

**Université Paris Diderot - Paris 7**  
**UFR de Mathématiques**

**Thèse**

pour obtenir le titre de

**Docteur de l'université Paris Diderot - Paris 7**

Spécialité mathématiques

présentée par

**Yann STROZECKI**

---

**Enumeration complexity and matroid decomposition**

---

Directeur de thèse : **Arnaud DURAND**

Soutenue publiquement le 8 décembre 2010, devant le jury composé de :

M. <b>Bruno</b>	<b>Courcelle</b>	Rapporteur
M. <b>Arnaud</b>	<b>Durand</b>	Directeur
M. <b>Miki</b>	<b>Hermann</b>	Co-Directeur
M. <b>Daniel</b>	<b>Král'</b>	Rapporteur
M. <b>Guillaume</b>	<b>Malod</b>	
M. <b>Michel</b>	<b>De Rougemont</b>	Rapporteur
M. <b>Gilles</b>	<b>Villard</b>	



# Remerciements

Je veux d'abord remercier Arnaud Durand de m'avoir accompagné pendant ces trois années et quelques de thèse. J'ai beaucoup apprécié sa gentillesse, sa disponibilité malgré son emploi du temps chargé et son ouverture sur un grand nombre de domaines de l'informatique théorique.

Je tiens à remercier Bruno Courcelle, Daniel Král' et Michel de Rougemont qui ont accepté d'être rapporteurs de ma thèse. Je suis extrêmement touché qu'ils soient tous présents pour ma soutenance, particulièrement Daniel Král', qui vient de Prague. Je remercie également les autres membres du jury – Guillaume Malod, Miki Hermann et Gilles Villard– d'avoir pris de leur temps pour lire mon manuscrit et assister à la soutenance.

Arnaud Durand, David Duris et Guillaume Malod ont suggéré un nombre incalculable de corrections et d'améliorations à mon manuscrit et surtout à mon anglais ! Bruno Courcelle a fait de nombreux commentaires qui m'ont aidé à améliorer le fond et la forme de la deuxième partie de cette thèse. Hervé Fournier et Sylvain Pérfel m'ont permis de rendre le chapitre trois présentable, et les discussions que j'ai eu avec eux et Guillaume pendant deux ans m'ont beaucoup appris : maintenant je sais ce qu'est un *circuit skew* (mais je suis un peu biaisé). Je les remercie tous pour m'avoir fait (un peu) progresser dans la dure discipline qu'est l'écriture scientifique.

Merci à tous les professeurs de mathématiques pour qui l'enseignement est plus une joie qu'une croix, je pense par exemple au sourire de M. Malric et au talent de beaucoup d'autres.

Ce long travail n'aurait pas été possible sans tous les moments de détente et de bonheur que j'ai pu avoir avec mes amis de Chevaleret ou d'ailleurs. Il est d'usage de remercier le bureau 5C06 (et un peu le bureau 5B01, on n'est pas raciste) pour l'ambiance de \*travail\* agréable. Sur le travail, je ne dirai rien, mais, pourtant, j'y suis allé tous les jours pendant trois ans, c'est dire la qualité de ses habitants, merci donc à :

Avenilde, pour avoir été un petit bout de Mexique à Paris et une grande amie dans mon cœur. David, pour quelques soirées surréalistes, des histoires dignes des feux de l'amour (promis je ne révèle rien) et pour son soutien. Fares, parce que c'est mon Kousin et qu'il vient de Berlin (il dit pré-géométrie pour dire matroïde, ces gens là ne parlent pas comme nous). Ana, à qui j'ai pu raconter tant d'histoires et vice-versa. Victor, pour avoir essayé de m'apprendre à danser. Au duo chilio-didacticien Carolina/Quelita, sept-soles Rémi, Brice le modeste trappeur, Clément et Gonenc, Karim, Laura ses pantoufles et son manchon, Marc le cohérent, François... Aux deux petits "nouveaux", one-eyed Pablo le mousquetaire et Julia, je sais qu'après mon départ le bureau sera plus que vivant grâce à eux. Aux visiteurs du midi, Jérôme a.k.a. Machete et Nini patte de lapin : je trouve que la fac a eu une excellente initiative en embauchant des clowns pour détendre ses thésards.

Merci à Pierre pour m'avoir cité dans ses remerciements et à Alexis, qui va bientôt m'inviter pour un post-doc au Chili. À Pablo W., pour ses idées saugrenues et ses petits

cafés. À Arnaud, de m'honorer de son amitié une fois par an (environ).

Merci à la Fabryk, pour m'avoir offert un espace de création, d'expression et surtout m'avoir fait rencontrer tellement de gens géniaux que je vais forcément en oublier beaucoup : Alexis, Fanny, Fatou, Sonia, Flora, Amel, Simon, Rémi, Sophie-anne, Charlotte, Thibault, mon coiffeur... et Manu. Merci à Catherine la patate, à qui j'ai dû faire subir les pires moments de ma thèse et qui m'a aidé à compenser en me gavant de quantités incroyables de Kouign-aman et autre morue séchée.

Généralement le CIES a mauvaise presse, mais ses ateliers théâtres m'ont permis de rencontrer Géraldine, JE, Benoît... qu'il en soit remercié à jamais.

Merci à mes amis de toujours :

Yanis même s'il perd en gold league, Nico K., Laurène et Lily, leur sous-produit, de m'avoir accueilli si souvent chez eux avec de bons "sandwichs". Thomas, qui a passé, sans le vouloir, sa vie à me suivre (ou est-ce le contraire). J'en déduis que je serai à Lyon l'année prochaine. Jade et Jules, on se boit quelques pintes ? Emily dont la gentillesse arrive encore à me surprendre et Fabien H. pour avoir essayé de me briser les bras pendant que je corrigeais ce manuscrit. Nico N. d'avoir tout partagé avec moi depuis treize ans, même ses amis : Benoît, Juju, Pouikipoo, Laure qui me fait sentir agréablement snob et intello quand je vais voir des expos avec elle, Fabien M. qui restera anonyme. Marie, qui, lorsqu'elle s'envole, veut bien parfois me prendre sous son aile. À Jérémie, qui est retenu sous une avalanche de choucroute depuis trois ans, mais que j'aime quand même.

Pour ne pas être redondant, je choisis cet emplacement pour dire merci à Jo, mon alter ego, mais il pourrait être dans chacun des autres paragraphes, travail, ami du bureau, ami d'ailleurs, presque famille. Je n'oublierai pas sa générosité à faire partager la musique qu'il aime (sa voisine non plus).

À Mia, sans sa maison je n'aurai jamais commencé à rédiger et sans elle je gagnerai moins souvent au Trivial Pursuit. À Dany, pour ses jolis chapeaux. À Nath, qui m'a aidé à développer mon alcoolisme en bonne compagnie, à Cyrille, qui m'a fait découvrir les plus belles fontaines en forme de dauphin du monde.

Merci à Camille, qui est comme du bon vin, il s'améliore tout les jours et il réconforte dans les moments difficiles. Merci à ma mère de se faire du souci parce que je ne m'en fais pas assez et à mon père de se faire du souci parce que je m'en fais trop. Merci à tous les deux de m'avoir appris, chacun à sa façon, à profiter de la vie et à être heureux.

En général merci, à la vie de valoir la peine d'être vécue et d'être remplie de tellement de beaux livres, films, chansons, théorèmes et rencontres qu'on se dit qu'on n'en aura jamais assez d'une.

Il y a dix-neuf occurrences de merci dans ce texte, en voilà une dernière pour tout ceux que j'ai oublié : Merci !

That's all.

# Contents

<b>1 Preliminaries</b>	<b>1</b>
1.1 Graphs, hypergraphs and matroids	1
1.1.1 Graphs	1
1.1.2 Hypergraphs	2
1.1.3 Matroids	3
1.1.4 Oriented matroids	5
1.2 Complexity	6
1.2.1 Problems	6
1.2.2 Model of computation	7
1.2.3 RAM machine for enumeration	8
1.2.4 Other complexity measures	9
1.2.5 Complexity classes: a short zoology	10
1.3 Logic	14
1.3.1 Structure	14
1.3.2 First-order logic	14
1.3.3 Second-order logic	15
1.3.4 The model-checking problem	16
1.4 Polynomials	16
1.4.1 Representation	16
1.4.2 Examples	18
<b>I Complexity</b>	<b>21</b>
<b>2 Enumeration</b>	<b>23</b>
2.1 Basics	24
2.2 Complexity measures and classes	25
2.2.1 Polynomial total time	26
2.2.2 Incremental polynomial time	28
2.2.3 Polynomial delay	29
2.3 Separation between classes	31
2.3.1 Unordered enumeration problems	32
2.3.2 Ordered enumeration problems	33

2.4	The power of ordering . . . . .	34
2.4.1	Hardness through a family of orders . . . . .	35
2.4.2	Hardness through one order . . . . .	35
2.4.3	Hardness through one enumerable order . . . . .	36
2.5	Operations on predicates and enumeration . . . . .	37
2.5.1	Union of predicates . . . . .	37
2.5.2	Subtraction of predicates . . . . .	39
2.5.3	Intersection of predicates . . . . .	40
2.6	An example: A-CIRCUIT . . . . .	40
<b>3</b>	<b>Enumeration of Monomials</b> . . . . .	<b>47</b>
3.1	Introduction . . . . .	47
3.1.1	Preliminaries . . . . .	49
3.2	Finding one monomial at a time . . . . .	49
3.2.1	The algorithm . . . . .	51
3.3	An incremental algorithm . . . . .	52
3.4	A polynomial delay algorithm . . . . .	53
3.4.1	Small values . . . . .	54
3.4.2	Large values . . . . .	55
3.4.3	Circuits . . . . .	56
3.4.4	The algorithm . . . . .	58
3.5	Complexity classes for randomized enumeration . . . . .	59
3.6	Higher degree polynomials . . . . .	62
3.6.1	KS algorithm . . . . .	63
3.6.2	Interpolation of fixed degree polynomials . . . . .	63
3.6.3	Comparison of interpolation methods . . . . .	66
3.7	Modest improvements . . . . .	67
3.7.1	Finite fields . . . . .	68
3.7.2	A method to decrease the degree . . . . .	68
3.7.3	Reduction of error and number of monomials . . . . .	69
3.7.4	Derandomization . . . . .	69
3.8	Hard questions for easy to compute polynomials . . . . .	70
3.8.1	Polynomials of unbounded degree . . . . .	71
3.8.2	Degree 3 polynomials . . . . .	72
3.8.3	Degree 2 polynomials . . . . .	72
3.8.4	Hardness regardless of the degree . . . . .	74
<b>4</b>	<b>Polynomials and Hypergraphs</b> . . . . .	<b>77</b>
4.1	Introduction to the Pfaffian Tree theorem . . . . .	77
4.2	Enumeration of the spanning hypertrees . . . . .	78
4.3	Parallelism . . . . .	79
4.4	Maximal spanning hypertree . . . . .	80

<b>II</b>	<b>Logic</b>	<b>83</b>
<b>5</b>	<b>Monadic Second-Order Logic</b>	<b>85</b>
5.1	Terms and trees . . . . .	85
5.2	Decomposition: the different notions of width . . . . .	86
5.2.1	Tree-width . . . . .	86
5.2.2	Branch-width . . . . .	88
5.2.3	Clique-width . . . . .	89
5.3	The logic MSO on graphs . . . . .	91
5.4	The logic MSO on higher order structures . . . . .	91
5.4.1	Hypergraphs . . . . .	91
5.4.2	Matroids . . . . .	92
<b>6</b>	<b>Decomposable Matroids</b>	<b>95</b>
6.1	Introduction . . . . .	95
6.2	Matroid decomposition . . . . .	96
6.2.1	Matroid branch-width . . . . .	96
6.2.2	Enhanced branch decomposition Tree . . . . .	98
6.3	Decision on an enhanced tree . . . . .	99
6.3.1	Signature . . . . .	99
6.3.2	From matroids to trees . . . . .	103
6.4	Extensions and applications . . . . .	106
6.4.1	Logic extension . . . . .	106
6.4.2	Spectra of $MSO_M$ formulas . . . . .	106
6.4.3	Enumeration . . . . .	108
6.5	Matroid operations . . . . .	109
6.5.1	Amalgam of boundaried matrices . . . . .	109
6.5.2	Series and parallel connections . . . . .	115
6.6	Discussion . . . . .	121
<b>7</b>	<b>Decomposable Hypergraphs</b>	<b>123</b>
7.1	Representation of a hypergraph . . . . .	123
7.2	Decomposition-width . . . . .	126
7.2.1	Structural properties . . . . .	128
7.3	Decision of MSO over decomposable hypergraphs . . . . .	129
7.3.1	Representation of hyperedges over a term . . . . .	130
7.4	Representation of hypergraph like structures . . . . .	131
7.4.1	Definable subclasses . . . . .	131
7.4.2	Encoding other decompositions . . . . .	133





# Introduction (en français)

Cette thèse s'intéresse à divers aspects de la théorie de la complexité. Celle-ci essaye de quantifier les "ressources" dont on a besoin pour résoudre un problème. Imaginons la situation suivante : Arthur veut remplir son sac à dos avec des objets utiles pour la quête du Graal, mais celui-ci est trop petit pour tout ranger. Comment va-t-il faire pour trouver une combinaison d'objets qui va remplir exactement son sac afin de ne pas perdre un espace précieux ? Le mieux qu'il puisse faire (en l'état actuel de nos connaissances) est de demander la réponse à Merlin, son enchanteur personnel. Celui-ci, grâce à ses talents de divination, trouve une solution. Par ailleurs, Arthur, qui n'a pas particulièrement confiance en Merlin –les mages ont leur propre agenda– peut facilement la vérifier : il met les objets proposés dans son sac et vérifie que celui-ci est parfaitement rempli. La ressource pour ce problème est donc un magicien douteux : c'est un problème NP.

Est-ce qu'Arthur aurait pu se débrouiller seul ? Il aurait pu essayer toutes les combinaisons d'objets, mais à ce rythme-là sa barbe serait devenue aussi blanche que celle de Merlin. De nombreux chercheurs spéculent qu'il ne peut pas faire mieux [Coo71], ce qu'on exprime par  $P \neq NP$ . En tout cas Arthur ne doit pas avoir honte de recourir à Merlin car aucun mathématicien n'a fait mieux que lui ces 40 dernières années. La difficulté de cette question peut paraître intimidante mais heureusement il existe de nombreux moyens de la contourner. La présente thèse en explore quelques-uns.

Le chapitre 1 fournit le vocabulaire pour poser et résoudre des problèmes de complexité. Il sert d'introduction technique à la thèse. Par ailleurs les principaux chapitres sont précédés d'une introduction plus détaillée que cette introduction générale.

**Énumération** Le petit problème d'Arthur est un problème de recherche : il veut obtenir une solution. S'il veut seulement savoir s'il y en a une, c'est un problème de décision, alors que s'il veut trouver le nombre de solutions on appelle cela un problème de comptage. Nous allons étudier dans les chapitres 2, 3 et 4 des problèmes d'énumération, c'est-à-dire qu'Arthur veut trouver toutes les solutions. Un problème difficile à décider l'est bien évidemment au moins autant dans sa version d'énumération. Ce nouveau point de vue permet donc de classer différemment des problèmes considérés comme faciles dans leur version de décision.

A Camelot, Arthur est confronté à un problème d'importance : en tant que souverain il doit organiser les mariages de ses dames et chevaliers. Après avoir considéré la naissance, le mérite et le thème astral de chacun, il sait quels couples feront la paire. Il doit maintenant résoudre le problème suivant : comment marier tout le monde sans décevoir personne (et est-ce seulement possible?). Comme précédemment il peut demander à Merlin une solution et il pourra ensuite rapidement décider si elle est correcte. Mais il peut faire mieux : il peut demander à Edmond [Edm65] et utiliser son algorithme pour trouver tout seul et rapidement une solution si il y en a une. Par contre, si il cherche le nombre de telles solutions (problème de comptage), il va avoir besoin d'un temps immense [Val79] (ou devenir plus riche d'un million de dollars<sup>1</sup>). S'il veut essayer chaque solution, il peut les énumérer avec un délai linéaire [Uno97] en le nombre de participants. Cet exemple illustre le fait qu'en changeant de point de vue sur un problème – décision, comptage ou énumération – on obtient différentes conclusions quant à sa complexité.

Les ressources considérées dans le cadre de l'énumération sont le temps pour trouver toutes les solutions et le délai entre deux solutions.

Dans le chapitre 2, nous étudions en détail les trois classes suivantes (et quelques autres) :

1. **TotalP** est la classe des problèmes énumérables en un temps polynomial en la taille de l'entrée et de la sortie
2. **IncP** est la classe des problèmes énumérables avec un délai polynomial en la taille de l'entrée et en le nombre de solutions déjà produites
3. **DelayP** est la classe des problèmes énumérables avec un délai polynomial en la taille de l'entrée

Quand on étudie des classes de complexité, on essaye de montrer qu'elles sont incluses strictement les unes dans les autres. Ici, comme dans beaucoup de cas, on n'arrive à montrer ce genre de résultat que relativement à des hypothèses comme  $P \neq NP$ . On obtient dans cette thèse la suite d'inclusions suivante :

$$\text{QueryP} \subsetneq \text{SDelayP} \subseteq \text{DelayP} \subseteq \text{IncP} \subsetneq \text{TotalP} \subsetneq \text{EnumP}$$

Malheureusement, les relations entre les deux classes les plus naturelles **IncP** et **DelayP** restent inconnues. On étudie aussi le rôle de l'ordre dans lequel l'énumération des solutions doit être fait, celui-ci pouvant créer des difficultés supplémentaires. Enfin on s'intéresse aux opérations ensemblistes sur les ensembles de solutions. On montre que seule l'union laisse stable la plupart des classes de complexité. Cela permet, par exemple, d'énumérer les solutions d'une formule sous forme normale disjonctive avec un délai linéaire (section 2.5).

<sup>1</sup>Un prix d'un million de dollars est offert par le Clay Mathematics Institute pour la résolution du problème  $P \neq NP$ .

**Hasard** Pour pouvoir résoudre plus efficacement des problèmes, on se permet d'utiliser un peu de hasard, c'est à dire qu'on a le droit de jouer à pile ou face. Cela permet de concevoir des stratégies ou protocoles qui ne pourraient pas fonctionner sans hasard. Par exemple Arthur, qui était daltonien, veut vérifier que Merlin ne lui joue pas un tour quand ce dernier prétend qu'il a mis des chaussettes dépareillées (une verte et une rouge). Il lui suffit de permuter aléatoirement (et secrètement) ses chaussettes et de demander à Merlin laquelle était à gauche. Si elles étaient de la même couleur, Merlin n'a aucune information et va se tromper une fois sur deux.

Dans le chapitre 3 on essaie de trouver une méthode qui permette à Arthur de gagner le jeu suivant : Merlin pense à un polynôme multivarié (comme tous les matins en se rasant) et Arthur doit le deviner. Dans ce but, il peut demander à Merlin la valeur du polynôme en un point. On appelle cela le problème d'interpolation pour un modèle à oracle. Les ressources sont le nombre de questions posées à Merlin et la quantité de calculs que doit effectuer Arthur. On peut montrer que si Arthur n'utilise pas de hasard, pour décider si le polynôme auquel pense Merlin est le polynôme nul (quel blagueur ce Merlin), il a besoin d'un nombre exponentiel de questions en le nombre de variables du polynôme. Comme nous voulons trouver le premier monôme rapidement, nous allons utiliser fondamentalement le hasard et un petit lemme [Zip79, Sch80].

Par contre, quand le polynôme a une représentation concrète, un petit circuit par exemple, il est peut-être possible de résoudre le problème sans hasard. En fait, le problème de décider si le polynôme est nul (POLYNOMIAL IDENTITY TESTING) sans utiliser de hasard (derandomization) est un sujet de recherche très actif ces dernières années. Il a été prouvé qu'on peut résoudre ce problème de manière déterministe sur des circuits de petite profondeur [KMSV10, SS10]. Le but est de montrer qu'on n'a pas besoin de hasard pour le résoudre sur les circuits en général. Cela aurait comme conséquence qu'il n'y a pas plus de problèmes qui puissent être résolus efficacement avec du hasard que sans. Ce qui, en langage mathématique, s'écrit  $P = BPP$ . En fait cette question est équivalente à la séparation de classes de circuits, c'est-à dire quelque chose de très proche de  $P \neq NP$  [IW97, KI04].

Le problème d'interpolation peut être vu comme un problème d'énumération : on veut donner les monômes du polynôme les uns après les autres dans un court délai. Le premier algorithme proposé dans cette thèse (section 3.3) permet d'interpoler les polynômes dont aucun monôme n'utilise exactement les mêmes variables avec un délai incrémental. Le second algorithme (section 3.4) interpole les polynômes multilinéaires (une sous-classe de la précédente) avec un délai polynomial. En outre, ces deux algorithmes ont une bonne complexité globale et utilisent dans leurs requêtes à Merlin des petits entiers. Ce dernier point est utile si Merlin pense à un polynôme sur un corps fini (ce qui arrive souvent car il a une mauvaise mémoire).

On peut encoder un certain nombre de problèmes combinatoires dans les monômes d'un polynôme facilement évaluable. Ainsi les deux précédents résultats peuvent fournir des algorithmes probabilistes pour des problèmes d'énumération

plus classiques, par exemple énumérer les couplages parfaits d'un graphe. On introduit donc les variantes probabilistes des classes d'énumération du chapitre 2, qu'on nomme **TotalPP**, **IncPP** et **DelayPP**. On montre ainsi, dans le chapitre 4, que le problème d'énumérer les hyperarbres couvrants d'un hypergraphe 3-uniforme est dans **DelayPP**.

Il existe déjà un algorithme qui permet d'interpoler les polynômes de degré quelconque avec un délai incrémental [KS01]. Nous présentons un algorithme alternatif (dans la section 3.6) pour résoudre ce problème, construit à partir des deux algorithmes précédemment proposés. Sa complexité est exponentiellement dépendante en le degré, il n'est donc meilleur que l'algorithme existant que pour des polynômes de degré inférieur à 10.

Enfin on essaye de comprendre pourquoi il est difficile de donner un algorithme à délai polynomial pour les polynômes de degré deux ou plus. On se restreint à l'étude de polynômes donnés par des circuits. On montre que des questions comme "est ce qu'un terme a un coefficient différent de zéro dans le polynôme ?" sont NP-dures pour les polynômes de degré deux. L'algorithme d'interpolation pour les polynômes multilinéaires ainsi que tout ceux que je peux imaginer peuvent résoudre cette question. Ceci suggère qu'il est difficile – impossible ? – de construire un algorithme d'interpolation plus général avec un délai polynomial.

**Parallélisme** Pour améliorer son pouvoir de résolution de problèmes, Arthur peut mettre à contribution ses sujets qui sont fort nombreux. Il leur donne un problème et ceux-ci doivent, en collaborant<sup>2</sup>, trouver sa solution. Si l'utilisation d'un grand nombre de cerveaux disponibles augmente la rapidité à trouver une solution, on dit que le problème est parallélisable.

Dans le chapitre 4, on étudie le problème d'énumération des hyperarbres couvrants d'un hypergraphe 3-uniforme, ce qui est l'occasion de proposer quelques notions de parallélisme adaptées à l'énumération.

On propose également un algorithme probabiliste pour trouver la taille du plus grand sous-hypergraphe acyclique d'un hypergraphe 3-uniforme. Cet algorithme est parallélisable, on dit que le problème est dans RNC. Cela signifie que si Arthur utilise un nombre de sujets polynomial en la taille de l'hypergraphe, il a besoin d'un temps polylogarithmique en cette taille pour obtenir la solution.

**Paramétrisation et logique** Quand un problème est NP-complet, on veut identifier ce qui le rend dur. C'est l'objet de la complexité paramétrée : on cherche un paramètre du problème qui une fois fixé le rend facile. Par exemple, dans le cas du sac à dos d'Arthur, si la taille des objets est fixée, le problème devient facile. De la même manière, si on fixe le degré des polynômes que l'on interpole, l'algorithme de la section 3.6 devient efficace.

---

<sup>2</sup>Pour que cette fable soit vraiment représentative de la définition formelle, il faudrait installer internet dans l'Angleterre arthurienne. En effet, tout les participants doivent pouvoir communiquer rapidement entre eux.

Il existe un cadre général, qui permet de décrire une très grande classe de problèmes qui deviennent faciles quand un certain paramètre est fixé. Pour comprendre cela, revenons à Arthur qui a récemment hérité d'un artefact puissant appelé automate d'arbre<sup>3</sup>. Il a également appris un langage magique appelé logique monadique du second ordre (ou *MSO* pour sauver des arbres). Tous les problèmes qu'il arrive à décrire dans ce langage et qui concernent des arbres, il peut les résoudre efficacement avec son automate d'arbre [TW68].

Souvent les problèmes qu'Arthur veut résoudre sont définis sur des graphes comme le problème des mariages et pas seulement sur des arbres. Il existe de nombreux paramètres de largeur de graphe –tree-width, branch-width, clique-width, rank-width ...– qui mesurent à quel point un graphe “ressemble” à un arbre. Certains de ces paramètres ont pour origine le “*Graph minor project*” [RS83, RS86, RS91, RS95] qui a permis de montrer que l'exclusion d'un nombre fini de mineurs est une manière universelle de définir l'identité d'un groupe de graphe<sup>4</sup>. De manière remarquable, on peut transférer les algorithmes efficaces qui existent sur les arbres en des algorithmes sur les graphes en augmentant leur complexité de manière importante mais dépendant uniquement du paramètre de largeur [Cou91].

Dans le chapitre 5, on fait un rapide tour d'horizon de différentes notions de largeur d'arbre et de leur relations. Cela sert d'introduction au chapitre suivant qui s'intéresse au même genre de problème mais sur une structure qui généralise les graphes.

Dans le chapitre 6, on s'intéresse aux matroïdes qui sont une manière d'axiomatiser la notion d'indépendance linéaire. Ils permettent d'abstraire l'espace des cycles d'un graphe ou des colonnes d'une matrice. De nombreux problèmes combinatoires sont en fait des cas particuliers de quelques problèmes comme l'intersection de matroïdes ou les mariages dans les matroïdes [Lov80]. Les matroïdes permettent aussi aux distributeurs automatiques de rendre leur monnaie aux gloutons.

La notion de branch-width se généralise facilement aux matroïdes et on voudrait obtenir des résultats comparables à ceux obtenus pour les graphes. Dans le cas des matroïdes représentés par des matrices sur des corps finis et de branch-width bornée, on peut décider la logique *MSO* en temps linéaire [Hli06]. Pour démontrer ce genre de résultat, il existe deux approches complémentaires :

- On donne des opérations, une grammaire qui permet de construire la classe d'objets qu'on étudie et on se sert des propriétés des opérations pour montrer le résultat recherché.
- On étudie des décompositions arborescentes qui caractérisent des bonnes propriétés de l'objet. Généralement cela veut dire qu'on peut couper la structure en petits morceaux tel que chacun “interagit” peu avec le reste.

<sup>3</sup>Un objet étrange : il parcourt les arbres de bas en haut et pourtant des feuilles vers la racine !

<sup>4</sup>Ce qui est mieux que pour les groupes d'humains ou l'identité est souvent définie par l'exclusion d'une infinité de personnes et pas que des jeunes.

La première partie du chapitre 6 est consacrée à une preuve alternative que la logique  $MSO$  est décidable en temps linéaire sur les matroïdes représentables de branch-width bornée. Pour démontrer cela, on crée un arbre de décomposition appelé *enhanced tree* qui encode de manière locale les informations permettant de décider de l'indépendance d'un ensemble dans un matroïde. Cette approche est plus simple et donne plus d'informations sur les matroïdes représentables que la preuve existante qui repose sur une grammaire.

