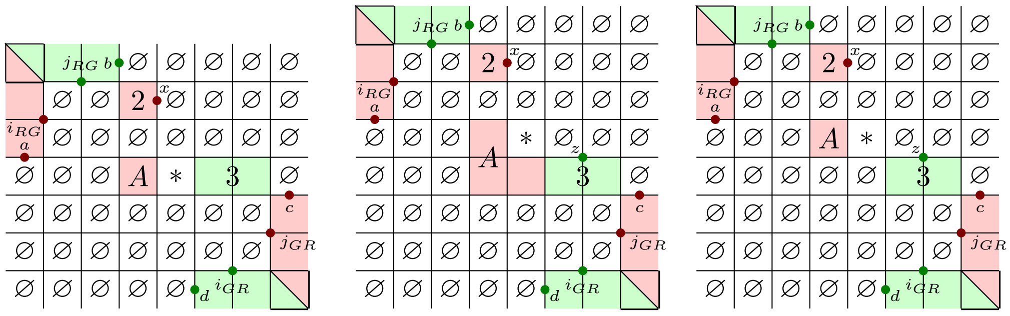


Figure 4.26: Zone 1 is empty and zone 4 is empty.

As σ is $\ominus$-indecomposable, zone 3 is non-empty. Let z be the topmost point of zone 3 (it may be to the left or to the right of i_{GR}). Applying rule (i) to z and d we obtain the second diagram. But (i_{RG}, j_{RG}) is the lowest right increasing sequence labeled RG, hence there is no point labeled R in the below-left quadrant of z –see diagram 3. But then σ is $\ominus$-decomposable which is forbidden.

Zone 1 is not empty Suppose that zone 1 of Figure 4.23 is non-empty. We prove that this case is also not possible. Define x as the lowest point of this zone as shown in the first diagram of Figure 4.27.

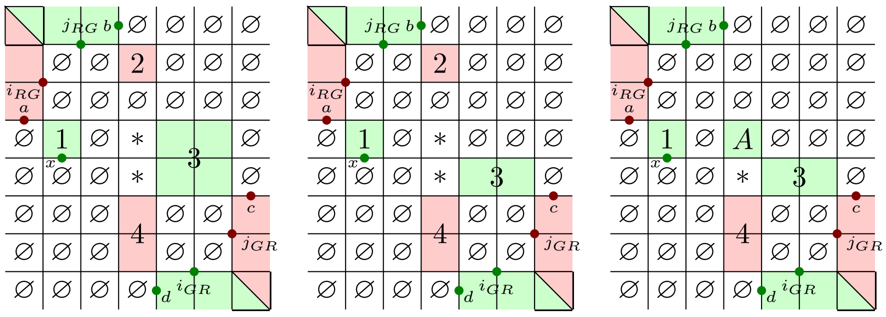


Figure 4.27: Zone 1 is not empty.

Rule (viii) applied to x and d implies the second diagram. Moreover, as (i_{GR}, j_{GR}) is the leftmost-top increasing sequence labeled GR, all points to the top right of x are in G, leading to the last diagram.

Zone 3 is not empty If zone 3 is not empty, let y be its topmost point (y may be to the left or to the right of i_{GR}) as pictured in Figure 4.28.

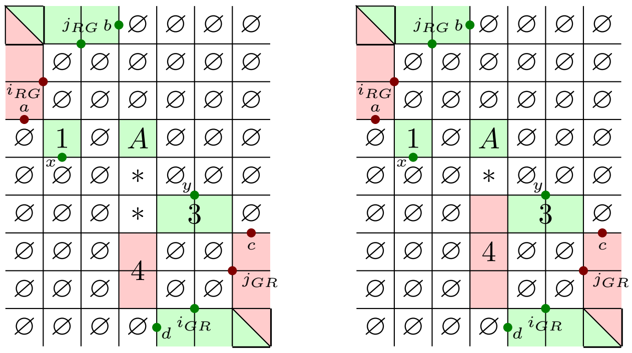


Figure 4.28: Zone 1 is not empty and zone 3 is not empty.

Rule (i) applied to d and y gives the second diagram. But (i_{RG}, j_{RG}) is the bottom-rightmost increasing sequence RG, hence no point in the lower left quadrant of y lies in R. Thus zone 4 is empty and σ is $\ominus$-decomposable which is forbidden.

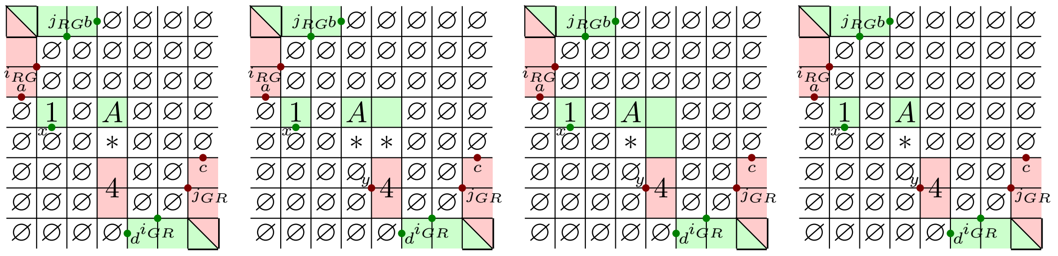
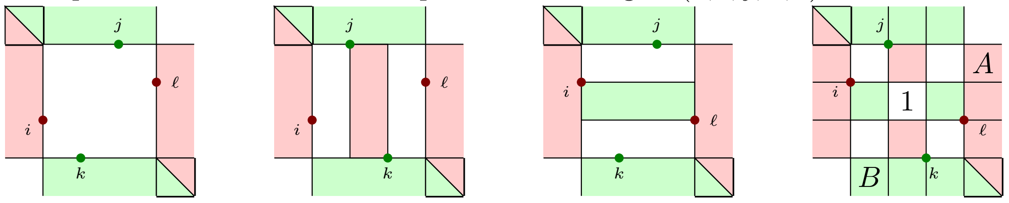


Figure 4.29: Zone 1 is not empty and zone 3 is empty

Zone 3 is empty Figure 4.29 illustrates the proof. As σ is $\ominus$-indecomposable, zone 4 is not empty. Let y be the leftmost point inside zone 4 – either above or below j_{GR} – as depicted in the second diagram. Rule (ii) applied to y and c leads to the third diagram. But (i_{RG}, j_{RG}) is the bottom-rightmost increasing sequence RG, hence no point of G lies in the top-right quadrant of y leading to the fourth diagram. So σ is $\ominus$-decomposable which is forbidden.

This ends the cases study, proving that zone A and B cannot be both empty. ■

Definition 4.62. Let σ be a permutation and i, j, k, ℓ four indices of σ such that $\sigma_i\sigma_j$ and $\sigma_k\sigma_\ell$ form patterns 12. We define the partial bicoloring $C_*(\sigma, i, j, k, \ell)$ of σ as follows.



If σ_i, σ_ℓ, σ_k and σ_j have a relative position corresponding to one of the above diagrams, then we define $C_*(\sigma, i, j, k, \ell)$ as the partial bicoloring of σ having the corresponding shape, where in the last diagram points of zone 1 are in R if zone A is nonempty, else in G if zone B is nonempty, and have no color otherwise.

Otherwise $C_*(\sigma, i, j, k, \ell)$ is the partial coloring with no point colored.

Proposition 4.63. *Let σ be a $\ominus$-indecomposable permutation and c a valid coloring of σ such that there exist an increasing sequence RG and an increasing sequence GR in c. Then $c = C_*(\sigma, i_{RG}, j_{RG}, i_{GR}, j_{GR})$.*

Proof: This is a consequence of Proposition 4.61 and Definition 4.62, noticing that if there exists a point x in zone A, then applying rule (ii) to ℓ and x, all points in zone 1 belong to R, and if there exists a point y in zone B, then applying rule (i) to y and k, all points in zone 1 belong to G. ■

4.3.4 A first polynomial algorithm

Algorithm 14: ColoringIndecomposable1(σ)

```
**Data**: $\sigma$ a $\ominus$-indecomposable permutation (whose size is denoted $n$).
**Result**: The set $E$ of valid colorings of $\sigma$
**for** $c$ bicoloring of $\sigma$ being one of
    $c$ is monochromatic $R$
    $c$ is monochromatic $G$
    $c = C_{GR}(\sigma, i, j)$ or $C_{RG}(\sigma, i, j)$ for $i \in [1..n]$ and $j \in [i..n]$ s.t. $\sigma_j > \sigma_i$
    $c = C_*(\sigma, i, j, k, \ell)$ for $i \in [1..n]$ and $j \in [i..n]$ s.t. $\sigma_j > \sigma_i$ and
                for $k \in [1..n]$ and $\ell \in [k..n]$ s.t. $\sigma_\ell > \sigma_k$
**do**
    If all points of $\sigma$ are colored and $c$ is valid then add $c$ to $E$;
```

Proposition 4.64. *Algorithm 14 computes in time $\mathcal{O}(n^5)$ the set of valid colorings of any $\ominus$-indecomposable permutation σ.*

Proof: Let σ be a $\ominus$-indecomposable permutation of size n and c a valid coloring of σ. Then from Propositions 4.54, 4.57, 4.60 and 4.63, c is either monochromatic, or $C_{GR}(\sigma, i, j)$ or $C_{RG}(\sigma, i, j)$ for some $i \in [1..n]$ and some $j \in [i..n]$ such that $\sigma_j > \sigma_i$, or $c = C_*(\sigma, i, j, k, \ell)$ for some $i \in [1..n]$, some $j \in [i..n]$ such that $\sigma_j > \sigma_i$, some $k \in [1..n]$ and some $\ell \in [k..n]$ such that $\sigma_\ell > \sigma_k$. Thus c is computed by Algorithm 14 and added to E as it is valid. Conversely, each coloring added to E is a valid bicoloring of σ.

Now consider the complexity of Algorithm 14. There are $\mathcal{O}(n^4)$ colorings computed. Indeed there are two monochromatic colorings, $\mathcal{O}(n^2)$ colorings $C_{GR}(\sigma, i, j)$ or $C_{RG}(\sigma, i, j)$ and $\mathcal{O}(n^4)$ colorings $C_*(\sigma, i, j, k, \ell)$. Moreover each coloring is computed in linear time using Propositions 4.51 and 4.52 and checking if the coloring is valid is done in linear time using Proposition 4.46. Hence Algorithm 14 runs in time $\mathcal{O}(n^5)$. ■

4.4 An optimal algorithm

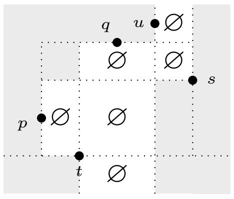
4.4.1 Rooting colorings

In this section we show how each diagram of Propositions 4.55, 4.58 and 4.61 can be rooted in a given point such that each point i_{GR}, i_{RG}, j_{GR} and j_{RG} can be deduced from this one. Moreover, given a diagram we show that we can assign colors to points of the permutations lying in a colored zone of the diagram in linear time.

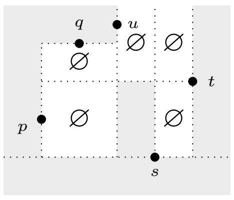
Definition 4.65. Let σ be a permutation and $s \in [1..|\sigma|]$. We set

$C_1(\sigma, s) = C_{GR}(\sigma, s, t)$ where $t = \min\{k \mid k > s \text{ and } \sigma_k > \sigma_s\}$
$C_2(\sigma, s) = C_{GR}(\sigma, t, s)$ where t is such that $\sigma_t = \max\{\sigma_k \mid k < s \text{ and } \sigma_k < \sigma_s\}$
$C_3(\sigma, s) = C_{RG}(\sigma, s, t)$ where t is such that $\sigma_t = \min\{\sigma_k \mid k > s \text{ and } \sigma_k > \sigma_s\}$
$C_4(\sigma, s) = C_{RG}(\sigma, t, s)$ where $t = \max\{k \mid k < s \text{ and } \sigma_k < \sigma_s\}$

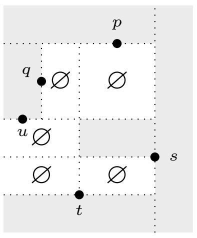
$C_5(\sigma, s) = C_*(\sigma, p, q, t, s)$ with
$t = \max\{k \mid k < u \text{ and } \sigma_k < \sigma_s\}$
with $u = \max\{k \mid k < s \text{ and } \sigma_k > \sigma_s\}$,
$p = \max\{k \mid k < t \text{ and } \sigma_t < \sigma_k < \sigma_s\}$ and
q such that $\sigma_q = \min\{\sigma_k \mid t < k \le u\}$



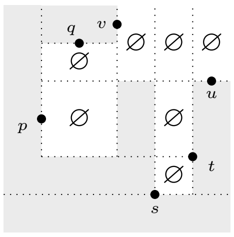
$C_6(\sigma, s) = C_*(\sigma, p, q, s, t)$ with
$t = \min\{k \mid k > s \text{ and } \sigma_k > \sigma_s\}$,
$u = \max\{k \mid k < t \text{ and } \sigma_k > \sigma_t\}$,
$p = \max\{k \mid k < u \text{ and } \sigma_s < \sigma_k < \sigma_t\}$ and
q such that $\sigma_q = \min\{\sigma_k \mid \sigma_k > \sigma_t \text{ and } p < k \le u\}$



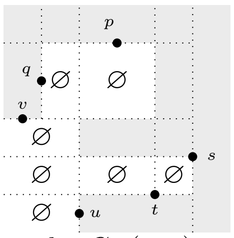
$C_7(\sigma, s) = C_*(\sigma, q, p, t, s)$ with
t such that $\sigma_t = \max\{\sigma_k \mid k < s \text{ and } \sigma_k < \sigma_s\}$,
u such that $\sigma_u = \min\{\sigma_k \mid k < t \text{ and } \sigma_k > \sigma_t\}$,
p such that $\sigma_p = \min\{\sigma_k \mid \sigma_k > \sigma_u \text{ and } t < k < s\}$ and
$q = \max\{k \mid k < p \text{ and } \sigma_u \le \sigma_k < \sigma_p\}$



$C_8(\sigma, s) = C_*(\sigma, p, q, s, t)$ with
$t = \min\{k \mid k > s \text{ and } \sigma_k > \sigma_s\}$,
u such that $\sigma_u = \max\{\sigma_k \mid \sigma_k > \sigma_t \text{ and } k > t\}$,
$v = \max\{k \mid k < u \text{ and } \sigma_k > \sigma_u\}$,
$p = \max\{k \mid k < v \text{ and } \sigma_t < \sigma_k < \sigma_u\}$ and
q such that $\sigma_q = \min\{\sigma_k \mid \sigma_k > \sigma_u \text{ and } p < k \le v\}$



$C_9(\sigma, s) = C_*(\sigma, q, p, t, s)$ with
t such that $\sigma_t = \max\{\sigma_k \mid k < s \text{ and } \sigma_k < \sigma_s\}$,
$u = \min\{k \mid k < t \text{ and } \sigma_k < \sigma_t\}$,
v such that $\sigma_v = \min\{\sigma_k \mid k < u \text{ and } \sigma_k > \sigma_u\}$,
p such that $\sigma_p = \min\{\sigma_k \mid \sigma_k > \sigma_v \text{ and } u < k < t\}$ and
$q = \max\{k \mid k < p \text{ and } \sigma_v \le \sigma_k < \sigma_p\}$



Note that if for some $m \in [1..9]$, the index t does not exist, then we take $C_m(\sigma, s)$ as the empty partial coloring (no point has a color). The same goes for indices p and q and $m \in [5..9]$.

Proposition 4.66. *Let σ be a $\ominus$-indecomposable permutation and c a valid coloring of σ which is not monochromatic. Then there exists $s \in [1..|\sigma|]$ and $m \in [1..9]$ such that $c = C_m(\sigma, s)$.*

Proof: As c is not monochromatic, then from Proposition 4.54 σ has at least a pattern 12 which is not monochromatic.

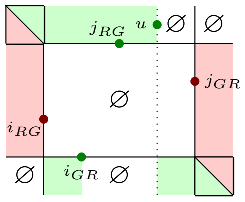
If there is no increasing subsequence RG in c then there is at least an increasing sequence GR in c. Thus from Proposition 4.57, $c = C_{GR}(\sigma, i_{GR}, j_{GR})$. Moreover, c has one of the three shapes described in Proposition 4.55. If the shape of c is one of the two first shapes, then j_{GR} is the leftmost point in the upper-right quadrant of i_{GR} and $c = C_1(\sigma, i_{GR})$.

Otherwise the shape of c is the third one and i_{GR} is the topmost point in the bottom-left quadrant of j_{GR} thus $c = C_2(\sigma, j_{GR})$.