Dans un deuxième temps, on introduit un cadre algébrique qui permet de définir la grammaire qui engendre les matroïdes représentables d'une certaine branch-width. On fait le lien avec la vision arbre de décomposition et on explique pourquoi il faut définir la grammaire sur les matrices plutôt que sur les matroïdes qu'elles représentent. Dans la section 6.5 on propose une grammaire, qui permet d'engendrer une classe de matroïdes qui ne sont pas nécessairement représentables et sur lesquels la décision de  $MSO$  est efficace. Les techniques de preuve de ce chapitre sont très uniformes et s'adaptent facilement à d'autres classes obtenues par des opérations similaires.

Enfin, dans le chapitre 7, on essaie d'abstraire les constructions du chapitre précédent. On introduit une représentation arborescente des hypergraphes qui généralise celles présentées pour les matroïdes. On peut de nouveau montrer que la logique monadique du second ordre est décidable en temps linéaire sur ces hypergraphes.

On étudie quelques propriétés du paramètre de largeur naturellement associé à cette représentation. Il se comporte notamment bien par rapport aux opérations simples sur les hypergraphes comme la complémentation, l'union disjointe ou la restriction des sommets. Enfin on montre comment un certain nombre de décompositions peuvent être vues comme des cas particuliers de celle-ci. Les résultats de ce chapitre ont donc surtout pour intérêt de donner un cadre simple pour imaginer d'autres décompositions arborescentes. Il reste, et c'est le plus difficile, à trouver des classes intéressantes d'hypergraphes <sup>5</sup> ainsi que des algorithmes pour produire leurs décompositions.

---

<sup>5</sup>sur lesquels la logique  $MSO$  permet d'exprimer des problèmes NP-complets

# Introduction (in english)

This thesis studies several aspects of computational complexity, the theory whose aim is to measure the “ressources“ one needs to solve a problem. Imagine the following situation: Arthur has a knapsack that he wants to load with useful objects in his quest to find the Grail. Which objects should he take with him in order to exactly fill its knapsack? The best he can do (to our current knowledge) is to ask Merlin, his personal wizard. Indeed, thanks to his magical powers, Merlin is able to guess the solution. However, Arthur who does not trust him much –wizards tend to be tricky– can easily check the solution: he just packs his knapsack with the suggested objects and checks wether it is exactly filled. The ressource for this problem is an untrustworthy wizard: it is a NP problem.

Could have Arthur found the solution himself? Well, he could have tried every combination of objects, but it is such a lengthy process that his beard would have become as white as Merlin’s. Many computer scientists think that he cannot do better [Coo71], a fact one writes  $P \neq NP$ . In this respect, Arthur must not be ashamed to ask for Merlin his help, since no mathematician has found a better way in the last 40 years. The hardness of this question may seem frightening, hopefully they are many ways to escape it. This thesis studies some of them.

Chapter 1 presents the vocabulary used to formulate and solve complexity problems. It is a technical introduction to the thesis. Most chapters also begin with a detailed and more specific introduction.

**Enumeration** Arthur’s problem is a search problem: he wants to find a solution. If he only wants to know if there is one, it is a decision problem, whereas if he wants to count the number of solutions, it is a counting problem. In the Chapters 2, 3 and 4 we study enumeration problems, that is Arthur wants to find all the solutions. An enumeration problem is obviously as hard, if not harder, as its decision version. This new point of view allows us to classify problems considered as easy in their decision version.

In Camelot, Arthur is responsible for the weddings of his knights and ladies. After much consideration about the birth, the virtue and the horoscope of everyone, he knows which couples will match. He must now solve the perfect matching problem: how to marry everyone without deceiving anyone (and is that even possible)? As before, Arthur may ask Merlin for the solution and then easily check if it works. But he can do better: he may ask Edmond [Edm65] for his algorithm

which gives a way to find quickly a solution if there is one. On the other hand, if he wants to count the number of solutions, it will require a huge time (or he becomes a million dollars wealthier<sup>6</sup>). If he needs all solutions, he can generate them with a linear delay in the number of ladies and knights to be married [Uno97]. This example illustrates the fact that by changing one's point of view on the problem –decision, counting or enumeration– we obtain contrasting conclusions on its complexity.

The considered resources in enumeration complexity are the time to find all solutions and the delay between two of them.

In Chapter 2, we study in detail the three following classes (and a few others):

1. **TotalP** is the class of enumeration problems solvable in polynomial time in the size of the input and the output
2. **IncP** is the class of enumeration problems solvable with a polynomial delay in the size of the input and in the number of already generated solutions
3. **DelayP** is the class of enumeration problems solvable with a polynomial delay in the size of the input

When one studies complexity classes, one tries to prove that they are properly included in one another. In this thesis, as in most of the literature, we prove this kind of result only under some hypothesis like  $P \neq NP$ . We obtain the following sequence of inclusions:

$$\text{QueryP} \subsetneq \text{SDelayP} \subseteq \text{DelayP} \subseteq \text{IncP} \subsetneq \text{TotalP} \subsetneq \text{EnumP}$$

Note that we still do not know if there is a proper inclusion between the two most natural classes **IncP** and **DelayP**. We also study the influence of the order in which the solutions have to be enumerated, since it can create some artificial hardness. Finally, we study set operations on enumeration problems. We prove that most complexity classes are stable under the union only. For instance, it enables us to enumerate the solutions of a formula in disjunctive normal form with a linear delay (Section 2.5).

**Randomness** To solve more efficiently some problems, we use a bit of randomness, that is to say we are allowed to flip a coin. It is useful to design strategy or protocols that wouldn't work otherwise. For instance Merlin tells Arthur, who is colorblind, that he is wearing a red sock and a green sock (and a king shall not wear red). Arthur wants to determine if Merlin is telling the truth and not playing one of his little tricks again. He only has to randomly (and secretly) swap his socks and ask Merlin which one was he wearing on his left foot. If they are of the same color, Merlin has no information to answer and his lie will be revealed every two times.

---

<sup>6</sup>A price of a million dollars is offered by the Clay Mathematics Institute for a solution to the problem  $P \neq NP$ .



In Chapter 3, we try to find a method to make Arthur win the following game: Merlin thinks about a multivariate polynomial (which proves that wizards and mathematicians are about the same) and Arthur has to guess it. To this aim, he can ask Merlin the value of the polynomial in any point. This is the interpolation problem for an oracle model. The resources are the number of questions asked to Merlin and the computations that Arthur does. One can prove that if Arthur does not use randomness, even to determine if Merlin's polynomial is zero (how funny of him), he needs an exponential number of questions in the number of variables of the polynomial. Since we want to find the first monomial quickly, we basically use randomness and a little lemma [Zip79, Sch80].

On the other hand, when the polynomial has an explicit representation, a small circuit for instance, it may be possible to solve the problem without randomness. In fact, the problem to decide if the polynomial is the zero polynomial (POLYNOMIAL IDENTITY TESTING) without using randomness (derandomization) is a very active topic of research. It has been proved that the problem can be solved deterministically for circuits of small depth [KMSV10, SS10]. The aim is to prove that one does not need randomness for any circuit. A consequence of this statement would be that there aren't more problems that can be solved efficiently with randomness than without. In mathematical terms, one says  $P = BPP$ . In fact this question is equivalent to the separation of class of circuits, that is to say something very close to  $P \neq NP$  [IW97, KI04].

The interpolation problem can be seen as an enumeration problem: one has to output the monomials of the polynomial one after another within a short delay. The first algorithm given in this thesis (Section 3.3) interpolates with incremental delay polynomials such that no two of their monomials use exactly the same variables. The second algorithm (Section 3.4) interpolates multilinear polynomials (and is thus less general) with a polynomial delay. Furthermore both algorithms have a good global complexity and use only small integers in their questions to Merlin. The last point is useful if Merlin thinks about a polynomial over a finite field (which happens often since he has a bad memory).

One can encode a good number of combinatorial problems into the monomials of an easy to compute polynomial. So the two previous algorithms can be used to solve more classical enumeration problems, for instance generating the perfect matchings of a graph. We introduce the probabilistic counterparts of the complexity classes of Chapter 2, that we name **TotalPP**, **IncPP** and **DelayPP**. As an illustration of these new classes, we prove in Chapter 4, that the problem of enumerating the spanning hypertrees of a 3-uniform hypergraph is in **DelayPP**.

There already is an algorithm to interpolate polynomials of any degree with an incremental delay [KS01]. We present an alternative algorithm (in Section 3.6) to solve this problem, inspired by the two algorithms built for restricted classes of polynomials. Its complexity depends exponentially on the degree, it is therefore better than the existing algorithm only for polynomials with a degree inferior to ten.

Finally, we try to understand why it is so difficult to find a polynomial delay

algorithm for polynomial of degree 2 or more. We prove that questions like "has a term got a coefficient different from zero in the polynomial?" are NP-hard for degree 2 polynomials. The interpolation algorithm for multilinear polynomials as well as all the polynomial delay ones I can think of are able to answer this question. This suggests that it is hard –impossible?– to build a more general interpolation algorithm with a polynomial delay.

**Parallelism** To improve his problem solving capacities, Arthur can rely on his numerous loyal subjects. He gives them a problem and they must work together<sup>7</sup> to find a solution. If the use of a great quantity of brains speed up the resolution of the problem, one says it is parallelizable.

In Chapter 4, one considers the problem of generating all spanning hypertrees of a 3-uniform hypergraph. It serves as an illustration for some notion of parallelism for enumeration that we introduce.

In Section 4.4, we provide a randomized algorithm which finds the size of the largest acyclic subhypergraph of a 3-uniform hypergraph. This algorithm can be parallelized, one says that the problem is in RNC. It means that if Arthur commands a number of subjects polynomial in the size of the hypergraph, he needs a polylogarithmic time in this size to solve the problem.

**Parametrized complexity and logic** When a problem is NP-complete, one wants to know what makes it hard. This is the problematic of parametrized complexity, that we can rephrase by what parameter can we fix to turn a hard problem into an easy one? For instance, filling Arthur's knapsack becomes easy if the size of the objects is fixed. In the same way, if we fix the degree of the polynomial we try to interpolate, the algorithm of Section 3.6 is efficient.

There is a framework to describe a great variety of problems made easy by fixing some parameters. To understand it, consider that Arthur just received from the Lady of the Lake, not Excalibur but a powerful artifact called a tree automaton<sup>8</sup>. He was also taught a magic language called monadic second order logic (or *MSO* to spare some trees). He can solve efficiently all the problems on trees he is able to describe in this language, thanks to its tree automaton [TW68].

Often, the problems that Arthur wants to solve are defined not on trees but on graphs, as the wedding problem. There are numerous width-parameters for graphs –tree-width, branch-width, clique-width, rank-width . . . – which measure how similar to a tree a graph is. Some of these parameters originate from the "Graph minor project" [RS83, RS86, RS91, RS95] which has proved that any hereditary class of graphs can be defined by exclusion of a finite number of minors. Quite surprisingly, one can use the efficient algorithms for trees on graphs by

---

<sup>7</sup>For this story to be accurate, one should bring the internet to medieval Britain. Indeed all the people working on the problem must be able to communicate quickly.

<sup>8</sup>A strange object which goes bottom-up but from the leaves to the root!

increasing their complexity in a way which depends only on the width parameter [Cou91].

In Chapter 5 some notions of width and their relationships are studied. It serves as an introduction to the next chapter which considers a related parameter on a structure which, in a way, generalizes graphs.

In Chapter 6 one considers matroids which are an axiomatization of the notion of linear dependency. A lot of combinatorial problems are particular instances of some problems on matroids like matroids intersection or matroid matching [Lov80]. Matroids are also used in vending machines to give their money back to greedy customers.

The notion of branch-width can be generalized to matroids and one wants results in the same vein as those obtained for graphs. In the case of matroids represented by matrices over finite fields and of bounded branch-width, one can decide the logic  $MSO$  in linear time [Hli06]. To prove this kind of result there are two complementary approaches:

1. One defines a grammar, that is to say a way to build a class of matroids from operations and a set of base matroids. One then uses the properties of the operations to prove a result on the whole class.
2. One studies tree-decompositions which characterize some good properties of a matroid. Usually it means that we can cut the matroid into small pieces such that they are mostly "isolated" from each other.

The first part of Chapter 6 is devoted to a new proof that the model-checking of  $MSO$  can be decided in linear time over representable matroids of bounded branch-width. To prove this, we introduce a decomposition tree called enhanced tree which locally encodes all necessary informations to decide the dependency of a set in a matroid. This approach is simpler and gives more informations than the existing proof which uses grammar techniques.

In the second part, we introduce an algebraic framework to define matroids grammar, in particular the one which defines representable matroids of bounded branch-width. We give the relationship between this grammar and the tree decomposition of the first part. We also explain why one should define the grammar on matrices rather than directly on the matroids they represent. In Section 6.5, we define a grammar which generates non necessarily representable matroids and for which the model-checking of  $MSO$  is still linear. The proof techniques of this chapter are very uniform and could be easily adapted to new operations on matroids.

In Chapter 7, the previous construction are abstracted. We introduce a tree representation of hypergraphs which generalizes the one given for matroids. We prove again that  $MSO$  is linear time decidable on these hypergraphs.

We then study some properties of the width-parameter naturally associated to this representation. This parameter is "regular" with regard to simple operations on hypergraphs like complementation, disjoint union or section. In Section 7.4, we

explain how graphs or matroids decompositions can be seen as particular instances of this hypergraph decomposition. This chapter can thus be helpful to imagine new tree-decompositions. We still have, and it is the hardest part, to find interesting classes of hypergraphs <sup>9</sup> and algorithms to find their decomposition.

---

<sup>9</sup>on which *MSO* can express NP-complete problems

# Chapter 1

## Preliminaries

This chapter provides some background and definitions about most of the objects and concepts which will later be used in this thesis. It should serve as both an introduction and a reference while reading the other chapters.

### 1.1 Graphs, hypergraphs and matroids

Here we introduce mathematical objects which are used to modelize a lot of computer science problems.

#### 1.1.1 Graphs

Some basic notions of graph theory are given in this subsection, for a more complete view on the subject one can read [Die05] among the numerous reference books available.

A graph  $G = (V, E)$  is a pair where  $V$  is a finite set of vertices and  $E \subseteq V \times V$  is the set of edges. This definition forbids loops, i.e. edges of the form  $\{v\}$ . We may also consider oriented graphs, where the edges are ordered pairs, that is to say the edges  $(u, v)$  and  $(v, u)$  are different.

A *path* of  $G$  is a sequence of distinct vertices  $\{x_1, \dots, x_k\}$  such that for all  $0 < i < k$ ,  $\{x_i, x_{i+1}\}$  is an edge of  $G$ . A *cycle* is a path such that  $x_1 = x_k$ . An *acyclic* graph is a graph with no cycle, we also call it a *tree* when it is connected and a *forest* when it is not.

A graph is called *connected* if for all two vertices  $u$  and  $v$ , there is a path from  $u$  to  $v$ . A graph is  $k$ -connected if, when we remove any set of  $k - 1$  edges, the graph is still connected. Equivalently, a graph is  $k$ -connected, if for all  $u$  and  $v$  there are  $k$  disjoint paths from  $u$  to  $v$ .

Let  $G = (V, E)$  be a graph, when  $V' \subseteq V$  and  $E' \subseteq E \cap V'^2$ , we say that  $G' = (V', E')$  is a *subgraph* of  $G$ . One says that  $G'$  *spans*  $G$  if  $V' = V$ . Let  $V' \subseteq V$ ,  $G_{V'}$  the subgraph induced by  $V'$  is the graph  $(V', \{e \in E | e \subseteq V'\})$ . The subgraph induced by  $E' \subseteq E$ , noted  $G_{E'}$ , is the graph  $(\{v \in V | \exists u \{u, v\} \in E'\}, E')$ .

A subgraph  $G'$  of  $G$  is called a *spanning tree* if it is a tree and it spans  $G$ . A *Hamiltonian path* is a path which spans  $G$ . Let  $E' \subseteq E$  be a set of edges of  $G$  such that no two edges share a vertex (are incident),  $G_{E'}$  is a *matching* of  $G$ . If  $G_{E'}$  spans  $G$  then it is a *perfect matching*.

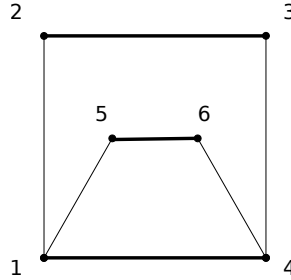


Figure 1.1: A graph on 6 vertices, with a perfect matching in bold

A graph is said to be a *clique*, if there is an edge between every vertex. If it has no edge at all, the graph is *independent*. A graph  $G = (V, E)$  is *bipartite* if we can find a partition of  $V$  into  $V_1$  and  $V_2$  such that  $G_{V_1}$  and  $G_{V_2}$  are independent.

A graph is *planar* if it can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. A graph  $H$  is a *minor* of another graph  $G$  if a graph isomorphic to  $H$  can be obtained from  $G$  by contracting some edges (merging of the two vertices of the edge), deleting some edges, and deleting some isolated vertices. A graph is planar if it does not contain the complete graph on 5 elements nor the complete bipartite graph on 6 elements as a minor (Kuratowski's theorem, see [Kur30]).

**Graph representations** We label the vertices of a graph  $G = (V, E)$  by the integers from 1 to  $|V|$ . The graph  $G$  is represented by its *adjacency matrix* defined by  $A_{i,j} = 1$  if  $\{i, j\} \in E$ , otherwise  $A_{i,j} = 0$ .

Assuming we have also indexed the edges by integers from 1 to  $|E|$ , we may also represent a graph by its *incidence matrix*, defined by  $I_{i,j} = 1$  if the vertex  $i$  is in the edge  $j$ , otherwise  $I_{i,j} = 0$ .

### 1.1.2 Hypergraphs

A hypergraph  $H = (V, E)$  is a pair where  $V$  is a finite set and  $E \subseteq \mathcal{P}(V)$ . We usually forbid the empty set or the loops to be in  $E$ . A hypergraph is said to be *k-uniform* if every element of  $E$  is of cardinal less or equal to  $k$ . A 2-uniform hypergraph is thus a graph.

There are several ways to generalize the notion of tree by defining different notions of acyclicity, they are all defined and extensively studied in [Dur09]. The most simple is called *Berge-acyclicity*. A *Berge-cycle* is a sequence of edges and vertices  $x_1, E_1, x_2, E_2, \dots, x_l, E_l, x_{l+1}$  such that for  $l \geq 2$ ,  $x_1 = x_{l+1}$  and for all  $i$ ,  $x_{i+1} \in E_i \cap E_{i+1}$ . A hypergraph without Berge cycle is called a (Berge) hypertree.

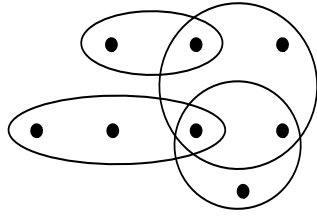


Figure 1.2: A 4-uniform hypergraph on 8 vertices

Let  $H = (V, E)$  be a hypergraph and let  $W \subseteq V$ . The *subhypergraph*  $H_W$  induced by  $W$  is defined as  $(W, \{e \cap W | e \in E\})$ . The *section hypergraph*  $H \times W$  induced by  $W$  is defined as  $(W, \{e \subseteq W | e \in E\})$ . Let  $F \subseteq E$ , the *subhypergraph*  $H_F$  induced by  $F$  is defined as  $(V, F)$ .

We say that the subhypergraph  $H$  spans  $G$  if  $V(H) = V(G)$ . Let  $H = (V, E)$  be a hypergraph, we say that  $S \subseteq V$  is a *hitting set* of  $H$  if it intersects every element of  $E$ . In other words, it is a vertex cover of the hypergraph, and the minimal hitting sets for inclusion, called *transversal*, have been extensively studied [EG95, FK96, EG].

### 1.1.3 Matroids

Matroids have been designed to abstract the notion of dependence that appears, for example, in graph theory or in linear algebra. All needed informations about matroids (and the proofs of what is stated in this section) can be found in the book Matroid Theory by J. Oxley [Oxl92].

**Definition 1.1.** A matroid is a pair  $(S, \mathcal{I})$  where  $S$  is a finite set, called the ground set, and  $\mathcal{I} \subseteq \mathcal{P}(S)$ . Elements of  $\mathcal{I}$  are said to be independent sets, the others are dependent sets. A matroid must satisfy the following axioms:

1.  $\emptyset \in \mathcal{I}$
2. If  $I \in \mathcal{I}$  and  $I' \subseteq I$ , then  $I' \in \mathcal{I}$
3. If  $I_1$  and  $I_2$  are in  $\mathcal{I}$  and  $|I_1| < |I_2|$ , then there is an element  $e$  of  $I_2 - I_1$  such that  $I_1 \cup e \in \mathcal{I}$ .

The matroids, like the graphs, may have *loops*, which are dependent singletons, but in all this thesis we assume that the matroids are loop free.

**Definition 1.2.** In a matroid, a *base* is a maximal independent set for inclusion. A *circuits* is a minimal dependent set for inclusion.

For each matroid  $M$ , we can define the dual matroid  $M^*$  by taking the same underlying set and calling a set a basis in  $M^*$  if and only if its complement is a basis in  $M$ . It is not difficult to verify that  $M^*$  is a matroid and that the dual of  $M^*$  is  $M$ .

We introduce two operations on the matroids, in order to define the notion of a minor of a matroid. Let  $M$  be a matroid,  $S$  a subset of its elements, the restriction of  $M$  to  $S$ , written  $M|S$  is the matroid  $(S, \mathcal{I})$ , such that a set is in  $\mathcal{I}$  if it is independent in  $M$  and contained in  $S$ . The contraction of  $M$  by  $S$  is the matroid  $(M^*|S)^*$ . One says that  $N$  is a minor of  $M$  if we can obtain  $N$  from  $M$  by a sequence of restrictions and contractions.

**Vector Matroid** Let  $A$  be a matrix, the *vector* matroid of  $A$  has for ground set the columns of  $A$  and a set of column vectors is independent if they are linearly independent.

**Definition 1.3.** A matroid  $M$  is representable over the field  $\mathbb{F}$  if it is isomorphic to a vector matroid of a matrix  $A$  with coefficients in  $\mathbb{F}$ . We also say that  $M$  is represented by  $A$  and that  $M$  is a  $\mathbb{F}$ -matroid.

The notion of representable matroid is central to Chapter 6. Note that there are matrices which are not similar<sup>1</sup> but represent the same matroid.

**Example 1.4.**

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The matrix  $A$  is defined over  $\mathbb{F}_2$ . The convention is to name a column vector by its position in the matrix. Here the set  $\{1, 2, 4\}$  is independent while  $\{1, 2, 3\}$  is dependent.

One can prove that the dual of a matroid representable over  $\mathbb{F}$  is also representable over  $\mathbb{F}$ . The matroids representable over  $\mathbb{F}_2$  are called *binary* matroids and those which are representable over any field are called *regular* matroids.

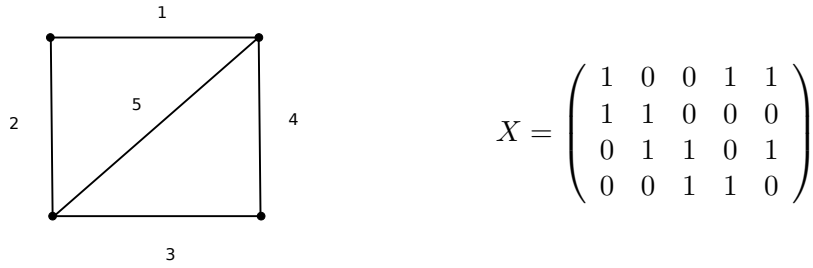
**Cycle Matroid** The second example is the *cycle* matroid; such matroids are said to be *graphic*. Let  $G$  be a graph, the ground set of its cycle matroid is the set of its edges. A set is said to be dependent if it contains a cycle. Here a base is a spanning tree if the graph is connected and a circuit is a cycle.

**Example 1.5.** In Fig. 1.3, the set  $\{1, 2, 4\}$  is independent whereas  $\{1, 2, 3, 4\}$  and  $\{1, 2, 5\}$  are dependent.

**Remark 1.6.** Any cycle matroid of a graph  $G$  is a binary matroid. To see this, one chooses an order on the edges and on the vertices of  $G$  then build the incidence matrix of  $G$  over the field  $\mathbb{F}_2$ . The dependence relation is the same over the edges and over the vectors representing the edges.

<sup>1</sup>a matrix  $A$  is similar to  $B$  if there is an invertible matrix  $S$  such that  $A = SBS^{-1}$



Figure 1.3: A graph and its representation by a matrix over  $\mathbb{F}_2$ 

**Other representations** The rank function on matroids, similar to the rank function on vector spaces, plays an important role. It is defined by:

$$\text{rank}(B) = \max\{|A| \mid A \subseteq B \text{ and } A \text{ independent}\}$$

This function is monotonic, that is for all subsets  $X$  and  $Y$ ,  $X \subseteq Y \Rightarrow r(X) \leq r(Y)$ . It is also submodular:

$$\text{rank}(A \cup B) + \text{rank}(A \cap B) \leq \text{rank}(A) + \text{rank}(B)$$

and it even leads to a characterization of matroids:

**Proposition 1.7.** *Let  $E$  be a finite set and  $r$  a function from  $2^E$  to  $\mathbb{Z}$ . The function  $r$  is the rank of a matroid if and only if it is submodular, monotonic and such that the rank of any element is 0 or 1.*

This function allows to define the closure of a set, which is the set of all elements which may be added to the set without increasing the rank. This construction shows how close to vector spaces matroids are, since we define a notion similar to the subspace generated by a set of elements.

We may also define matroids by the set of bases, by the dependent sets or by the circuits. Let  $E$  be a finite set and let  $\mathcal{C} \subseteq \mathcal{P}(E)$ , it is the collection of the circuits of a matroid with ground set  $E$  if and only if:

1.  $\emptyset \notin \mathcal{C}$
2.  $C_1, C_2 \in \mathcal{C}^2$  if  $C_1 \subseteq C_2$  then  $C_1 = C_2$
3.  $C_1, C_2 \in \mathcal{C}^2$ , if  $C_1 \neq C_2$  and  $e \in C_1 \cap C_2$  then  $\exists C \in \mathcal{C}, C \subseteq C_1 \cup C_2 \setminus \{e\}$

#### 1.1.4 Oriented matroids

Here we briefly present an extension of the matroids, namely the oriented matroids. Oriented matroids have been introduced as an abstraction of cycles of oriented graphs, like the matroids abstract the cycles spaces of undirected graphs. They are used in combinatorial geometry and optimization (arrangement of hyperplanes, linear programming).

**Definition 1.8** (Signed part). Let  $E$  be a finite set, a signed part  $C = (C^+, C^-)$  of  $E$  is a pair of two disjoint sets of  $E$ . We note  $-C$  the signed part  $(C^-, C^+)$  and  $\underline{C}$  the set  $C^+ \cup C^-$ .

We now introduce oriented matroids, which are defined by axioms very similar to those of matroids, but represent more geometric objects. We give an axiomatization of the circuits of an oriented matroid, since it is simpler.