If there is an increasing subsequence RG in c but no increasing sequence GR, then from Proposition 4.60 $c = C_{RG}(\sigma, i_{RG}, j_{RG})$. Moreover c has one of the 5 shapes described in Proposition 4.58. If the shape of c is one of the three first shapes, then j_{RG} is the lowest point in the upper-right quadrant of i_{RG} and $c = C_3(\sigma, i_{RG})$. Otherwise the shape of c is one of the two last shapes and i_{RG} is the rightmost point in the bottom-left quadrant of j_{RG} thus $c = C_4(\sigma, j_{RG})$.

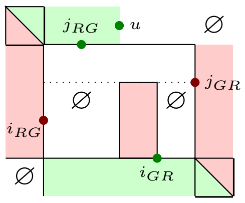
We are left with the case where there is an increasing subsequence RG and an increasing sequence GR in c. Then from Proposition 4.63, $c = C_*(\sigma, i_{RG}, j_{RG}, i_{GR}, j_{GR})$. Moreover c has one of the 5 shapes described in Proposition 4.61.

If the shape of c is the first one, let u be the rightmost point in the top left quadrant of j_{GR} (maybe $u = j_{RG}$). Then applying rule (ii) to i_{GR} and u, c has the following shape:



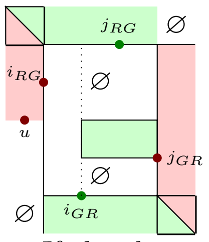
Thus i_{GR} is the rightmost point on the left of u below j_{GR}. Moreover i_{RG} is the rightmost point on the top left quadrant of i_{GR} below j_{GR}. Finally j_{RG} is the lowest point on the right of i_{GR} and on the left of u. Hence $c = C_5(\sigma, j_{GR})$.

If the shape of c is the second one of Proposition 4.61, let u be the rightmost point in the top right quadrant of j_{RG} (maybe $u = j_{RG}$). From rule (viii) applied to j_{RG} and i_{GR}, $u < i_{GR}$. Then applying rule (ii) to i_{RG} and j_{GR} and applying rule (i) to j_{RG} and u if $u \neq j_{RG}$, c has the following shape:



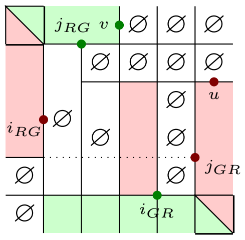
Thus j_{GR} is the leftmost point in the upper right quadrant of i_{GR} and u is the rightmost point in the upper left quadrant of j_{GR}. Moreover i_{RG} is the rightmost point to the left of u, below j_{GR} and above i_{GR}. Finally, j_{RG} is the lowest point in the upper left quadrant of j_{GR} and to the right of i_{RG}. Hence $c = C_6(\sigma, i_{GR})$.

If the shape of c is the third one of Proposition 4.61, let u be the lowest point in the lower left quadrant of i_{RG} (maybe $u = i_{RG}$). From rule (vii) applied to i_{RG} and j_{GR}, $\sigma_u > \sigma_{j_{GR}}$. Then applying rule (i) to i_{GR} and j_{RG} and applying rule (ii) to u and i_{RG} if $u \neq i_{RG}$, c has the following shape:



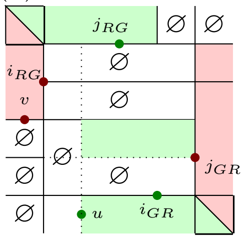
Thus i_{GR} is the topmost point in the lower left quadrant of j_{GR} and u is the lowest point in the upper left quadrant of i_{GR}. Moreover j_{RG} is the lowest point above u, to the right of i_{GR} and to the left of j_{GR}. Finally, i_{RG} is the rightmost point to the lower left of j_{RG} and above u. Hence $c = C_7(\sigma, j_{GR})$

If the shape of c is the fourth one of Proposition 4.61, let u be the topmost point to the upright quadrant of j_{GR} and v be the rightmost point to the top-right quadrant of j_{RG} (maybe $v = j_{RG}$). Note that u is above i_{RG} as u is above x (u is the topmost point) which is above i_{RG}. Then applying rule (ii) to i_{RG} and u and applying rule (iii) to j_{RG} and v if $v \neq j_{RG}$, c has the following shape:



Thus j_{GR} is the leftmost point in the up right quadrant of i_{GR}. Point u is the topmost point in the upper right quadrant of j_{GR}. Point v is the rightmost point in the upper left quadrant of u. Then i_{RG} is the rightmost point to the left of v, below u and above i_{GR}. At last, j_{RG} is the lowest point above u, to the right of i_{RG} and to the left of v. Hence $c = C_8(\sigma, i_{GR})$

If the shape of c is the last one of Proposition 4.61, let u be the leftmost point in the lower left quadrant of i_{GR} and v be the lowest point in the lower left quadrant of i_{RG} (maybe $v = i_{RG}$). Note that u is to the left of j_{RG} as it is to the left of y (u is the leftmost point) and y is to the left of j_{RG}. Then applying rule (i) to u and j_{RG} and applying rule (ii) to v and i_{RG} if $v \neq i_{RG}$, c has the following shape:



Thus i_{GR} is the topmost point in the lower left quadrant of j_{GR} and u is the leftmost point in the lower left quadrant of i_{GR}. Moreover v is the lowest point in the upper left quadrant of u and j_{RG} is the lowest point above v and to the right of u and to the left of i_{GR}. Finally, i_{RG} is the rightmost point in the lower left quadrant of j_{RG} and above v. Hence $c = C_9(\sigma, j_{GR})$, concluding the proof. ■

Proposition 4.67. *Let σ be a permutation, $s \in [1..|\sigma|]$ and $m \in [1..9]$. Then we can compute $C_m(\sigma, s)$, test whether all points of σ are colored and check whether $C_m(\sigma, s)$ is valid in linear time w.r.t. $|\sigma|$.*

Proof: Knowing m and s, we can find in linear time the corresponding points t, p and q of Definition 4.65. The result is then a consequence of Propositions 4.51, 4.52 and 4.46. ■

4.4.2 Algorithm for $\ominus$-indecomposable permutations

In this section, we give a quadratic algorithm computing the set of valid colorings of a $\ominus$-indecomposable permutation, and prove that this algorithm is optimal.

This algorithm relies on Proposition 4.66 which proves that any $\ominus$-indecomposable permutation has at most a linear number of valid colorings, all of the kind $C_m(\sigma, s)$. It is then sufficient to check these colorings. By definitions, colorings $C_m(\sigma, s)$ are partial colorings. However if a point is uncolored, it means that it lies in a zone of the diagram that should be empty (as depicted in Propositions 4.55,4.58 and 4.61) hence the coloring has to be rejected. More precisely, the algorithm is the following:

Algorithm 15: ColoringIndecOptimal(σ)

```
Data: $\sigma$ a $\ominus$-indecomposable permutation
Result: The set $E$ of valid colorings of $\sigma$
for $c$ bicoloring of $\sigma$ monochromatic $R$ or monochromatic $G$ do
    If $c$ is valid then add $c$ to $E$;
for $s$ from 1 to $|\sigma|$ do
    for $m$ from 1 to 9 do
        $c = C_m(\sigma, s)$;
        If all points of $\sigma$ are colored and $c$ is valid then add $c$ to $E$;
```

Theorem 4.68. *A $\ominus$-indecomposable permutation of size n has at most $9n$ valid colorings. Those colorings can be computed using Algorithm 15 in time $\mathcal{O}(n^2)$ which is optimal.*

Proof: This is a consequence of Propositions 4.66 and 4.67, except for the optimality (note that we may expect $9n + 2$ instead of $9n$ in the statement of the theorem, but if a valid coloring $c = C_m(\sigma, s)$ with $m \in [5..9]$ then $s \neq 1$; and $2 + 4n + 5(n-1) \leq 9n$). Finally Proposition 4.69 below implies that the set of valid colorings of the identity of size n has size $2n^2$ (it is a set of $2n$ coloring having each n colored points), proving the optimality. ■

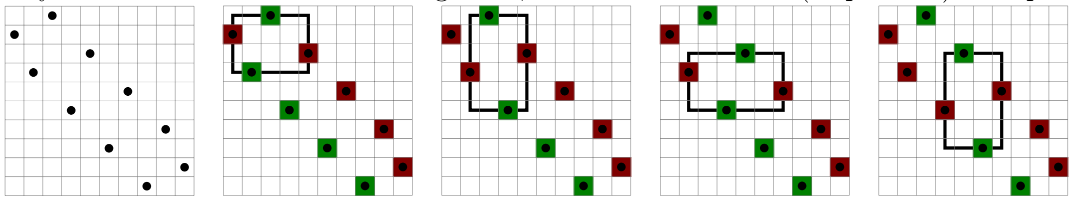
Proposition 4.69. *For all n the identity of size n has exactly $2n$ valid colorings.*

Proof: Let σ be the identity of size n. For all k between 1 and n let C^k_{RG} (resp. C^k_{GR}) be the coloring of σ such that for all i, σ is in R (resp. G) if $i \leq k$ and in G (resp. R) otherwise. Then it is straightforward to check that C^k_{RG} (resp. C^k_{GR}) is a valid coloring of σ, i.e. has no pattern 132, 213, $1X2$ or $2/13$. Conversely if c is a valid coloring of the identity, rules (iii) and (iv) of $\mathcal{R}_8$ imply that there are at most one pair of consecutive points whose colors are different. So c is some C^k_{RG} or some C^k_{GR}. ∎

The property of having a linear number of colorings is not a special case of the identity. Indeed there are some simple permutations that also have a linear number of colorings, as shown in the next proposition.

Proposition 4.70. *Permutations $\sigma^{(n)} = (2n-1)(2n-3)(2n)(2n-5)(2n-2)(2n-7)(2n-4)\dots 5\,8\,3\,6\,1\,4\,2$ of size $2n$ have at least $2n-3$ valid colorings.*

Proof. To prove the result, we exhibit $2n-3$ colorings. We look at set of four points of σ whose indices (resp. values) are consecutive and which form a pattern 2 4 1 3 (resp. 3 1 4 2). Notice that they can be taken to be $\{i_{RG}, i_{GR}, j_{RG}, j_{GR}\}$ in a valid coloring of σ respecting to the third (resp. second) diagram of Proposition 4.61, as shown in the figure below. This way we obtain $2n-3$ valid colorings of σ, since there are $n-2$ (resp. $n-1$) such patterns.

 ∎

4.4.3 Final algorithm

In this section, we give a quadratic algorithm computing an encoding of the set of all pushall sorting processes of a given permutation.

Recall first that if a permutation σ is $\ominus$-decomposable, then it is 2-stack pushall sortable if and only if each of the blocks of its decomposition is 2-stack pushall sortable. Moreover to obtain a sorting process of σ we can just push elements of the first block according to any pushall sorting process of it, then elements of the second and so on, before popping out all the elements. This means that the set of valid colorings of a $\ominus$-decomposable permutation is the product of valid colorings of each block, as stated in Proposition 4.71 below. When π is a pattern of σ and c a coloring of σ, we denote by $c|_\pi$ the restriction of c to π.

Proposition 4.71. *Let σ be a permutation and $Col(\sigma)$ the set of valid colorings of σ. If $\sigma = \ominus[\pi_1, \dots, \pi_k]$ then the map $c \rightarrow (c|_{\pi_1}, \dots, c|_{\pi_k})$ is a bijection from $Col(\sigma)$ into $Col(\pi_1) \times \cdots \times Col(\pi_k)$.*

Proof: Let c be a valid coloring of σ, then c avoids patterns 132, 213, $2/13$ and $1X2$. Thus for all i, $c|_{\pi_i}$ avoids patterns 132, 213, $2/13$ and $1X2$ hence is a valid coloring of π_i. Conversely let $c_i \in Col(\pi_i)$ for all i. Then coloring points of σ according to $(c_1, \dots c_k)$ (i.e. according to c_1 for the $|\pi_1|$ first points of σ, according to c_2 for the $|\pi_2|$ following points and so on) leads to a coloring c of σ which is valid. Indeed assume that c is not valid. Then c has a pattern 132, 213, $2/13$ or $1X2$. Let p be such a pattern. Then p is not inside a block π_i as c_i is a valid coloring for all i. If all points of p are in different blocks π_i then p is 321 which is excluded. Thus there are one point of p in a block π_i and two points of p in a block π_j. If $i < j$ then p begins with its greatest point, which is excluded as p is 132, 213, 2/13 or $1X2$. If $i > j$ then p ends with its smallest point, which is excluded as p is 132, 213, 2/13 or $1X2$. As a consequence such a pattern p does not exists and $c \in Col(\sigma)$, concluding the proof. ■

Using Proposition 4.71, we can compute the set of valid colorings of any permutation using Algorithm 15 which computes the set of valid colorings of a $\ominus$-indecomposable permutation.

Algorithm 16: Colorings(σ)

```
Data: $\sigma$ a permutation
Result: A linear description of the set $Col(\sigma)$ of valid colorings of $\sigma$
Compute the $\ominus$-decomposition of $\sigma$: $\sigma = \ominus[\pi_1, \dots, \pi_k]$ with $\pi_i$ $\ominus$-indecomposable;
for $i$ from 1 to $k$ do
    Compute $Col(\pi_i)$ thanks to Algorithm 15;
Return $(Col(\pi_1), \dots, Col(\pi_k))$;
```

Proposition 4.72. *Let σ be a permutation of size n. Then Algorithm 16 gives a description of $Col(\sigma)$ in time $\mathcal{O}(n^2)$. More precisely, Algorithm 16 gives an integer $k \leq n$ and k sets $C_1, \dots C_k$ such that $Col(\sigma) = C_1 \times \cdots \times C_k$ and the sum of the cardinals $\sum_{i=1}^{k} |C_i|$ is less than $9n$.*

Proof. The algorithm computes the $\ominus$-decomposition of σ: $\sigma = \ominus[\pi_1, \dots, \pi_k]$ with π_i $\ominus$-indecomposable. This is done in linear time. If $k = 1$ then σ is $\ominus$-indecomposable and $Col(\sigma) = Col(\pi_1)$. We concludes thanks to Theorem 4.68. If $k > 1$ then from Proposition 4.71, $Col(\sigma) \approx Col(\pi_1) \times \cdots \times Col(\pi_k)$. From Theorem 4.68, $Col(\pi_i)$ has a size is smaller than $9|\pi_i|$ for all i and is computed in $\mathcal{O}(|\pi_i|^2)$. We concludes the proof noticing that $9|\pi_1| + \cdots + 9|\pi_k| = 9|\sigma|$ and $|\pi_1|^2 + \cdots + |\pi_k|^2 \leq |\sigma|^2$. ■

Notice that $Col(\sigma)$ may have a cardinality exponential w.r.t. $|\sigma|$. For instance with $\sigma = n(n-1)\dots 21$ (the reverse of the identity of size n), σ has 2^n valid colorings but $Col(\sigma)$ can be represented in linear space since it is the Cartesian product of n sets of size 2: each block of σ is the trivial permutation 1 which has 2 valid colorings, the two monochromatic ones.

We are now able to state the main theorem of this chapter:

Theorem 4.73. *Using Algorithm 16, we can decide in time $\mathcal{O}(n^2)$ whether a permutation σ of size n is 2-stack pushall sortable.*

Moreover Algorithm 16 computes in time $\mathcal{O}(n^2)$ an encoding of all pushall sorting processes of σ.

Proof. By Theorem 4.45, a permutation σ is 2-stack pushall sortable if and only if it admits a valid coloring. Thus using Proposition 4.72 all we need is to test whether each set $Col(\pi_i)$ returned by Algorithm 16 is non-empty with $\sigma = \ominus[\pi_1, \dots, \pi_k]$ being the $\ominus$-decomposition of σ.

We conclude the proof using Proposition 4.72 and 4.18 and Theorem 4.43. ■

We believe that pushall sorting defined in this chapter is of interest *per se*, but most importantly, with the results on pushall sorting given in this chapter we are able to give in the next chapter a polynomial algorithm deciding whether a permutation is sortable with two stack in series, although this problem was conjectured to be NP-complete in the literature.

Chapter 5

A polynomial algorithm deciding if a permutation is 2-stack sortable

This chapter deals with deciding whether a permutation is sortable with two stacks in series. Whether this decision problem lies in P or is NP-complete is a longstanding open problem since the introduction of serial compositions of stacks by Knuth in *The Art of Computer Programming* [Knu73a] in 1973. We hereby prove that this decision problem lies in P by giving a polynomial algorithm to solve it. This algorithm strongly relies on pushall sorting which is defined and studied in the previous chapter.

5.1 Introduction

Lots of fascinating questions arose out of the seminal work of Knuth about sorting devices such as stacks and queues; many are still unanswered more than 40 years later, despite a considerable literature on the subject.

One of these still open problems is the following: How many permutations of length n can be sorted by two stacks connected in series?

A classical strategy when trying to solve this kind of question is to study words encoding sorting processes. In general there are many words corresponding to a sortable permutation, but we can try to find rules limiting the number of words associated to a given permutation.