**Definition 1.9.** Let  $E$  be a finite set and let  $\mathcal{C}$  be a set of signed parts of  $E$ . The set  $\mathcal{C}$  is the set of oriented circuits of an oriented matroid if it satisfies the following axioms:

- $\emptyset$  is not in  $\mathcal{C}$
- if  $X$  is in  $\mathcal{C}$ , then so is  $-X$
- for all  $X, Y$  in  $\mathcal{C}$ , such that  $\underline{X} \subseteq \underline{Y}$ ,  $X = Y$  or  $X = -Y$
- let  $X, Y$  be two distinct circuits and  $e \in X^+ \cap Y^-$ , then there is a  $Z \in \mathcal{C}$  such that  $Z^+ \subseteq X^+ \cup Y^+ \setminus \{e\}$  and  $Z^- \subseteq X^- \cup Y^- \setminus \{e\}$

**Oriented matroids from directed graphs** The most simple example of an oriented matroid is, like in the unoriented case, a graph and its cycles. Let  $G$  be a directed graph, the ground set of the matroid  $M_G$  is the set of its edges  $E$ . The signed part  $(C^+, C^-)$  is a circuit of  $M_G$  if  $\underline{C}$  is an undirected cycle of  $G$  and for one of its orientations  $C^+$  is the set of edges which are directed in the same way as this orientation and  $C^-$  is the set of edges in the other direction.

## 1.2 Complexity

In computability, one of the fundamental question concerns the decidability of a language. Put in an other way, is the characteristic function of the language recursive or equivalently computable by a Turing/RAM machine ? The complexity theory proposes a refinement to this question: once we know that a language  $L$  is decidable, what is the computational cost to decide wether  $x \in L$ , that is what is the running time of a Turing/RAM machine computing its characteristic function?

This section is dedicated to the formalization of computational complexity, which is the main subject of this thesis. For a good and general treatment of the subject, there are numerous reference books, two of those are [PP94, AB09].

### 1.2.1 Problems

Let  $\Sigma$  be a finite alphabet, it can be of any size larger than one, and often we use  $\{0, 1, \#\}$ . If  $x$  and  $y$  are  $\{0, 1\}$  words,  $x\#y$  denotes the pair  $(x, y)$ . Let  $x \in \{0, 1\}^*$ , the *Hamming weight* of  $x$ , denoted by  $w(x)$  is the number of 1 in  $x$ . We write  $\Sigma^n$  for the set of words of size  $n$  and  $\Sigma^{>n}$  for the set of words strictly greater than  $n$ . We denote by  $|x|$  the length of the word  $x$ .

**Definition 1.10.** A *polynomially balanced* binary predicate  $A$ , is a subset of  $\Sigma^* \times \Sigma^*$ , such that if  $A(x, y)$  then  $|y| < Q(|x|)$ , for  $Q$  a polynomial. Argument  $x$  is called an instance of  $A$ , and  $y$  is called a solution (or a witness) if  $A(x, y)$  holds. The finite set  $\{y|A(x, y)\}$  is denoted by  $A(x)$ .

From a polynomially balanced predicate, we define four different problems:

1. **Decision Problem.** The decision problem associated to  $A$ , denoted by  $\exists.A$ , is the function which to  $x \in \Sigma^*$  associates 0 if  $A(x) = \emptyset$ , 1 otherwise. Sometimes we identify the function with the language  $L = \{x|A(x) \neq \emptyset\}$ .
2. **Search Problem.** A search problem associated to  $A$ , is any function which to  $x \in \Sigma^*$  associates  $y \in A(x)$  if  $A(x) \neq \emptyset$ , a reserved element of the alphabet otherwise, that we denote by “None”.
3. **Counting Problem.** The counting problem associated to  $A$ , denoted by  $\sharp.A$ , is the function which to  $x \in \Sigma^*$  associates the cardinal of  $A(x)$ .
4. **Enumeration Problem.** The enumeration problem associated to  $A$ , denoted by  $\text{ENUM}.A$ , is the function which to  $x \in \Sigma^*$  associates the set  $A(x)$ .

An algorithm solves a problem, if it computes the function defined by the problem. The complexity of a problem is a measure of the “ressources“ used by the algorithms which solve it, and we now present a model of computation to formalize this.

### 1.2.2 Model of computation

In this thesis we consider the RAM machine as computation model and we follow the definition of [Bag]. More details may be found herein and in the series of articles about this model [Gra96, GS02, GO04].

**Definition 1.11** (RAM machine). A RAM machine is:

- an infinite sequence of input registers  $I(0), I(1), \dots$ , of computation registers  $R(0), R(1), \dots$  and two special registers  $A$  and  $B$  often called accumulators. The input registers contain the input and the others are initialized to the integer 0.
- a finite sequence of indexed instructions (the program) of the following type:
  1.  $A \leftarrow I(A)$
  2.  $A \leftarrow R(A)$
  3.  $B \leftarrow A$
  4.  $R(A) \leftarrow B$
  5.  $A \leftarrow A + B$

6.  $A \leftarrow A - B$
7. IF  $A > 0$  GOTO( $i$ )
8. STOP

The semantic of this machine is simple, it executes sequentially its program, except when it encounters a IF  $A > 0$  GOTO( $i$ ), which makes it jump to the  $i^{\text{th}}$  instruction if the content of  $A$  is more than 0. The arrow  $\leftarrow$  denotes the affectation of the value on its right to the register on its left. The machine stops when it encounters a STOP instruction. We assume that the machine stops on every input.

An input of the machine is a word  $x = x_1x_2 \dots x_n$ , with  $x_i \in \{0, 1\}$  for all  $i \leq n$ . The size  $n$  is stored in the register  $I(0)$ , and for all  $1 \leq i \leq n$ ,  $I(i) = x_i$ . The output of the machine is the binary word  $w = w_1w_2 \dots w_k$  where  $w_i$  is the binary representation of the integer in  $R(i)$  when the machine encounters the STOP instruction and  $k$  is the last index such that  $R(k)$  is non zero.

There are several different models of RAM machines and of ways to define their complexity (see [vEB91]), but they are essentially equivalent for the problems we are dealing with. To study classes of low complexity, such as linear time, one has to be particularly careful with the way one represents the input, or the cost of an arithmetic operation, but it is not relevant in this thesis and we adopt the uniform measure for its simplicity.

In this model, every instruction is counted as one step, regardless of the size of the values operated on. The time spent by a machine  $M$  on the input  $x$  is the number of steps before stopping, we denote it by  $T(M, x)$ . We want to relate the time to the size of  $x$  rather than to  $x$ : we say that a machine is of time complexity  $f(n)$  if  $\max_{|x|=n} T(M, x) = O(f(n))$ .

The other classical measure is the space used by  $M$  on the instance  $x$ . It is defined as the maximum of the accessed registers indices and of the integers stored during the computation of  $M$  on  $x$ .

A RAM machine computes a function from  $\{0, 1\}^*$  to  $\{0, 1\}^*$ . Hence it can compute a decision, a search or a counting problem, by encoding the relevant alphabet in  $\{0, 1\}^*$ . For the enumeration problems, we change the model, since we do not want to compute a word which represents the set of solutions and stop, but rather to output the solutions one after the other.

### 1.2.3 RAM machine for enumeration

To the previously defined RAM machine, we add an instruction OUTPUT. The output of such a RAM machine is the sequence of words  $(w_1, \dots, w_n)$ , where  $w_i$  is the binary encoding of the integer in  $A$  when the the  $i^{\text{th}}$  OUTPUT instruction is executed.

Let  $M$  be such a machine and  $x$  a word, we write  $M(x)$  the result of the computation of  $M$  on  $x$ . Sometimes we will use  $M(x)$  to denote the set of outputs although it is a sequence. We note  $|M(x)|$  the size of the output sequence, which

is equal to the number of OUTPUT instructions executed. Given a RAM machine  $M$  and an input  $x$ , we note  $T(M, x, i)$  the number of instructions executed before the  $i^{\text{th}}$  OUTPUT. This function is defined for  $1 \leq i \leq |M(x)|$  and we extend it at  $i = 0$  by 0 and at  $i = |M(x)| + 1$  by the number of instructions before  $M$  stops.

---

**Discussion:** The RAM machine model is usually chosen over the Turing Machine model, when one wants to define low complexity classes and also because it is closer to a real computer. Most of the time, complexity theorists do not care about the model because a Turing machine may be simulated by a RAM machine with only a polynomial overhead and conversely. But this simulation is not uniform, in the sense that if we isolate a part of a computation done by a RAM machine, the Turing Machine may need an exponentially bigger time to achieve the same task. More specifically, with a RAM machine, we may deal with an exponential size structure such as a binary search tree in polynomial time, whereas it is not possible with a Turing machine. This is useful for enumeration problems, and has been implicitly used to enumerate the maximal independent sets of a graph in lexicographic order in [JPY88].

---

We study two complexity measures, the first one is the same as for the classical RAM machine:

**Definition 1.12.** Let  $M$  be a RAM machine, and let  $x$  be a word. The *total time* taken by  $M$  on  $x$  is  $T(M, x, |M(x)| + 1)$ . We say that the machine  $M$  has a total time  $f(n)$  if  $\max_{|x|=n} T(M, x, |M(x)| + 1) = O(f(n))$ .

The next measure is specific to enumeration and its study is the main objective of enumeration complexity.

**Definition 1.13.** Let  $M$  be a RAM machine, let  $x$  be a word and  $i$  be an integer. The *delay* between the  $i^{\text{th}}$  and the  $(i + 1)^{\text{th}}$  OUTPUT is the quantity  $T(M, x, i + 1) - T(M, x, i)$ , when it is defined. We say that the machine  $M$  has a delay  $f(n)$  if  $\max_{|x|=n} \max_{i \leq |M(x)|} T(M, x, i + 1) - T(M, x, i) = O(f(n))$ .

#### 1.2.4 Other complexity measures

In some settings, it can be useful to use another computation model and therefore a different complexity measure. One convenient model is to define an object by some of its properties that we can test in constant time. It is often called a black box or an oracle model. We must adapt the RAM machine to the case of a computation with an oracle.

**Definition 1.14.** A *RAM machine with oracle* has a sequence of new registers  $O(0), O(1), \dots$  and a new instruction ORACLE. The semantic of the machine depends on the oracle, which is a function  $f$  from  $\{0, 1\}^*$  to  $\{0, 1\}^*$ . We call  $w$  the word which is encoded in the register  $O(0), O(1), \dots$ . When the instruction ORACLE is executed, the machine writes in the register  $A$  the value of  $f(w)$ .

The complexity measures of this particular model, in addition to the total time and delay are the number of calls to the oracle and the size of the words which are given as arguments to the oracle. One usually uses this model to work with multivariate polynomials since it helps give lower bounds and it abstracts away the operation of evaluating the polynomial. In Chapter 3, most of the results are obtained for this black box model with  $f$  a multivariate polynomial.

When one devises an algorithm to solve a problem whose instances are matroids, the representation of the matroids matters. One may ask a matroid to be represented by the collection of its independent sets (or circuits, bases, ...). This collection can be exponential in the number of elements of the matroid. Most complexity problems become trivial in this setting, and the usual way to address this is to assume we have a black box deciding in constant time if a set is independent or not, see [KBE<sup>+</sup>05]. We may also consider subclasses of matroids, for which we do not need the explicit set of independent sets, because we can decide if a set is independent or not in polynomial time. The cycle and vector matroids have this convenient property, which allows to turn a black box algorithm into a classical one. Two additional classes of matroids, with efficient decision of independence are given in Chapter 6.

### 1.2.5 Complexity classes: a short zoology

Since complexity theorists are tidy people, they like to put problems in big boxes with a nice label on it, and when there are none available, they create one with an exotic name. Thoses boxes or "classes" are sets of problems, which can be solved by machines whose capacities and running time are restricted.

For instance, we say that a language  $L$  (or a decision problem) is in the class P, if there is an integer  $k$  and a machine which computes 1 on the input  $x$  when  $x \in L$  and 0 otherwise in a time  $O(|x|^k)$  for all inputs  $x$ . In the same vein, we call FP the class of functions computable by a RAM machine in polynomial time.

**Reductions** A fundamental tool to study complexity classes is the notion of reduction, which relates the complexity of one problem to another. The idea is that if a problem  $A$  is reducible to  $B$ , then an algorithm which solves  $B$  can be used to solve  $A$ .

**Definition 1.15** (Polynomial-time many-one reduction). Let  $L_1$  and  $L_2$  be two languages, we say that  $L_1$  is many-one reducible to  $L_2$ , if there is a polynomial time computable function  $f$  such that  $x \in L_1$  iff  $f(x) \in L_2$ .

To study low complexity classes, we may change the condition that  $f$  is polynomial time computable into  $f$  is log-space computable. The polynomial-time many-one reduction is only useful to compare two decision problems. On the other hand, the next reduction is adapted to counting and enumeration problems.

**Definition 1.16** (Parsimonious reduction). We say that the predicate  $A$  reduces parsimoniously to the predicate  $B$  if there are two polynomial time computable functions  $f$  and  $g$  such that  $A(x, y)$  if and only if  $B(f(x), g(x, y))$  and  $g(x, \cdot)$  is a bijection from  $A(x)$  to  $B(x)$ .

For each reduction and class of problems we define the notion of *hardness*. We say that a problem  $A$  is hard for a class  $\mathcal{C}$ , if all problems in  $\mathcal{C}$  reduce to  $A$ . If  $A$  is also in  $\mathcal{C}$ , we say that  $A$  is complete for  $\mathcal{C}$ . One says that a class  $\mathcal{C}$  is stable under reduction, if any problem reducible to a problem in  $\mathcal{C}$  is also in  $\mathcal{C}$ .

**The class NP and the polynomial hierarchy** One of the highest achievements of the complexity theory is the introduction of the class of NP problems and all the work done to understand them. We present them without introducing the non deterministic machines, since the “logical” characterization, a.k.a. witness checking, is more adapted to our exposition. The most well-known question of the complexity theory is whether  $P \neq NP$ . This is widely believed to be true, but has still resisted any attempt to prove it. We relate several questions to this one in Section 2.3.

**Definition 1.17.**

1. The class NP is the set of decision problems associated to predicates decidable in polynomial time.
2. The class #P is the set of counting problems associated to predicates decidable in polynomial time.

There is an analog to NP for enumeration problems, we present it and other complexity classes for enumeration with much details in Chapters 2 and 3. We have seen how to define four problems from a predicate (decision, search, counting and enumeration), the next example shows how their complexity may be different.

**Example 1.18.** Consider the predicate  $\text{MATCHING}(M, G)$  which is true when  $M$  is a perfect matching of the graph  $G$ . The problem  $\exists.\text{MATCHING}$  is solvable in polynomial time thanks to Edmond’s algorithm [Edm65] which is an improvement of the simple algorithm which works for bipartite graphs. It solves in fact the search problem, since it builds a matching by the augmenting path technique.

On the other hand, the problem  $\sharp.\text{MATCHING}$  is #P-complete, even for bipartite graphs [Val79], this result is proved by a sequence of Turing reductions from  $\sharp.\text{SAT}$ .

Finally, there is a very efficient algorithm to enumerate perfect matching in bipartite graphs with a total time linear in the number of matchings [Uno97].

The most celebrated problem in NP is SAT. It is complete for the class NP [Coo71] under parsimonious reduction. It is defined by:

SAT

*Input:* a propositional formula  $\phi$  in conjunctive normal form

*Output:* accept if there is a truth value assignment of the variables such that  $\phi$  is satisfied

The following restrictions of propositional formulas give rise to satisfaction problems of different complexities.

- A clause is ternary if it is the conjunction of three literals. The problem of the satisfaction of a conjunction of ternary clauses is denoted by 3CNF-SAT or 3CNF and is NP-complete.
- A clause is binary if it is the conjunction of two literals. The problem of the satisfaction of a conjunction of binary clauses is denoted by 2CNF-SAT or 2CNF and is in P.
- A clause is Horn if it contains at most one positive literal. The problem of the satisfaction of a conjunction of Horn clauses is denoted by HORN-SAT and is P-complete for log-space reductions.
- An affine clause is an equation over the variables seen as elements of  $\mathbb{F}_2$ . For instance,  $x_1 \oplus \neg x_2 \oplus x_3 = 1$  is an affine clause. The problem of the satisfaction of a conjunction of affine clauses is denoted by AFFINE-SAT and is in P.

Let  $y_1$  and  $y_2$  be assignments of the variables of a formula. If we want to compare them, one can say that  $y_1 < y_2$  if all the variables set to true in  $y_1$  are also set to true in  $y_2$  (it is the pointwise order). A solution greater than any other for this order is said to be *maximum*.

The Hamming weight of a solution is the number of variables set to 1. We define a partial order by saying that  $y_1 < y_2$  iff  $w(y_1) < w(y_2)$ . A solution greater than any other for this order is said to be *maximal*. Remark that a maximal solution is also maximum.

Usually finding a maximal solution is harder than finding a maximum solution. For instance, one can find a maximum independent set in a graph in polynomial time, while finding a maximal one is NP-hard.

**Polynomial hierarchy** The class of the complement of languages in NP is denoted by coNP. Alternatively, from a polynomially balanced predicate  $A$ , one defines the problem  $\forall.A$ , as the set of  $x \in \Sigma^*$  such that  $Q$  is a polynomial and  $\forall y \in \Sigma^{Q(|x|)} A(x, y)$ . The class coNP is then the set of problems  $\forall.A$ , with  $A$  decidable in polynomial time.



One can further generalize the definition of NP and coNP by alternating the quantifiers in front of the predicate. A language is in  $\Sigma_k^P$  if it is the set of  $x \in \Sigma^*$  such that  $Q_1 y_1 Q_2 y_2 \dots Q_k y_k A(x, y_1 y_2 \dots y_k)$  and  $Q_1 = \exists$ . When  $Q_1 = \forall$ , it is in the class  $\Pi_k^P$ . There is a natural complete problem for  $\Sigma_k^P$ , the quantified boolean satisfaction problem with  $k$  alternations of quantifiers denoted by  $QBF_k$ . It is the problem to decide whether  $Q_1 y_1 y_2 \dots Q_k y_k f$  is true, where  $f$  is a propositional formula in the variables  $y_1, \dots, y_k$  and  $Q_1 = \exists$ .

The polynomial hierarchy, denoted by PH, is the union of all  $\Sigma_k^P$  and  $\Pi_k^P$ . It is easy to prove that if  $P = NP$  then the whole hierarchy collapses:  $P = PH$ . Toda has proved a beautiful theorem [Tod91], which states that, up to a particular reduction, PH is included in  $\#P$ .

**Other classes** Here we give some classes defined by different complexity measures and computational models. We say that a problem is in PSPACE if it is decidable by a machine which uses a space  $O(|x|^k)$  for all instances  $x$ . The problem QBF, defined like  $QBF_k$  but with no restriction on the alternation of quantifiers, is PSPACE-complete. One can also prove that  $\#P \subseteq PSPACE$ .

Since it is widely believed that NP-complete problems can only be solved in exponential time, complexity theorists have tried to extend the class of easy or “tractable” problems. One idea is to increase the power of a machine, here by giving access to randomness, and to compute problems with a good probability.

A probabilistic RAM machine has the additional instruction RAND which writes with probability one half 0 or 1 in a special register. We say that a probabilistic RAM machine computes the word  $w$  on the instance  $x$  with probability  $p$  if  $p$  is the number of runs on  $x$  for which the machine computes  $w$  divided by the total number of runs. The class we now define is considered to be the probabilistic equivalent of P.

**Definition 1.19** (BPP). A language  $L$  is decidable in **B**ounded-**e**rror **P**robabilistic **P**olynomial time, BPP if there is a probabilistic machine  $M$  always running in polynomial time such that:

- $\forall x \in L, Pr(M(x) = 1) \geq \frac{2}{3}$
- $\forall x \notin L, Pr(M(x) = 1) \leq \frac{1}{3}$

It is conjectured that  $BPP = P$  but the best result so far is the inclusion  $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$  [Sip83]. In fact, most of the problems in BPP are also in one of its subclasses, called RP, that we now introduce. The class RP is easily proved to be in NP (but still not in P).

**Definition 1.20** (RP). A language  $L$  is decidable in **R**andomized **P**olynomial time, RP if there is a probabilistic machine  $M$  always running in polynomial time such that:

- $\forall x \in L, Pr(M(x) = 1) \geq \frac{2}{3}$

- $\forall x \notin L, Pr(M(x) = 1) = 0$

Another idea to speed-up exponentially a computation is to allow parallel computation on a polynomial number of processors. The class NC is the set of decision problems decidable in polylogarithmic time on a parallel computer with a polynomial number of processors. It does not depend on the model of parallel computer and can be alternatively defined to be the decision problems decidable by a uniform family of Boolean circuits with polylogarithmic depth and a polynomial number of gates. A problem is in randomized NC, denoted by RNC, if it is decided with probability  $\frac{3}{4}$  by a family of randomized Boolean circuits with polylogarithmic depth and a polynomial number of gates. For more details on parallel computation see [GHR95].

### 1.3 Logic

We recall a few key concepts about logic, a good and thorough exposition of different logics and their connections with complexity can be found in [Lib04].

#### 1.3.1 Structure

A signature  $\sigma = \{R_1, \dots, R_k, f_1, \dots, f_l\}$  is a set of relation symbols  $R_1, \dots, R_k$ , a set of function symbols  $f_1, \dots, f_l$  and the arity of each symbol, *i.e.* the number of its arguments.

A structure is defined as a pair  $\mathcal{A} = (A, \{S^A | S \in \sigma\})$  where  $A$  is a finite set called the *domain* of  $\mathcal{A}$  and  $S^A$  is the *interpretation* of the relation or function  $S$ . If  $S$  is a relation of arity  $r$ ,  $S^A$  is a subset of  $A^r$ . If  $S$  is a function of arity  $r$ ,  $S^A$  is a function from  $A^r$  to  $A$ . We say that a structure is relational if there is no function symbol in its signature.

**Example 1.21.** Let  $\sigma = \{E\}$  where  $E$  is a relation symbol of arity two. Let  $\mathcal{A} = (D, E^A)$  a finite  $\sigma$  structure, it represents a graph  $G$ . The vertices of  $G$  are the elements of the domain  $D$  and there is an edge  $(x, y)$  in  $G$  if and only if  $E(x, y)$ . In fact, every graph is represented by such a structure and it is the canonical way to do it (but not the only one).

#### 1.3.2 First-order logic

The first order logic or *FO* is the name of a family of formulas and a way to interpret them over structures. The first order formulas are defined inductively on a signature  $\sigma$ . They are built from an infinite set of *variables*  $x, y, z, \dots$ , the negation, conjunction and disjunction symbols  $\neg, \wedge$  and  $\vee$ , the quantifier symbols  $\exists x$  and  $\forall x$  and the elements of  $\sigma$ . A *term* is either a variable or a function symbol of  $\sigma$  applied to terms. An *atomic formula* is an equality between two terms or a relation applied to terms. A *formula* is in the closure of the atomic formulas by logical connectives and quantifiers.

The set of free variables of a formula is defined as usual and we note  $\phi(x_1, \dots, x_n)$  a formula with free variables  $x_1, \dots, x_n$

A *prenex formula* is of the form  $Q_1 \dots Q_n \psi$  where  $Q_i$  is a quantifier and  $\psi$  has no quantifier. It is well known that every formula can be transformed into an equivalent prenex formula.

One defines interesting fragments of *FO*, by restricting the number of quantifiers in a formula. For instance, the set of formulas of the form  $\exists x_1, \dots, x_n \psi$  where  $\psi$  has no quantifier is denoted by  $\exists FO$ .

Finally, the semantic of the formulas, that is to say their interpretation on a model, is defined as usual, by interpreting a variable as an element of the domain, the relations and functions of the language by their interpretation. We write  $(\mathcal{A}, \bar{a}) \models \phi(\bar{x})$ , the fact that  $\phi(\bar{a})$  is true on the model  $\mathcal{A}$ .

A *property* is a class of models with the same signature. Let  $\mathcal{P}$  be a property, it is *expressible* in a logic, here *FO*, if there is a formula  $\phi \in FO$  such that:

$$M \in \mathcal{P} \Leftrightarrow M \models \phi$$

**Example 1.22.** On a structure which represents a graph, as in Example 1.21, the formula  $\phi$  is true if the graph has no triangle.

$$\phi = \forall x \forall y \forall z \neg (Exy \wedge Eyz \wedge Ezx)$$

### 1.3.3 Second-order logic

One extends the first order logic by second-order variables denoted by capital letters  $X, Y, Z, \dots$  which represent relations on the domain instead of elements. The second-order logic, written *SO*, is defined like *FO* except it also has  $X(t_1, \dots, t_k)$  for atomic formula where the  $t$ 's are terms and  $X$  is a second-order variable of arity  $k$ . We also take the closure of atomic formulas by second-order quantifiers  $\exists X$  and  $\forall X$ .

We study a restriction of *SO*, the monadic second-order logic *MSO*, where all second-order variables are of arity one and thus represent sets. One can express properties in *MSO* logic, which are not definable in *FO*.

**Example 1.23.** It can be shown that graph connectivity cannot be expressed in *FO* (see [Lib04]). On the other hand, it is expressible in *MSO* by the negation of the following formula:

$$\exists X (\exists x X(x) \wedge \exists x \neg X(x) \wedge (\forall x \forall y (X(x) \wedge \neg X(y)) \rightarrow \neg E(x, y)))$$

The notion of expressivity enables us to characterize complexity classes by a logic (implicit complexity). For instance, Fagin has proved that  $\exists SO$  captures NP [Fag74], that is the problems on finite models which are in NP are expressible in  $\exists SO$ .

### 1.3.4 The model-checking problem

The bridge between logic and complexity is the problem of *model-checking*: given a formula  $\phi$  in a logic  $\mathcal{L}$  and a structure  $\mathcal{A}$ , does  $\mathcal{A}$  satisfy  $\phi$  ?

It is a natural problem, which appears in many fields of computer science such as verification, database, constraint satisfaction ... The complexity of the model-checking has been extensively studied for various logics and type of structures. Moreover, one can study the complexity of such problems in terms of  $|\mathcal{A}|$  and  $|\phi|$  (combined complexity), but also in term of  $|\mathcal{A}|$  only ( $\phi$  is fixed: data complexity) or in term of  $|\phi|$  ( $|\mathcal{A}|$  is fixed: expression complexity).

We can characterize the complexity class NP thanks to model-checking. Let  $S = (\{0, 1\}, \emptyset)$  be the model with a 2 element domain over an empty signature. The model-checking of the logic  $\exists FO$  over  $S$  is nothing but the problem SAT. If we want to decide all  $FO$  over  $S$ , the model checking is the problem QBF.

Finally the model checking of  $MSO$  on trees has a deep relationship with tree automata and this can be used to study more complex objects such as graphs and matroids but we postpone this subject to Chapter 5.

## 1.4 Polynomials

Polynomials are certainly the most used functions in mathematics and are ubiquitous in complexity theory where they serve to bound the resources of computations, as a model of computation, or to encode solutions of classical combinatorial problems like the Tutte polynomial.

We consider polynomials with  $n$  variables and coefficients in a ring, which is most of the time a finite field,  $\mathbb{Z}$  or  $\mathbb{Q}$ . Usually, we call the variables  $X_1, \dots, X_n$  or all at once  $\vec{X}$ . A sequence of  $n$  positive integers  $\vec{e} = (e_1, \dots, e_n)$  defines the *term*  $\vec{X}^{\vec{e}} = X_1^{e_1} X_2^{e_2} \dots X_n^{e_n}$ . A *monomial* is a term multiplied by a scalar, which is its *coefficient*, and a polynomial is a sum of monomials.

The *degree* of a monomial  $\lambda \vec{X}^{\vec{e}}$  is  $\max_{i=1, \dots, n} e_i$ . The *total degree* of this monomial is  $\sum_{i=1, \dots, n} e_i$ . The (total) degree of a polynomial is the maximum of the (total) degrees of its monomials. All the examples we give here are multilinear polynomials, meaning that their degree is 1.