In the article *Permutations generated by stacks and deques*[AAL10], Albert, Atkinson and Linton give lower and upper bounds for the the number of permutations of length n generated by two stacks in series, two stacks in parallel, and a general deque (as already mentioned, deciding whether a permutation can be generated by two stacks in series and deciding whether a permutation is 2-stack sortable is equivalent up to composition with the inverse permutation).

They use the theory of finite automata and regular languages applied to words encoding sorting processes to obtain approximations of the growth rate of these classes.

Their lower and upper bounds for two stacks in parallel (7.535 and 8.3461) and for a deque (7.890 and 8.352) are rather close, while for two stacks in series (8.156 and 13.374) they are more distant.

This article ends by commenting on the difficulty of sorting with two stacks in series:

> There is a great similarity between two stacks in parallel and a deque.[...] By contrast we seem to understand two stacks in series rather less. It is possible that this system is intrinsically more complex than either a deque or two parallel stacks. One reason for believing this is that we have no efficient test for whether a given permutation can be generated by two stacks in series, whereas the membership problem for deques and two parallel stacks is in the complexity class P (see [RT84, EI71]). It is possible that the problem of deciding whether a given permutation is the product of two stack permutations (which is exactly the same as being generated by two serial stacks) is NP-complete.

This chapter deals with deciding whether a permutation is sortable with two stacks in series. We prove that this decision problem lies in P by giving a polynomial algorithm to solve it. This algorithm is rather complicated, and we believe as the authors of [AAL10] that sorting with two stacks in series is intrinsically complex.

When only one stack is considered, there exists a natural algorithm to decide whether a permutation is sortable or not. Indeed, there is a unique way to sort a permutation using only one stack, and a greedy algorithm gives a decision procedure. For two stacks in series, a permutation can be sorted in numerous ways. Take for example the permutation 4321. Each element can be pushed in either stacks H or V and output the identity at the end. This way the decreasing permutation of size n has at least 2^{n-1} ways to be sorted i.e. at least 2^{n-1} sorting words.

In fact, for two-stack sorting, finding a canonical sorting is hard. Thus many slightly modified models appear: In the literature, several variants of the greedy algorithm for one stack were introduced, but none allows to sort every 2-stack sortable permutation. For example, in his PhD-thesis [Wes90] West introduced a right greedy model. Permutations sortable with this model, called West-2-stack sortable permutations, are characterized and enumerated. The left greedy algorithm introduced in [AMR02] sorts permutations sortable with two *decreasing* stacks in series, which are also characterized and enumerated.

For our unrestricted case, sometimes called in the literature the *general 2-stack sorting problem*, no characterization of sortable permutations and no polynomial algorithm to decide if a permutation is sortable is known (except the one given in this chapter). A common mistake when trying to sort a given permutation is to pop out the smallest element not yet output, i, as soon as it lies in the stacks. This operation may indeed move other elements if i is not the topmost element of H. The elements above it are then transferred into V before i can be popped out. But sometimes, it can be necessary to take some elements of σ from the input and push them onto H or V before this transfer. Take for example the permutation 324617985. Trying to pop out the smallest element as soon as it is in the stacks leads to a dead-end. However, this permutation can be sorted using the word $\rho_3\rho_2\lambda_2\rho_4\rho_6\rho_1\lambda_1\mu_1\mu_2\rho_7\lambda_7\lambda_6\lambda_4\lambda_3\mu_3\mu_4\rho_9\rho_8\rho_5\lambda_5\mu_5\mu_6\mu_7\lambda_8\mu_8\lambda_9\mu_9$ (here for clarity we indicate on which integer each operation is performed). But we prove that this natural idea of popping out smallest elements as soon as possible can be adapted considering right-to-left minima of the permutation.

We saw that a sorting process can be described as a word on the alphabet $\{\rho, \lambda, \mu\}$. In this chapter, we will also describe a sorting in a different way. Take the prefix of a stack word: it corresponds to moving some elements from the permutation to the stacks or outputting them. At the end of the prefix some elements may be in the stacks. We can take a picture of the stacks (such a picture is called a stack configuration) and indeed, we will show that considering such pictures for all the prefixes that correspond to the entry of a right-to-left (RTL) minimum of the permutation in H is sufficient to decide the sortability.

Considering step by step the times t_i where the i-th right-to-left minimum enters stack H is one key of our algorithm. The second one is to encode sets of stack configurations by a graph.

Before going into details, we set definitions and notations.

5.2 Definitions and notations

We defined permutations of the set $[1..n]$ but we can define more generally permutations of any set of integers: Let I be a set of integers. A permutation of I is a bijection from I onto I. We write a permutation σ of I as the word $\sigma = \sigma_1\sigma_2\dots\sigma_n$ where $\sigma_i = \sigma(i_1)$ with $I = \{i_1 \dots i_n\}$ and $i_1 < i_2 < \cdots < i_n$. The size of the permutation is the integer n and if not specified, $I = [1..n]$. Notice that given the word $\sigma_1\sigma_2\dots\sigma_n$ we can deduce the set I and the map σ. With this definition, a permutation is a word whose letters are distinct integers. For any subset J of I, $\sigma_{|J}$ denotes the permutation such that the word corresponding to $\sigma_{|J}$ is the subword of the word corresponding to σ formed with the letters of J. For instance with $\sigma = 25413$, $\sigma_{|\{2,3,5\}} = 253$.

Recall that a permutation σ is sortable if, taking σ as input, there exists a sequence of operations ρ, λ, μ as depicted in Figure 4.1 (p.149) that leads to writing as output the elements of σ in increasing order.

Each sorting process is encoded by a unique word on the alphabet $\{\rho, \lambda, \mu\}$. For example, the sorting process of 2431 given in Example 4.1 (p.150) is encoded by the word $w = \rho\rho\lambda\rho\lambda\rho\lambda\mu\lambda\mu\mu\mu$. We can also decorate the word to specify the element on which each operation is performed. The *decorated word* for w and 2431 is $\hat{w} = \rho_2\rho_4\lambda_4\rho_3\lambda_3\rho_1\lambda_1\mu_1\lambda_2\mu_2\mu_3\mu_4$. Note that (σ, w) and $\hat{w}$ give the same information. The decorated word associated to (σ, w) is denoted $\hat{w}^\sigma$. Notice that in a decorated word each letter ρ_i, λ_i or μ_i appears only once.

In this chapter we often use the decomposition of permutations into blocks already seen in the previous chapters, but it is more convenient here not to renormalize the elements. That's why we give an alternative definition of $\ominus$-decomposition without renormalization:

Recall that a block B of a permutation $\sigma = \sigma_1\sigma_2\dots\sigma_n$ is a factor $\sigma_i\sigma_{i+1}\dots\sigma_j$ of σ such that the set of values $\{\sigma_i,\dots,\sigma_j\}$ forms an interval. Notice that by definition of a factor, the set of indices $\{i,\dots,j\}$ also forms an interval. Given two blocks B and B' of σ, we write $B < B'$ if and only if $\sigma_i < \sigma_j$ for all $\sigma_i \in B$, $\sigma_j \in B'$. A permutation σ is $\ominus$-decomposable if we can write it as $\sigma = B_1\dots B_k$ such that $k \geq 2$ and for all i, $B_i > B_{i+1}$ in terms of blocks. Otherwise we say that σ is $\ominus$-indecomposable. When each B_i is $\ominus$-indecomposable, we write $\sigma = \ominus[B_1,\dots,B_k]$ and call this the $\ominus$-decomposition of σ. Notice that we do not renormalize the elements of B_i thus except B_k, the B_i are permutations but not of the set $[1..|B_i|]$. Nevertheless, B_i can be seen as a permutation of $[1..|B_i|]$ by decreasing all its elements by $|B_{i+1}| + \dots + |B_k|$.

The RTL (right-to-left) minima of a permutation are the elements σ_k such that there do not exist j satisfying $j > k$ and $\sigma_j < \sigma_k$. We denote by σ_{k_i} the i^{th} right-to-left (RTL) minimum of σ. If σ has r RTL minima and $|\sigma| = n$, then $\sigma = \dots\sigma_{k_1}\dots\sigma_{k_2}\dots\sigma_{k_r}$ with $\sigma_{k_1} = 1$ and $k_r = n$.

Take for example the permutation $\sigma = 6\,5\,8\,7\,4\,1\,3\,2$. The $\ominus$-decomposition of σ is $\sigma = \ominus[6\,5\,8\,7, 4, 1\,3\,2]$. Furthermore σ has 2 RTL-minima which are $\sigma_6 = 1$ and $\sigma_8 = 2$.

Definition 5.1. We denote $\sigma^{(i)} = \{\sigma_j \mid j < k_i \text{ and } \sigma_j > \sigma_{k_i}\}$ the restriction of σ to the elements in the upper left quadrant of the i^{th} right-to-left (RTL) minimum σ_{k_i}. The $\ominus_i$-decomposition of σ is the $\ominus$-decomposition of $\sigma^{(i)} = \ominus[B_1^{(i)},\dots,B_{s_i}^{(i)}]$. In the sequel, s_i always denotes the number of blocks of $\sigma^{(i)}$ and $B_j^{(i)}$ the j^{th} block in the $\ominus_i$-decomposition.

There are two key ideas in this chapter. First, among all possible sorting words for a 2-stack sortable permutation, we show that there exists a sorting word satisfying some property denoted $\mathcal{P}$. In particular we prove that if a permutation σ is sortable then there exists a sorting process in which the elements that lie in the stacks just before a right-to-left minimum k_i enters the stacks are exactly the elements of $\sigma^{(i)}$. A formal definition is given in Definition 5.15.

The second idea is to encode the different sortings of a permutation satisfying $\mathcal{P}$ by a sequence of graphs $\mathcal{G}^{(i)}$ in which each node represents a stack configuration of a block $B_j^{(i)}$ and edges give compatibility between these partial configurations to obtain a possible total stack configuration for $\sigma^{(i)}$. The index i is taken from 1 to the number of right-to-left minima of the permutation.

This chapter is organized as follows: Section 5.3 studies general properties of two-stack sorting and describes which elements can move at each moment of a sorting process. Section 5.4 introduces the sorting graph $\mathcal{G}^{(i)}$ which encodes possible stack configuration for $\sigma^{(i)}$ at a given time t_i and gives an algorithm to compute this graph iteratively for all i from 1 to the number of right-to-left minima, leading to an algorithm deciding whether a permutation is 2-stack sortable. Then Section 5.5 focuses on complexity analysis. To conclude, we give in Section 5.6 some natural continuations of our work.

5.3 Study of two-stack sorting processes

5.3.1 Stack configurations and accessibility

We begin with some definitions.

Definition 5.2 (prefix stack word)**.** A *prefix stack word* is a word $w \in \{\rho, \lambda, \mu\}^*$ such that for any prefix v of w, $|v|_\rho \geq |v|_\lambda \geq |v|_\mu$.

Intuitively, prefix stack words are words describing some operations ρ, λ, μ starting with empty stacks and an arbitrarily long input and there may be some elements in the stacks at the end of these operations, whereas stack words (defined p.150) are words encoding a complete sorting process (stacks are empty at the beginning and at the end of the process).

Definition 5.3 (subword)**.** Let I be a set of integers.

For any decorated word u we define $u_{|I}$ as the subword of u made of letters ρ_i, λ_i, μ_i with $i \in I$. For example, if $u = \rho_3\mu_5\lambda_3\rho_6\rho_7\lambda_6$ then $u_{|\{5,6\}} = \mu_5\rho_6\lambda_6$.

We extend this definition to prefix stack words: given a permutation σ and a prefix stack word w, $w_{|I}$ is the word of $\{\rho, \lambda, \mu\}^*$ obtained from $\hat{w}^\sigma_{|I}$ by deleting indices from letters ρ_i, λ_i, μ_i.

Intuitively, $w_{|I}$ is the subword of w made of the operations of w that act on integers of I.

Lemma 5.4. *For any stack word (resp. prefix stack word) w, $w_{|I}$ is also a stack word (resp. prefix stack word).*

Proof. Let σ be a permutation. As w is a prefix stack word, for all i from 1 to $|\sigma|$, ρ_i appears before λ_i which itself appears before μ_i in $\hat{w}^\sigma_{|I}$. Therefore for any prefix v of $w_{|I}$, $|v|_\rho \geq |v|_\lambda \geq |v|_\mu$. If, moreover, w is a stack word, let $\alpha \in \{\rho, \lambda, \mu\}$; then for any letter α_i in $\hat{w}^\sigma_{|I}$, ρ_i, λ_i and μ_i appear each exactly once in $\hat{w}^\sigma_{|I}$, thus $|w_{|I}|_\rho = |w_{|I}|_\lambda = |w_{|I}|_\mu$. ■

Lemma 5.5. *Let σ be a permutation and I a subset of* $[1..|\sigma|]$. *If w is a sorting word for σ, then $w_{|I}$ is a sorting word for $\sigma_{|I}$.*

Now we turn to stack configurations, beginning with linking prefix stack words to stack configurations.

Definition 5.6. Let w be a prefix stack word. Starting with a permutation σ as input, the stack configuration reached after performing operations described by the word w is denoted $c_\sigma(w)$.

With this definition, a stack configuration c is reachable for σ if there exists a prefix stack word w such that $c = c_\sigma(w)$ (recall that a stack configuration is reachable for σ if, taking σ as input, there exists a sequence of operations ρ, λ, μ leading to this configuration).

Recall also that a stack configuration is poppable if the elements in stacks H and V can be output in increasing order using operations λ and μ. From Theorem 4.13, a stack configuration is poppable if and only if it avoids the three unsortable stack-patterns $|\ |132|$, $|12|\ |$ and $|2|13|$. Moreover recall that there is a unique way to pop the elements out in increasing order in terms of stack operations. We will denote by $out_c(I)$ the word that consists of the operations that output in increasing order the elements of the set of values I from a poppable stack configuration c.

Lemma 5.7. *If $\sigma = \ominus[B_1, \dots B_k]$ then in any poppable stack configuration reachable for σ, the elements of B_i are below the elements of B_j in the stacks for all $i < j$ (see Figure 5.1).*

Proof. Notice that by definition of a stack, in any stack configuration reachable for σ the elements of H are in increasing order of indices from bottom to top. Moreover in any poppable stack configuration the elements of V are in decreasing order of values from bottom to top since from Theorem 4.13 the stack configuration avoids the pattern $|12|\ |$. This leads to the claimed property. ■

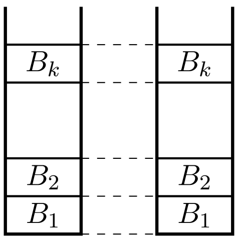
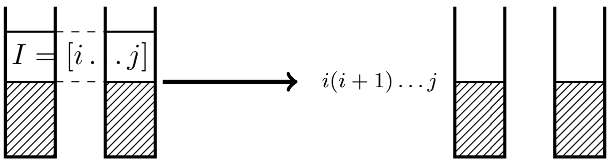


Figure 5.1: Poppable stack configuration reachable for $\ominus[B_1, \dots B_k]$.

Lemma 5.8. *Let σ be a 2-stack sortable permutation and $w = uv$ be a sorting word for σ. Assume that after performing operations of u, the elements $1 \dots i-1$ have been output and the elements $i \dots j$ are at the top of the stacks. Then there exists a sorting word $w' = uu'u''$ for σ such that u' consists only of moving the elements $i \dots j$ from the stacks to the output in increasing order without moving any other elements.*



Proof. We claim that $u' = v_{|[i..j]}$ and $u'' = v_{|![i..j]}$ satisfy the desired property, where $![i..j]$ is the set of integers $[1..|\sigma|] \setminus [i..j]$. This can be checked using decorated words associated to w and w' and noticing that $v_{|[i..j]} = out_{c_\sigma(u)}([i..j])$ and $v_{|![i..j]} = v_{|>j}$ since by hypothesis after performing operations of u, the elements $1, \dots, i-1$ have been output and the elements $i, \dots, j$ are at the top of the stacks. ∎

The stack configurations for a sorting process encode the elements that are currently in the stacks. But some elements are still waiting in the input and some elements have been output. To fully characterize a configuration, we define an *extended* stack configuration of a permutation σ of size n to be a pair (c, i) where $i \in \{1, \dots n+1\}$ and c is a poppable stack configuration made of all elements within $\sigma_1, \sigma_2, \dots, \sigma_{i-1}$ that are greater than a value p. The elements $\sigma_i, \dots, \sigma_n$ are still in the input and the elements $\sigma_j < p, j < i$ have already been output. Notice that we don't need the configuration to be reachable.

Definition 5.9. Let σ be a permutation and (c, i) be an extended stack configuration of σ. Then the extended stack configuration (c', j) of σ is *accessible* from (c, i) if the stack configuration (c', j) can be reached starting from (c, i) and performing operations ρ, λ and μ such that operations μ performed output the elements of $c \cup \{\sigma_i \dots \sigma_n\}$ in increasing order.

For example, if $\sigma = 2\,3\,1\,6\,5\,8\,4\,7$ then $(\begin{array}{|c|}8\\\hline\end{array}\,\begin{array}{|c|}5\\6\\\hline\end{array}, 7)$ is accessible from $(\begin{array}{|c|}2\\3\\\hline\end{array}\,\sqcup, 4)$ by the sequence of operations $\mu_2\mu_3\rho_6\rho_5\rho_8\lambda_8$. But $(\begin{array}{|c|}2\\3\\\hline\end{array}\,\begin{array}{|c|}6\\\hline\end{array}, 5)$ is not accessible from $(\begin{array}{|c|}1\\2\\\hline\end{array}\,\begin{array}{|c|}3\\\hline\end{array}, 4)$.

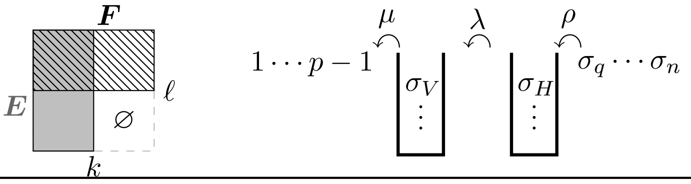
Notice that the question of whether a permutation is 2-stack sortable can be reformulated as:
Is $(\sqcup\,\sqcup, n+1)$ accessible from $(\sqcup\,\sqcup, 1)$?

To solve this problem is the main goal of this chapter and is somehow hard; however some special cases are easier to deal with. The following Lemma gives conditions on the involved configurations under which the accessibility decision problem is linear and can be solved by the *isAccessible* procedure given in Algorithm 17. In the last sections, we show how more general cases can be solved using this Lemma.

Lemma 5.10. *Let σ be a permutation of size n and (c,i), (c',j) two extended stack configurations of σ with $i<j$. Let E be the set of elements of c and F those of c'.*

- *If there exist $k,\ell \in \{1\dots n\}$ such that $E=\{\sigma_m \mid m\le k\}$ and $F=\{\sigma_m \mid \sigma_m \ge \ell\}$*
- *If moreover $E\cup F=\sigma$,*

then we can decide in linear time whether (c',j) is accessible from (c,i) using Algorithm 17.



Algorithm 17: isAccessible$\big((c,i),(c',j),\sigma\big)$

```
Data: $\sigma$ a permutation and $(c,i),(c',j)$ two stack configurations of $\sigma$ satisfying
      conditions of Lemma 5.10
Result: true or false depending on whether the configuration $c'$ is accessible from
        $c$
begin
    Put configuration $c$ in the stacks $H$ and $V$;
    $p \leftarrow$ the smallest element of $c \cup \{\sigma_i \dots \sigma_n\}$ (next element to be output);
    $q \leftarrow i$ (next index of $\sigma$ that must enter the stacks);
    We denote by $V(c')$ the set of elements of $V$ in configuration $c'$ and by $\sigma_V$ the
    top of $V$ in the current configuration (the same goes for $H$).
    while $q < j$ or $p < \ell$ or $\sigma_H \in V(c')$ do
        if $\sigma_V = p$ then
            Perform $\mu$; $p \leftarrow p+1$;
        else
            if $\sigma_H < \ell$ then
                Perform $\lambda$;
            else
                if $H = \varnothing$ or $\sigma_H \in H(c')$ then
                    Perform $\rho$; $q \leftarrow q+1$;
                else
                    if $\sigma_q \in H(c')$ or $\sigma_H > \sigma_q$ then
                        Perform $\lambda$;
                    else
                        Perform $\rho$; $q \leftarrow q+1$;
    Return $(H,V) == c'$;
```

Proof. We prove by case study that there is no choice between operations ρ,λ,μ at each time step. This is illustrated by Algorithm 17. We first prove its correctness before studying its complexity.

We start with configuration $curr = c$. By studying specific elements of the current configuration $curr$, we prove that we can always decide which operation should be performed to transform $curr$ into c'. If at any step this operation is forbidden then c' is not accessible from *curr*. Thus repeating the following process will eventually lead to deciding whether c' is accessible from c.

Notice that by definition, c and c' are poppable thus *curr* has to be poppable, hence must avoid the three unsortable patterns of Definition 4.12. Let p be the next element to be output, i.e. the smallest element of $c \cup \{\sigma_i \dots \sigma_n\}$. Let σ_H (resp. σ_V) be the topmost element of H (resp. of V) and σ_q be the element waiting in the input to be pushed onto H (σ_q may not exist and in that case $\sigma_q = \varnothing$; at the beginning $\sigma_q = \sigma_i$).

- If $\sigma_V = p$ then we perform μ (from Lemma 5.8).
- Otherwise operation μ is forbidden. We have to choose between ρ and λ. Moreover $p \notin V$ as V is in decreasing order from bottom to top.

  1. Suppose that $\sigma_H < \ell$. This means that $\sigma_H \notin F$ i.e. $\sigma_H \notin c'$. Notice that by definition of p, $p \leq \sigma_H$ and thus $p \notin c'$. Moreover $p \notin V$ thus $p \in H$. If $p = \sigma_H$ then, by Lemma 5.8, we can pop out p. Thus we perform λ. If $\sigma_H \neq p$, then we will prove that all elements x such that $p \leq x \leq \sigma_H$ form an interval at the top of the stacks. Those elements are all in the stacks by definition of ℓ and p. As V is decreasing, the elements of $[p \dots \sigma_H]$ belonging to V are at the top of it. Consider now the position of those elements in H.

     Suppose that it is not an interval. Then there exists an element x in H such that $x < \sigma_H$ and there is an element $y > \sigma_H$ between x and σ_H. But in that case, the elements $xy\sigma_H$ form the pattern $1\,3\,2$ and *curr* is not poppable so any operation ρ, λ or μ is allowed here because we will never reach c'.

     Suppose now that the elements $[p \dots \sigma_H]$ form an interval in H and V. Then as $p \in H$ is the smallest element, by Lemma 5.8, we want to pop out the elements $[p \dots \sigma_H]$, hence we perform λ.

     In conclusion, if $\sigma_H < \ell$ we perform λ.
  2. If not, then $\sigma_H \geq \ell$ and thus $\sigma_H \in c'$. Once again there are different cases:

     (a) If $H = \varnothing$ then λ is forbidden, thus we perform ρ.

     (b) If $\sigma_H \in H(c')$, it must stay in H thus λ is forbidden and we perform ρ.

     (c) Else $\sigma_H \in V(c')$.

     - If $\sigma_q \in H(c')$ then ρ is forbidden because σ_q would prevent σ_H from moving. Thus we perform λ.
     - Else $\sigma_q \in V(c')$. If $\sigma_H > \sigma_q$, as $\sigma_H \in V(c')$, ρ is forbidden otherwise we cannot put σ_q above σ_H in V. Thus we perform λ.
     - Otherwise $\sigma_H, \sigma_q \in V(c')$ and $\sigma_H < \sigma_q$. λ is forbidden otherwise we cannot put σ_H above σ_q in V. Thus we perform ρ.

We have proved that at each step of the algorithm we know which operation we have to do if we want to reach c'. Moreover while $q < j$ or $p < \ell$ or $\sigma_H \in V(c')$, it is impossible that $curr = c'$ so we have to continue. Conversely if $q \geq j$ and $p \geq \ell$ and $\sigma_H \notin V(c')$ then ρ and μ and λ are forbidden and we have to stop. Then if $curr = c'$, c' is accessible from c, otherwise c' is not accessible from c.

Finally there are at most $3n$ steps since at each step of the algorithm we perform an operation ρ, λ or μ. Moreover each step takes a constant time, therefore the algorithm runs in linear time. ■

In the sequel of this chapter, we do not compute all possible stack configurations during a sorting process of a given permutation σ but indeed focus on specific steps of the sorting.

We study the possible stack configurations at each time step t_i corresponding to the moment just before the right-to-left minimum σ_{k_i} is pushed onto stack H. Those configurations are configurations (c, k_i) accessible from $(\sqcup \bigsqcup, 1)$.

We will prove that we can add two different restrictions on these configurations. First, (c, k_i) must be a pushall stack configuration of $\sigma^{(i)}$ (see below). Second (c, k_i) must be accessible from some configuration (c', k_{i-1}) between time t_{i-1} and t_i.

Definition 5.11 (pushall configuration)**.** A stack configuration is a *pushall* stack configuration of σ if it is poppable, total and reachable for σ.

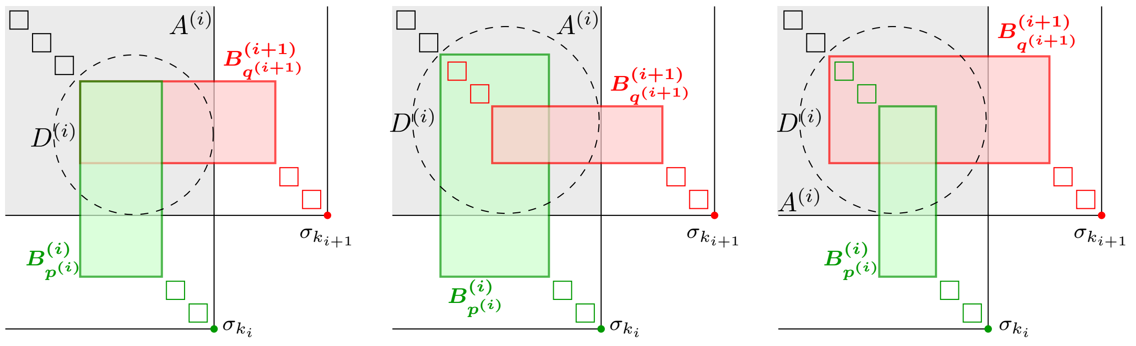
We have seen in the previous chapter that pushall stack configurations characterize pushall sorting processes since there is exactly one way to push the elements into the stacks and then to pop them out in increasing order.

5.3.2 From time t_i to time t_{i+1}

Thanks to the previous decomposition into different time steps corresponding to each moment a right-to-left minimum is pushed onto H and thanks to the results of Chapter 4 on 2-stack pushall sortable permutations, we can give a polynomial algorithm deciding whether a permutation is 2-stack sortable. Indeed, we will prove that it is enough to consider sorting processes such that for each time t_i the only elements in the stacks are exactly those of $\sigma^{(i)}$. But the stack configuration at time t_i is then a pushall stack configuration of $\sigma^{(i)}$; moreover Proposition 4.72 states that we can compute all pushall stack configurations in quadratic time. When a permutation is $\ominus$-indecomposable, Theorem 4.68 states that the number of possible pushall stack configurations is linear in the size of the permutation. This will ensure that our algorithm runs in polynomial time. Using this result, we now have the possible total stack configurations at time t_1.

The key idea for computing the set of possible stack configurations at time t_i relies on Lemma 5.14 below. Informally, it is possible to decide whether a configuration at time t_i can evolve into a given configuration at time t_{i+1}. Moreover, during this transition, only a few operations are undetermined. Indeed the largest elements won't move, the smallest ones will be output in increasing order and the remaining ones form a $\ominus$-indecomposable permutation. This will allow us to exhibit a polynomial algorithm.

First of all we denote by $A^{(i)}$ the common part of the permutations $\sigma^{(i)}$ and $\sigma^{(i+1)}$, that is, $A^{(i)} = \sigma^{(i)} \bigcap \sigma^{(i+1)} = \{\sigma_j \mid j < k_i \text{ and } \sigma_j > \sigma_{k_{i+1}}\}$. This sub-permutation $A^{(i)}$ intersects $\ominus$-indecomposable blocks of $\sigma^{(i)}$ and $\sigma^{(i+1)}$. Let $p^{(i)}$ (resp. $q^{(i+1)}$) be the index such that $B^{(i)}_{p^{(i)}}$ (resp. $B^{(i+1)}_{q^{(i+1)}}$) contains the smallest value of $A^{(i)}$. Let $D^{(i)} = (B^{(i)}_{p^{(i)}} \bigcup B^{(i+1)}_{q^{(i+1)}}) \bigcap A^{(i)}$ (see below for $p^{(i)} = q^{(i+1)}$, $p^{(i)} < q^{(i+1)}$ and $p^{(i)} > q^{(i+1)}$.



Lemma 5.12. *For any* $j < \min(p^{(i)}, q^{(i+1)})$, $B^{(i)}_j = B^{(i+1)}_j$.

Lemma 5.13. *Let $\sigma_\ell \in A^{(i)}$. During a sorting process of σ, elements σ_m such that $\sigma_m > \sigma_\ell$ and $m < \ell$ do not move between t_i and t_{i+1}.*

Proof. Let σ_m be an element such that $m < \ell$ and $\sigma_m > \sigma_\ell$. As $\sigma_\ell \in A^{(i)}$, $\sigma_\ell > \sigma_{k_{i+1}}$ and $j < k_i$, so does $\sigma_m > \sigma_{k_{i+1}}$ and $m < k_i$. Hence both elements σ_m, σ_ℓ lie in the stacks between t_i and t_{i+1} (they cannot be output as $\sigma_{k_{i+1}}$ must be output first). Suppose that σ_m is in H at time t_i. As $m < \ell$, element σ_ℓ is pushed after σ_m into the stacks, thus either σ_ℓ is above σ_m in H or lies in V at time t_i and t_{i+1}. So, σ_m cannot move into V, otherwise σ_ℓ would be under it in V and V would contain a pattern 12. So, σ_m stay in H.

Suppose now that σ_m is in V at time t_i. As noticed previously, this element is not output at time t_{i+1}. So it also lies in stack V at time t_{i+1}, proving the lemma. ■

In the following we study conditions for two total pushall stack configurations c and c' corresponding to stack configuration of $\sigma^{(i)}$ and $\sigma^{(i+1)}$ to be accessible one from the other, that is, we can move elements starting from c at time t_i to obtain c' at time t_{i+1}.

Lemma 5.14. *Let (c, k_i) (resp. (c', k_{i+1})) be a total stack configuration of $\sigma^{(i)}$ (resp. $\sigma^{(i+1)}$). Let $\pi = \sigma_{|B^{(i)}_{p^{(i)}} \bigcup B^{(i+1)}_{q^{(i+1)}}}$. Then (c', k_{i+1}) is accessible from (c, k_i) for σ iff:*

1. *$(c'_{|\pi}, |\pi| + 1)$ is accessible from $(c_{|\pi}, \sharp(D^{(i)} \bigcup B^{(i)}_{p^{(i)}}) + 1)$ for π.*
2. *$\forall j < min(p^{(i)}, q^{(i+1)})$, $c_{|B^{(i)}_j} = c'_{|B^{(i)}_j}$.*
3. *$\forall j > q^{(i+1)}, c'_{|B^{(i+1)}_j}$ is a reachable configuration for $\sigma_{|B^{(i+1)}_j}$.*

Proof. Suppose first that (c', k_{i+1}) is accessible from (c, k_i). This means that we can go from c to c' using operations represented by a decorated word $\hat{w}$. These operations are stable, that is, for all I, $c'_{|I}$ is accessible from $c_{|I}$. To do so, we just extract operations corresponding to the elements of I. Indeed the decorated word $\hat{w}_{|I}$ allows us to transform c into c'. This proves the first point of Lemma 5.14.

Let $\sigma_\ell \in B^{(i)}_p$. Lemma 5.13 ensures that the elements of $B^{(i)}_j$ with $j < p^{(i)}$ do not move between t_i and t_{i+1} proving the second point of Lemma 5.14.

Finally, the elements of $B^{(i+1)}_j$ for $j > q^{(i+1)}$ are pushed iteratively when going from c to c'. Those elements stay in the stacks as $\sigma_{k_{i+1}}$, which is smaller, is pushed after them. Thus they correspond to a pushall configuration.

Conversely, suppose that we have the 3 different points above, we must prove that (c', k_{i+1}) is accessible from (c, k_i) for σ. We start by taking the stack configuration c and we will prove that we can obtain c' by moving elements. First of all, as c is a pushall stack configuration, and as the elements of B_ℓ for $\ell > p$ are the smallest ones and have been pushed last into the stacks, they are at the top of the stacks (see Lemma 5.7). Thus we can pop them and output them in increasing order using Lemma 5.8.

The remaining elements in the stacks don't move in the preceding operation, thus stay in the same position as in c. In that configuration, the elements of $B^{(i)}_{p^{(i)}}$ are the smallest ones and have been pushed most recently into the stacks. Hence they lie at the top of the stacks.