### 1.4.1 Representation

There are several ways of representing a polynomial. First we can store the list of coefficients of its monomials or its explicit representation in any other base. This representation does not give an efficient way of computing or even storing the polynomial. Indeed, a polynomial with  $n$  variables and degree  $d$  may have up to  $d^n$  monomials. For instance  $\sum_{I \subseteq [n]} \prod_{i \in I} X_i$  is a multilinear polynomial with  $2^n$  monomials.

We can store the factorized form of a polynomial: it is represented by a list of polynomials of which it is the product. The former polynomial may be written  $\prod_{i \in [n]} (X_i + 1)$  which is the product of only  $n$  degree one polynomials.

The generalization of this approach is to represent a polynomial by an *arithmetic circuit*. It is a directed acyclic graph, whose nodes of indegree zero are called input gates and are labeled by either a variable or a ring element. All the other nodes are of indegree two and labeled by either  $+$  or  $\times$ . An *arithmetic formula* is a circuit whose underlying graph is a directed tree.

The polynomial computed by an *arithmetic circuit* is defined inductively:

- for an input gate, it is the label
- for a node labeled  $+$ , which receives edges from two nodes computing  $f$  and  $g$ , it is  $f + g$
- for a node labeled  $\times$ , which receives edges from two nodes computing  $f$  and  $g$ , it is  $f \times g$

One defines the *formal degree* of a circuit inductively. The degree of an input gate is one, the degree of a  $+$  node is the maximum of the degrees of its inputs and the degree of a  $\times$  node is the sum of the degrees of its inputs. Notice that there is a simple algorithm to evaluate a polynomial represented by a circuit on any input. The evaluation algorithm is polynomial in the size of the circuit, the size of the input and the formal degree.

We may further abstract the notion of representation of a polynomial by saying that any algorithm which computes the same function as a polynomial represents this polynomial. Finally, we may assume that we do not know the algorithm which computes the polynomial, but that we have access to one. This model of polynomial given by a *black box* is used for the following problem:

#### POLYNOMIAL IDENTITY TESTING

*Input:* a polynomial given as a black box.

*Output:* decides if the polynomial is zero.

One is also interested in finding an explicit representation of a polynomial given by a black box, this problem is called POLYNOMIAL INTERPOLATION.

We will now give three examples of polynomials, with  $n^2$  variables noted  $(X_{i,j})_{(i,j) \in [1,n]^2}$ , which can thus be seen as applied to the  $n \times n$  matrix  $X$ . By choosing a good representation of a graph by a matrix, we can also associate those polynomials to graphs.

### 1.4.2 Examples

**Determinant** The *Determinant* has been defined in linear algebra to characterize systems of linear equations or matrices of full rank.

$$\det \vec{X} = \sum_{\sigma \in \Sigma_n} \text{sg } \sigma \prod_{i=1}^n X_{i, \pi(i)}$$

Here  $\text{sg}$  denotes the signature of a permutation, which is either 1 or  $-1$ .

To a graph  $G$ , we associate the matrix  $M_G$  such that  $M_{i,j} = X_{i,j}$  if  $(i, j)$  is an edge of  $G$  and 0 otherwise. We write  $\det(M_G)$  for the determinant of the matrix  $M_G$ , which is a projection of the full determinant in  $n^2$  variables. There is a bijection between the cycle covers of the graph  $G$  and the monomials of  $\det(M_G)$ .

We can also relate the determinant to the spanning trees of a graph  $G$ , by defining its Kirchhoff matrix  $K(G)$ : for  $i \neq j$   $K(G)_{i,j} = -X_{i,j}$  and  $K(G)_{i,i} = \sum_{(i,j) \in E(G)} X_{i,j}$ . The Matrix-Tree theorem (see [Jer03]) is the following equality where  $\mathcal{T}$  is the set of spanning trees of  $G$ :

$$\det(K(G)) = \sum_{T \in \mathcal{T}} \prod_{(i,j) \in T} X_{i,j}$$

The Determinant is easily computable, by gaussian elimination in  $O(n^3)$  algebraic operations. We can also use a dynamic programming algorithm [Rot01], which does not use divisions and needs  $O(n^4)$  algebraic operations. The most elaborate algorithms [KV05] give the evaluation of the Determinant in  $(n^{2,7})^{1+o(1)}$  algebraic operations. There are also efficient randomized algorithms to deal with the cost of the arithmetic operations. Any of these algorithms can be seen as a representation of the Determinant.

We can also represent a Determinant of  $n^2$  variables by an arithmetic circuit with divisions of size  $n^3$  or by an arithmetic formula of size  $2^{O(\log^2 n)}$  using Gaussian elimination [Ber84]. This means that the computation of the Determinant is in the class NC.

**Pfaffian** The *Pfaffian* is related to the Determinant, and may be seen as one of its generalizations. One considers the set  $\Pi$  of all permutations  $\pi$  on  $[2n]$  such that  $\pi(2t) = i_t$  and  $\pi(2t+1) = j_t$  with  $i_k < j_k$  and  $i_1 < \dots < i_{2n}$ .

$$\text{Pf}(\vec{X}) = \sum_{\pi \in \Pi} \text{sg}(\pi) X_{i_1, j_1} \dots X_{i_n, j_n}$$

Let  $G$  be a graph, we build the matrix  $A_G$  such that  $(A_G)_{i,j}$  is

- $X_{i,j}$  when  $(i, j)$  is an edge of  $G$  and  $i < j$
- $-X_{i,j}$  when  $(i, j)$  is an edge of  $G$  and  $i > j$

- 0 when  $(i, j)$  is not an edge

The relation between the Pfaffian and the Determinant is  $\text{Pf}(A_G)^2 = \det(A_G)$  [BR91]. Therefore the Pfaffian is as easy to compute as the Determinant. Each monomial of  $\text{Pf}(A_G)$  is of the form  $\epsilon(M) \prod_{(i,j) \in M} X_{i,j}$ , where  $M$  is a perfect matching of  $G$  and  $\epsilon(M)$  is 1 or  $-1$ .

Moreover one generalizes the definition of  $A_G$  to an orientation of  $G$  by saying that  $(A_{\vec{G}})_{i,j}$  is  $X_{i,j}$  if  $(i, j)$  is an edge and  $-X_{i,j}$  if  $(j, i)$  is an edge. One can prove that for every so called Pfaffian orientation of  $G$ , the coefficients  $\epsilon(M)$  in  $\text{Pf}(A_{\vec{G}})$  are all the same, that is to say the Pfaffian is the perfect matching polynomial up to a sign. Since every planar graph admits a Pfaffian orientation, we can use this fact to compute the number of perfect matchings in a planar graph in polynomial time [Kas61].

### Permanent

$$\text{Per} \vec{X} = \sum_{\sigma \in \Sigma_n} \sigma \prod_{i=1}^n X_{i, \pi(i)}$$

The *Permanent* looks like the Determinant and its monomials are also in bijection with the cycle covers of a graph. Alternatively, one may consider that its monomials are in bijection with the perfect matchings of a bipartite graph. Its evaluation is #P complete over  $\mathbb{Z}$  [Val79], since it may be used to count the number of perfect matching of a bipartite graph. However, over  $\mathbb{F}_2$ , it is equal to the determinant and thus easy to compute.





Part I

Complexity



## Chapter 2

# Enumeration

In the last 30 years, numerous results on enumeration complexity have been published but they are of different nature and this field is still somewhat fragmented.

First, a lot of enumeration algorithms of interesting combinatorial objects have been designed. For some authors, the considered complexity measure is the total time or the amortized time (total time divided by the number of solutions). For instance, we know how to enumerate efficiently with respect to this measure the perfect matchings [Uno97], the spanning trees and the maximal matchings [AF96], the linear extensions of a partial order [PR94], . . . . There are also results with an emphasis on the delay, although the resulting algorithms often seem of less practical interest: independent sets of matroids [KBE<sup>+</sup>05], maximal acyclic sub-hypergraphs [DH09], independent sets [JPY88], assignments of a 2-SAT formula [KSS00], . . .

The enumeration of the transversals of a hypergraph has also attracted a lot of attention, since this problem seems to have a lot of applications in different fields of computer science [KS93, EG95, FK96, EG].

If one introduces concepts like precomputation and local transformations, one can study algorithms with a very low delay (linear or constant). There are a number of results which state that a query –the enumeration of the satisfying assignments of a formula– can be enumerated with such a low delay. For instance, first-order queries on structures of bounded degree are computable with constant delay [DG07] and one can even compute the  $j^{\text{th}}$  solution of such queries in linear time [BDGO08]. In the same vein, *MSO* queries on terms and thus on graphs or matroids of bounded branch-width can be enumerated with linear delay ([Cou09] and Section 6.4).

Finally, there have been some attempts to classify the complexity of enumeration problems [JPY88, KSS00]. In this chapter our aim is to extend this classification by introducing new classes and to prove some simple but useful properties of these classes. We inspire ourselves from all tools, classes and concepts introduced for classical complexity and try to adapt them to this particular context.

## 2.1 Basics

We recall here the definition of an enumeration problem:

**Definition 2.1** (Enumeration Problem). Let  $A \subseteq \Sigma^* \times \Sigma^*$  be a polynomially balanced binary predicate, we write  $A(x)$  for the set of  $y$  such that  $A(x, y)$  holds. The enumeration problem  $\text{ENUM}\cdot A$  is the function which associates  $A(x)$  to  $x$ . A RAM machine solves  $\text{ENUM}\cdot A$  if, for each  $x \in \Sigma^*$ , it computes a sequence  $y_1, \dots, y_n$  such that:

1.  $\{y_1, \dots, y_n\} = A(x)$
2.  $i \neq j \Rightarrow y_i \neq y_j$ , for all  $i, j$

One might require the elements of the enumeration to be generated in a particular order. This constraint increases the complexity of the enumeration problem and is sometimes considered in the literature [ASY, JPY88, DH09].

**Definition 2.2** (Enumeration order). Let  $\Sigma$  be a finite alphabet and let  $\{\langle_x\}_{x \in \Sigma^*}$  be a sequence of total orders on  $\Sigma^*$  that we write  $\langle$ . We say that  $\langle$  is an enumeration order if there is a polynomial time algorithm, which given  $x, y_1$  and  $y_2$  decides if  $y_1 \langle_x y_2$ .

**Example 2.3.**

- The lexicographic order for enumeration, denoted by  $\langle_{lex}$ , is the set  $\{\langle_x\}_{x \in \Sigma^*}$ , where each  $\langle_x$  is the lexicographic order on  $\Sigma^*$ .
- Let  $\Sigma = \{0, 1\}$ , the enumeration order  $\langle_w$  is the set of orders  $\{\langle_x\}_{x \in \Sigma^*}$ , where  $u \langle_x v$  if  $w(u) < w(v)$  ( $w$  is the Hamming weight).

In these two examples, the enumeration order is one order instead of a family. However, the possibility to have a family of order is useful, since the order on solutions which is considered in many problems often depends on the instance.

**Definition 2.4** (Ordered Enumeration Problem). Let  $\text{ENUM}\cdot A$  be an enumeration problem and let  $\langle$  be an enumeration order. The pair  $(A, \langle)$  is an *ordered enumeration problem*, which is denoted by  $\text{ENUM}\cdot(A, \langle)$ . A RAM machine which solves  $\text{ENUM}\cdot A$  also solves  $\text{ENUM}\cdot(A, \langle)$ , if for every  $x$  it generates a sequence  $y_1, \dots, y_n$  sorted in increasing order according to  $\langle_x$ .

---

**Discussion:** We could unify ordered and unordered enumeration problems by defining partially ordered enumeration problems, where the order  $\langle_x$  is partial and may then be empty or total. But a lot of properties or their demonstrations are different in the ordered and unordered case. Therefore a lot of results would be stated for the two special cases and the unification would be artificial. Moreover, the case of a partial order, which is neither total or empty, is sometimes very difficult to deal with, when it comes to classification and reduction of enumeration problems.

---

We define a complexity class similar to NP or #P for enumeration, by a restriction of the predicates the problems are defined from.

**Definition 2.5.** The problem  $\text{ENUM}\cdot A$  (respectively an ordered enumeration problem  $\text{ENUM}\cdot(A, <)$ ) is in the class **EnumP** (resp. **EnumP<sup>o</sup>**) if the predicate  $A(x, y)$  is decidable in polynomial time.

Note that the order plays no role in the definition of the class, then if  $\text{ENUM}\cdot A \in \mathbf{EnumP}$  then for all enumeration orders  $<$ ,  $\text{ENUM}\cdot(A, <)$  is in **EnumP<sup>o</sup>**. We recall now the notion of parsimonious reduction, since it is adapted to enumeration problems and to this class especially.

**Definition 2.6** (Parsimonious reduction). Let  $\text{ENUM}\cdot A$  and  $\text{ENUM}\cdot B$  be two enumeration problems (resp.  $\text{ENUM}\cdot(A, <_1)$  and  $\text{ENUM}\cdot(B, <_2)$  two ordered enumeration problem). A parsimonious reduction from  $\text{ENUM}\cdot A$  to  $\text{ENUM}\cdot B$  (resp. from  $\text{ENUM}\cdot(A, <_1)$  to  $\text{ENUM}\cdot(B, <_2)$ ) is a pair  $(f, g)$  of polynomial time computable functions such that  $A(x, y)$  if and only if  $B(f(x), g(x, y))$  and  $g(x, \cdot)$  is a bijection (resp. an increasing bijection) from  $A(x)$  to  $B(x)$ .

For obvious reason, the class **EnumP** is closed under parsimonious reductions. Moreover we know that every NP problem is parsimoniously reducible to the problem SAT. Therefore the problem  $\text{ENUM}\cdot\text{SAT}$  is **EnumP**-complete.

**Remark 2.7.** We would like to find a complete enumeration problem, whose decision is not hard. This question is inspired by the fact that counting the matchings of a graph is #P-complete, whereas deciding if there is one is in P. To answer this question we have to propose new reductions adapted to enumeration problems.

We give here an artificial problem which is not complete for parsimonious reductions, but which has an efficient enumeration algorithm if and only if  $\text{ENUM}\cdot\text{SAT}$  has one. Let  $A$  be the predicate such that  $A(x, 0)$  is true and  $A(x, 1y)$  is true if  $\text{SAT}(x, y)$ . This problem cannot be complete for parsimonious reductions, because the decision problem is trivial (always true). However  $\text{ENUM}\cdot A$  can be seen as “complete” since we can simulate any other enumeration problem only by removing the 0 solution. It suggests a kind of subtractive reduction, but we will see, in Section 2.5, that it does not enjoy as good properties as for counting problems [DHK05].

## 2.2 Complexity measures and classes

One of the most studied problems in enumeration is the following:

$\text{ENUM}\cdot\text{MAXIS}$

*Input:* a graph

*Output:* all the maximal (for the inclusion) independent sets of the graph

In an article by Johnson et al. [JPY88] about  $\text{ENUM}\cdot\text{MAXIS}$ , the notions of *polynomial total time*, *incremental polynomial time* and *polynomial delay* are introduced. They correspond to the first three classes of the five we define in this section.

In this thesis, all considered enumeration problems will be in the class **EnumP**, i.e. the predicates which define them are all decidable in polynomial time.

---

**Discussion:** The separation of the classes defined in this section are made harder (but more interesting) by this hypothesis. In fact, the time hierarchy theorem is enough to separate unconditionally all classes if we do not ask the problems to be in **EnumP**.

This hypothesis is not too strong since most studied enumeration problems, if not all, satisfy it. Furthermore the problems in **EnumP** enjoy better properties, for instance stability by union (Section 2.5) and the randomized classes we define in Chapter 3 are more robust when intersected with **EnumP**.

---

### 2.2.1 Polynomial total time

Let  $M$  be a RAM machine, recall that  $T(M, x, i)$  is the number of steps before the  $i^{\text{th}}$  instruction OUTPUT. From now on, the machine  $M$  will be clear from the context and we will write  $T(x, i)$  instead of  $T(M, x, i)$ .

**Definition 2.8.** A problem  $\text{ENUM}\cdot A$  (resp.  $\text{ENUM}\cdot(A, <)$ ) is computable in polynomial total time, written **TotalP** (resp. **TotalP<sup>o</sup>**) if there is a polynomial  $Q(x, y)$  and a machine  $M$  which solves  $\text{ENUM}\cdot A$  (resp.  $\text{ENUM}\cdot(A, <)$ ) and satisfies for all  $x$ ,  $T(x, |M(x)| + 1) < Q(|x|, |M(x)|)$ .

**Remark 2.9.** It has been remarked in [JPY88] that to study **TotalP<sup>o</sup>**, the order is not relevant. Indeed, one can always generate all solutions in any order, sort them and then output them in the desired order. This means that when  $\text{ENUM}\cdot A \in \text{TotalP}$ , for all enumeration orders  $<$ ,  $(\text{ENUM}\cdot A, <) \in \text{TotalP<sup>o</sup>}$ . However, this method requires to store all solutions before sorting them. Therefore this algorithm can use a space exponential in the input, while an algorithm which does not respect the order could be better for this measure.

One can imagine a “better“ way of dealing with the order, which uses less space but slightly more time. Let  $(\text{ENUM}\cdot A, <)$  be an ordered enumeration problem and let  $M$  be a machine which solves in polynomial total time the problem  $\text{ENUM}\cdot A$ . We want to find the minimal element of  $A(x)$  larger than a given  $y$  for the order  $<_x$ . To do this, one only has to enumerate the whole set  $A(x)$  thanks to  $M$  and use an auxiliary variable to store the smallest element larger than  $y$ .

Then one recursively calls this algorithm on  $x$  and  $y$  set to the last generated solution. It stops when there are no solutions larger than  $y$ . This algorithm outputs the solution of  $A(x)$  in the order  $<_x$  in a time which is the time of a run of  $M$  on  $x$  multiplied by  $|A(x)|$ . On the other hand, one only uses two additional variables in addition to the space  $M$  uses. Hence, if  $M$  works in polynomial space and polynomial total time, this algorithm has the same space and time complexity.

**Example 2.10.**

ENUM·TRANSVERSAL

*Input:* a hypergraph

*Output:* all the transversals of the hypergraph

One can decide if a set of elements is a transversal in polynomial time, therefore ENUM·TRANSVERSAL is in **EnumP**. It is known that one can enumerate the transversals of a hypergraph in subexponential total time, that is to say a total time  $n^{o(\log n)}$  where  $n$  is a polynomial in the size of the input and the number of solutions [FK96]. Several restrictions to this problem have also been proved to be in **TotalP**, for instance if the hypergraphs are all  $k$ -uniform. The question of proving whether the general ENUM·TRANSVERSAL problem is in **TotalP** has consequences in boolean logic, database theory and artificial intelligence [EG95].

From any problem ENUM· $A$  we define the following characteristic decision problem:

ALLSOLUTION $_A$

*Input:* an instance  $x$  and a set  $S$  included in  $A(x)$

*Output:* accept if  $A(x) \setminus S = \emptyset$ , else reject

We can relate the complexity of this decision problem to the complexity of ENUM· $A$  as in [KSS00].

**Lemma 2.11.** *If* ENUM· $A \in$  **TotalP** *then* ALLSOLUTION $_A \in$  P.

*Proof.* Let  $M$  be a machine which solves ENUM· $A$  in time  $Q(|x|, |A(x)|)$  where  $Q$  is a polynomial. Let  $(x, S)$  be an instance of ALLSOLUTION $_A$ . One runs  $M$  for a time  $Q(|x|, |S|)$ . If  $M$  ends and outputs exactly the set of solutions  $S$  then accept, else reject. This algorithm is polynomial in  $|x|, |S|$  and it solves ALLSOLUTION $_A$ , therefore ALLSOLUTION $_A \in$  P.  $\square$

This problem is the first example of the relations between decision and enumeration, which helps us to transfer classical decision results in enumeration. In fact this lemma is used in [KSS00] to prove that the problem of enumerating all maximal models of a Horn Formula is not in **TotalP** if  $P \neq NP$ . The converse of this lemma is not known (nor believed) to be true.

## 2.2.2 Incremental polynomial time

In the next subsections, we define classes which deal with the dynamic of the enumeration and are thus at the heart of enumeration complexity.

**Definition 2.12.** A problem  $\text{ENUM}\cdot A$  (resp.  $\text{ENUM}\cdot(A, <)$ ) is computable in incremental polynomial time, written **IncP** (resp. **IncP<sup>o</sup>**) if there is a polynomial  $Q(x, y)$  and a machine  $M$  which solves  $\text{ENUM}\cdot A$  (resp.  $\text{ENUM}\cdot(A, <)$ ) such that for all  $x$  and  $i$ ,  $T(x, i + 1) - T(x, i) \leq Q(|x|, i)$ .

This class corresponds to the problems whose solutions are easy to find at the beginning but for which it becomes harder and harder to find new ones. Most of the algorithms which are known so far for problems of this class work by increasing the set of solutions by means of a polynomial time rule, until saturation.

**Example 2.13.** The problem of enumerating the circuits of a matroid has been proved to be in **IncP** [KBE<sup>+</sup>05]. The proof is given for a model, where the matroid is given as an oracle. As soon as we can compute in polynomial time if a set is independent or not, like for representable matroids given by a matrix, the result holds. In fact, the harder problem of enumerating all the circuits to which belongs a given element is also in **IncP**. We study it in more details in Section 2.6.

Let  $\text{ENUM}\cdot A$  be an enumeration problem, we define the following corresponding search problem:

$\text{ANOTHERSOLUTION}_A$

*Input:* an instance  $x$  of  $A$  and a subset  $S$  of  $A(x)$

*Output:* an element of  $A(x) \setminus S$  and a special value if  $A(x) = S$

One extends this definition of  $\text{ANOTHERSOLUTION}$  to an ordered enumeration problem in the following way:

$\text{ANOTHERSOLUTION}_{(A, <)}$

*Input:* an instance  $x$  of  $A$  and a subset  $S$  of  $A(x)$

*Output:* the minimal element of  $A(x) \setminus S$  for  $<_x$  and a special value if  $A(x) = S$

The classes **IncP** and **IncP<sup>o</sup>** may be defined only by using these  $\text{ANOTHERSOLUTION}$  problems, a proof for the ordered case is given below.

**Proposition 2.14.**  $\text{ANOTHERSOLUTION}_{(A, <)} \in \text{FP} \Leftrightarrow \text{ENUM}\cdot(A, <) \in \text{IncP}^{\circ}$

*Proof.* Assume  $\text{ANOTHERSOLUTION}_{(A, <)}$  is computable in polynomial time, we describe an algorithm which solves  $\text{ENUM}\cdot(A, <)$  on the input  $x$ . Let  $S$  be the empty set. One applies the algorithm solving  $\text{ANOTHERSOLUTION}_{(A, <)}$  to  $x$  and to  $S$ . If it outputs a solution, we add it to  $S$  and call recursively this algorithm on  $x$  and  $S$ . If not, the algorithm stops.

Since  $\text{ANOTHERSOLUTION}_{(A, <)}$  gives a minimal solution for  $<_x$ , the described enumeration algorithm respects the order. The delay between the  $i^{\text{th}}$  and the



$i + 1^{\text{th}}$  solution is bounded by the execution time of the algorithm  $\text{ANOTHERSOLUTION}_{(A, <)}$  which is polynomial in  $|x|$  and  $|S|$  the number of already produced solutions.

Conversely, assume that  $\text{ENUM}\cdot(A, <) \in \mathbf{IncP}^{\circ}$ . On an instance  $(x, S)$  of  $\text{ANOTHERSOLUTION}_{(A, <)}$  we want to find a minimal solution which is not in  $S$  if it exists. Assume that there is a solution in  $A(x) \setminus S$ , one enumerates  $|S| + 1$  solutions thanks to the  $\mathbf{IncP}^{\circ}$  algorithm, in time polynomial in  $|S|$  and  $|x|$ . The first generated solution which is not in  $S$  is a minimal element of  $A(x) \setminus S$  and one outputs it. If  $S = A(x)$ , the enumeration will end in time polynomial in  $|S|$  and  $|x|$ . In this case, one outputs "None" meaning there is no other solution.  $\square$

### 2.2.3 Polynomial delay

For practical applications, one wants the delay to be bounded uniformly, that is it must not depend on the number of already enumerated solutions. This property is captured by the following classes.

**Definition 2.15.** A problem  $\text{ENUM}\cdot A$  (resp.  $\text{ENUM}\cdot(A, <)$ ) is computable in polynomial delay, written  $\mathbf{DelayP}$  (resp.  $\mathbf{DelayP}^{\circ}$ ), if there is a polynomial  $Q(x, y)$  and a machine  $M$  which solves  $\text{ENUM}\cdot A$  (resp.  $\text{ENUM}\cdot(A, <)$ ) and satisfies for all  $x$  and all  $i$ ,  $T(x, i + 1) - T(x, i) \leq Q(|x|)$ .

One alternative to this definition is to ask for the delay to be polynomially bounded in the size of one output instead of the instance. This choice may be different in some cases (see an example in [Cou09]). We may also ask the space to be bounded polynomially, which would forbid some methods we describe in the next sections.

The problem  $\text{ENUM}\cdot \text{MAXIS}$  paired up with the lexicographic order is known to be in  $\mathbf{DelayP}^{\circ}$  [JPY88]. In this paper, although the delay is polynomial, the space used during the enumeration is exponential. Indeed the algorithm maintains a priority queue, which stores a potentially exponential number of solutions, and this justifies our choice of a RAM machine model.

On the other hand, there is an algorithm which solves the unordered problem [TIAS77] in polynomial space. In fact, it is even memoryless meaning that during the enumeration, it uses the last generated solution and no other information to find the next solution. The following class tries to formalize this property, which is shared by many other algorithms.

**Definition 2.16.** A problem  $\text{ENUM}\cdot A$  is computable in strong polynomial delay, written  $\mathbf{SDelayP}$ , if for every  $x$  there is a total order  $<_x$  such that the following problems are in  $\text{FP}$ :

1. given  $x$ , output the first element of  $A(x)$  for  $<_x$
2. given  $x$  and  $y \in A(x)$  output the next element of  $A(x)$  for  $<_x$  or "None" if there is none

If the set  $\{<_x\}_{x \in \Sigma^*}$  is an enumeration order denoted by  $<$ , we say that  $(\text{ENUM}\cdot A, <)$  is in the class **SDelayP**<sup>o</sup>.

**Example 2.17.** The enumeration of the minimal spanning trees or of the maximal matchings of a weighted graph are in **SDelayP** (see [AF96, Uno97]). In fact, a problem which can be solved by a traversal of an implicit tree of solutions, without storing global information is in **SDelayP**. We will see such an example in Section 3.4.

Let  $A$  be a predicate such that  $\exists.A$  is in  $P$  and the associated search problem is computable by self-reduction. Then,  $\text{ENUM}\cdot A$  is in **SDelayP**. For a precise definition of this notion and an example of a problem not self-reducible, see [KV91]. We now explain this property for the problem SAT.

Let  $\phi(x_1, \dots, x_n)$  be a formula and let  $\text{SAT}(\phi(x_1, \dots, x_n))$  be the set of satisfying assignments of  $\phi(x_1, \dots, x_n)$ . The set  $\text{SAT}(\phi(x_1, \dots, x_n))$  is equal to  $\text{SAT}(\phi(0, x_2, \dots, x_n)) \cup \text{SAT}(\phi(1, x_2, \dots, x_n))$ . Both formulas  $\phi(0, \dots, x_n)$  and  $\phi(1, \dots, x_n)$  can be transformed into two logically equivalent formulas  $\psi(x_2, \dots, x_n)$  and  $\chi(x_2, \dots, x_n)$ , which depend on the  $n - 1$  variables  $x_2, \dots, x_n$ . We have then reduced the decision of SAT on the instance  $\phi$  on the decision of SAT on two instances with less variables. If one can decide if one of the two formulas has a satisfying assignment, we can use the same procedure on it and eventually build a satisfying assignment.