Then using point 1 of our hypothesis, we can move those elements together with pushing the elements of $B^{(i+1)}_{q^{(i+1)}} \setminus B^{(i)}_{p^{(i)}}$ so that all those elements (that is the elements of π) are in the same position as in c'. Then, by hypothesis item 3, $\forall j > q^{(i+1)}, c'_{|B^{(i+1)}_j}$ is a reachable configuration. Thus we can push its elements into the stacks in the same relative order as in c' (see Lemma 5.7). During these operations we ensure that the elements of B_ℓ with $\ell \geq min(p^{(i)}, q^{(i+1)})$, $c_{|B_j^{(i)}}$ are in the same position in our configuration than in c'. Point 2 ensures that we indeed obtain c'. ■

The preceding Lemma describes exactly which elements can move between t_i and t_{i+1} and how they move. But the hypotheses of Lemma 5.14 are restrictive, that is, configurations c and c' must be two total stack configurations of $\sigma^{(i)}$ and $\sigma^{(i+1)}$. Thus, we first prove that among all sortings of a 2-stack sortable permutation, there exists at least one for which the stack configuration at time t_i contains exactly the elements of $\sigma^{(i)}$ for all i.

Definition 5.15 (Properties (P_i) and (P))**.** Let σ be a permutation and w a sorting word for σ. We say that w verifies (P_i) if and only if

(i) $\rho_{\sigma_{k_i}} \lambda_{\sigma_{k_i}} \mu_{\sigma_{k_i}}$ is a factor of w,

(ii) μ_{σ_j} appears before $\rho_{\sigma_{k_i}}$ for all $\sigma_j < \sigma_{k_i}$,

(iii) All operations μ_{σ_ℓ} with $\sigma_\ell \in B_j^{(i)}$ and $j \in [p^{(i)} + 1..s_i]$ appear before $\rho_{\sigma_{k_{i+1}}}$ in w,

where σ_{k_i} is the i^{th} right-to-left minima of the permutation and $\sigma^{(i)} = \ominus[B_1^{(i)}, \ldots, B_{s_i}^{(i)}]$.

If a word w verifies Property (P_i) for all i then we say that w verifies Property (P).

Lemma 5.16. *If the sorting word encoding a sorting process of σ verifies Property (P_i), then at time t_i the elements currently in the stacks are exactly those of $\sigma^{(i)}$.*

Proof. By definition of time t_i (just before σ_{k_i} enters the stacks) each element in the stacks has an index smaller than k_i. Moreover among elements of index smaller than k_i, those of value greater than σ_{k_i} cannot have been output by definition of a sorting, and those of value smaller than σ_{k_i} have already been output since w satisfies item (ii) of Property (P_i). ■

Lemma 5.17. *Let w be a sorting word for a permutation σ, r be the number of RTL-minima of σ and $\ell \in [1..r]$. If w verifies (P_i) for $i \in [1..\ell - 1]$ then there exists a sorting word w' for σ that verifies (P_i) for $i \in [1..\ell]$.*

Proof. Consider the sorting process of σ encoded by w. The key idea is to prove that the smallest elements are at the top of the stacks so that we can transform the word w thanks to Lemma 5.8.

Property (ii) for (P_ℓ) states that μ_{σ_j} should appear before $\rho_{\sigma_{k_\ell}}$ for all $\sigma_j < \sigma_{k_\ell}$. Suppose that there still exists an element σ_j with $\sigma_j < \sigma_{k_\ell}$ in the stacks just before σ_{k_ℓ} is pushed into the stacks. We prove that this element can be popped out before σ_{k_ℓ} is pushed. Let σ_{j_0} be the smallest element still in the stacks just before $\rho_{\sigma_{k_i}}$. By definition, the elements smaller than σ_{j_0} have already been output. Consider interval $I = [\sigma_{j_0}, \sigma_{k_\ell} - 1]$. Those elements are still in the stacks. If they are at the top of the stacks they can be output using Lemma 5.8. If not, there exists in the stacks an element $x \notin I$ above an element $y \in I$. As V is decreasing, those elements are in H. Moreover $x > \sigma_{k_\ell} > y$. Then σ_{k_ℓ} cannot be pushed as it will create a pattern 132 in H with the elements x and y. Thus I is at the top of the stacks and we can output it before σ_{k_ℓ} is pushed onto H: using Lemma 5.8, we build from w a sorting word $w^{(1)}$ for σ satisfying (P_i) for $i \in [1..\ell - 1]$ and Property (ii) of (P_ℓ). This means that $w^{(1)}$ can be decomposed as $w^{(1)} = u\rho_{\sigma_{k_\ell}} v$ such that the stack configuration $c_\sigma(u)$ respects the following constraint: the elements $1, \ldots, \sigma_{k_\ell}$ are not in the stacks.

So if we consider the stack configuration $c_\sigma(u\rho_{\sigma_{k_\ell}})$, element σ_{k_ℓ} is at the top of H and since $out_{c_\sigma(u\rho_{\sigma_{k_\ell}})}(\sigma_{k_\ell}) = \lambda_{\sigma_{k_\ell}}\mu_{\sigma_{k_\ell}}$ we can use Lemma 5.8 to change the sorting word $w^{(1)}$ into a sorting word $w^{(2)} = u\rho_{\sigma_{k_\ell}}\lambda_{\sigma_{k_\ell}}\mu_{\sigma_{k_\ell}}v'$, satisfying Property (i) for (P_ℓ).

Now we show considering the stack configuration $c = c_\sigma(u\rho_{\sigma_{k_\ell}}\lambda_{\sigma_{k_\ell}}\mu_{\sigma_{k_\ell}})$ how to transform the word $w^{(2)}$ into a word $w' = u\rho_{\sigma_{k_\ell}}\lambda_{\sigma_{k_\ell}}\mu_{\sigma_{k_\ell}}v^{(1)}v^{(2)}$ with $v^{(1)} = out_c(B^{(\ell)}_{p^{(\ell)}+1} \cup \cdots \cup B^{(\ell)}_{s_\ell})$. This will conclude the proof.

Notice that the elements of c are exactly those of $\sigma^{(\ell)}$ since the last operations performed are $\rho_{\sigma_{k_\ell}}\lambda_{\sigma_{k_\ell}}\mu_{\sigma_{k_\ell}}$ and elements are pushed into the stacks in increasing order of indices and output in increasing order of values. Thus $out_c(B^{(\ell)}_{p^{(\ell)}+1} \cup \cdots \cup B^{(\ell)}_{s_\ell}) = out(B^{(\ell)}_{s_\ell}) \dots out(B^{(\ell)}_{p^{(\ell)}+1})$ (see Lemma 5.7). We show by induction on j from s_ℓ to $p^{(i)}+1$ that we can build a sorting word for σ of the form $u\rho_{\sigma_{k_\ell}}\lambda_{\sigma_{k_\ell}}\mu_{\sigma_{k_\ell}}v^{(1,j)}v^{(2,j)}$ with $v^{(1,j)} = out(B^{(\ell)}_{s_\ell}) \dots out(B^{(\ell)}_{j})$. For $j = s_\ell$ that is a word in which the elements of block B_{s_ℓ} are output immediately after σ_{k_ℓ} has been output. By definition of s_ℓ and because the elements of c are exactly those of $\sigma^{(\ell)}$, all elements of B_{s_ℓ} lie in the stacks in configuration c, are the smallest elements in this configuration and lie at the top of the stacks in configuration c (see Lemma 5.7). Hence, using Lemma 5.8, there exists a sorting word $w^{(3)}$ for σ such that $w^{(3)} = u\rho_{\sigma_{k_\ell}}\lambda_{\sigma_{k_\ell}}\mu_{\sigma_{k_\ell}}out(B_{s_\ell})v''$. Repeating this operation for all blocks B_j with j from $s_\ell - 1$ to $p^{(i)}+1$, we have Property (iii). ■

Notice that Property (P_0) is an empty property satisfied by any sorting word. Using recursively Lemma 5.17 we can transform any sorting word into a sorting word satisfying Property (P), leading with Lemma 5.16 to the following theorem:

Theorem 5.18. *If σ is 2-stack sortable then there exists a sorting word of σ satisfying Property (P). In particular, in the sorting process that this word encodes, the elements currently in the stacks at time t_i are exactly those of $\sigma^{(i)}$.*

Theorem 5.18 ensures that if a permutation is sortable then there exists a sorting in which at each time step t_i, the elements in the stacks are exactly those of $\sigma^{(i)}$. Thus stack configurations at time t_i and t_{i+1} satisfy the hypotheses of Lemma 5.14 and we can apply it to decide if a permutation is 2-stack sortable.

5.4 An iterative algorithm

5.4.1 A first naive algorithm

From Theorem 5.18 a permutation σ is 2-stack sortable if and only if it admits a sorting process satisfying Property (P). The main idea is to compute the set of sorting processes of σ satisfying Property (P) and then to decide whether σ is 2-stack sortable by testing its emptiness.

Verifying (P) means verifying (P_j) for all j from 1 to r, r being the number of right-to-left minima (whose indices are denoted k_j). The algorithm proceeds in r steps: for i from 1 to r we iteratively compute the sorting processes of $\sigma_{\leq k_i}$ verifying (P_ℓ) for all ℓ from 1 to i. As $\sigma_{\leq k_r} = \sigma$, the last step gives the sorting processes of σ satisfying Property (P).

By "compute the sorting processes of $\sigma_{\leq k_i}$" we mean compute the stack configuration just before σ_{k_i} enters the stacks in such a sorting process. Note that this is also the stack configuration just after σ_{k_i} has been output since $\rho_{\sigma_{k_i}}\lambda_{\sigma_{k_i}}\mu_{\sigma_{k_i}}$ is a factor of any word verifying (P).

Definition 5.19. We call a *P_i-stack configuration* of σ a stack configuration $c_\sigma(w)$ for which there exists u such that the first letter of u is $\rho_{\sigma_{k_i}}$ and wu is a sorting word of $\sigma_{\leq k_i}$ verifying (P) for $\sigma_{\leq k_i}$ (that is, verifying (P_ℓ) for all ℓ from 1 to i).

Lemma 5.20. *For any i from 1 to r, $\sigma_{\leq k_i}$ is 2-stack sortable if and only if the set of P_i-stack configurations of σ is nonempty. In particular, σ is 2-stack sortable if and only if the set of P_r-stack configurations of σ is nonempty.*

Proof. This is a direct consequence of Definition 5.19 and Theorem 5.18. ■

Lemma 5.21. *Any P_i-stack configuration of σ is a pushall stack configuration of $\sigma^{(i)}$ accessible from some P_{i-1}-stack configurations of σ.*

Proof. By definition of (P), each P_i-stack configuration of σ is accessible from some P_{i-1}-stack configuration of σ (take the prefix of w that ends just before $\rho_{\sigma_{k_{i-1}}}$). Moreover it is a pushall stack configuration of $\sigma^{(i)}$ from Lemma 5.16. ■

As explained above, the algorithm proceeds in r steps such that after step i we know every P_i-stack configuration of σ and we want to compute the P_{i+1}-stack configurations of σ at step $i+1$. As configurations for $i+1$ are a subset of pushall stack configurations of $\sigma^{(i+1)}$, a possible algorithm is to take every pair of configurations (c, c') with c being a P_i-stack configuration of σ (computed at step i) and c' being any pushall stack configuration of $\sigma^{(i+1)}$ (given by Algorithm 16). Then we can use Algorithm 17 to decide whether c' is accessible from c for σ. This leads to the following algorithm deciding whether a permutation σ is 2-stack sortable:

Algorithm 18: *isSortableNaive*

```
Data: $\sigma$ a permutation
Result: true or false depending on whether $\sigma$ is 2-stack sortable
begin
    $E, F$ two empty sets;
    $E \leftarrow$ PushallConfigs($\sigma^{(1)}$);
    for $i$ from 2 to $r$ do
        $F \leftarrow \emptyset$;
        for $c$ in $E$ do
            for $c'$ in PushallConfigs($\sigma^{(i)}$) do
                if isAccessible($(c, k_i), (c', k_{i+1}), \sigma$) then
                    $F \leftarrow F \cup c'$;
        $E \leftarrow F$;
    if $E$ is empty then
        return false;
    else
        return true;
```

Notice that at step i, the set E computed contains all P_i-stack configurations of σ but may contain some other configurations. However since each configuration of E is a pushall configuration of $\sigma^{(i)}$ and is accessible for σ from some pushall configurations of $\sigma^{(i-1)}$, each configuration of E indeed corresponds to some sorting procedure of $\sigma_{\leq k_i}$, proving the correctness of Algorithm 18.

But this algorithm is not polynomial. Indeed the number of P_i-stack configurations of σ is possibly exponential. However this set can be described by a polynomial representation as a graph $\mathcal{G}^{(i)}$ and we can adapt Algorithm 18 to obtain a polynomial algorithm. In this adapted algorithm, the set E computed at step i is exactly the set of P_i-stack configurations of σ.

5.4.2 Towards the sorting graph

We now explain how to adapt Algorithm 18 to obtain a polynomial algorithm. Instead of computing all P_i-stack configurations of σ (which are pushall stack configurations of $\sigma^{(i)}$), we compute the restriction of such configurations to blocks $B_j^{(i)}$ of the $\ominus$-decomposition of $\sigma^{(i)}$. By Lemma 5.7, those configurations are stacked one upon the others. The stack configurations of any block $B_j^{(i)}$ are labeled with an integer which is assigned when the configuration is computed. Those pairs (configuration,integer) will be the vertices of the graph $\mathcal{G}^{(i)}$ which we call a *sorting graph*, the edges of which represent the configurations that can be stacked one upon the other. Vertices of the graph $\mathcal{G}^{(i)}$ are partitioned into levels corresponding to blocks $B_j^{(i)}$. To ensure the polynomiality of the representation, we will prove that a given integer label could only appear once per level of the graph $\mathcal{G}^{(i)}$. As those numbers are assigned to configurations when they are created, each integer corresponding to a pushall stack configuration, from Chapter 4 there exists only a polynomial number of distinct integers thus of vertices. This will be explained in detail in the next section. The integer indeed can be seen as the memory of the configuration that encodes its history since it has been created: two configurations which have the same label come from the same initial pushall configuration.

More precisely a sorting graph $\mathcal{G}^{(i)}$ for a permutation σ of size n and an index i verifies the following properties:

- Vertices of $\mathcal{G}^{(i)}$ are partitioned into s_i subsets $V_j^{(i)}$ with $j \in [1 \dots s_i]$ called levels.
- For any $j \in [1 \dots s_i], |V_j^{(i)}| \leq 9n + 2$.
- Each vertex $v \in \mathcal{G}^{(i)}$ is a pair (c, ⓛ) with c a stack configuration and ⓛ an index called *configuration index.*
- All configuration indices are distinct inside a graph level $V_j^{(i)}$
- (c, ⓛ) $\in V_j^{(i)} \Rightarrow c$ is a pushall stack configuration of $B_j^{(i)}$ accessible for σ.
- There are edges only between vertices of adjacent levels $V_j^{(i)}$, $V_{j+1}^{(i)}$.
- Paths between vertices of $V_1^{(i)}$ and $V_{s_i}^{(i)}$ correspond to stack configurations of $\sigma^{(i)}$. More precisely such paths are in one-to-one correspondence with P_i-stack configurations of σ (that is, stack configurations corresponding to a sorting of $\sigma_{\leq k_i}$ satisfying (P) just before σ_{k_i} is pushed onto H).
- For any vertex v of $\mathcal{G}^{(i)}$, there is a path between vertices of $V_1^{(i)}$ and $V_{s_i}^{(i)}$ going through v.

Though the definition of sorting graph is complex, its use will be quite understandable and easy. Look for example at the permutation $\sigma = 4321$. There is only one right-to-left minimum which is 1. Compute all possible stack configurations just after 1 enters H. At this time, all elements are in the stacks since the first element which must be output is 1. More formally, we are looking at the pushall stack configurations of σ with 1 in H.

There are 8 different such configurations which are:

$$\sqcup\begin{array}{|c|}1\\2\\3\\4\\\hline\end{array}\quad \begin{array}{|c|}2\\\hline\end{array}\begin{array}{|c|}1\\3\\4\\\hline\end{array}\quad \begin{array}{|c|}3\\\hline\end{array}\begin{array}{|c|}1\\2\\4\\\hline\end{array}\quad \begin{array}{|c|}4\\\hline\end{array}\begin{array}{|c|}1\\2\\3\\\hline\end{array}\quad \begin{array}{|c|}3\\4\\\hline\end{array}\begin{array}{|c|}1\\2\\\hline\end{array}\quad \begin{array}{|c|}2\\4\\\hline\end{array}\begin{array}{|c|}1\\3\\\hline\end{array}\quad \begin{array}{|c|}2\\3\\\hline\end{array}\begin{array}{|c|}1\\4\\\hline\end{array}\quad \begin{array}{|c|}2\\3\\4\\\hline\end{array}\begin{array}{|c|}1\\\hline\end{array}$$

The $\ominus$-decomposition of σ is $\sigma = \ominus[4,3,2,1]$. We build a graph with 4 levels, each level corresponding to pushall stack configurations of a block.