We write  $\text{SAT}(\mathcal{C})$  the problem SAT on the formulas in  $\mathcal{C}$ . If SAT is in  $P$ , one can find the first satisfying assignment for the lexicographic order of  $\phi(x_1, \dots, x_n)$  in polynomial time. By doing a depth-first traversal in the tree of the partial assignments of variables, which let the formula satisfiable, one enumerates all satisfying assignments in **SDelayP**.

**Proposition 2.18** (Creignou and Herbrard [CH97]). *The problem  $\text{ENUM}\cdot\text{SAT}(\mathcal{C})$  is in **SDelayP** when  $\mathcal{C}$  is one of the following classes:*

1. *Horn formulas*
2. *anti-Horn formulas*
3. *affine formulas*
4. *bijunctive formulas*

*Sketch of the proof:* One proves this proposition by checking that the result of a substitution of a truth value to a variable in a formula of one of these four types of formula is equivalent to a formula of the same type. Since the decision problem on these classes of formula is in  $P$ , one can enumerate all solutions in lexicographic order by the method provided before.  $\square$

The enumeration version of such problems have also been studied with the constraint of giving only minimal or maximal solutions (for inclusion), which make them hard most of the time [KSS00].

Finally we introduce the class of enumeration problems, such that we have access to any solution in polynomial time.

**JTHSOLUTION<sub>A</sub>**

*Input:* an integer  $i$  given in binary and  $x \in \Sigma^*$

*Output:* the  $i^{\text{th}}$  element of  $A(x)$  in some total order  $<_x$  or a special value if  $|A(x)| < i$

**Definition 2.19.** A problem **ENUM·A** is in the class **QueryP** if **JTHSOLUTION<sub>A</sub>** is in **FP**. If the set  $\{<_x\}_{x \in \Sigma^*}$  of orders in the definition of **JTHSOLUTION<sub>A</sub>** is an enumeration order, denoted by  $<$ , we say that  $(\text{ENUM·A}, <) \in \text{QueryP}^\circ$ .

**Example 2.20.**

**ENUM·LINEARSYSTEM**

*Input:* a  $n \times m$  matrix  $M$  and a vector  $y$  of size  $m$ , both over the finite field  $\mathbb{F}$

*Output:* the set of vectors  $y$  such that  $My = b$

The problem **ENUM·LINEARSYSTEM** is in **QueryP**. The solution of the equation  $My = b$  is a vector subspace and one can compute one of its basis  $(y_1, \dots, y_k)$  in polynomial time. The set of solutions is the set of vectors equal to  $\lambda_1 y_1 + \dots + \lambda_k y_k$  for all  $(\lambda_1, \dots, \lambda_k) \in \mathbb{F}^k$ . One orders these solutions by the lexicographic order on the words  $\lambda_1 \dots \lambda_k$ . One finds the  $i^{\text{th}}$  such word (representation of  $i$  in base  $k$ ) and computes the corresponding vector in polynomial time.

Most of the known examples of **QueryP** problems are satisfaction problems for restricted logics or over models with good properties. For instance, one the enumeration of the satisfying assignments of a first order formula over structures of bounded degree is in **QueryP** [BDGO08]. This is also true for a monadic second-order formula over trees and over graphs or matroids of bounded branch-width ([Cou09] and Section 6.4).

Finally, one can easily count the number of solutions of a problem in **QueryP**. It is generally not the case for the enumeration problems outside this class.

**Proposition 2.21.** *Let  $A$  be a polynomially balanced predicate decidable in polynomial time. If  $\text{ENUM·A} \in \text{QueryP}$  then  $\#A \in \text{FP}$ .*

*Proof.* First remark that  $|A(x)| \leq 2^{Q(|x|)}$  for some polynomial  $Q$ . Since  $\text{ENUM·A} \in \text{QueryP}$ , one can decide if  $|A(x)| < i$  in polynomial time in  $|i|$  and  $|x|$ . Thus by a dichotomic search, one finds the exact size of  $A(x)$ , with  $Q(|x|)$  dichotomic steps, which proves that the counting problem is in **FP**  $\square$

## 2.3 Separation between classes

We prove in this section that most of the previously defined classes are different under reasonable complexity assumptions. The only exception concerns the two most natural classes, **IncP** and **DelayP**, which are not known to be different.

First notice that, when the number of solutions of a problem is polynomially bounded, if the problem is in **TotalP** it is also in **QueryP**! Therefore most of the separation results will use problems with a potentially super polynomial number of solutions. However, there are “hard” search problems with few solutions. For instance the problem of finding a prime factor of an integer is not proved to be in FP but its decision version is in P [AKS04]. The associated enumeration problem has a very small number of solutions to enumerate (linear in the size of the input), and thus it cannot be in **TotalP** without being in **QueryP**. It is a good candidate to separate **EnumP** from **TotalP** without relying on the hardness of the related decision problem.

### 2.3.1 Unordered enumeration problems

The first proposition illustrates the collapse of the enumeration complexity classes when  $P = NP$ . Conversely, the next ones explore the consequences of the collapse of enumeration classes on decision classes.

**Proposition 2.22.** *If  $P = NP$  then  $\text{EnumP} = \text{SDelayP}$ .*

*Proof.* Recall that every problem in NP is parsimoniously reducible to SAT. The same reduction proves that  $\text{ENUM}\cdot\text{SAT}$  is complete for **EnumP**. We assume that  $P = NP$ , therefore we can decide SAT in polynomial time. Since SAT is self-reducible one can design an algorithm in **SDelayP** for  $\text{ENUM}\cdot\text{SAT}$ , thus  $\text{EnumP} = \text{SDelayP}$ .  $\square$

**Proposition 2.23.**  *$P = NP$  if and only if  $\text{TotalP} = \text{EnumP}$ .*

*Proof.* Assume that  $\text{ENUM}\cdot\text{SAT}$  is in **TotalP**, then by Lemma 2.11  $\text{ALLSOLUTIONS}_{\text{SAT}}$  is in P. Since SAT reduces to  $\text{ALLSOLUTIONS}_{\text{SAT}}$ , we have  $P = NP$ .

Conversely assume that  $P = NP$ , Proposition 2.22 implies that  $\text{TotalP} = \text{EnumP}$ .  $\square$

**Proposition 2.24.** *If  $\text{TotalP} = \text{IncP}$  then  $P = \text{coNP} \cap \text{NP}$ .*

*Proof.* Let  $L$  be a language in  $\text{coNP} \cap \text{NP}$ : there is a predicate  $A$  such that  $L = \{x \mid A(x) \neq \emptyset\}$  and a predicate  $B$  such that  $\bar{L} = \{x \mid B(x) \neq \emptyset\}$ . Let  $Q$  be a polynomial such that  $y \in A(x)$  or  $y \in B(x)$  implies that  $|y| \leq Q(|x|)$ . For simplicity, say that  $L$  is a language in  $\{0, 1\}^*$ .

Let  $\Pi(x, y\#w)$  be the predicate which is true if and only if  $A(x, y) \vee B(x, y)$  and  $|w| \leq Q(|x|)$ , that is to say we take the union of  $A$  and  $B$  and do some padding.

The set  $\Pi(x)$  is never empty, since if  $x \in L$  there is a  $y$  such that  $A(x, y)$  holds and if not there is a  $y$  such that  $B(x, y)$  holds. Because of the padding, there are more than  $2^{|w|} = 2^{Q(|x|)}$  elements in  $\Pi(x)$  for each  $y$  satisfying either  $A$  or  $B$ . Therefore the naive exhaustive enumeration algorithm is polynomial in the number of solutions, which proves that the enumeration problem  $\Pi$  is in **TotalP**.

Assume  $\text{IncP} = \text{TotalP}$ , we have an incremental algorithm for the problem  $\text{ENUM}\cdot\Pi$ . In particular, it gives, on any instance  $x$ , the first solution  $y\#w$

in polynomial time and we can decide whether  $y$  satisfies  $A(x, y)$  or  $B(x, y)$  in polynomial time. This procedure decides if  $x \in L$  in polynomial time therefore  $P = \text{coNP} \cap \text{NP}$ .  $\square$

Contrarywise to the previous proposition, we only have an implication. To prove an equivalence, one may try to strengthen the conclusion of the last proposition to  $P = \text{NP}$  or to prove that  $P = \text{coNP} \cap \text{NP}$  implies  $\mathbf{TotalP} = \mathbf{IncP}$ . It is unlikely that the two above statements can be easily proved at the same time or we would have the seemingly difficult theorem  $P = \text{NP} \Leftrightarrow P = \text{coNP} \cap \text{NP}$ .

**Remark 2.25.** At this point the separation of  $\mathbf{DelayP}$  and  $\mathbf{IncP}$  seems elusive. Remark however that the total time spent by a  $\mathbf{DelayP}$  algorithm is linear in the number of solutions whereas it is polynomial for an  $\mathbf{IncP}$  algorithm. In fact, if we generalize these classes to enumeration problems defined by predicates, which are not necessarily polynomial time decidable, we can prove that they are unconditionally different thanks to the time hierarchy theorem.

The enumeration of circuits of a matroid studied in [KBE<sup>+</sup>05] is in  $\mathbf{IncP}$  and does not seem to be in  $\mathbf{DelayP}$ . It could be a good candidate to separate  $\mathbf{DelayP}$  and  $\mathbf{IncP}$ .

**Proposition 2.26.** *If  $\#\text{P} \neq \text{FP}$  then  $\mathbf{QueryP} \subsetneq \mathbf{SDelayP}$ .*

*Proof.* A careful analysis of the algorithm of [Uno97] which enumerates the perfect matchings of a graph reveals that this problem is in  $\mathbf{SDelayP}$ . If  $\mathbf{QueryP} = \mathbf{SDelayP}$ , the counting of the perfect matchings is in  $\text{FP}$  by proposition 2.21. Since it is a  $\#\text{P}$ -complete problem, we have proved that  $\#\text{P} = \text{FP}$ .  $\square$

**Corollary 2.27.** *If  $P \neq \text{coNP} \cap \text{NP}$ , the following inclusions hold:*

$$\mathbf{QueryP} \subsetneq \mathbf{SDelayP} \subseteq \mathbf{DelayP} \subseteq \mathbf{IncP} \subsetneq \mathbf{TotalP} \subseteq \mathbf{EnumP}.$$

### 2.3.2 Ordered enumeration problems

Notice that all separation results for unordered problems hold also for ordered problems. Nevertheless, we need a new proof of the collapse of the enumeration classes when  $P = \text{NP}$ .

**Proposition 2.28.** *If  $P = \text{NP}$  then  $\mathbf{EnumP}^\circ = \mathbf{SDelayP}^\circ$ .*

*Proof.* Let  $\text{ENUM}\cdot(A, <) \in \mathbf{EnumP}^\circ$ , we define the predicate  $\Pi(x\#w, y)$  which is true when  $y$  is the successor of  $w$  for  $<_x$  in  $A(x)$ :

$$\Pi(x\#w, y) = \forall z [A(x, y) \wedge A(x, w) \wedge w <_x y \wedge (A(x, z) \Rightarrow (z <_x w \vee y \leq z))]$$

Since the predicate  $A$  and the order  $<_x$  are polynomial time decidable, the decision problem  $\exists y \Pi(x\#w, y)$  is in  $\Sigma_2^P$ . Hence, it parsimoniously reduces to the complete problem  $\Sigma_2\text{SAT}$ . It means that the reduction preserves the set of  $y$  which satisfy  $\Pi(x\#w, y)$  (which is here a singleton).

Assume now that  $P = NP$ , the polynomial hierarchy collapses and  $\Sigma_2\text{SAT}$  is decidable in polynomial time. The problem  $\Sigma_2\text{SAT}$  is self-reducible for the same reasons that  $\text{SAT}$  is. Therefore the search problem associated to  $\Sigma_2\text{SAT}$  is in  $\text{FP}$ . Since  $\exists y \Pi(x \# w, y)$  parsimoniously reduces to  $\Sigma_2\text{SAT}$ , one can find  $y$  such that  $\Pi(x \# w, y)$  in polynomial time. We have proved that from a solution  $w$  of  $A(x)$ , we can find the next one according to  $<_x$  in polynomial time. We can also find the first for  $<_x$  in the same way, thus  $\text{ENUM}\cdot(A, <) \in \mathbf{SDelayP}^\circ$ .  $\square$

For some separation results we can also do better than in the unordered case. For instance one can relax the hypothesis needed to prove  $\mathbf{IncP} \neq \mathbf{TotalP}$ .

**Proposition 2.29.**  $P = NP$  if and only if  $\mathbf{IncP}^\circ = \mathbf{TotalP}^\circ$ .

*Proof.* It is known that  $\text{ENUM}\cdot\text{MAXIS} \in \mathbf{DelayP}$ . Hence the problem is in  $\mathbf{TotalP}$  and also in  $\mathbf{TotalP}^\circ$  when paired with any enumeration order  $<$ . The problem of deciding if a subgraph is the first maximal independent set of a graph in the anti-lexicographic order is NP-complete [JPY88]. If  $\mathbf{IncP}^\circ = \mathbf{TotalP}^\circ$ , the enumeration of the maximal independent sets of a graph in the anti-lexicographic order can be done in incremental time. Therefore we can find the first in polynomial time and  $P = NP$ .

The converse holds because of Proposition 2.28.  $\square$

We can also use the results on  $\text{ENUM}\cdot\text{MAXIS}$  to get a new separation of classes:

**Proposition 2.30.**  $P = NP$  if and only if  $\mathbf{SDelayP}^\circ = \mathbf{DelayP}^\circ$ .

*Proof.* The problem of enumerating all maximal independent sets of a graph in the lexicographic order is in  $\mathbf{DelayP}^\circ$  [JPY88]. In the same article it is proved that given a maximal independent set, finding the next in the lexicographic order is NP-complete. Assume  $\mathbf{SDelayP}^\circ = \mathbf{DelayP}^\circ$ , then by definition of  $\mathbf{SDelayP}^\circ$  the previous problem is also in  $P$ , thus  $P = NP$ .

The converse holds because of Proposition 2.28.  $\square$

**Corollary 2.31.** Assume  $P \neq NP$ , we have the following inclusions:

$$\mathbf{QueryP}^\circ \subsetneq \mathbf{SDelayP}^\circ \subsetneq \mathbf{DelayP}^\circ \subseteq \mathbf{IncP}^\circ \subsetneq \mathbf{TotalP}^\circ \subsetneq \mathbf{EnumP}^\circ.$$

## 2.4 The power of ordering

Here we briefly study how an enumeration problem can be made difficult by the order alone. The fundamental example is the problem  $\text{ENUM}\cdot\text{MAXIS}$  which is hard for the reverse lexicographic order but easy for the lexicographic order. It suggests restrictions to the kind of order allowed to define an enumeration problem.

### 2.4.1 Hardness through a family of orders

We use the predicate  $SAT(x, y)$  which defines the NP-complete SAT problem. The assignments  $y$  of a formula  $x$  are encoded in such a way that  $|y| \leq |x|$ . Let  $O_1(x, y)$  be the predicate true for all  $(x, y)$  such that  $|y| \leq |x|$ . The problem  $ENUM \cdot O_1$  is trivial: it is in **QueryP**. We denote by  $\overline{SAT}(x)$  the set  $\Sigma^{|x|} \setminus SAT(x)$ .

**Definition 2.32.** Let  $<_x^1$  be an order on  $\Sigma^*$  for each  $x \in \Sigma^*$  with the following properties:

- the restriction of  $<_x^1$  to the sets  $SAT(x)$ ,  $\overline{SAT}(x)$  and  $\Sigma^{>|x|}$  is the lexicographic order
- any element of  $SAT(x)$  is greater than any element of  $\overline{SAT}(x)$ , which is greater than any element of  $\Sigma^{>|x|}$

Since the three sets are a partition of  $\Sigma^*$ ,  $<_x^1$  is a total order on  $\Sigma^*$ . Given  $x$ ,  $y_1$  and  $y_2$ , one can decide in polynomial time in which of the three sets  $SAT(x)$ ,  $\overline{SAT}(x)$  and  $\Sigma^{>|x|}$   $y_1$  and  $y_2$  are. One also computes the lexicographic order of  $y_1$  and  $y_2$  in linear time. From these informations, one can compare  $y_1$  and  $y_2$  for  $<_x^1$  in polynomial time. Therefore, the set  $\{<_x^1\}_{x \in \Sigma^*}$  that we write  $<^1$  is an enumeration order.

**Proposition 2.33.** *If  $P \neq NP$ , the problem  $(O_1, <^1)$  is not in **IncP<sup>o</sup>**.*

*Proof.* For each  $x \in \Sigma^*$ , the first element of  $O_1(x)$  for  $<_x^1$  is the least lexicographic element of  $SAT(x)$  if it is not empty. Assume  $ENUM \cdot (O_1, <^1)$  is in **IncP<sup>o</sup>**, one can find in polynomial time the first element of  $O_1(x)$ . One decides if it is in  $SAT(x)$  in polynomial time, which is equivalent to decide the emptiness of  $A(x)$ , that is to say the problem SAT in polynomial time.  $\square$

### 2.4.2 Hardness through one order

After reading the previous subsection, one may think that the definition of an enumeration order is not restrictive enough. Indeed, the hardness may come from the lack of uniformity, since we have one order for each instance of the problem. Assume now that an enumeration order does not depend anymore on  $x$ , i.e. it is only a total order on  $\Sigma^*$  computable in polynomial time.

We now consider the predicate  $O_2(x, y)$  which is true when  $y = x\#w$  and  $|w| \leq |x|$ . Again, the enumeration problem  $ENUM \cdot O_2$  is very easy: it is in **QueryP**. We denote by  $S$  the set of words of the form  $y = x\#w$  with  $|w| \leq |x|$ .

**Definition 2.34.** The order  $<^2$  has the following properties:

- the restriction of  $<^2$  to  $\Sigma^* \setminus S$  is the lexicographic order
- let  $y_1 = x_1\#w_1$  and  $y_2 = x_2\#w_2$  be two elements of  $S$ . If  $x_1 \neq x_2$  then  $y_1 <_2 y_2$  if and only if  $x_1 <_{lex} x_2$

- if  $x_1 = x_2$ , then  $y_1 <_2 y_2$  if and only if  $w_1 <_{x_1}^1 w_2$ , where  $<_{x_1}^1$  is the order defined in the previous section.
- the elements of  $S$  are all smaller than the elements of  $\Sigma^* \setminus S$  for  $<^2$

The order  $<^2$  is total and computable in polynomial time for the same reasons that  $<^1$  is. The following proposition holds; its proof is similar to that of Proposition 2.33.

**Proposition 2.35.** *If  $P \neq NP$ , the ordered problem  $(O_2, <^2)$  is not in  $\mathbf{IncP}^o$ .*

### 2.4.3 Hardness through one enumerable order

The previous example shows that we have yet to find a restriction on enumeration orders strong enough to avoid these artificial hard problems. We could ask the order itself to be enumerable, meaning that given an element  $y$  we can find in polynomial time its successor in the order. Note that  $<^1$  and  $<^2$  do not satisfy this condition, while the lexicographic order does.

We now give one problem which is made more difficult by such an order. This time a part of the hardness comes from the problem, since we cannot encode easily a NP-complete problem in such an order.

Let  $O_3(x, y)$  be the predicate which holds when:

- $y = 0\#w$ ,  $|w| \leq |x|$  and  $\text{SAT}(x, w)$
- $y = 1\#w$  and  $|w| \leq |x|$

The idea is that  $O_3(x)$  contains all the elements in  $\text{SAT}(x)$  and an exponential number of trivial solutions.

**Proposition 2.36.** *The problem  $\text{ENUM} \cdot O_3$  is in  $\mathbf{DelayP}$ . On the other hand, if  $P \neq NP$ ,  $\text{ENUM} \cdot (O_3, <_{lex})$  is not in  $\mathbf{IncP}^o$ .*

*Proof.* There is a polynomial delay algorithm for the unordered problem  $\text{ENUM} \cdot O_3$ . For  $x \in \Sigma^*$ , one enumerates all  $w \in \Sigma^*$  with  $|w| \leq |x|$ . For each of those  $w$ , one tests in polynomial time if  $\text{SAT}(x, w)$  holds, if yes it outputs  $0\#w$ , and it also always outputs  $1\#w$ .

Assume  $\text{ENUM} \cdot (O_3, <_{lex}) \in \mathbf{IncP}^o$ , one can find in polynomial time the minimal solution in lexicographic order of  $O_3(x)$ . If  $x$  is in  $\text{SAT}$ , this solution is  $0\#w$  with  $\text{SAT}(x, w)$  and  $w$  is minimal in lexicographic order for this property. If  $x$  is not in  $\text{SAT}$ , the first solution is  $1\#$ . Hence, one decides  $\text{SAT}$  in polynomial time and  $P = NP$ .  $\square$

We now give other less artificial problems which are known to be in  $\mathbf{DelayP}$  without order but not even in  $\mathbf{IncP}^o$  for a specific enumeration order.



- The problem  $\text{ENUM}\cdot\text{MAXIS}$  is in  $\mathbf{DelayP}$ , but once paired with the reverse lexicographic order, it is not in  $\mathbf{IncP}^\circ$  if  $P \neq \text{NP}$ . Notice that, though one polynomial delay algorithm for  $\text{ENUM}\cdot\text{MAXIS}$  in the lexicographic order is known, it uses an exponential space.
- Let  $<$  be an enumeration order on words representing graphs, such that for two graphs  $G_1$  and  $G_2$ ,  $G_1 < G_2$  if  $G_1$  has less vertices than  $G_2$ . The problem  $\text{ENUM}\cdot(\text{MAXIS}, <)$  is not in  $\mathbf{IncP}^\circ$  if  $P \neq \text{NP}$ . Indeed, the problem to find the size of a maximum independent set of a graph is NP-complete.
- A polynomial delay algorithm to enumerate all maximal acyclic subhypergraphs of a hypergraph is given in [DH09]. In the same article, it is also proved that the problem of finding the first lexicographic maximal acyclic subhypergraph is coNP-complete, therefore the enumeration problem is not in  $\mathbf{IncP}^\circ$  if  $P \neq \text{NP}$ .
- See Section 2.6 for a problem on matroids which can be solved in incremental time, but not for the enumeration order  $<_w$ .

## 2.5 Operations on predicates and enumeration

In this section we study the behavior of the enumeration problems with regard to simple set operations. The first result on union of predicates is of interest since it allows to design algorithms with a good delay even when there is no orders on the solutions.

### 2.5.1 Union of predicates

**Definition 2.37.** Let  $A(x, y)$  and  $B(x, y)$  be two polynomially balanced predicates. The union of  $A$  and  $B$ , denoted by  $[A \cup B]$ , is defined by: for all  $x, y$ ,  $[A \cup B](x, y)$  holds if and only if  $A(x, y)$  holds or  $B(x, y)$  holds.

**Proposition 2.38** (Parallel enumeration). *Let  $\text{ENUM}\cdot A$  and  $\text{ENUM}\cdot B$  be two problems of  $\mathbf{EnumP}$ , then the problem  $\text{ENUM}\cdot[A \cup B]$  is also in  $\mathbf{EnumP}$ . Moreover, there is a polynomial  $Q$  such that, if  $M_A$  is a RAM machine solving  $\text{ENUM}\cdot A$  with delay  $f$  and  $M_B$  is a RAM machine solving  $\text{ENUM}\cdot B$  with delay  $g$ , there is a RAM machine solving  $\text{ENUM}\cdot[A \cup B]$  with delay  $f + g + Q$ .*

*Proof.* Let  $M_A$  and  $M_B$  be two RAM machines, which solve  $\text{ENUM}\cdot A$  and  $\text{ENUM}\cdot B$ . One builds a machine  $M_{A \cup B}$  which solves  $\text{ENUM}\cdot[A \cup B]$  by running  $M_A$  and  $M_B$  in parallel on the instance  $x$ .

At each step,  $M_{A \cup B}$  produces a new solution  $y$  of  $A(x)$  thanks to  $M_A$ . It then tests if  $y \in B(x)$  in polynomial time because  $B \in \mathbf{EnumP}$ . If  $y \notin B(x)$  it outputs it, otherwise it is discarded and the next solution of  $B(x)$  given by  $M_B$

is computed and outputted<sup>1</sup>. If there is no solution left in  $A(x)$  (resp.  $B(x)$ ), it finishes the enumeration thanks to  $M_B$  (resp.  $M_A$ ).

Remark that if  $M_{A \cup B}$  has enumerated  $k$  elements of  $B(x)$  thanks to  $M_B$  then it has also found and discarded  $k$  elements of  $A(x) \cap B(x)$  given by  $M_A$ . Therefore if  $M_{A \cup B}$  has outputted all  $B(x)$ , it has used  $M_A$  to produce  $|B(x)|$  elements of  $A(x) \cap B(x)$ , which must then satisfy  $B(x) = A(x) \cap B(x)$ . Therefore the enumeration of the remaining elements of  $A(x)$  does not create any repetition. Moreover all elements of  $A(x) \cap B(x)$  are enumerated only by  $M_B$ , thus the algorithm makes no repetition.

The delay of  $M_{A \cup B}$  is bounded by the sum of the delays of  $M_A$  and  $M_B$  plus a polynomial, because at each step of the algorithm we simulate  $M_A$  and  $M_B$  enough time to let them produce one solution. The polynomial overhead comes from the cost of running the two machines in parallel and of checking if a solution is in  $B(x)$ .  $\square$

One says that a class  $\mathcal{C}$  is stable under an operation  $f$  defined over  $\mathcal{C} \times \mathcal{C}$ , if for all  $a, b \in \mathcal{C}$ , one has  $f(a, b) \in \mathcal{C}$ .

**Corollary 2.39.** *The classes **TotalP**, **IncP**, **DelayP** are stable under union.*

We now give one simple application of this proposition to a classical problem. One says that a formula is in disjunctive normal form if it is the disjunction of clauses, where each clause is the conjunction of variables or negation of variables called literals.

ENUM·DNF

*Input:* a formula in disjunctive normal form

*Output:* all satisfying assignments of the formula

**Proposition 2.40.** *The problem ENUM·DNF is in **DelayP**.*

*Proof.* A DNF formula is a disjunction of clauses. The enumeration of the models of one clause is easy, since a clause fixes the value of each variable which appears in it and any truth value of the other variables gives a satisfying assignment. Thus one can enumerate the models of a clause with linear delay. Remark that the disjunction of two clauses has for models the union of the models of the clauses. Thus using Proposition 2.38 as many times as they are clauses, we can enumerate the models of the disjunction of all clauses with a delay which is proportional to the number of clauses times the size of a model plus a polynomial overhead because of the parallelization.  $\square$

We can improve the delay to a linear delay, by designing an algorithm specific to ENUM·DNF. We now adapt Proposition 2.38 to the case of ordered enumeration problem. In fact, the result is even better, because one does not need the hypothesis that the problems are in **EnumP**.

---

<sup>1</sup>note that it can be  $y$  itself

**Proposition 2.41.** *Let  $\text{ENUM}\cdot A$  and  $\text{ENUM}\cdot B$  be two enumeration problems, and let  $<$  be an enumeration order. There is a polynomial  $Q$  such that, if  $M_A$  is a RAM machine solving  $\text{ENUM}\cdot(A, <)$  with delay  $f$  and  $M_B$  is a RAM machine solving  $\text{ENUM}\cdot(B, <)$  with delay  $g$ , there is a RAM machine solving  $\text{ENUM}\cdot([A \cup B], <)$  with delay  $f + g + Q$ .*

*Proof.* The proof of this proposition is nothing but the description of a variant of the merge procedure of the famous merge-sort algorithm.

Let  $M_A$  and  $M_B$  be two machines which solve  $\text{ENUM}\cdot(A, <)$  and  $\text{ENUM}\cdot(B, <)$ , we describe a machine  $M_{A \cup B}$  which solves  $\text{ENUM}\cdot([A \cup B], <)$ . It runs  $M_A$  and  $M_B$  in parallel on the instance  $x$  and uses two variables  $y_A$  and  $y_B$  which serve as buffers for solutions of  $M_A$  and  $M_B$  respectively.

The machine begins with the first solution produced by  $M_A$  in  $y_A$  and the first solution produced by  $M_B$  in  $y_B$ . At the beginning of each step it compares  $y_A$  and  $y_B$ .

- If they are equal it outputs their value once and puts in  $y_A$  and  $y_B$  the next solution generated by respectively  $M_A$  and  $M_B$ .
- If  $y_A < y_B$ , it outputs  $y_A$  and puts in  $y_A$  the next solution generated by  $M_A$ .
- If  $y_B < y_A$ , it outputs  $y_B$  and puts in  $y_B$  the next solution generated by  $M_B$ .

When at a point of the algorithm, let say w.l.o.g.  $M_A$  has generated all its solutions, one outputs all solutions produced by  $M_B$  and then stops.

This algorithm clearly gives every solution of  $[A \cup B]$ . The sequence of solutions in respectively  $y_A$  and  $y_B$  are increasing for  $<_x$  by definition of  $M_A$  and  $M_B$ . Since we always output first the smallest value between  $y_A$  and  $y_B$ ,  $M_{A \cup B}$  outputs the solution in strictly increasing order. It also proves that there is no repetition.  $\square$

## 2.5.2 Subtraction of predicates

We study also the subtraction between two predicates in the spirit of the subtractive reduction forged for the counting problems [DHK05].

**Definition 2.42.** Let  $A(x, y)$  and  $B(x, y)$  be two polynomially balanced predicates such that for all  $x$ ,  $B(x) \subseteq A(x)$ . The predicate  $[A \setminus B](x, y)$  is defined by: for all  $x, y$ ,  $[A \setminus B](x, y)$  holds if and only if  $y \in A(x) \setminus B(x)$ .

**Proposition 2.43.** *If  $\text{P} \neq \text{NP}$  then the classes **DelayP**, **IncP** and **TotalP** are not stable by subtraction.*

*Proof.* The idea of the proof is that disjunctive formulas are the dual of conjunctive formulas, whose models are hard to enumerate. We need the following problem:

**ENUM-ALL**

*Input:* a formula in disjunctive normal form

*Output:* all assignments of the formula

Let  $\phi$  be a formula in conjunctive normal form, the formula  $\neg\phi$  is equivalent to a formula  $\Psi$  of the same size in disjunctive normal form. Furthermore, let us remark that  $SAT(\phi) = ALL(\phi) \setminus DNF(\neg\phi)$ . The problems  $ENUM\cdot ALL$  and  $ENUM\cdot DNF$  are in **SDelayP**. Their substraction is the problem  $ENUM\cdot SAT$ , which is **EnumP**-complete for parsimonious reductions. Thus if the class **SDelayP** is stable under substraction  $SDelayP = EnumP$  which implies  $P = NP$ . The same is true for the classes above **SDelayP**.  $\square$

**Remark 2.44.** Let  $ENUM\cdot(A, <)$  and  $ENUM\cdot(B, <)$  two ordered enumeration problems with the same enumeration order, there are a few cases in which an algorithm with good delay for  $ENUM\cdot(A, <)$  and  $ENUM\cdot(B, <)$  yields an algorithm with good delay for  $ENUM\cdot([A \setminus B], <)$ :

1. when one of the problem has always a polynomial number of solutions
2. when  $B(x)$  is always a final segment of  $A(x)$
3. when  $B(x)$  is not too dense in  $A(x)$ , that is each initial segment  $I$  of  $A(x)$  does not contain too much elements of  $B(x)$ . For instance, if there is  $c > 0$  and a polynomial  $Q$  such that for all  $I$ ,  $|I \setminus B(x)| > \frac{|I|^c}{Q(|x|)}$ , one can solve  $ENUM\cdot([A \setminus B], <)$  with polynomial delay if  $ENUM\cdot(A, <)$  and  $ENUM\cdot(B, <)$  are in **DelayP<sup>o</sup>**.

### 2.5.3 Intersection of predicates

The third natural operation on sets, the intersection does not let the enumeration classes stable either.

**Definition 2.45.** Let  $A(x, y)$  and  $B(x, y)$  be two polynomially balanced predicates. The intersection of  $A$  and  $B$ , denoted by  $A \cap B$ , is defined by: for all  $x, y$ ,  $[A \cap B](x, y)$  holds if and only if  $A(x, y)$  holds and  $B(x, y)$  holds.

**Proposition 2.46.** *If  $P \neq NP$  then the classes **SDelayP**, **DelayP**, **IncP** and **TotalP** are not stable by intersection.*

*Proof.* The enumeration problems  $ENUM\cdot HORN$  and  $ENUM\cdot AFFINE$  are both in **SDelayP**. The problem  $ENUM\cdot[HORN \cap AFFINE]$  is the problem of satisfiability of the conjunction of Horn and affine clauses, since the intersection of the satisfying assignments of two formulas are the satisfying assignments of the conjunction. This problem is NP-complete thanks to the classification done in Schaeffer dichotomy theorem [CKS01]. Assume that **SDelayP** is stable under intersection, then  $SDelayP = EnumP$  and thus  $P = NP$ .  $\square$

## 2.6 An example: A-Circuit

Let us consider the following problem:

## A-CIRCUIT

*Input:* a matroid  $M$  and a set  $A$  of its elements

*Output:* accept if there is a circuit  $C$  of  $M$  such that  $A \subseteq C$

This problem and its enumeration version, ENUM·A-CIRCUIT, are later used as illustrations of the fixed parameter tractable algorithms developed in Chapter 6. We now give the known complexity results for this problem, they mostly come from [KBE<sup>+</sup>05]. Only matroids with an independence predicate decidable in polynomial time are considered.

1. If  $|A| = 1$  or  $2$ , the problem is decidable in polynomial time. For the particular case of a vector matroid see [DH03], in general use a matroid separation algorithm.
2. If  $|A| = 3$ , the question is open.
3. If  $|A| = k$  is fixed and the matroid is a cycle matroid then it is decidable in polynomial time by reduction to the problem of finding  $k$  disjoint paths in a graph [RS95].
4. If  $|A|$  is unbounded, even if the matroid is only a cycle matroid, the question is NP-complete by reduction from the Hamiltonian Path problem.

Consider now the associated enumeration problem ENUM·A-CIRCUIT. Assume that  $|A| = 1$  and that the matroid is representable on a finite field or on  $\mathbb{Q}$ . Then, this problem is equivalent to enumerate all minimal solutions (for inclusion of the support) of a linear system. If the field is  $\mathbb{F}_2$ , it is equivalent to produce all the minimal (for the pointwise order) solutions of an affine formula, which is an affine variation of the circumscription problem for propositional formula studied in artificial intelligence [McC80]. The question of the complexity of this problem is asked in [KSS00]. At this time, it was not even known to be in **IncP**, but it was later proved in [KBE<sup>+</sup>05] for all matroids with an independence predicate decidable in polynomial time.

We would like to have an algorithm with polynomial delay rather than incremental. It is the case when  $|A| = k$  is fixed and the matroid is a cycle matroid [RT75]. Our aim is to design a polynomial delay algorithm for a class of matroids broader than the cycle matroids and/or for unbounded  $|A|$ . It is only possible for a subclass of the vector matroids when  $|A|$  is unbounded since the decision problem is then NP-complete. We give below a polynomial delay algorithm for a very restricted class of representable matroids. Furthermore, in Chapter 6, a few interesting classes of matroids are presented on which the decision problem and the delay of the enumeration one are linear.

First remark that it is much easier to enumerate free families (and even bases) rather than circuits.

**Proposition 2.47** (Folklore). *Let  $M$  be a matroid with ground set  $V = \{v_1, \dots, v_n\}$  and a dependency predicate decidable in polynomial time. Let  $A \subseteq V$  of size  $k$ , the*

enumeration of the free families of  $M$  containing  $A$  can be done with polynomial delay.

*Proof.* Without loss of generality, one can say that the elements of  $A$  are  $\{v_1, \dots, v_k\}$ . It is easy to check that Algorithm 1 enumerates the free families of  $M$  containing the set  $A$ .  $\square$

---

**Algorithm 1:** Enumeration of the free families containing a fixed set

---

**Data:** A matroid  $M$  with ground set  $\{v_1, \dots, v_n\}$  and an integer  $k$

**Result:** The free families of  $M$  containing  $\{v_1, \dots, v_k\}$

```

begin
   $F \leftarrow A \cup \{v_{k+1}\}$ 
   $i \leftarrow k + 1$ 
  while  $|F| \geq k$  do
    if  $i \geq n$  then
       $t \leftarrow$  largest index of the vectors in  $F$ 
       $F \leftarrow F \setminus \{v_t\}$ 
       $i \leftarrow t + 1$ 
    else
      if  $F \cup v_i$  is independent then
         $F \leftarrow F \cup \{v_i\}$ 
        Output( $F \cup \{v\}$ )
       $i \leftarrow i + 1$ 
  end

```

---

Let us consider representable matroids over a field  $\mathbb{F}_2$ : they are given by a set of vector  $V$  in a vector space  $E$  over the field  $\mathbb{F}_2$ . The elements of  $V$  are noted  $\{v_1, \dots, v_n\}$ , they are implicitly ordered by their indices.

**Lemma 2.48.** *Let  $F \subseteq V$  a free family of vectors and  $v = \sum_{f \in F} f$  then  $F \cup \{v\}$  is a minimal dependent set.*

*Proof.* Assume  $F \cup \{v\}$  is not minimal, then there is  $w \in F$  such that  $F \setminus \{w\} \cup \{v\}$  is dependent. It is easy to generate  $w$  from  $(F \setminus \{w\}) \cup \{v\}$ , hence this set generates the same vector space as the family  $F$ . Since it is of the same size as  $F$ , it cannot be dependent.  $\square$

From now on, we work with  $V$  a vector subspace of  $E$ . This is a very “dense” matroid, if its of dimension  $d$ , it has  $2^d$  elements. If  $F$  is a free family of  $V$ , then  $v = \sum_{f \in F} f$  is also in  $V$  since it is a vector space. Thus the set  $F \cup \{v\}$  is a circuit of  $V$ . Conversely, if an element of a circuit is removed, one obtains a free family, where elements sum up to  $v$  over  $\mathbb{F}_2$ . We say that  $F$  and  $F \cup \{v\}$  are associated.

The strategy to enumerate the circuits containing the fixed set  $A$  is thus to enumerate the free families containing  $A$  by means of Algorithm 1. But it is not enough to generate every solution, the algorithm must also avoid the repetitions.

A circuit  $C$  is generated by the free families  $C \setminus \{v_i\}$  for all  $v_i \in C$ . One must choose among these families to output the circuit  $C$  only once. In the next algorithm, one chooses  $C \setminus \{v_i\}$  such that  $i > j$  for all  $v_j \in C \setminus \{v_i\}$ . The procedures FIRST\_FREE\_FAMILY and NEXT\_FREE\_FAMILY are given by Algorithm 1 and do what their name suggest.

---

**Algorithm 2:** Enumeration of the circuits of a vector subspace
 

---

**Data:** A vector subspace  $V = \{v_1, \dots, v_n\}$  and an integer  $k$

**Result:** The circuits of  $V$  containing  $\{v_1, \dots, v_k\}$

**begin**

$F \leftarrow \text{FIRST\_FREE\_FAMILY}(V, k)$

**while**  $F \neq \emptyset$  **do**

**if**  $v = \sum_{f \in F} f$  has a biggest index than the elements of  $F$  **then**

└ **Output**( $F \cup \{v\}$ )

$F \leftarrow \text{NEXT\_FREE\_FAMILY}(V, k)$

**end**

---

**Proposition 2.49.** *Algorithm 2 solves the problem ENUM·A-CIRCUIT restricted to vector subspaces over  $\mathbb{F}_2$  with incremental delay and polynomial space.*

*Proof.* Algorithm 2 follows the proposed strategy. Let  $F_1, \dots, F_l$  be the successive values taken by  $F$  during the run of Algorithm 2. It is the list of free families which contains  $A$ .

Let  $C$  be a circuit containing  $A$ . Remark that, since  $|C| \leq n$ , by a previous remark, it is associated to at most  $n$  free families containing  $A$ . Moreover, we output a circuit  $C$  at step  $i$  when the sum of elements of  $F_i$  is of index larger than the elements of  $F_i$ . Because of the order on free families induced by Algorithm 1, the other free families associated to  $C$  appear later in the enumeration.

Thanks to these properties, we can prove that the delay of the algorithm is incremental. Assume that Algorithm 2 has just produced the  $k^{\text{th}}$  solution and that it was generated from  $F_i$ . The set of free families  $\{F_1, \dots, F_{(k+1)n}\}$  is of size  $(k+1)n$ , thus there must be at least  $k+1$  different circuits associated to these families. Since Algorithm 2 outputs a circuit the first time it encounters an associated free family, when it encounters  $F_{(k+1)n}$  it must have produced at least  $k+1$  solutions.

Finally, the delay between the  $k^{\text{th}}$  and the  $(k+1)^{\text{th}}$  solution is bounded by the time to generate  $\{F_1, \dots, F_{(k+1)n}\}$  which is polynomial in  $n$  and  $k$ . We have thus proved that ENUM·A-CIRCUIT is in **IncP** when we restrict the input matroids to the class of vector subspaces over  $\mathbb{F}_2$ .  $\square$

This result may seem disappointing since we already know that **ENUM·A·CIRCUIT** for any representable matroids and  $|A| = 1$  is in **IncP**. We have generalized it to every size of  $|A|$  but for very few representable matroids.

First, we could easily extend the result to  $V$  a vector subspace of dimension  $d$  over any finite field  $\mathbb{F}$  of size  $k$ . Given a free family  $F$ , there are  $(k-1)^{|F|}$  vectors  $v$  such that  $F \cup \{v\}$  is dependent. It is bounded by  $k^n$  which is the size of  $V$ . Therefore one can find all  $v$  such that  $F \cup \{v\}$  is a circuit in polynomial time. Finally, one decides which circuits to output by a condition on the order of  $v$  relatively to  $F$  to avoid repetitions as in the previous algorithm.

Second, if we are willing to use an exponential space, we can improve the delay of the algorithm. This gives an example of a space-time tradeoff, obtained by amortizing the delay.

**Proposition 2.50.** *The problem **ENUM·A·CIRCUIT** restricted to vector subspaces over  $\mathbb{F}_2$  is in **DelayP**.*

*Proof.* One uses a queue, in which one stores the solutions produced by Algorithm 2. More precisely: run Algorithm 2, each time it outputs a solution push it into the queue. Every  $n$  free families built by Algorithm 2, pop the first element out of the queue to output it. When Algorithm 2 stops, output the remaining elements of the queue.

We have seen that, when Algorithm 2 has computed  $kn$  free families, it has produced at least  $k$  solutions. This proves that the queue is never empty before Algorithm 2 stops. Therefore the delay is bounded by  $n$  times the delay between the construction of two free families, which is polynomial in  $n$ .  $\square$

**Remark 2.51.** Assume you have an enumeration algorithm in polynomial delay but each solution is repeated in the enumeration. On the other hand, the number of repetitions is bounded by a polynomial in the input. If one uses the same technique as in the previous proposition, with a binary search tree instead of a queue, the algorithm would become a polynomial delay one. Here we have a way to distinguish the first time we output a solution, therefore we only need a queue, but in both cases we use potentially an exponential space.

Now that we have a result for **ENUM·A·CIRCUIT** for vector spaces, we want to generalize it to representable matroids built from vector spaces. For instance, **ENUM·A·CIRCUIT** for direct sum of vector spaces is in **DelayP**, we just do the enumeration independently in every vector space. In particular, if  $E_1$  and  $E_2$  are subspaces of the vector space  $E$  and  $E_1 \cap E_2 = \{0\}$ , then we can solve **ENUM·A·CIRCUIT** on  $E_1 \cup E_2$ . The question is: what happens if  $E_1 \cap E_2 \neq \{0\}$ ? In the following proposition, we try to characterize the circuits of  $E_1 \cup E_2$  to answer this question.

**Proposition 2.52.** *Let  $E_1$  and  $E_2$  be two vector spaces and let  $B = E_1 \cap E_2$ . The circuits of the matroid represented by  $E_1 \cup E_2$  are of the following types:*



1. a circuit of  $E_1$
2. a circuit of  $E_2$
3. equal to  $C_1 \cup C_2$  and there is a  $b \in B$  such that  $C_1 \cup \{b\}$  is a circuit of  $E_1$  and  $C_2 \cup \{b\}$  is a circuit of  $E_2$
4. equal to  $C_1 \cup C_2 \cup C_3$  and there are  $b_1, b_2 \in B$  such that  $C_1 \cup \{b_1\}$  is a circuit of  $E_1$ ,  $C_2 \cup \{b_2\}$  is a circuit of  $E_2$  and  $\{b_1, b_2\} \cup C_3$  is a circuit of  $B$ .

*Proof.* Let  $C$  be a circuit of  $E_1 \cup E_2$ . We partition  $C$  into  $C_1 = C \cap (E_1 \setminus B)$ ,  $C_2 = C \cap (E_2 \setminus B)$  and  $C_3 = C \cap B$ . There is a linear combination of the elements of  $C$  equal to 0 and by grouping the elements according to the partition, we have  $c_1 + c_2 + c_3 = 0$  where  $c_i \in C_i$ . Notice that for  $i = 1, 2, 3$ ,  $c_i$  can be equal to zero but only if  $C_i$  is empty, because  $C$  is a circuit. By definition,  $c_3$  is in  $B$  and  $c_2$  is in  $E_2$ . Therefore  $c_2 + c_3$  is also in  $E_2 = E_2 \cup B$ . Since  $c_1 = -c_2 - c_3$ , we have  $c_1 \in E_2$  which implies that  $c_1 \in B$ . By symmetry,  $c_2$  is also in  $B$ .

If  $c_1 = 0$ , then  $C_1$  is empty and  $C$  is a circuit of  $E_1$ . We also have that if  $c_2 = 0$ , then  $C_2$  is empty and  $C$  is a circuit of  $E_2$ . If we assume that  $c_3 = 0$  and that  $c_1 \neq 0$ , then  $c_1 = -c_2$  and because of the previous remark,  $c_1 \in B$ . Therefore there are linear combinations of elements of  $C_1 \cup \{c_1\}$  and  $C_2 \cup \{c_2\}$  which are equal to zero. These two sets have thus to be circuits, because if it was not the case it would contradict the fact that  $C$  is a circuit.

Let now consider the case where  $c_1, c_2, c_3$  are all different from zero. Remark that  $C_3 \cup \{c_1, c_2\}$  is a circuit, because we have a linear combination of its elements equal to 0 and that if it is not minimal, then  $C$  is not a circuit. Moreover, we have proved that  $c_1, c_2 \in B$  hence  $C_3 \cup \{c_1, c_2\}$  is a circuit of  $B$ . For obvious reasons,  $C_1 \cup \{c_1\}$  is a circuit of  $E_1$  and  $C_2 \cup \{c_2\}$  is a circuit of  $E_2$ , which achieves the proof.  $\square$

**Corollary 2.53.** *Let  $M$  be a  $\mathbb{F}_2$ -matroid represented by  $E_1 \cup E_2$  such that  $E_1 \cap E_2$  is of dimension 1 (equal to the space  $\{0, b\}$ ). The circuits of  $M$  are the disjoint union of the following sets:*

1. the circuits of  $E_1$
2. the circuits of  $E_2$
3. the circuits  $C_1 \cup C_2$  such that  $C_1 \cup \{b\}$  is a circuit of  $E_1$  and  $C_2 \cup \{b\}$  is a circuit of  $E_2$

*Proof.* Since  $B$  is of dimension 1, the fourth case of Proposition 2.52 never happens. Therefore a circuit is one of the three first types of Proposition 2.52.

Conversely a circuit of  $E_1$  or  $E_2$  is always a circuit of  $E_1 \cup E_2$ . Assume we have  $C_1 \cup \{b\}$  a circuit of  $E_1$  and  $C_2 \cup \{b\}$  a circuit of  $E_2$ . The sum of elements of  $C_1$  and the sum of elements of  $C_2$  are equal to  $b$ . Therefore the sum of elements of

$C_1 \cup C_2$  is equal to 0. Moreover, if the sum of elements of a strict subset of  $C_1 \cup C_2$  is equal to 0, then a strict subset of either  $C_1$  or  $C_2$  has its sum of elements equal to  $b$ . It contradicts the fact that  $C_1 \cup \{b\}$  is a circuit of  $E_1$  or that  $C_2 \cup \{b\}$  is a circuit of  $E_2$ .  $\square$

We have proved here that  $E_1 \cup E_2$  represents the parallel connection of  $E_1$  and  $E_2$  along the element  $b$ . Indeed, the previous corollary is basically the definition of the parallel connection that one finds in Chapter 6 with applications to matroid decomposition.

**Proposition 2.54.** *The problem  $\text{ENUM}\cdot A\text{-Circuit}$  is in  $\mathbf{DelayP}$  over  $\mathbb{F}_2$ -matroids of the form  $E_1 \cup E_2$ , where  $E_1$  and  $E_2$  are vector spaces and  $\dim E_1 \cap E_2 = 1$ .*

*Proof.* Let  $A \subseteq E_1 \cup E_2$ , we want to enumerate the set of circuits which extend  $A$ . Let  $0$  and  $b$  be the two elements of  $E_1 \cap E_2$  and let  $A_1 = A \cap E_1$ ,  $A_2 = A \cap E_2$ . The algorithm to enumerate the circuits of  $E_1 \cup E_2$  does these three successive steps:

1. it enumerates the circuits of  $E_1$  which contains  $A$
2. it enumerates the circuits of  $E_2$  which contains  $A$
3. if  $b \notin A$ , it enumerates the circuits of  $E_1$  which contain  $A_1 \cup \{b\}$  and for each such circuit  $C_1 \cup \{b\}$ , it enumerates the circuits  $C_2 \cup \{b\}$  of  $E_2$  which contain  $A_2 \cup \{b\}$  and return  $C_1 \cup C_2$

Because of the previous proposition, each circuit is generated in one of the three steps. Moreover, the sets of circuits enumerated in each of the three steps are disjoint, hence there is no repetition. Since the enumeration of circuits in a vector space can be done in polynomial delay, this algorithm has the same polynomial delay.  $\square$

**Open question:** is  $\text{ENUM}\cdot A\text{-Circuit}$  in  $\mathbf{DelayP}$  over the spaces of the form  $E_1 \cup E_2$  such that  $\dim(E_1 \cap E_2) = k$ ? If true, one can even imagine a decomposition of any vector matroid into union of vector spaces.

## Chapter 3

# Enumeration of Monomials

### 3.1 Introduction

In this chapter, we revisit the famous problem of polynomial interpolation, that is to say finding the monomials of a polynomial from its values, with an emphasis on the delay of the enumeration. The question of polynomial interpolation is a good framework for studying enumeration problems since it subsumes many interesting questions. To solve this problem we have to introduce probabilistic enumeration and that yields new interesting questions on probabilistic complexity classes.

It has long been known that a finite number of evaluation points is enough to interpolate a polynomial and efficient procedures (both deterministic and probabilistic) have been studied by several authors [BO88, Zip90, KLL00, GS09]. The complexity depends mostly on the number of monomials of the polynomial and on an a priori bound on this number which may be exponential in the number of variables. The deterministic methods rely on prime numbers as evaluation points, with the drawback that they may be very large. The probabilistic methods crucially use the Schwarz-Zippel lemma, which is also a tool in this chapter, and efficient solving of particular linear systems.

As a consequence of a result about random efficient identity testing [KS01], Klivans and Spielman give an interpolation algorithm, which happens to have an incremental delay. In this vein, the present chapter studies the problem of generating the monomials of a polynomial with the best possible delay. In particular we consider natural classes of polynomials such as multilinear polynomials, for which we prove that interpolation can be done with polynomial delay. Similar restrictions have been studied in other works about POLYNOMIAL IDENTITY TESTING for a quantum model [AMAM09], for depth 3 circuits which thus define almost linear polynomials [KS08] and for multilinear polynomials defined by a depth 4 circuit [KMSV10] (for a survey on this problem see [Sax09]). Moreover, a lot of interesting polynomials are multilinear like the Determinant, the Pfaffian, the Permanent, the elementary symmetric polynomials or anything which may be defined by a syntactically multilinear arithmetic circuit.

In Section 3.3 we present an algorithm which works for polynomials such that no two of their monomials use the same set of variables. It is structured as in [KS01] but is simpler and has better delay, though polynomially related. In Section 3.4 we propose a second algorithm which works for multilinear polynomials. It has a delay polynomial in the number of variables, which makes it exponentially better than known algorithms so far. It is also easily parallelizable. In addition, both algorithms enjoy a global complexity as good or better than the algorithms of the literature and use only small evaluation points making them suitable to work over finite fields.

In the first sections, we give algorithms in the black box model (presented in Chapter 1). They are efficient, but they do not take into account the cost of evaluations of the polynomial (considered to be in unit time). In Section 3.5, we consider polynomials which can be evaluated in polynomial time. Over these polynomials, the different interpolation algorithms correspond to three complexity classes for enumeration, namely **TotalPP**, **IncPP**, **DelayPP** which are probabilistic variants of the classes introduced in Chapter 2. We use our probabilistic algorithms on polynomials encoding well-known combinatorial problems in their monomials. It proves that some problems such as the enumeration of the spanning trees of a graph are in the classes we have just defined. This idea allows us also to propose new algorithms for two hypergraph problems in Chapter 4.

We generalize the techniques of the previous sections in Section 3.6 to build an incremental interpolation algorithm for fixed degree polynomials. We give the idea of the algorithm of [KS01] for comparison and we explain why it is good for high degree polynomials whereas our algorithm is better for low degree. We also compare our two algorithms with several classical ones and show that they are good with regard to parameters like number of calls to the black box or size of the evaluation points (see Table 3.6.3).

In Section 3.7 we propose several methods to improve the complexity of the previous algorithms. They are related to different parameters of the problem, namely the degree of the polynomial, the field and the error bound. We also explain how to derandomize our algorithms on a class of polynomials as soon as one can derandomize the POLYNOMIAL IDENTITY TESTING problem on the same class.

In Section 3.8 we present four easy to compute polynomials which encode hard combinatorial questions in their monomials. Thanks to them, we prove that some decision and search problems are hard to solve, even for low degree polynomials. For instance the problem of finding a specified monomial in a degree 2 polynomial is proved hard by encoding a restricted version of the Hamiltonian path problem in a polynomial given by the Matrix-Tree theorem (see [Aig07]). As a consequence, there is no polynomial delay interpolation algorithm for degree 2 polynomials similar to the one for degree 1 because it would solve the latter problem and would imply  $\text{RP} = \text{NP}$ .

A part of this chapter has been presented at MFCS 2010 [Str10].