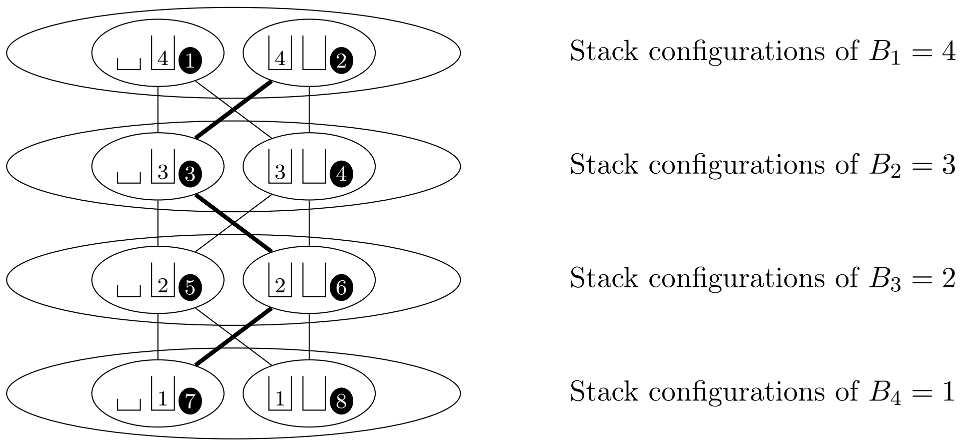

Stack configurations of $B_1 = 4$
Stack configurations of $B_2 = 3$
Stack configurations of $B_3 = 2$
Stack configurations of $B_4 = 1$

Figure 5.2: Graph encoding pushall stack configurations of $\sigma = 4321$.

Then the 8 configurations of σ are found taking each of the 8 different paths going from any configuration of B_1 to configuration $\sqcup\begin{array}{|c|}1\\\hline\end{array}$ of B_4. In Figure 5.2, the thick path gives the stack configuration $\begin{array}{|c|}2\\4\\\hline\end{array}\begin{array}{|c|}1\\3\\\hline\end{array}$ by stacking the selected configuration of B_4 above the configuration of B_3 and so on.

But in the last level B_4 we only consider configuration $\sqcup\begin{array}{|c|}1\\\hline\end{array}$ so this level is useless. The sorting graph $\mathcal{G}^{(1)}$ for $\sigma = 4321$ encodes pushall stack configurations of $\sigma^{(1)} = 432$, corresponding to stack configurations just *before* 1 enters H (and not after as above).

There are 8 different such configurations which are:

$$\sqcup\begin{array}{|c|}2\\3\\4\\\hline\end{array}\quad \begin{array}{|c|}2\\\hline\end{array}\begin{array}{|c|}3\\4\\\hline\end{array}\quad \begin{array}{|c|}3\\\hline\end{array}\begin{array}{|c|}2\\4\\\hline\end{array}\quad \begin{array}{|c|}4\\\hline\end{array}\begin{array}{|c|}2\\3\\\hline\end{array}\quad \begin{array}{|c|}3\\4\\\hline\end{array}\begin{array}{|c|}2\\\hline\end{array}\quad \begin{array}{|c|}2\\4\\\hline\end{array}\begin{array}{|c|}3\\\hline\end{array}\quad \begin{array}{|c|}2\\3\\\hline\end{array}\begin{array}{|c|}4\\\hline\end{array}\quad \begin{array}{|c|}2\\3\\4\\\hline\end{array}\sqcup$$

As the $\ominus$-decomposition of $\sigma^{(1)}$ is $\sigma^{(1)} = \ominus[4,3,2]$, the sorting graph $\mathcal{G}^{(1)}$ has 3 levels.

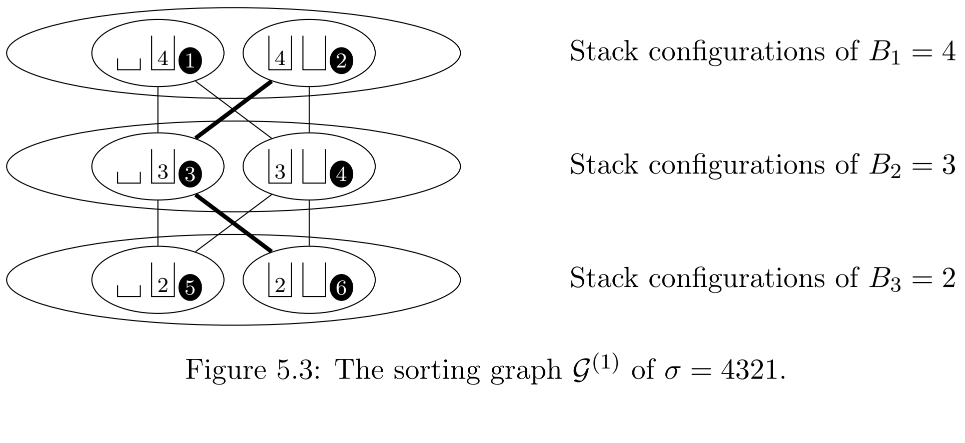

Stack configurations of $B_1 = 4$
Stack configurations of $B_2 = 3$
Stack configurations of $B_3 = 2$

Figure 5.3: The sorting graph $\mathcal{G}^{(1)}$ of $\sigma = 4321$.

Then the 8 configurations of σ are found taking each of the 8 different paths going from any configuration of B_1 to any configuration of B_3. In Figure 5.3, the thick path gives the stack configuration $\begin{array}{|c|}2\\4\\\hline\end{array}\ \begin{array}{|c|}3\\\hline\end{array}$ by stacking the selected configuration of B_3 above the configuration of B_2 and so on.

We transform Algorithm 18 into a polynomial algorithm by computing at step i not all P_i-stack configurations of σ, but instead the sorting graph $\mathcal{G}^{(i)}$ encoding them. The graph $\mathcal{G}^{(i)}$ is computed iteratively from the graph $\mathcal{G}^{(i-1)}$ for any i from 2 to r. The way $\mathcal{G}^{(i)}$ is computed from $\mathcal{G}^{(i-1)}$ depends on the relative values of $p^{(i)}$ and $q^{(i+1)}$. By definition of a sorting graph given p.204, if at any step $\mathcal{G}^{(i)}$ is empty, it means that $\sigma_{\leq k_i}$ is not sortable (from Theorem 5.18) and so is σ thus the algorithm returns `false`. This is summarized in Algorithm 19.

Algorithm 19: *isSortable*

```
Data: $\sigma$ a permutation
Result: true or false depending on whether $\sigma$ is 2-stack sortable
begin
    $\mathcal{G} \leftarrow ComputeG1$;
    for $i$ from 2 to $r$ do
        if $p^{(i)} = q^{(i+1)}$ then
            $\mathcal{G} \leftarrow iteratepEqualsq(\mathcal{G})$ or return false
        else
            if $p^{(i)} < q^{(i+1)}$ then
                $\mathcal{G} \leftarrow iteratepLessThanq(\mathcal{G})$ or return false
            else
                $\mathcal{G} \leftarrow iteratepGreaterThanq(\mathcal{G})$ or return false
    return true
```

In the next subsections we describe the sub-procedures used in our main algorithm *isSortable*(σ).

5.4.3 First step: $\mathcal{G}^{(1)}$

In this subsection, we show how to compute the P_1-stack configurations of σ, that is, the stack configurations corresponding to time t_1 for sorting words of $\sigma_{\leq k_1}$ that satisfy (P) for $\sigma_{\leq k_1}$.

From Lemma 5.21, such a stack configuration is a pushall stack configuration of $\sigma^{(1)}$. Conversely since $\sigma_{k_1} = 1$, $\sigma^{(1)} = \sigma_{<k_1}$ and each sorting word of $\sigma_{\leq k_1}$ satisfies (P_1) for $\sigma_{\leq k_1}$. Thus the set of P_1-stack configurations of σ is the set of pushall stack configurations of $\sigma^{(1)}$.

By Proposition 4.71 of Chapter 4, these stack configurations are described by giving the set of stack configurations for each block of the $\ominus$-decomposition of $\sigma^{(1)}$. More precisely, with $\sigma^{(1)} = \ominus[B_1^{(1)}, \dots, B_{s_1}^{(1)}]$ there is a bijection from $pushallConfigs(B_1^{(1)}) \times \cdots \times pushallConfigs(B_{s_1}^{(1)})$ onto $pushallConfigs(\sigma^{(1)})$ by stacking configurations one upon the other (as in Lemma 5.7). As a consequence, from Lemma 5.20 $\sigma_{\leq k_1}$ is not sortable if and only if a set $pushallConfigs(B_j^{(1)})$ is empty.

Moreover it will be useful to label the configurations computed so that we attach a distinct integer to each stack configuration when computed.

At this point, we have encoded all configurations corresponding to words satisfying P up to the factor $\rho_1\lambda_1\mu_1$.

The obtained graph is $\mathcal{G}^{(1)}$. This step is summarized in Algorithm 20.

Algorithm 20: ComputeG1

```
Data: $\sigma$ a permutation, $num$ a global integer variable
Result: false if $\sigma_{\leq k_1}$ is not sortable, the sorting graph $\mathcal{G}^{(1)}$ otherwise.
begin
    $E = \varnothing$;
    Compute $\sigma^{(1)}$ and its $\ominus$-decomposition $\ominus[B_1^{(1)}, \dots, B_{s_1}^{(1)}]$;
    for $j$ from 1 to $s_1^{(1)}$ do
        $V_j^{(1)} \leftarrow \varnothing$;
        $S = pushallConfigs(B_j^{(1)})$;
        if $S = \varnothing$ then
            return false;
        else
            for $s \in S$ do
                $V_j^{(1)} \leftarrow V_j^{(1)} \bigcup \{(s, num)\}$;
                $num \leftarrow num + 1$;
            if $j > 1$ then
                $E = E \bigcup \{(s, s'), s \in V_j^{(1)}, s' \in V_{j-1}^{(1)}\}$
    return $\mathcal{G}^{(1)} = (\bigcup_{j \in [1..s_1^{(1)}]} V_j^{(1)}, E)$
```

5.4.4 From step i to step $i+1$

After step i we know the graph $\mathcal{G}^{(i)}$ encoding every P_i-stack configuration of σ and we want to compute the graph $\mathcal{G}^{(i+1)}$ encoding P_{i+1}-stack configurations of σ at step $i+1$. From Lemma 5.21 we have to check the accessibility of pushall stack configuration of $\sigma^{(i+1)}$ from P_i-stack configurations of σ. We want to avoid to check every pair of configurations (c, c') with c being a P_i-stack configuration and c' be a pushall stack configuration of $\sigma^{(i+1)}$ because the number of such pair of configurations is possibly exponential. Thus our algorithm focuses not on stack configurations of some $\sigma^{(\ell)}$ but on sets of stack configurations of blocks $B_j^{(\ell)}$, making use of Lemma 5.14.

Using Lemma 5.21, Lemma 5.14 can be rephrased as:

Lemma 5.22. *Let c' be a total stack configuration of $\sigma^{(i+1)}$, $p = p^{(i)}$ and $q = q^{(i+1)}$. Then c' is a P_{i+1}-stack configuration of σ if and only if:*

- *For any $j \leq q$, $c'_{|B_j^{(i+1)}}$ is a pushall stack configuration of $\sigma_{|B_j^{(i+1)}}$, and*
- *There exists a P_i-stack configuration c of σ such that:*
  - *$c'_{|B^{(i)}_{\min(p,q)} \cup \dots \cup B^{(i)}_q}$ is accessible from $c_{|B^{(i+1)}_{\min(p,q)} \cup \dots \cup B^{(i+1)}_p}$ for $\sigma_{|B^{(i)}_p \bigcup B^{(i+1)}_q}$ and*
  - *$c'_{|B_1^{(i+1)} \cup \dots \cup B^{(i+1)}_{\min(p,q)-1}} = c_{|B_1^{(i)} \cup \dots \cup B^{(i)}_{\min(p,q)-1}}$*

Recall that a P_i-stack configuration of σ is encoded by a path in the sorting graph $\mathcal{G}^{(i)}$, corresponding to the $\ominus$-decomposition of the permutation $\sigma^{(i)}$ into blocks $B_j^{(i)}$. The last point of Lemma 5.22 ensures that the first levels (1 to $min(p^{(i)}, q^{(i+1)}) - 1$) are the same in

$\mathcal{G}^{(i+1)}$ than in $\mathcal{G}^{(i)}$. The first point of Lemma 5.22 ensures that the last levels ($> q^{(i+1)}$) of $\mathcal{G}^{(i+1)}$ form a complete graph whose vertices are all pushall stack configurations of corresponding blocks. So the only unknown levels for $\mathcal{G}^{(i+1)}$ are those between $\min(p^{(i)}, q^{(i+1)})$ and $q^{(i+1)}$ and we can compute them by testing accessibility.

There are distinct cases depending on the relative values of $p^{(i)}$ and $q^{(i+1)}$. To lighten the notations in the following, we sometimes write p (resp. q) instead of $p^{(i)}$ (resp. $q^{(i+1)}$).

Case $p^{(i)} = q^{(i+1)}$

If $p^{(i)} = q^{(i+1)}$ then $B^{(i+1)}_{q^{(i+1)}} \cap A^{(i)} = B^{(i)}_{p^{(i)}} \cap A^{(i)}$ (see Figure 5.4).

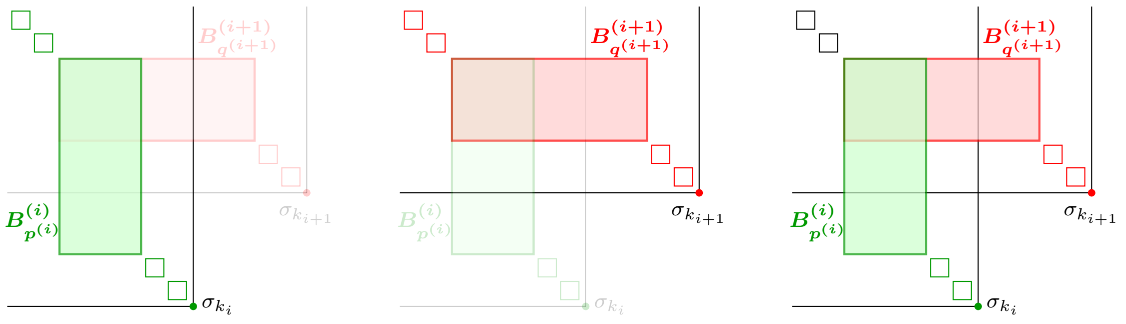


Figure 5.4: Block decomposition of $\sigma^{(i)}$ and of $\sigma^{(i+1)}$ when $p^{(i)} = q^{(i+1)}$

We have the sorting graph $\mathcal{G}^{(i)}$ encoding all P_i-stack configurations of σ and we want to compute the sorting graph $\mathcal{G}^{(i+1)}$ encoding all P_{i+1}-stack configurations of σ assuming that $p^{(i)} = q^{(i+1)} = min(p^{(i)}, q^{(i+1)})$.

In this case, from Lemma 5.22 we only have to check accessibility of pushall configurations of $B^{(i+1)}_q$ from configurations of $B^{(i)}_p$ belonging to level p of $\mathcal{G}^{(i)}$. Indeed from the definition of a sorting graph given p.204, for any vertex v of $\mathcal{G}^{(i)}$ there is a path between vertices of $V^{(i)}_1$ and $V^{(i)}_{s_i}$ going through v, and such a path corresponds to a P_i-stack configurations of σ. Thus for any configurations x of $B^{(i)}_p$ belonging to a vertex v of level p of $\mathcal{G}^{(i)}$, there is at least one P_i-stack configurations c of σ such that $c_{|B^{(i)}_p} = x$, and $c_{|B^{(i)}_1 \cup \dots \cup B^{(i)}_{\min(p,q)-1}}$ is encoded by a path from v to level p of $\mathcal{G}^{(i)}$ (which go through each level $< p$).

If there is no pushall configuration of $B^{(i+1)}_q$ accessible from some configurations of $B^{(i)}_p$ belonging to level p of $\mathcal{G}^{(i)}$, or if $\sigma^{(i+1)}$ has no pushall configuration, then σ has no P_{i+1}-stack configuration and $\sigma_{\leq k_{i+1}}$ is not sortable (from Lemma 5.20).