### 3.1.1 Preliminaries

In this chapter we interpret the problem of interpolating a polynomial given by a black box as an enumeration problem. It means that we try to find all the monomials of a polynomial given by the number of its variables and an oracle which allows to evaluate the polynomial on any point in unit time. This problem is denoted by ENUM·POLY but will be solved in this chapter only on restricted classes of polynomials.

Most of the algorithms we are going to consider are probabilistic. We denote by  $\epsilon$  their error bound, where  $\epsilon$  is a small positive real. Since  $\epsilon$  is sometimes given as input, it can be thought as a small rational.

In all this chapter, the multivariate polynomials are taken in  $\mathbb{Q}[X_1, \dots, X_n]$  and they have a degree  $d$  and a total degree  $D$ . In Sec. 3.4 we assume that the polynomial is multilinear i.e.  $d = 1$  and  $D$  is thus bounded by  $n$ . In most of this chapter  $d$  is considered to be a fixed constant and as such it almost never appears in the complexity of the algorithms.

We assume that the maximum of the bitsize of the coefficients appearing in a polynomial is  $O(n)$  to simplify the statement of some results. In the examples of Sec. 3.5 it is even  $O(1)$ . When analyzing the delay of an algorithm solving ENUM·POLY, we are interested in both the number of calls to the black box and the time spent between two generated monomials. We are also interested in the size of the integers used in the calls to the oracle, since in real cases the complexity of the evaluation depends on it.

**Definition 3.1.** The support of a monomial is the set of indices of variables which appear in the monomial.

Let  $L$  be a set of indices of variables, then  $f_L$  is the homomorphism of  $\mathbb{Q}[X_1, \dots, X_n]$  defined by:

$$\begin{cases} f_L(X_i) = X_i & \text{if } i \in L \\ f_L(X_i) = 0 & \text{otherwise} \end{cases}$$

From now on, we denote  $f_L(P)$  by  $P_L$ . It is the polynomial obtained by substituting 0 to every variable of index not in  $L$ , that is to say all the monomials of  $P$  which have their support in  $L$ . We call  $\vec{X}^L$  the multilinear term of support  $L$ , which is the product of all  $X_i$  with  $i$  in  $L$ .

**Example 3.2.** Let  $P$  be the polynomial  $2X_1^2 - X_4X_5X_6^3 + 3X_1X_3X_7 + X_2^2X_1$ . The support of  $3X_1X_3X_7$  is  $L = \{1, 3, 7\}$  and  $P_L = 2X_1^2 + 3X_1X_3X_7$ .

## 3.2 Finding one monomial at a time

The first problem we want to solve is to decide if a polynomial given by a black box is the zero polynomial a.k.a. POLYNOMIAL IDENTITY TESTING. We are especially interested in the corresponding search problem, i.e. giving explicitly one term and

its coefficient. Indeed, we show in Sec. 3.3 how to turn any algorithm solving this problem into an incremental interpolation algorithm.

For any  $t$  evaluation points, one may build a polynomial with  $t$  monomials which vanishes at every point (see [Zip90]). Therefore, if we do not have any a priori bound on  $t$ , we must evaluate the polynomial on at least  $(d+1)^n$   $n$ -tuples of integers to determine it. Since such an exponential complexity is not satisfying, we introduce probabilistic algorithms, which nonetheless have a good and manageable bound on the error.

**Lemma 3.3** (Schwarz-Zippel [Sch80]). *Let  $P$  be a non zero polynomial with  $n$  variables of total degree  $D$ , if  $x_1, \dots, x_n$  are randomly chosen in a set of integers  $S$  of size  $\frac{D}{\epsilon}$  then the probability that  $P(x_1, \dots, x_n) = 0$  is bounded by  $\epsilon$ .*

A classical probabilistic algorithm to decide if a polynomial  $P$  of total degree bounded by  $D$  is identically zero can be derived from this lemma.

---

**Algorithm 3:** The procedure *not\_zero*( $P, \epsilon$ )

---

**Data:** A polynomial  $P$  with  $n$  variables, its total degree  $D$  and the error bound  $\epsilon$

```

begin
  pick  $x_1, \dots, x_n$  randomly in  $[\frac{D}{\epsilon}]$ 
  if  $P(x_1, \dots, x_n) = 0$  then
    | return False
  else
    | return True
end

```

---

Remark that the algorithm never gives a false answer when the polynomial is zero. The probability of error when the polynomial is non zero is bounded by  $\epsilon$  thanks to Lemma 3.3. The problem POLYNOMIAL IDENTITY TESTING for explicit polynomials computable in polynomial time is thus in RP.

This procedure makes exactly one call to the black box on points of size  $\log(\frac{D}{\epsilon})$ . The error rate may then be made exponentially smaller by increasing the size of the points. There is an other way to achieve the same reduction of error. Repeat the previous algorithm  $k$  times for  $\epsilon = \frac{1}{2}$ , that is to say the points are randomly chosen in  $[2D]$ . If all runs return zero, then the algorithm decides that the polynomial is zero else it decides it is not zero. Since the random choices are independent in each run, the probability of error of this algorithm is bounded by  $2^{-k}$ . Hence, to achieve an error bound of  $\epsilon$ , we have to set  $k = \log(\epsilon^{-1})$ . We always use this latter variant in this chapter and we denote it also by *not\_zero*( $P, \epsilon$ ). It uses slightly more random bits but it only involves numbers less than  $2D$ .

To solve the search problem we use the following lemma.

**Lemma 3.4.** *Let  $P$  be a polynomial without constant term and whose monomials have distinct supports and  $L$  a minimal set (for inclusion) of variables such that*

$P_L$  is not identically zero. Then  $P_L$  is a monomial of support  $L$ .

In order to use the lemma, up to Section 3.4 all polynomials have monomials with distinct supports and no constant term. This class of polynomials contains the multilinear polynomials but is much larger. Moreover being without constant term is not restrictive since we can always replace a polynomial by the same polynomial minus its constant term that we compute beforehand by a single oracle call to  $P(0, \dots, 0)$ .

### 3.2.1 The algorithm

We now give an algorithm which finds, in randomized polynomial time, a monomial of a polynomial  $P$ , thanks to Lemma 3.4 and the procedure *not\_zero*. In this algorithm,  $L$  is a set of indices of variables and  $i$  an integer used to denote the index of the current variable.

---

**Algorithm 4:** The procedure *find\_monomial*( $P, \epsilon$ )

---

**Data:** A polynomial  $P$  with  $n$  variables, its total degree  $D$  and the error bound  $\epsilon$

**Result:** A monomial of  $P$

**begin**

$L \leftarrow \{1, \dots, n\}$

**if** *not\_zero*( $P, \frac{\epsilon}{n+1}$ ) **then**

**for**  $i = 1$  **to**  $n$  **do**

**if** *not\_zero*( $P_{L \setminus \{i\}}, \frac{\epsilon}{n+1}$ ) **then**

$L \leftarrow L \setminus \{i\}$

**return** *The monomial of support*  $L$

**else**

**return** “Zero”

**end**

---

Once the algorithm has found the support  $L$  of a monomial, that we write  $\lambda \vec{X}^{\vec{e}}$ , it must find  $\lambda$  and  $\vec{e}$ . The evaluation of  $P_L$  on  $(1, \dots, 1)$  returns  $\lambda$ . For each  $i \in L$ , the evaluation of  $P_L$  on  $X_i = 2$  and for  $j \neq i$ ,  $X_j = 1$  returns  $\lambda 2^{e_i}$ . From these  $n$  calls to the black box, one can find  $\vec{e}$  in linear time and thus output  $\lambda \vec{X}^{\vec{e}}$ .

We analyze this algorithm, assuming first that the procedure *not\_zero* never makes a mistake and that  $P$  is not the zero polynomial. In this case, the algorithm does not answer “Zero” at the beginning. Therefore  $P_L$  is not zero at the end of the algorithm, because an element is removed from  $L$  only if this condition is respected. Since removing any other element from  $L$  would make  $P_L$  zero by construction, the set  $L$  is minimal for the property of  $P_L$  being non zero. Then by Lemma 3.4 we know that  $P_L$  is a monomial of  $P$ , which allows us to output it as previously explained.

Errors only appear in the procedure *not\_zero* with probability  $\frac{\epsilon}{n+1}$ . Since we use this procedure  $n + 1$  times we can bound the total probability of error by  $\epsilon$ . The total complexity of this algorithm is  $O(n \log(\frac{n}{\epsilon}))$  since each of the  $n$  calls to the procedure *not\_zero* makes  $O(\log(\frac{n}{\epsilon}))$  calls to the oracle in time  $O(1)$ . We summarize the properties of this algorithm in the next proposition.

**Proposition 3.5.** *Given a polynomial  $P$  as a black box, whose monomials have distinct supports, Algorithm 4 finds, with probability  $1 - \epsilon$ , a monomial of  $P$  by making  $O(n \log(\frac{n}{\epsilon}))$  calls to the black box on entries of size  $\log(2D)$ .*

### 3.3 An incremental algorithm for polynomials with monomials of distinct supports

We build an algorithm which enumerates the monomials of a polynomial incrementally once we know how to find a monomial of a polynomial in polynomial time. The idea is to subtract the monomial to the polynomial and recurse. The procedure *find\_monomial* defined in Proposition 3.5 is used to find the monomial.

We need a procedure *subtract*( $P, Q$ ) which acts as a black box for the polynomial  $P - Q$  when  $P$  is given as a black box and  $Q$  as an explicit set of monomials. Let  $D$  be the total degree of  $Q$ ,  $C$  is a bound on the size of its coefficients and  $i$  is the number of its monomials. One evaluates the polynomial *subtract*( $P, Q$ ) on points of size  $m$  as follows:

1. compute the value of each monomial of  $Q$  in time  $O(D \max(C, m))$
2. add the values of the  $i$  monomials in time  $O(iD \max(C, m))$
3. call the black box to compute  $P$  on the same points and return this value minus the one we have computed for  $Q$

---

#### Algorithm 5: Incremental computation of the monomials of $P$

---

**Data:** A polynomial  $P$  with  $n$  variables and the error bound  $\epsilon$

**Result:** The set of monomials of  $P$

```

begin
   $Q \leftarrow 0$ 
  while not_zero(subtract( $P, Q$ ),  $\frac{\epsilon}{2^{n+1}}$ ) do
     $M \leftarrow$  find_monomial(subtract( $P, Q$ ),  $\frac{\epsilon}{2^{n+1}}$ )
    Output( $M$ )
     $Q \leftarrow Q + M$ 
end

```

---

**Theorem 3.6.** *Let  $P$  be a polynomial whose monomials have distinct supports with  $n$  variables,  $t$  monomials and a total degree  $D$ . Algorithm 5 computes the set*



of monomials of  $P$  with probability  $1 - \epsilon$ . The delay between the  $i^{\text{th}}$  and  $i + 1^{\text{th}}$  monomials is bounded by  $O(iDn^2(n + \log(\epsilon^{-1})))$  in time and  $O(n(n + \log(\epsilon^{-1})))$  calls to the oracle. The algorithm performs  $O(tn(n + \log(\epsilon^{-1})))$  calls to the oracle on points of size  $\log(2D)$ .

*Proof. Correctness.* We analyze this algorithm under the assumption that the procedures *not\_zero* and *find\_monomial* do not make mistakes.

We have the following invariant of the while loop :  $Q$  is a subset of the monomials of  $P$ . It is true at the beginning because  $Q$  is zero. Assume that  $Q$  satisfies this property at a certain point of the while loop, since we know that *not\_zero(subtract(P,Q))*,  $P - Q$  is not zero and is then a non empty subset of the monomials of  $P$ . The outcome of *find\_monomial(subtract(P,Q))* is thus a monomial of  $P$  which is not in  $Q$ , therefore  $Q$  plus this monomial still satisfies the invariant. Remark that we have also proved that the number of monomials of  $Q$  is increasing by one at each step of the while loop. The algorithm must then terminate after  $t$  steps and when it does *not\_zero(subtract(P,Q))* gives a negative answer meaning that  $Q = P$ .

**Probability of error.** The probability of failure is bounded by the sum of the probabilities of error coming from *not\_zero* and *find\_monomial*. These procedures are both called  $t$  times with an error bounded by  $\frac{\epsilon}{2^{n+1}}$ . There are  $2^n$  different supports, hence there is at most  $2^n$  monomials in  $P$ . The total probability of error is bounded by  $\frac{2t}{2^{n+1}}\epsilon \leq \epsilon$ .

**Complexity.** The procedure *not\_zero* is called  $t$  times and uses the oracle  $n + \log(\epsilon^{-1})$  times, whereas *find\_monomial* is called  $t$  times but uses  $n(n + \log(\epsilon^{-1}))$  oracle calls, which adds up to  $t(n + 1)(n + \log(\epsilon^{-1}))$  calls to the oracle. In both cases the evaluation points are of size  $O(\log(D))$ .

The delay between two solutions is the time to execute *find\_monomial*. It is dominated by the execution of *subtract(P, Q)* at each oracle call on points of size  $\log(2D)$ . The algorithm calls *subtract(P, Q)*  $n(n + \log(\epsilon^{-1}))$  times and each of these calls needs a time  $O(iD \max(C, \log(D)))$ . We assume here that  $d = O(2^n)$  and  $C = O(n)$  thus  $\max(C, \log(D)) = O(n)$ . Hence the delay is  $O(iDn^2(n + \log(\epsilon^{-1})))$ .  $\square$

The complexity could be improved by using a smarter way to evaluate  $Q$  in *subtract(P, Q)*, like fast or iterated multiplication. One other possible improvement would be to do all computations modulo a random prime as it is done in the IP protocol for the permanent [LFK92]. The idea of using a small finite field is developed in Sec. 3.7.

### 3.4 A polynomial delay algorithm for multilinear polynomials

In this section we first solve the problem of finding the degree of a polynomial with regard to a set of variables.

**Definition 3.7.** Let  $n$  be an integer and  $S \subseteq [n]$ . Let  $d_S(\vec{X}^{\vec{e}}) = \sum_{i \in S} e_i$  be the degree of the term  $\vec{X}^{\vec{e}}$  with regard to  $S$ . We write  $d_S(P)$  for the degree of a polynomial  $P$  with regard to  $S$ , it is the maximum of the degrees of its monomials with regard to  $S$ .

Remark that  $d_{\{i\}}(P)$  is the degree of  $X_i$  in  $P$ , while  $d_{[n]}(P)$  is the total degree of  $P$ . We present a method to efficiently find the degree of a polynomial with regard to any set of variables. It transforms the multivariate polynomial into an univariate one, which is then interpolated. To achieve this, one uses a polynomial number of calls to the black box on small points. As a corollary, one gives an efficient algorithm for the following problem, when the polynomial is multilinear.

MONOMIAL-FACTOR

*Input:* a polynomial given as a black box and a term  $\vec{X}^{\vec{e}}$

*Output:* accept if  $\vec{X}^{\vec{e}}$  divides a monomial in the polynomial

We also give a second algorithm, which solves this problem with only one call to the black box but which uses an exponentially larger point in the call and a third one specialized to circuits. We then design an algorithm, which enumerates the monomials of a multilinear polynomial with polynomial delay. This algorithm has the interesting property of being easily parallelizable, which is obviously not the case of the incremental one.

### 3.4.1 Small values

Let  $P(\vec{X})$  be a  $n$  variables polynomial over  $\mathbb{Q}$ , and let  $(S_1, S_2)$  be a partition of  $[n]$ . We can see  $P$  as a polynomial with variables  $(X_i)_{i \in S_1}$  over the ring  $\mathbb{Q}[(X_i)_{i \in S_2}]$ . In fact, the total degree of  $P$  as a polynomial over this ring is equal to  $d_{S_1}(P)$ . We introduce a new variable  $X_{n+1}$ , the polynomial  $\tilde{P}$  is the polynomial  $P$  where  $X_{n+1} \cdot X_i$  is substituted to  $X_i$  when  $i \in S_1$ . We have the equality  $d_{\{n+1\}}(\tilde{P}) = d_{S_1}(P)$ .

**Proposition 3.8.** *Let  $P(\vec{X})$  be a non zero polynomial with  $n$  variables, a total degree  $D$  and let  $S$  be a subset of  $[n]$ . There is an algorithm which computes  $d_S(P)$  with probability greater than  $\frac{2}{3}$  in time polynomial in  $n$ ,  $D$  and the size of the coefficients of  $P$ .*

*Proof.* We define the polynomial  $\tilde{P}$  with  $n+1$  variables from  $P$  and  $S$ , as explained previously. It is equal to  $\sum_{i=0}^d X_{n+1}^i Q_i(\vec{X})$  where  $Q_d$  is a non zero polynomial.

Here  $d$  is both  $d_S(P)$  and the degree of  $\tilde{P}$  seen as a univariate polynomial over  $\mathbb{Q}[X_1, \dots, X_n]$ .

Now choose randomly in  $[3D]$  a value  $x_i$  for each  $X_i$  such that  $i \neq n+1$ . The polynomial  $\tilde{P}(x_1, \dots, x_n, X_{n+1})$  is a univariate polynomial, and the coefficient of  $X_{n+1}^d$  is  $Q_d(\vec{x})$ . By Lemma 3.3, the probability that  $Q_d(\vec{x})$  is zero is bounded by  $\frac{1}{3}$ .

The polynomial  $\tilde{P}(x_1, \dots, x_n, X_{n+1})$  can be interpolated from its value on the integers  $1, \dots, D$ , because it is of degree less or equal to  $d_S(P) \leq D$ . The value of  $\tilde{P}(x_1, \dots, x_n, x_{n+1})$  is equal to  $P(x'_1, \dots, x'_n)$  where  $x'_i = x_{n+1}x_i$  if  $i \in S$  and  $x_i$  otherwise. The time to interpolate  $\tilde{P}(x_1, \dots, x_n, X_{n+1})$  with  $s$  a bound on the size of  $\tilde{P}(x_1, \dots, x_n, x_{n+1})$  for  $0 \leq x_{n+1} \leq D$  is  $O(D^2 \log(s))$ . Remark that  $s$  is polynomial in the size of the coefficients of  $P$  and in  $D$ .

Finally, the interpolation of  $\tilde{P}(x_1, \dots, x_n, X_{n+1})$  gives its degree, which is equal to  $d_S(P)$  with probability greater than  $\frac{2}{3}$ .  $\square$

We now give a solution to the problem MONOMIAL-FACTOR for terms of the form  $\vec{X}^S$  in a multilinear polynomial as a corollary. In fact to obtain a better complexity, we do not use directly Proposition 3.8, but rather the idea of its proof.

**Corollary 3.9.** *Let  $P(\vec{X})$  be a multilinear polynomial with  $n$  variables, a total degree  $D$  and let  $S$  be a subset of  $[n]$ . There is an algorithm which solves the problem MONOMIAL-FACTOR on the polynomial  $P$  and the term  $\vec{X}^S$  with probability  $1 - \epsilon$ . It does  $|S| \log(\epsilon)$  calls to the black box on points of size  $\log |S|$  and uses a time  $O(\log(\epsilon)|S|^2(n + D \log(D)))$ .*

*Proof.* The polynomial  $P(\vec{X})$  can be written  $\vec{X}^S P_1(\vec{X}) + P_2(\vec{X})$ , where  $\vec{X}^S$  does not divide  $P_2(\vec{X})$ . Since  $P$  is multilinear, one of its monomials is divided by  $\vec{X}^S$  if and only if  $d_S(P) = |S|$ .

We do a simpler substitution than in the previous proof: one substitutes  $X_{n+1}$  to  $X_i$  when  $i \in S_1$ . Let us remark that  $P_1$  does not depend on variables of indices in  $S$ , since  $\vec{X}^S P_1$  is multilinear. Therefore  $P_1$  is not zero after the substitution if and only if it was not zero before substitution.

One then chooses randomly a value in  $[2D]$  for each  $X_i$  with  $i \notin S$  and interpolates the obtained univariate polynomial. The bound on the total degree of the polynomial is  $|S|$ , therefore we need  $|S|$  calls to the black box on points of size  $\log |S|$ . The time used is  $O(|S|^2 s)$ , where  $s$  is a bound on the size of the values of the univariate polynomial on  $1, \dots, |S|$ . The polynomial  $P$  has at most  $2^n$  monomials, its coefficients are of size  $O(n)$  and the value of a term on a point less than  $2D$  is less than  $(2D)^D$ , therefore  $s$  is  $O(n + D \log(D))$ .

Finally to bring the probability of error from  $\frac{1}{2}$  to  $\epsilon$  one repeat the procedure  $\log(\epsilon)$  times.  $\square$

We could generalize this corollary to terms  $(\vec{X}^S)^d$  in a polynomial of degree  $d$  without changing the proof. We will use this idea in Sec. 3.6.

### 3.4.2 Large values

We prove here directly a proposition similar to the corollary of the previous subsection. If needed, we could adapt it to give an algorithm which decides the degree of a polynomial with regard to a set.

Note that in this case we need an a priori bound on the coefficients of the polynomial, but it is not so demanding since in most applications those coefficients are constant. We also assume that the coefficients are in  $\mathbb{Z}$  to simplify the proof.

**Proposition 3.10.** *Let  $P(\vec{X})$  be a multilinear polynomial with  $n$  variables, a total degree  $D$ , its coefficients of size at most  $C$  and let  $S$  be a subset of  $[n]$ . There is an algorithm which solves the problem MONOMIAL-FACTOR on the polynomial  $P$  and the term  $\vec{X}^S$  with probability  $1 - \epsilon$ . It does one call to the black box on a point of size  $O(n + D \log(\frac{D}{\epsilon}))$  and uses a time  $O(|S|(n + D \log(\frac{D}{\epsilon})))$ .*

*Proof.* We write  $P = \vec{X}^S P_1(\vec{X}) + P_2(\vec{X})$  and we want to decide if  $P_1(\vec{X})$  is the zero polynomial. We let  $\alpha$  be the integer  $2^{2(n+C+D \log(\frac{2D}{\epsilon}))}$  and do a call to the oracle on the values  $(x_i)_{i \in [n]}$ :

$$\begin{cases} x_i = \alpha & \text{if } i \in S \\ x_i \text{ is randomly chosen in } [\frac{2D}{\epsilon}] & \text{otherwise} \end{cases}$$

The value of a variable whose index is not in  $S$  is bounded by  $\frac{2D}{\epsilon}$ , therefore a monomial of  $P_2$  (which contains at most  $|S| - 1$  variables of index in  $S$ ) has its contribution to  $P(x_1, \dots, x_n)$  bounded by  $2^C (\frac{2D}{\epsilon})^D \alpha^{|S|-1}$ . Since  $P_2$  has at most  $2^n$  monomials, its total contribution is bounded in absolute value by  $2^{n+C+D \log(\frac{2D}{\epsilon})} \alpha^{|S|-1}$  which is equal to  $\alpha^{|S|-\frac{1}{2}}$ . If  $P_1(x_1, \dots, x_n)$  is zero, this also bounds the absolute value of  $P(x_1, \dots, x_n)$ .

Assume now that  $P_1(x_1, \dots, x_n)$  is not zero, it is at least 1 since it is defined on  $\mathbb{Z}$  and evaluated on integers. Moreover  $\vec{x}^S$  is equal to  $\alpha^{|S|}$ , thus the absolute value of  $\vec{x}^S P_1(x_1, \dots, x_n)$  has  $\alpha^{|S|}$  for lower bound. By the triangle inequality

$$\begin{aligned} |P(x_1, \dots, x_n)| &> |\vec{x}^S P_1(x_1, \dots, x_n)| - |P_2(x_1, \dots, x_n)| \\ |P(x_1, \dots, x_n)| &> \alpha^{|S|} - \alpha^{|S|-\frac{1}{2}} > \alpha^{|S|-\frac{1}{2}} \end{aligned}$$

We can then decide if  $P_1(x_1, \dots, x_n)$  is zero by comparison of  $P(x_1, \dots, x_n)$  to  $\alpha^{|S|-\frac{1}{2}}$ . Remark that  $P_1(x_1, \dots, x_n)$  may be zero even if  $P_1$  is not zero. Nonetheless  $P_1$  only depends on variables which are in  $S$  and are thus randomly taken in  $[\frac{2D}{\epsilon}]$ . By Lemma 3.3, the probability that the polynomial  $P_1$  is not zero although  $P_1(x_1, \dots, x_n)$  has value zero is bounded by  $\epsilon$ .

The algorithm which solves MONOMIAL-FACTOR, only does one call to the oracle, then computes  $\alpha^{|S|}$  and compares it to the result in time linear in the size of  $\alpha^{|S|}$ .  $\square$

### 3.4.3 Circuits

In this section, we give a solution to the problem MONOMIAL-FACTOR in a different model. We assume here that a polynomial is given by a multilinear arithmetic circuit  $C$  of size  $s$ . We say that a circuit is multilinear<sup>1</sup> if the polynomial computed

<sup>1</sup>It is a semantic condition, which cannot be easily verified. One often works with circuits such that the polynomials computed at each of their node is multilinear. They are called syntactically multilinear.

at the output node is multilinear. We want to solve MONOMIAL-FACTOR with input  $C$  and a set  $S$  representing the term  $\vec{X}^S$ . Thanks to a transformation of  $C$  into its homogeneous components with regard to  $S$ , we obtain an arithmetic circuit which represents a polynomial different from zero if and only if the answer to MONOMIAL-FACTOR is positive.

**Proposition 3.11.** *Let  $C$  be a multilinear arithmetic circuit of size  $c$ , with total degree  $D$  and let  $S$  be a set of size  $s$ . There is an algorithm which solves the problem MONOMIAL-FACTOR with probability  $1 - \epsilon$  in time  $O(s^2 c \log(D\epsilon^{-1}))$ .*

*Proof.* For each vertex  $v$  of  $C$ , we write  $P_v^i$  for the sum of the monomials  $\vec{X}^{\vec{e}}$  computed by  $C$  at the vertex  $v$  such that  $d_S(\vec{X}^{\vec{e}}) = i$ . For each vertex  $v$ , we introduce the vertices  $v_0, \dots, v_s$  labeled as  $v$  is. We build a new circuit  $C'$  on these vertices (and some others) such that the polynomial computed by  $v_i$  is  $P_v^i$ . To do that we have to define how the vertices are connected according to their type (variable, constant,  $+$  or  $\times$ ).

- Let  $u$  be an input node. If it is labeled by a constant or a variable of index not in  $S$ , then  $u_0$  is labeled by this constant or variable. If it is a variable of index in  $S$ , then  $u_1$  is labeled by this variable. All other variables  $u_i$  are labeled by 0.
- Let  $u$  be a sum node with inputs  $v$  and  $w$ . We have  $P_u^i = P_v^i + P_w^i$ , therefore the circuit  $C'$  has the edges  $(u_i, v_i)$  and  $(u_i, w_i)$  for all  $i \leq s$ .
- Let  $u$  be a product node with inputs  $v$  and  $w$ . We have  $P_u^i = \sum_{j+k=i} P_v^j * P_w^k$ .

We need some new sum vertices to implement the sum above by a binary tree of length at most  $O(\log(s))$  and of size  $O(s)$ . We say that this binary tree has  $u_i^{(j,k)}$  as leaves for all  $(j, k)$  such that  $i = j + k$  and  $u_i$  as root. The vertices  $u_i^{(j,k)}$  are product vertices, while the vertices  $u_i$  are sum vertices. The circuit  $C'$  has the edges  $(u_i^{(j,k)}, v_k)$  and  $(u_i^{(j,k)}, w_j)$  for all  $i$  and all  $(j, k)$  such that  $i = j + k$ .

The circuit  $C'$  is of size  $O(s^2 c)$ . Let  $v$  be the output gate of  $C$  and let  $P$  be the polynomial computed at  $v$  by  $C$ . By an easy induction, we can prove that  $C'$  computes at  $v_s$  the polynomial  $P_v^s$ . Since  $P$  is multilinear, the polynomial  $P_v^s$  is different from zero if and only if  $\vec{X}^S$  divides a monomial of  $P$ .