This leads to the following algorithm:

```
Algorithm 21: $iteratepEqualsq(\mathcal{G}^{(i)})$
Data: $\sigma$ a permutation and $\mathcal{G}^{(i)}$ the sorting graph at step $i$
Result: false if $\sigma_{\leq k_{i+1}}$ is not sortable, the sorting graph $\mathcal{G}^{(i+1)}$ otherwise.
begin
    $\mathcal{G}$ an empty sorting graph with $s_{i+1}$ levels;
    $\mathcal{G}' \leftarrow ComputeG1(\sigma^{(i+1)})$ (pushall sorting graph of $\sigma^{(i+1)}$) or return false;
    Copy levels $q+1 \dots s_{i+1}$ of $\mathcal{G}'$ into the same levels of $\mathcal{G}$;
    for $(c, \textcircled{\ell})$ in level $p$ of $\mathcal{G}^{(i)}$ do
        $\mathcal{H}$ the subgraph of $\mathcal{G}^{(i)}$ induced by $(c, \textcircled{\ell})$ in levels $< p$;
        for $(c', \textcircled{\ell'})$ in level $q$ of $\mathcal{G}'$ do
            if $isAccessible(c, c', \sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}})$ then
                Add $(c', \textcircled{\ell'})$ in level $q$ of $\mathcal{G}$ (if not already done);
                Merge $\mathcal{H}$ in levels $\leq q$ of $\mathcal{G}$ with $(c', \textcircled{\ell'})$ as origin;
    if level $q$ of $\mathcal{G}$ is empty then
        return false;
    for $(c', \textcircled{\ell'})$ in level $q$ of $\mathcal{G}$ do
        Add all edges from $(c', \textcircled{\ell'})$ to each vertex of level $q+1$ of $\mathcal{G}$;
    return $\mathcal{G}$
```

Case $p^{(i)} < q^{(i+1)}$

If $p^{(i)} < q^{(i+1)}$ then $B^{(i+1)}_{q^{(i+1)}} \cap A^{(i)} \subsetneq B^{(i)}_{p^{(i)}} \cap A^{(i)}$ (see Figure 5.5).

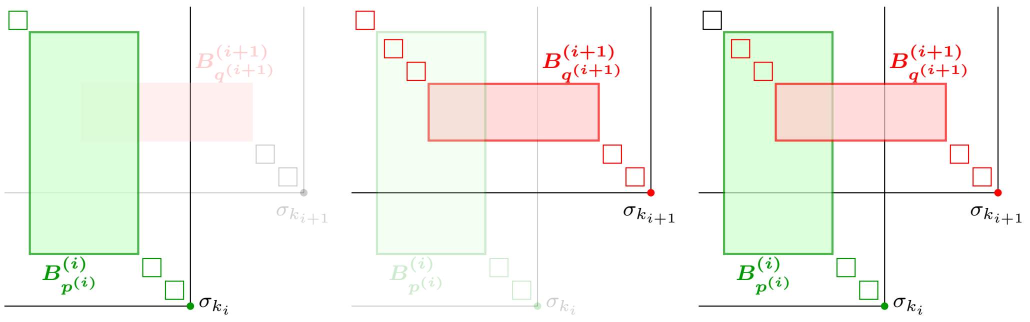


Figure 5.5: Block decomposition of $\sigma^{(i)}$ and of $\sigma^{(i+1)}$ when $p^{(i)} < q^{(i+1)}$

Again, Lemma 5.22 ensures that the first $p-1$ levels of $\mathcal{G}^{(i+1)}$ come from those of $\mathcal{G}^{(i)}$ and the levels $> q$ are all pushall stack configurations of the blocks $B^{(i+1)}_{>q}$ of $\sigma^{(i+1)}$. The difficult part is from level p to level q. As in the preceding case, by Lemma 5.22, we have to select among pushall stack configurations of blocks $p, p+1, \dots, q$ of $\sigma^{(i+1)}$ those accessible from a configuration of $B_p^{(i)}$ that appears at level p in $\mathcal{G}^{(i)}$. We can restrict the accessibility test from configurations of $B_p^{(i)}$ appearing in graph $\mathcal{G}^{(i)}$ to pushall stack configurations of $B_q^{(i+1)}$. Indeed, Lemma 5.13 ensures that the elements of blocks $B_j^{(i+1)}$ for j from p to $q-1$ are in the same stack at time t_i and at time t_{i+1}. Thus configurations of $B_j^{(i+1)}$ for j from p to $q-1$ are restrictions of configurations of $B_p^{(i)}$. We keep the same label in the vertex to encode that those configurations of $B_p^{(i+1)}, B_{p+1}^{(i+1)}, \dots, B_{q-1}^{(i+1)}$ come from the same configuration of $B_p^{(i)}$ and we build edges between vertices of $B_{j+1}^{(i+1)}$ and $B_j^{(i+1)}$ that come from the same configuration of $B_p^{(i)}$. It is because of this case $p = q$ that we have to label configurations in our sorting graph. Indeed two different stack configurations c_1 and c_2 of $B_p^{(i)}$ may have the same restriction to some block $B_j^{(i+1)}$ but not be compatible with the same configurations, thus we want the corresponding vertices of level j of $\mathcal{G}^{(i+1)}$ to be distinct; that's why we use labels.

More precisely we have the following algorithm.

Algorithm 22: *iteratepLessThanq*($\mathcal{G}^{(i)}$)

```
Data: $\sigma$ a permutation and $\mathcal{G}^{(i)}$ the sorting graph at step $i$
Result: false if $\sigma_{\leq k_{i+1}}$ is not sortable, the sorting graph $\mathcal{G}^{(i+1)}$ otherwise.
begin
    $\mathcal{G}$ an empty sorting graph with $s_{i+1}$ levels;
    $\mathcal{G}' \leftarrow ComputeG1(\sigma^{(i+1)})$ (pushall sorting graph of $\sigma^{(i+1)}$) or return false;
    Copy levels $q+1, \dots, s_{i+1}$ of $\mathcal{G}'$ into the same levels of $\mathcal{G}$;
    for $(c, \ell)$ in level $p$ of $\mathcal{G}^{(i)}$ do
        $\mathcal{H}$ the subgraph of $\mathcal{G}^{(i)}$ induced by $(c, \ell)$ in levels $< p$;
        for $(c', \ell')$ in level $q$ of $\mathcal{G}'$ do
            if $isAccessible(c, c', \sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}})$ then
                Add $(c', \ell')$ in level $q$ of $\mathcal{G}$ (if not already done);
                for $j$ from $q-1$ downto $p$ do
                    Add $(c_{|B_j^{(i+1)}}, \ell)$ in level $j$ of $\mathcal{G}$;
                    Add an edge between $(c_{|B_j^{(i+1)}}, \ell)$ and $(c_{|B_{j+1}^{(i+1)}}, \ell)$ in $\mathcal{G}$.
                Merge $\mathcal{H}$ in levels $\leq p$ of $\mathcal{G}$ with $(c_{|B_p^{(i+1)}}, \ell)$ as origin;
    if level $q$ of $\mathcal{G}$ is empty then
        return false;
    for $(c', \ell')$ in level $q$ of $\mathcal{G}$ do
        Add all edges from $(c', \ell')$ to each vertex of level $q+1$ of $\mathcal{G}$;
    return $\mathcal{G}$;
```

Note that in Algorithm 22, before calling $isAccessible(c, c', \sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}})$ we extend configuration c' to $D^{(i)} \bigcup B_q^{(i+1)}$ by assigning the same stack than in c to points of $D^{(i)} \setminus B_q^{(i+1)}$. This is justified by Lemma 5.13.

Case $p^{(i)} > q^{(i+1)}$

If $p^{(i)} > q^{(i+1)}$ then $B_{p^{(i)}}^{(i)} \cap A^{(i)} \subsetneq B_{q^{(i+1)}}^{(i+1)} \cap A^{(i)}$ (see Figure 5.6).

This case is very similar to the preceding one except that $B_p^{(i)}$ is not cut into pieces but glued together with preceding blocks. As a consequence, when testing accessibility of a configuration of $B_q^{(i+1)}$, we should consider every corresponding configuration in $\mathcal{G}^{(i)}$, that is every configuration obtained by stacking configurations at level $q, q+1, \dots, p$ in $\mathcal{G}^{(i)}$. Unfortunately this may give an exponential number of configurations, but noticing that by Lemma 5.13 the elements of blocks $B_q^{(i)}, B_{q+1}^{(i)} \dots B_{p-1}^{(i)}$ are exactly in the same stack at time t_i and at time t_{i+1}, it is sufficient to check the accessibility of a pushall configuration

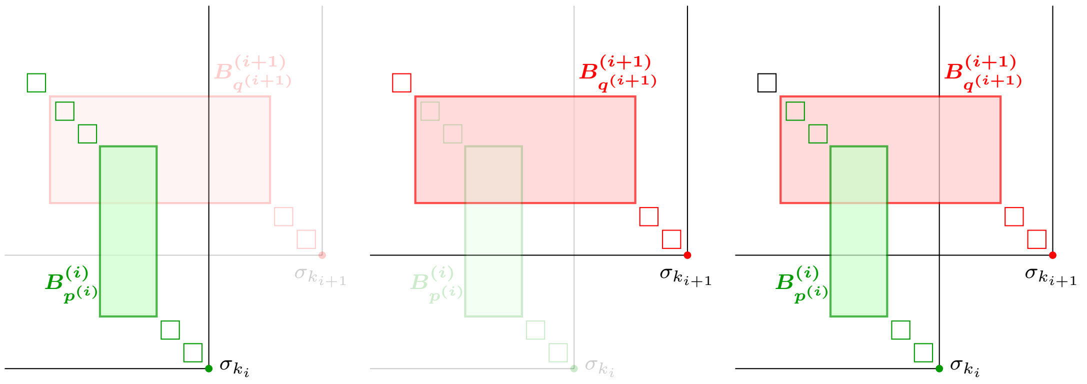


Figure 5.6: Block decomposition of $\sigma^{(i)}$ and of $\sigma^{(i+1)}$ when $p^{(i)} > q^{(i+1)}$

c' of $B_q^{(i+1)}$ from a configuration c of $B_p^{(i)}$ and verify afterwards whether the configuration c has ancestors in $\mathcal{G}^{(i)}$ that match exactly the configuration c'. This leads to the following algorithm.

Algorithm 23: *iteratepGreaterThanq*$(\mathcal{G}^{(i)})$

```
Data: $\sigma$ a permutation and $\mathcal{G}^{(i)}$ the sorting graph at step $i$
Result: false if $\sigma_{\leq k_{i+1}}$ is not sortable, the sorting graph $\mathcal{G}^{(i+1)}$ otherwise
begin
    $\mathcal{G}$ an empty sorting graph with $s_{i+1}$ levels;
    $\mathcal{G}' \leftarrow ComputeG1(\sigma^{(i+1)})$ (pushall sorting graph of $\sigma^{(i+1)}$) or return false;
    Copy levels $q+1, \dots, s_{i+1}$ of $\mathcal{G}'$ into the same levels of $\mathcal{G}$;
    for $(c, \ell)$ in level $p$ of $\mathcal{G}^{(i)}$ do
        for $(c', \ell')$ in level $q$ of $\mathcal{G}'$ do
            if $isAccessible(c, c', \sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}})$ then
                if there is a path $(c, \ell) \leftrightarrow (c'_{|B_{p-1}^{(i)}}, \ell_1) \leftrightarrow \dots \leftrightarrow (c'_{|B_q^{(i)}}, \ell_k)$ in $\mathcal{G}^{(i)}$
                then
                    Add $(c', \ell')$ in level $q$ of $\mathcal{G}$ (if not already done);
                    $\mathcal{H}$ the subgraph of $\mathcal{G}^{(i)}$ induced by $(c'_{|B_q^{(i)}}, \ell_k)$ in levels $< q$;
                    Merge $\mathcal{H}$ in levels $\leq q$ of $\mathcal{G}$ with $(c', \ell')$ as origin;
    if level $q$ of $\mathcal{G}$ is empty then
        return false;
    for $(c', \ell')$ in level $q$ of $\mathcal{G}$ do
        Add all edges from $(c', \ell')$ to each vertex of level $q+1$ of $\mathcal{G}$;
    return $\mathcal{G}$;
```

Note that in Algorithm 23, before calling $isAccessible(c, c', \sigma_{|B_p^{(i)} \bigcup B_q^{(i+1)}})$ we extend configuration c to $D^{(i)} \bigcup B_p^{(i)}$ by assigning the same stack as in c' to points of $D^{(i)} \setminus B_p^{(i)}$. This is justified by Lemma 5.13.

Now that we have described all steps of our algorithm, we turn to the study of its complexity.

5.5 Complexity Analysis

In this section we study the complexity of our main algorithm: *isSortable(σ)* (Algorithm 19). The key idea for the complexity study relies on a bound on the size of each graph $\mathcal{G}^{(i)}$, as described in the following lemma.

Lemma 5.23. *For any $i \in [1..r]$, the maximal number of vertices in a level of $\mathcal{G}^{(i)}$ is $9n$ where n is the size of the input permutation.*

Proof. From Theorem 4.68 of Chapter 4, the maximal number of pushall stack configurations of a $\ominus$-indecomposable permutation π is $9|\pi|$.

By definition of $\mathcal{G}^{(1)}$, the vertices of a level correspond to pushall stack configurations of a given block of the $\oplus_1$-decomposition of the input permutation σ (i.e. the $\oplus$-decomposition of $\sigma^{(1)}$). Thus the cardinality of a level is bounded by $9k$ where k is the size of the corresponding block. As $k \leq n$, the result holds for $i = 1$.

Suppose now that the result is true for a given $\mathcal{G}^{(i)}$; we show that it is then true for $\mathcal{G}^{(i+1)}$. The graph $\mathcal{G}^{(i+1)}$ is build from $\mathcal{G}^{(i)}$ using Algorithms 21, 22 or 23. In each case for a level j of $\mathcal{G}^{(i+1)}$ we have:

If $j > q^{(i+1)}$ then the vertices of the level j of $\mathcal{G}^{(i+1)}$ are the pushall stack configurations corresponding to the block $B_j^{(i+1)}$ of the $\oplus_{i+1}$-decomposition of σ. Thus Theorem 4.68 ensures that the cardinality of level j is bounded by $9n$.

If $j = q^{(i+1)}$ then the vertices of the level j of $\mathcal{G}^{(i+1)}$ are a subset of the pushall stack configurations corresponding to the block $B_j^{(i+1)}$ of the $\oplus_{i+1}$-decomposition of σ. Again Theorem 4.68 ensures that the cardinality of level j is bounded by $9n$.

If $j < p^{(i)}$ then the vertices of the level j of $\mathcal{G}^{(i+1)}$ are a subset of vertices of the level j of $\mathcal{G}^{(i)}$. By induction hypothesis the cardinality of level j is bounded by $9n$.

If $p^{(i)} \leq j < q^{(i+1)}$ then the vertices of the level j of $\mathcal{G}^{(i+1)}$ are restrictions of a subset of vertices of the level j of $\mathcal{G}^{(i)}$. By the induction hypothesis the cardinality of level j is bounded by $9n$, concluding the proof. ∎

Lemma 5.24. *For any $i \in [1..r]$, the number of vertices of $\mathcal{G}^{(i)}$ is $\mathcal{O}(n^2)$ and the number of edges of $\mathcal{G}^{(i)}$ is $\mathcal{O}(n^3)$, where n is the size of the input permutation.*

Proof. The result follows from Lemma 5.23 as there are at most n levels and there are edges only between consecutive levels. ∎

Theorem 5.25. *Given a permutation σ, Algorithm 19* isSortable(σ) *decides whether σ is sortable with two stacks in series in polynomial time w.r.t. $|\sigma|$.*

Proof. Algorithm 19 involves four other subroutines: *ComputeG1* (Algorithm 20), *iteratepEqualsq* (Algorithm 21), *iteratepLessThanq* (Algorithm 22) and *iteratepGreaterThanq* (Algorithm 23).

Each for-loop in these algorithms is executed at most a linear number of times by Lemma 5.23.

Moreover each included operation is polynomial by Lemmas 5.24 and 5.10. ∎

A more precise analysis of complexity leads to an overall complexity of $\mathcal{O}(n^5)$.

5.6 Conclusion and perspectives

In the second part of this thesis, we have defined a new restriction of 2-stack sorting, namely 2-stack pushall sorting. We provide an $\mathcal{O}(n^2)$ algorithm which computes an encoding of all pushall sortings of a given permutation, which thus decides whether a permutation is 2-stack pushall sortable. We proved that this complexity is optimal. Moreover we characterize 2-stack pushall sortable permutations as permutations whose diagram admits a valid coloring, which is a particular bicoloring defined by means of excluded colored patterns.

More studies remain to be done on 2-stack pushall sorting. First, a simpler characterization of 2-stack pushall sortable permutations would be interesting. Second, it would be nice to enumerate 2-stack pushall sortable permutations by finding a closed formula for the number of 2-stack pushall sortable permutations of size n or by finding their generating function. Our study of the precise form of valid bicolorings should help to find an enumeration. However a 2-stack pushall sortable permutation may admit several valid bicolorings. Nevertheless we strongly believe that our description of valid bicolorings will help to find bounds or asymptotic formulas. In particular we have ideas to find the growth rate of 2-stack pushall sortable permutations thanks to a bijection with ternary trees.

To find the enumeration or because of interest *per se*, it would be interesting to study more deeply the number of valid colorings of a given permutation, the number of pushall sortings of a given permutation, and the number of valid colorings corresponding to a given pushall sorting process, and to find general bounds on these numbers. We already know that each 2-stack pushall sortable permutation admits at least one valid coloring, that each valid coloring corresponds to exactly one sorting process, and that the number of valid colorings corresponding to a given sorting process is one plus the number of moves λ between the last move ρ and the first move μ. We also know that a $\ominus$-decomposable permutation of size n has at most $9n$ valid colorings (thus at most $9n$ pushall sorting processes) but that the decreasing permutation of size n has 2^n valid colorings and 2^{n-1} pushall sorting processes. It would also be nice to be able to define a canonical valid coloring or a canonical pushall sorting for the 2-stack pushall sortable permutations.

We also showed that pushall sorting and general sorting are closely linked, and provide characterizations of the elements of both bases whose node of the decomposition tree is linear. It would be nice to have a complete characterization of both bases (which are infinite).

Thanks to the link between pushall sorting and general sorting, we have provided a polynomial algorithm deciding whether a permutation σ is 2-stack sortable, settling a long standing open problem. Moreover this algorithm gives an encoding of all sorting processes of σ satisfying a particular property (Property (P) of Definition 5.15). We first showed that any 2-stack sortable permutation admits a sorting process satisfying Property (P). Then the algorithm proceeds in r steps, r being the number of right-to-left minima of σ: At step i the algorithm gives an encoding of all sorting processes of $\sigma_{\leq k_i}$ satisfying Property (P), $\sigma_{\leq k_i}$ being the prefix of σ ending by the i-th right-to-left minimum of σ.

The exact complexity of this algorithm is still to be analyzed. In particular we do not know if it is optimal.

We hope that this polynomial algorithm will allow a better understanding of 2-stack sortable permutations. In particular, can one find a characterization of 2-stack sortable permutations, for example by a property of their diagram (as for 2-stack pushall sortable permutation)? Can one enumerate 2-stack sortable permutations, or at least find asymptotics? We know [AAL10] that the growth rate of 2-stack sortable permutations is between 8.156 and 13.374. Can we refine these bounds thanks to our work?

Another natural continuation of our work is to generalize it for t stacks in series with $t > 2$. The notion of pushall sorting can be directly translated with t stacks in series: A permutation is t-stack pushall sortable if it admits a sorting with t stacks in series in two parts; in the first part we are not allowed to pop out from the last stack and in the second part we are not allowed to push elements from the input. In other words, the first part consists only of putting all the elements into the stacks while the second step consists only of outputting all the elements in increasing order.

This notion of pushall sorting is still closely linked with general sorting even when $t > 2$. Indeed we still have that σ is t-stack pushall sortable if and only if $\ominus[\sigma, 1]$ is t-stack sortable. In particular, most of the results of subsection 4.2.3 and some of the results of subsection 4.2.3 still hold when $t > 2$.

However, it is not obvious that we can use pushall sorting to find a polynomial algorithm deciding whether a permutation is t-stack sortable for $t > 2$.

Indeed, assume that we have a polynomial algorithm deciding whether a permutation is t-stack pushall sortable. To use it for general sorting, we can as for $t = 2$ consider time t_i when the i-th right-to-left minimum enters the first stack, and we still have to consider pushall sortings of the permutation $\sigma^{(i)}$ defined in Definition 5.1. But deciding the accessibility between configurations at time t_i and configurations at time t_{i+1} seems difficult when $t > 2$.

Neither is it obvious that deciding whether a permutation is t-stack pushall sortable is polynomial for $t > 2$.

Indeed for $t = 2$ the polynomial algorithm relies on the characterization of 2-stack pushall sortable permutations by means of a bicoloring. We could imagine characterizing t-stack pushall sortable permutations by means of a t-coloring. However the bijection with bicoloring in the case $t = 2$ comes from the fact that the stacks are ordered during a sorting procedure: the elements of the first stack H are in increasing order of indices from bottom to top and the elements of the second stack V are in decreasing order of values from bottom to top.

For $t > 2$ it is still true that the elements of the first stack are in increasing order of indices from bottom to top and the elements of the last stack are in decreasing order of values from bottom to top, but we do not know anything about the elements in the other stacks.

Nevertheless it is still true that a t-stack pushall sorting process can be viewed as two steps of a $(t-1)$-stack sorting process, considering first the last stack as the output and then the first stack as the input. However generalizing $(t-1)$-stack sorting to t-stack sorting may be very difficult, since generalizing 1-stack sorting to 2-stack sorting has proved to be so complex.

For t parallel stacks, we know that the decision problem can be answered in time $\mathcal{O}(n \log n)$ for $t \leq 3$, while for $t > 3$ it is NP-complete [EI71, Ung92]. We may wonder if for t serial stacks there also exists an integer k such that the decision problem is polynomial for $t \leq k$ and NP-complete for $t > k$.

More generally, given a permutation σ we may wonder how many stacks in series are necessary to sort σ. There is a naive algorithm to answer this question: we know that σ can be sorted by $\log_2(n)$ stacks where $n = |\sigma|$. Thus we can test for each t from 1 to $\log_2(n)$

whether σ is t-stack sortable by checking all stack words. However such an algorithm is highly inefficient. Is there a polynomial algorithm answering this question?

List of Figures and Tables

List of Algorithms

Bibliography

[AA05] Michael H. Albert and Mike D. Atkinson. Simple permutations and pattern restricted permutations. *Discrete Mathematics*, 300(1–3):1–15, 2005. 12, 14, 15, 16, 23, 24, 25, 27, 28, 31, 32, 36, 39, 54, 55, 114, 119, 120, 121, 123, 124, 126, 139, 140

[AAAH01] Michael H. Albert, Robert E. L. Aldred, Mike D. Atkinson, and Derek A. Holton. Algorithms for pattern involvement in permutations. In *ISAAC '01: Proceedings of the 12th International Symposium on Algorithms and Computation*, volume 2223 of *Lecture Notes in Computer Science*, pages 355–366, London, UK, 2001. Springer-Verlag. 15, 50, 55, 59, 108

[AAB$^+$10] Michael H. Albert, Mike D. Atkinson, Robert Brignall, Nik Ruškuc, Rebecca Smith, and Julian West. Growth rates for subclasses of Av(321). *Electronic Journal of Combinatorics*, 17:Paper R141, 2010. 31

[AAB$^+$13] Michael H. Albert, Mike D. Atkinson, Mathilde Bouvel, Nik Ruškuc, and Vincent Vatter. Geometric grid classes of permutations. *Transactions of the American Mathematical Society*, To appear, 2013. 31

[AAK03] Michael H. Albert, Mike D. Atkinson, and Martin Klazar. The enumeration of simple permutations. *Journal of Integer Sequences*, 6, 2003. 23, 31, 47

[AAL10] Michael Albert, Mike D. Atkinson, and Steve Linton. Permutations generated by stacks and deques. *Annals of Combinatorics*, 14:3–16, 2010. 145, 148, 192, 214

[ABM] Michael H. Albert and Mireille Bousquet-Mélou. Permutations sortable by two stacks in parallel. In preparation. 145

[AC75] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6), June 1975. 89

[ALR05] Michael H. Albert, Steve Linton, and Nik Ruškuc. The insertion encoding of permutations. *Electronic Journal of Combinatorics*, 12:Paper R47, 2005. 31

[AMR02] Mike D. Atkinson, M. M. Murphy, and N. Ruskuc. Sorting with two ordered stacks in series. *Theor. Comput. Sci.*, 289:205–223, October 2002. 145, 146, 193

[ARS11] Mike D. Atkinson, Nik Ruškuc, and Rebecca Smith. Substitution-closed pattern classes. *Journal of Combinatorial Theory, Series A*, 118(2):317–340, 2011. 31

[AS02] Mike D. Atkinson and Timothy Stitt. Restricted permutations and the wreath product. *Discrete Mathematics*, 259(1–3):19–36, 2002. 27

[B0́3] Miklós Bóna. A survey of stack-sorting disciplines. *The Electronic Journal of Combinatorics*, 9(2), 2003. 21

[Bar09] Jean-Luc Baril. More restrictive Gray codes for some classes of pattern avoiding permutations. *Inform. Process. Lett.*, 109:799–804, 2009. 114

[BBL98] Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65:277–283, 1998. 21, 114

[BBPR10] Frédérique Bassino, Mathilde Bouvel, Adeline Pierrot, and Dominique Rossin. Deciding the finitness of the number of simple permutations contained in a wreath-closed class is polynomial. *Pure Mathematics and Applications*, 21(2):119–135, 2010. 31, 55, 62, 93

[BBR11] Frédérique Bassino, Mathilde Bouvel, and Dominique Rossin. Enumeration of pin-permutations. *Electronic Journal of Combinatorics*, 18, 2011. 16, 53, 54, 55, 56, 61, 65, 66, 67, 68, 69, 73, 74, 88, 108, 109

[BCMR08] Anne Bergeron, Cédric Chauve, Fabien de Montgolfier, and Mathieu Raffinot. Computing common intervals of K permutations, with applications to modular decomposition of graphs. *SIAM Journal on Discrete Mathematics*, 22(3):1022–1039, June 2008. 55, 108

[BCMR11] Mathilde Bouvel, Cédric Chauve, Marni Mishna, and Dominique Rossin. Average-case analysis of perfect sorting by reversals. *Discrete Mathematics, Algorithms and Applications*, 3(3), 2011. 108

[Ber68] Claude Berge. *Principes de combinatoire.* Dunod, Paris, 1968. 12

[BHV08a] Robert Brignall, Sophie Huczynska, and Vincent Vatter. Decomposing simple permutations, with enumerative consequences. *Combinatorica*, 28(4):385–400, jul 2008. 31

[BHV08b] Robert Brignall, Sophie Huczynska, and Vincent Vatter. Simple permutations and algebraic generating functions. *Journal of Combinatorial Theory, Series A*, 115(3):423–441, 2008. 16, 31, 55

[BM03] Mireille Bousquet-Mélou. Four classes of pattern-avoiding permutations under one roof: Generating trees with two labels. *Electr. J. Comb.*, 9(2), 2003. 22, 31

[Bón02] Miklós Bóna. A survey of stack-sorting disciplines. *Electr. J. Comb.*, 9(2), 2002. 144, 145

[BR06] Mathilde Bouvel and Dominique Rossin. The longest common pattern problem for two permutations. *Pure Mathematics and Applications*, 17(1–2):55–69, 2006. 36

[Bri10] Robert Brignall. A survey of simple permutations. In Steve Linton, Nik Ruškuc, and Vincent Vatter, editors, *Permutation Patterns, St Andrews 2007*, volume 376 of *London Mathematical Society Lecture Note Series*, pages 41–65. Cambridge University Press, Cambridge, 2010. 14, 23, 31

[Bri12] Robert Brignall. Grid classes and partial well order. *Journal of Combinatorial Theory, Series A*, 119:99–116, 2012. 31

[BRV07] Mathilde Bouvel, Dominique Rossin, and Stéphane Vialette. Longest common separable pattern among permutations. In *CPM '07: Proceedings of the 18th annual symposium on Combinatorial Pattern Matching*, volume 4580 of *Lecture Notes in Computer Science*, pages 316–327, Berlin, Heidelberg, 2007. Springer-Verlag. 50

[BRV08] Robert Brignall, Nik Ruškuc, and Vincent Vatter. Simple permutations: decidability and unavoidable substructures. *Theoret. Comput. Sci.*, 391(1–2):150–163, 2008. 15, 16, 31, 32, 53, 54, 55, 58, 59, 60, 61, 67, 108, 140

[BXHP05] Binh-Minh Bui Xuan, Michel Habib, and Christophe Paul. Revisiting T. Uno and M. Yagiura's algorithm. In *Lecture notes in computer science*, volume 3827, pages 146–155. ISAAC, Springer, 2005. 108

[Cib09] Josef Cibulka. On constants in the Füredi–Hajnal and the Stanley–Wilf conjecture. *Journal of Combinatorial Theory, Series A*, 116(2):290–302, 2009. 31

[DFLS04] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4–5):577–625, 2004. 114, 115, 117, 118, 139

[DFMV08] W. M. B. Dukes, M. F. Flanagan, T. Mansour, and V. Vajnovski. Combinatorial gray codes for classes of pattern avoiding permutations. *Theoretical Computer Science*, 396:35–49, 2008. 114

[DV] Phan Thuan Do and Vincent Vajnovszki. Exhaustive generation of some classes of pattern avoiding permutations using succession functions. Conference in honor of Donald E. Knuth, Bordeaux, France (2007). 114

[EI71] S. Even and A. Itai. Queues, stacks, and graphs. In *Theory of Machines and Computations*, pages 71–86. Academic Press, 1971. 145, 192, 214

[Eli04] Sergi Elizalde. *Statistics on pattern-avoiding permutations.* PhD thesis, MIT, 2004. 22, 31

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics.* Cambridge University Press, Cambridge, 2009. 32, 33, 114, 115, 116, 139

[FZVC94] Philippe Flajolet, Paul Zimmerman, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1–2):1–35, 1994. 117

[HS01] Steffen Heber and Jens Stoye. Finding all common intervals of k permutations. In *12th Annual Symposium Combinatorial Pattern Matching, (CPM 2001)*, volume 2089 of *Lecture Notes in Computer Science*, pages 207–218. Springer Verlag, 2001. 25

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979. 59, 88, 89, 91, 110

[KM03] Sergey Kitaev and Toufik Mansour. A survey on certain pattern problems. available at http://www.ru.is/kennarar/sergey/publications.html, 2003. 22, 31, 114

[Knu68] Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms.* Addison-Wesley, 1968. 13, 143, 144

[Knu73a] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, 1973. 143, 145, 156, 191

[Knu73b] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming.* Addison-Wesley, Reading MA, 3rd edition, 1973. 31

[LS90] Venkatramani Lakshmibai and B Sandhya. Criterion for smoothness of Schubert varieties in $SL(n)/B$. In *Proceedings of the Indian Academy of Sciences-Mathematical Sciences*, volume 100, pages 45–52. Springer, 1990. 21

[MR84] Rolf H. Möhring and Franz J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Math*, 19:257–356, 1984. 22

[MT04] Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley-Wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004. 22, 31, 114

[Mur02] Maximillian M. Murphy. *Restricted permutations, anti chains, atomic classes and stack sorting.* Phd thesis, University of St Andrews, 2002. 17, 145, 148, 150, 162

[PR12] Adeline Pierrot and Dominique Rossin. Simple permutation poset. Preprint available at `http://arxiv.org/abs/1201.3119`, 2012. 31

[Pra73] Vaughan R. Pratt. Computing permutations with double-ended queues, parallel stacks and parallel queues. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison a nd Richard M. Karp, and H. Raymond Strong, editors, *STOC - Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 268–277. ACM, 1973. 144, 145

[PSS08] C. Pivoteau, B. Salvy, and M. Soria. Boltzmann oracle for combinatorial systems. In *Algorithms, Trees, Combinatorics and Probabilities*, pages 475–488. Discrete Mathematics and Theoretical Computer Science, 2008. Proceedings of the Fifth Colloquium on Mathematics and Computer Science. Blaubeuren, Germany. September 22-26, 2008. 118

[RT84] Pierre Rosenstiehl and Robert Endre Tarjan. Gauss codes, planar hamiltonian graphs, and stack-sortable permutations. *J. Algorithms*, 5(3):375–390, 1984. 192

[ST93] James H. Schmerl and William T. Trotter. Critically indecomposable partially ordered sets, graphs, tournaments and other binary relational structures. *Discrete Mathematics*, 113:191–205, 1993. 14, 35, 36, 37, 38, 39, 40

[Tar72] Robert Endre Tarjan. Sorting using networks of queues and stacks. *J. ACM*, 19(2):341–346, 1972. 144

[Úlf11] Henning Úlfarsson. Describing West-3-stack-sortable permutations with permutation patterns, 2011. 146

[Ung92] Walter Unger. The complexity of colouring circle graphs (extended abstract). In Alain Finkel and Matthias Jantzen, editors, *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 389–400. Springer, 1992. 145, 214

[Vat08] Vincent Vatter. Enumeration schemes for restricted permutations. *Combinatorics, Probability and Computing*, 17(1):137–159, 2008. 22, 31

[Vat10] Vincent Vatter. Permutation classes of every growth rate above 2.48188. *Mathematika*, 56:182–192, 2010. 31, 140

[Vat11] Vincent Vatter. Small permutation classes. *Proceedings of the London Mathematical Society*, 103:879–921, 2011. 31

[VW11] Vincent Vatter and Steve Waton. On partial well-order for monotone grid classes of permutations. *Order*, 28:193–199, 2011. 31

[Wes90] Julian West. *Permutations with forbidden subsequences and Stack sortable permutations.* Phd thesis, Massachusetts Institute of Technology, 1990. 145, 192

[Wes93] Julian West. Sorting twice through a stack. *Theor. Comput. Sci.*, 117(1&2):303–313, 1993. 145

[Zei92] Doron Zeilberger. A proof of Julian West's conjecture that the number of 2-stack sortable permutations of length n is 2(3n)!/((n + 1)!(2n + 1)!). *Discrete Mathematics*, 102(1):85–93, 1992. 146