Therefore, to decide the problem MONOMIAL-FACTOR, we must solve the problem POLYNOMIAL IDENTITY TESTING on a circuit of size  $O(s^2 c)$ . It can be done thanks to Lemma 3.3, by evaluation of  $C'$  on random points taken in  $[D\epsilon^{-1}]$ . The time complexity of this algorithm is  $O(s^2 c \log(D\epsilon^{-1}))$ .  $\square$

The construction which is given in the proof is used in the considerations about derandomization of Section 3.7. Moreover, it can be a more efficient way than the one of Corollary 3.9 to solve the problem MONOMIAL-FACTOR when the input is a circuit.

### 3.4.4 The algorithm

Let  $P$  be a multilinear polynomial with  $n$  variables and a total degree  $D$ . Let  $L_1$  and  $L_2$  be two disjoint sets of indices of variables, we want to determine if there is a monomial of  $P$ , whose support contains  $L_2$  and is contained in  $L_1 \cup L_2$ .

Let us consider the polynomial  $P_{L_1 \cup L_2}$ , its monomials are the monomials of  $P$  such that their supports are included in  $L_1 \cup L_2$ . Obviously  $P$  has a monomial whose support contains  $L_2$  and is contained in  $L_1 \cup L_2$  if and only if  $(P_{L_1 \cup L_2}, \vec{X}^{L_2}) \in \text{MONOMIAL-FACTOR}$ . Let us call *not\_zero\_improved*( $L_1, L_2, P, \epsilon$ ) the algorithm given by Corollary 3.9, which solves this question in polynomial time with probability  $1 - \epsilon$ .

We now describe a binary tree such that there is a bijection between the leaves of the tree and the monomials of  $P$ . The nodes of this tree are pairs of lists  $(L_1, L_2)$  such that there exists a monomial of support  $L$  in  $P$  with  $L_2 \subseteq L \subseteq L_1 \cup L_2$ . Consider a node labeled by  $(L_1, L_2)$ , we note  $i$  the smallest element of  $L_1$ , it has for left child  $(L_1 \setminus \{i\}, L_2)$  and for right child  $(L_1 \setminus \{i\}, L_2 \cup \{i\})$  if they exist. The root of this tree is  $([n], \emptyset)$  and the leaves are of the form  $(\emptyset, L_2)$ . A leaf  $(\emptyset, L_2)$  is in bijection with the monomial of support  $L_2$ .

To enumerate the monomials of  $P$ , Algorithm 6 does a depth first search in this tree using *not\_zero\_improved* and when it visits a leaf, it outputs the corresponding monomial thanks to the procedure *coefficient*( $P, L$ ) that we now describe. Let  $L$  be the support of a monomial, we want to find its coefficient. One uses the same procedure as in Corollary 3.9 (substitution, interpolation) and outputs the coefficient of the monomial of the highest degree. Indeed, after the substitution of Corollary 3.9, we obtain the univariate polynomial  $\tilde{P}_L$  whose monomial of degree  $|L|$  is the image of the monomial of support  $L$  in  $P$  and has thus the same coefficient.

---

**Algorithm 6:** A depth first search of the support of monomials of  $P$  (recursive description)

---

**Data:** A multilinear polynomial  $P$  with  $n$  variables and the error bound  $\epsilon$

**Result:** All monomials of  $P$

**begin**

    Monomial( $L_1, L_2, i$ ) =

**if**  $i = n + 1$  **then**

        | **Output**(coefficient( $P, L_2$ ))

**else**

        | **if** *not\_zero\_improved*( $L_1 \setminus \{i\}, L_2, P, \frac{\epsilon}{2^{n-n}}$ ) **then**

            | Monomial( $L_1 \setminus \{i\}, L_2, i + 1$ )

        | **if** *not\_zero\_improved*( $L_1 \setminus \{i\}, L_2 \cup \{i\}, P, \frac{\epsilon}{2^{n-n}}$ ) **then**

            | Monomial( $L_1 \setminus \{i\}, L_2 \cup \{i\}, i + 1$ )

    in Monomial( $[n], \emptyset, 1$ )

**end**

---

**Theorem 3.12.** *Let  $P$  be a multilinear polynomial with  $n$  variables,  $t$  monomials and total degree  $D$ . Algorithm 6 computes the set of monomials of  $P$  with probability  $1 - \epsilon$ . The delay between the  $i^{\text{th}}$  and  $i + 1^{\text{th}}$  monomials is bounded in time by  $O(D^2 n^2 \log(n)(n + \log(\epsilon^{-1})))$  and by  $O(nD(n + \log(\epsilon^{-1})))$  oracle calls. The whole algorithm performs  $O(tnD(n + \log(\epsilon^{-1})))$  calls to the oracle on points of size  $O(\log(D))$ .*

*Proof.* Between the visit of two leaves, the procedure *not\_zero\_improved* is called at most  $n$  times and the procedure *coefficient* once. Both of them have an equivalent complexity cost. By Corollary 3.9, we know that one call to *not\_zero\_improved* on a term of support of size at most  $D$  with an error parameter  $\frac{\epsilon}{n^{2^n}}$  uses a time  $O(D^2 n \log(n)(n + \log(\epsilon^{-1})))$  and  $O(D(n + \log(\epsilon^{-1})))$  calls to the oracle on points of size  $\log(D)$ .

Since we call the procedures *not\_zero\_improved* and *coefficient* less than  $nt$  times during the algorithm, the error is bounded by  $nt \frac{\epsilon}{n^{2^n}} < \epsilon$ .  $\square$

There is a possible trade-off in the way *not\_zero\_improved* and *coefficient* are implemented: if one knows a bound on the size of the coefficients of the polynomial, then the algorithm of Proposition 3.10 can be used. The number of calls in the algorithm is less than  $tn$  which is close to the optimal  $2t$ .

**Remark 3.13.** When a polynomial is monotone (coefficients all positive or all negative) and is evaluated on positive points, the result is zero if and only if it is the zero polynomial. Algorithms 5 and 6 may be modified to work deterministically for monotone polynomials. The term  $(n + \log(\epsilon^{-1}))$  in the time complexity and number of calls of both algorithms disappears, since there are no more repetitions of the procedures *not\_zero* or *not\_zero\_improved* to exponentially decrease the error. For more on derandomization, see Section 3.7.

### 3.5 Complexity classes for randomized enumeration

In this section the results about interpolation in the black box formalism are transposed into more classical complexity results. We are interested in enumeration problems defined by predicates  $A(x, y)$  such that, for each  $x$ , there is a polynomial  $P_x$  whose monomials are in bijection with  $A(x)$ . If  $P_x$  is efficiently computable, an interpolation algorithm gives an effective way of enumerating its monomials and thus to solve  $\text{ENUM}\cdot A$ .

**Example 3.14.** We associate to each graph  $G$  the determinant of its adjacency matrix  $M_G$ . The monomials of this multilinear polynomial are in bijection with the cycle covers of  $G$ . Hence the problem of enumerating the monomials of  $\det(M_G)$  is equivalent to enumerating the cycle covers of  $G$ .

The specialization of different interpolation algorithms to efficiently computable polynomials naturally correspond to the probabilistic counterparts of **TotalP**,



**IncP** and **DelayP**. We present several problems related to a polynomial like in Example 3.14 to illustrate how easily the interpolation methods described in this chapter produce enumeration algorithms for combinatorial problems.

In all the following definitions, we assume that a predicate which defines an enumeration problem is decidable in polynomial time. In other words, all defined classes are included in **EnumP**.

**Definition 3.15.** A problem  $\text{ENUM}\cdot A$  is computable in probabilistic polynomial total time, written **TotalPP**, if there is a polynomial  $Q(x, y)$  and a machine  $M$  which solves  $\text{ENUM}\cdot A$  with probability greater than  $\frac{2}{3}$  and satisfies for all  $x$ ,  $T(x, |M(x)|) < Q(|x|, |M(x)|)$ .

The class **TotalPP** is very similar to the class **BPP** for decision problems. For both classes, the choice of  $\frac{2}{3}$  is arbitrary, everything greater than  $\frac{1}{2}$  would do. To achieve this, repeat a polynomial number of times an algorithm working in total polynomial time and return the set of solutions generated in the majority of runs. The probability of error has decreased exponentially, to prove this classical result, one has to use Chernoff's bound, which is stated and proved in [AB09].

A refinement [KLL00] of Zippel's algorithm [Zip90] solves  $\text{ENUM}\cdot \text{POLY}$  in a time polynomial in the number of monomials. If one uses this algorithm to interpolate the polynomial of Example 3.14 associated to a graph  $G$ , one can output the cycle covers of  $G$ . Since the evaluation of a Determinant can be done in polynomial time, Zippel's algorithm proves that the enumeration of the cycle covers of a graph is in **TotalPP**.

If one knows the number of monomials of a polynomial (or if this number can be approximated within a polynomial factor), then both [KS01] and [GS09] provide a deterministic algorithm which solves  $\text{ENUM}\cdot \text{POLY}$ . While it seems to provide **TotalP** algorithms, even in the case of polynomials represented by circuits, the computation of the number of monomials is a #P-complete problem.

**Definition 3.16.** A problem  $\text{ENUM}\cdot A$  is computable in probabilistic incremental time, written **IncPP**, if there is a polynomial  $Q(x, y)$  and a machine  $M$  which solves  $\text{ENUM}\cdot A$  with probability greater than  $\frac{2}{3}$  and satisfies for all  $x$ ,  $T(x, i + 1) - T(x, i) \leq Q(|x|, i)$ .

We describe here a way to improve the error bound of **IncPP** algorithms, but it would work equally well on **DelayPP** ones. Note that in both cases we need an exponential space and there is a slight time overhead. Moreover, one really uses the fact that the problems are in **EnumP**.

**Proposition 3.17.** *If  $\text{ENUM}\cdot A$  is in **IncPP** then there is a polynomial  $Q$  and a machine  $M$  which for all  $\epsilon$  computes the solution of  $\text{ENUM}\cdot A$  with probability  $1 - \epsilon$  and satisfies for all  $x$ ,  $T(x, i + 1) - T(x, i) \leq Q(|x|, i) \log(\epsilon^{-1})$ .*

*Proof.* Since  $\text{ENUM}\cdot A$  is in **IncPP**, there is a machine  $M$  which computes the solution of  $\text{ENUM}\cdot A$  with probability  $\frac{2}{3}$  and a delay bounded by  $Q(|x|, i)$ . Since



$A(x, y)$  may be tested in polynomial time, we can assume that every output of  $M$  is a correct solution: one checks  $A(x, y)$  before outputting  $y$  and stops if  $A(x, y)$  does not hold.

We now simulate  $k$  runs of the machine  $M$  on input  $x$  in parallel. Let  $S_1$  and  $S_2$  be two sets of elements of  $A(x)$ . We use  $S_1$  as a list of the solutions already outputted by the algorithm and  $S_2$  as a buffer of solutions found but not yet outputted. At the beginning of the algorithm,  $S_1$  and  $S_2$  are empty.

In each of the run of  $M$ , when a solution should be outputted, the algorithm tests if it is in  $S_1$  and if not add it to the set  $S_2$ . These operations can be done in polynomial time even if  $S_1$  and  $S_2$  are exponential, if we implement these sets with a binary search tree.

Assume the algorithm has just outputted the  $i^{\text{th}}$  solution. It let the  $k$  runs be simulated for another  $Q(|x|, i)$  steps each before outputting an element of  $S_2$ . This element is removed from  $S_2$  and added to  $S_1$ . If  $S_2$  is empty, the algorithm stops. This algorithm clearly works in incremental polynomial time and if one of the runs finds all solutions, it also finds all solutions. Hence, the probability of finding all solutions is more than  $1 - \frac{1}{3}^k$ . If we set  $k = \lceil \frac{\log(\epsilon^{-1})}{\log(3)} \rceil$ , we have a probability of  $1 - \epsilon$ , which completes the proof.  $\square$

**Remark 3.18.** Assume the order in which the solutions of  $\text{ENUM}\cdot A$  are generated by  $M$  does not depend on the alea and is computable in polynomial time. Then one can use the algorithm to enumerate the solutions of a union of ordered problems (Proposition 2.41), to achieve the same reduction of error. In that case, the space overhead is not exponential but only polynomial.

The class **IncPP** may also be related to the problems  $\text{ANOTHERSOLUTION}$ . We adapt Proposition 2.14 to the class **IncPP**. A search problem has a solution in probabilistic polynomial time, if we have an algorithm which computes, with probability  $\frac{2}{3}$ , a solution in time polynomial in  $|x|$ .

**Proposition 3.19.** *There is an algorithm which computes with probability  $\frac{2}{3}$  a solution of  $\text{ANOTHERSOLUTION}_A$  in polynomial time if and only if  $\text{ENUM}\cdot A \in \text{IncPP}$ .*

*Proof.* Assume  $\text{ANOTHERSOLUTION}_A$  is computable in probabilistic polynomial time, we want to enumerate the solution of  $\text{ENUM}\cdot A$  on the input  $x$ . The number of solutions of  $\text{ENUM}\cdot A$  is bounded by  $2^{Q(|x|)}$  with  $Q$  a polynomial. We repeat the algorithm which solves  $\text{ANOTHERSOLUTION}_A$  at most  $Q(|x|)$  times. Each time, we test if the produced solution is correct, it can be done in polynomial time because  $\text{ENUM}\cdot A \in \text{EnumP}$ . If it is not the case we stop. This algorithm returns either a solution or “None” with probability of error bounded by  $\frac{1}{3 \cdot 2^{Q(|x|)}}$ .

We apply this algorithm to  $x$  and the empty set, we add the found solution to the set of solutions and we go on like this until we have found all solutions. The delay between the  $i^{\text{th}}$  and the  $i + 1^{\text{th}}$  solutions is bounded by the execution time of the algorithm  $\text{ANOTHERSOLUTION}$  which is polynomial in  $|x|$  and  $i$  the size of

the set of produced solutions. Moreover the probability of error is bounded by  $\frac{1}{3} < |A(x)| \times \frac{1}{3 \cdot 2^{Q(|x|)}}$ . This proves that  $\text{ENUM}\cdot A$  is in **IncPP**.

Conversely if  $\text{ENUM}\cdot A \in \text{IncPP}$ , on an instance  $(x, S)$  of  $\text{ANOTHERSOLUTION}_A$  we want to find a solution which is not in  $S$ . One enumerates  $|S| + 1$  solutions by the **IncPP** algorithm, in time polynomial in  $|S|$  and  $|x|$ . If one of these solutions is not in  $S$ , it is the output of the algorithm. If  $S$  is the set of all solutions, the enumeration will end in time polynomial in  $S$  and  $x$  and the algorithm outputs “None”.  $\square$

Since Zippel’s algorithm finds all monomials in its last step –even for multilinear polynomials– it seems hard to turn it into an incremental algorithm. On the other hand Algorithm 5 whose design has been inspired by Proposition 3.19 does the interpolation in incremental delay.

**Example 3.20.** To a graph  $G$ , we associate the polynomial  $\text{PerfMatch}(G)$ , whose monomials represent the perfect matchings of  $G$ . We write  $\mathcal{C}$  the set of perfect matching of  $G$ .

$$\text{PerfMatch}(G) = \sum_{C \in \mathcal{C}} \prod_{(i,j) \in C} X_{i,j}$$

For graphs with a “Pfaffian” orientation, such as the planar graphs, this polynomial is related to a Pfaffian and is then efficiently computable. Moreover all the coefficients of this polynomial are positive, therefore we can use Algorithm 5 to interpolate it deterministically with incremental delay. We have proved that the enumeration of perfect matching of planar graphs is in **IncP**. However, such algorithms already exist [Uno97] and are more efficient, since they are specifically designed for this purpose.

**Definition 3.21.** A problem  $\text{ENUM}\cdot A$  is computable in probabilistic polynomial delay **DelayPP** if there is a polynomial  $Q(x, y)$  and a machine  $M$  which solves  $\text{ENUM}\cdot A$  with probability greater than  $\frac{2}{3}$  and satisfies for all  $x$ ,  $T(x, i + 1) - T(x, i) \leq Q(|x|)$ .

**Example 3.22.** We have seen in Chapter 1 the Matrix-Tree theorem. It proves that a polynomial whose monomials are in bijection with the spanning trees of a graph is equal to a Determinant and thus is computable in polynomial time. Since this polynomial is also monotone, we can use Algorithm 6 to prove that the enumeration of the spanning trees of a graph is in **DelayP**.

**Open question:** Is there any inclusion between a probabilistic enumeration complexity class and a deterministic one?

### 3.6 Higher degree polynomials

In this section we sketch the method of Klivans and Spielman, which enables us to interpolate a polynomial in incremental delay with a polynomial dependency

in the degree. We also give a method derived from the ideas of both Algorithms 5 and 6 to interpolate a polynomial with incremental delay, which is good only for polynomials of small, fixed degree.

### 3.6.1 KS algorithm

In this part we explain an elaborate result on Identity Testing [KS01], which is used to interpolate sparse polynomials. The key of the method is a clever transformation of a multivariate polynomial into a univariate polynomial.

**Theorem 3.23** (Theorem 5 of [KS01]). *There exist a randomized polynomial time algorithm which maps the zero polynomial to itself and any non zero polynomial  $P$  with  $n$  variables and total degree at most  $D$  to a non zero univariate polynomial  $P'$  of degree at most  $D^4 n^6$  with probability  $\frac{2}{3}$ .*

The idea of the proof is to map a variable  $X_i$  to  $h^{z_i}$  where  $h$  is a new variable and the  $z_i$ 's are well chosen linear forms. Each monomial of  $P$  is thus mapped in  $P'$  to  $h$  to the power of a sum of the linear forms  $z_i$ . A generalized isolation lemma, proves that among these sums of linear forms evaluated on random points, there is one which is minimum with high probability.

Therefore the monomial of lowest degree in  $P'$  comes with probability  $\frac{2}{3}$  from a unique monomial of  $P$ , denoted by  $\lambda \vec{X}^{\vec{e}}$ . One interpolates the polynomial  $P'$  in polynomial time to find  $\lambda h^l$  the lowest degree monomial. This gives  $\lambda$  but one still needs to find  $\vec{e}$ .

One builds a new polynomial  $P''$ , where one substitutes  $p_i h^{z_i}$  to  $X_i$  where  $p_i$  is the  $i^{\text{th}}$  prime. One evaluates the linear form  $z_i$  on the same points as to find  $\lambda$ , therefore the lowest degree monomial of  $P''$  is  $\lambda \prod p_i^{e_i} h^{z_i}$ . One then interpolates  $P''$ , recovers  $\vec{e}$  and returns the whole monomial  $\lambda \vec{X}^{\vec{e}}$ .

**Theorem 3.24** (Adapted from Theorem 12 of [KS01]). *There is a randomized polynomial time algorithm which given a black box access to a non zero polynomial  $P$  with  $n$  variables and total degree  $D$  returns a monomial of  $P$  with probability  $\frac{2}{3}$  in a time polynomial in  $n$ ,  $D$  and with  $O(n^6 D^4)$  calls to the oracle.*

Using this procedure to implement *find\_monomial* in Algorithm 3.6, we obtain an incremental interpolation algorithm for any polynomial.

### 3.6.2 Interpolation of fixed degree polynomials

We describe in this subsection another algorithm which finds one monomial of any polynomial. However its dependency in the degree is exponential. It could be useful to interpolate a family of polynomials with a fixed degree. It is based on a generalization of the ideas used to design *not\_zero\_improved* and Algorithm 5.

**Proposition 3.25.** *Let  $P$  be a polynomial with  $n$  variables of degree  $d$  and of total degree  $D$ . There is an algorithm, which returns a set  $L$  of cardinal  $l$ , maximal such*

that  $(\vec{X}^L)^d$  divides a monomial of  $P$  with probability greater than  $1 - \epsilon$ . It uses  $ldn$  calls to the oracle on points of size less than  $\log(Dn\epsilon^{-1})$ .

*Proof.* By the same method as Corollary 3.9, we can solve MONOMIAL-FACTOR for the polynomial  $P$  and the term  $(\vec{X}^L)^d$ . We call the procedure solving this problem  $exist\_monomial(P, L, \epsilon)$ . Recall that it works with probability  $\epsilon$  by choosing random values in  $[D\epsilon^{-1}]$  for the variables not in  $L$ . The other variables are mapped into one variable and the obtained univariate polynomial of degree  $ld$  is interpolated. To this aim, the algorithm evaluates it on every integer of  $[ld]$ , so it calls the oracle  $ld$  times. The points on which  $P$  is evaluated are either in  $[ld]$  or are randomly chosen in  $[D\epsilon^{-1}]$ . They are less than  $D\epsilon^{-1}$ , since  $ld \leq D$  and  $\epsilon < 1$ .

---

**Algorithm 7:** The procedure  $max\_monomial(P, \epsilon)$

---

**Data:** A multilinear polynomial  $P$  with  $n$  variables, degree  $d$  and the error bound  $\epsilon$   
**Result:** A list of indices  $L$   
**begin**  
     $L \leftarrow \emptyset$   
    **for**  $i = 1$  **to**  $n$  **do**  
        **if**  $exist\_monomial(P, L \cup \{i\}, \frac{\epsilon}{n+1})$  **then**  
             $L \leftarrow L \cup \{i\}$   
    **return**  $L$   
**end**

---

Algorithm 7 finds a set  $L$  maximal for the property that  $(\vec{X}^L)^d$  divides a monomial of  $P$ . It works in the same way Algorithm 4 does. In total, it does  $ldn$  calls to the oracle and the randomly chosen points are of size  $\log(Dn\epsilon^{-1})$ .  $\square$

After one run of  $max\_monomial$ , we have  $L$  such that  $P = (X^L)^d P_1 + P_2$ . Since  $P$  is of degree  $d$  and that  $L$  is maximal for the property that  $(X^L)^d$  divides a monomial of  $P$ ,  $P_1$  is of degree  $d-1$  at most. If we find a way to evaluate  $P_1$ , it is then easy to apply  $max\_monomial$  recursively and to eventually find a monomial of  $P$ .

**Proposition 3.26.** *Let  $P$  be an  $n$  variables polynomial of degree  $d$ . Assume that  $P = (X^L)^d P_1 + P_2$  where  $P_1$  is not zero and  $P_2$  is of degree less than  $d$ . There is an algorithm, denoted by  $restriction(P, L)$ , which acts as a black box computing  $P_1$ . To do one evaluation of  $P_1$  on points of size less than  $s$ , it does  $ld$  calls to  $P$  on points of size less than  $\max(s, \log(ld))$ .*

*Proof.* W.l.o.g. say that  $L$  is the set  $[l]$ , the polynomial  $P_1$  depends only on the variables  $X_{l+1}, \dots, X_n$ . One want to compute  $P_1$  on the values  $x_{l+1}, \dots, x_n$ , whose size is bounded by  $s$ . Let  $H(Y)$  be the polynomial  $P$ , where  $Y$  is substituted to  $X_i$  if  $i \in [l]$ , otherwise  $X_i = x_i$ . We have  $H(Y) = Y^{ld} P_1(\vec{x}) + P_2(\vec{x}, Y)$  where

$P_2(\vec{x}, Y)$  is an univariate polynomial of degree less than  $ld$ . The coefficient of  $Y^{ld}$  in  $H(Y)$  is equal to the evaluation of  $P_1$  on the desired values. To compute it, one has only to interpolate  $H(Y)$ , with  $ld$  calls to the oracle on points of size bounded by  $\log(ld)$  and  $s$ .  $\square$

---

**Algorithm 8:** A recursive algorithm finding one monomial of  $P$

---

**Data:** A polynomial  $P$  with  $n$  variables, a degree  $d$  and the error bound  $\epsilon$

**Result:** A monomial of  $P$

**begin**

  Monomial( $Q, i$ ) =

**if**  $i = 0$  **then**

    | **Return**( $Q(0)$ )

**else**

    |  $L \leftarrow \text{max\_monomial}(Q, \frac{\epsilon}{n})$  ;

    | **Return**  $L$ ;

    | Monomial( $\text{restriction}(Q, L), i - 1$ ) ;

  in Monomial( $P, d$ ) ;

**end**

---

**Theorem 3.27.** *Let  $P$  be a polynomial with  $n$  variables of degree  $d$  and of total degree  $D$ . Algorithm 8 returns the sets  $L_d, \dots, L_1$  and the integer  $\lambda$  such that  $\lambda \prod_{i=1}^d (\vec{X}^{L_i})^i$  is a monomial of  $P$ , with probability  $1 - \epsilon$ . It performs  $O(nD^d)$  calls to the oracle on points of size  $\log(\frac{n^2 D}{\epsilon})$ .*

*Proof.* Let  $Q_d, \dots, Q_1$  be the sequence of polynomials on which the procedure Monomial of Algorithm 8 is recursively called. We denote by  $l_i$  the size of  $L_i$ . By a simple induction, using Prop 3.25 and Prop. 3.26, we have that  $Q_i$  is of degree  $i$  and we note its number of variable  $n_i$ . The correction of the algorithm derives from the construction of procedures *max\\_monomial* and *restriction*, that is Proposition 3.25 and 3.26.

We now bound the number of oracle calls this algorithm performs. Again, by an induction using the complexity of *restriction* given in Prop. 3.26, we see that one evaluation of  $Q_i$  requires  $\prod_{j=i+1}^d j l_j$  calls to  $P$ . The points used are of size less than  $\log(n^2 D \epsilon^{-1})$  for all  $i$ .

In Algorithm 8, the procedure *max\\_monomial* computes  $L_i$  from  $Q_i$  which, by Prop. 3.25, requires  $in_i l_i$  calls to the oracle giving  $Q_i$ . By the previous remark, these calls to the oracle giving  $Q_i$  are in fact  $n_i \prod_{j=i}^d j l_j$  calls to  $P$ .

It holds that  $\sum_{j=i}^d jl_j \leq D$  because the term  $\prod_{j=i}^d (\vec{X}^{L_j})^j$  divides a monomial of  $P$  and thus its total degree is less than the total degree of  $P$ . Therefore the maximum of  $\prod_{j=i}^d jl_j$  is obtained if all terms of the sum have the same value, i.e.  $l_j j = \frac{D}{d-i+1}$ .

Therefore  $\prod_{j=i}^d jl_j \leq \left(\frac{D}{d-i+1}\right)^{d-i+1}$ . Since  $n \leq n_i$ , one obtains that the number of calls to  $P$  when executing *max\_monomial* on  $Q_i$  is bounded by  $n \left(\frac{D}{d-i+1}\right)^{d-i+1}$ .

The total number of calls to  $P$  in the algorithm is the sum of calls done by *max\_monomial* on each polynomial  $Q_i$  for  $1 \leq i \leq d$ . Hence it is bounded by  $\sum_{i=1}^d n \left(\frac{D}{d-i+1}\right)^{d-i+1} \leq dnD^d = O(nD^d)$ .  $\square$

One can improve the analysis of Algorithm 8. For instance, its complexity depends on the number of degrees at which the variables of the polynomial appear and not on the degree itself. It means that, if we want to find a monomial of a polynomial whose variables are either at the power one or hundred, the previous algorithm does  $O(nD^2)$  calls and not  $O(nD^{100})$ .

Algorithm 8 finds a monomial of a polynomial of any degree and can thus be used to implement *find\_monomial* in Algorithm 5 to obtain an interpolation algorithm. However, we have to bound its probability of error by  $\frac{\epsilon}{(d+1)^n}$ , and we use then rather large evaluation points. To compare it fairly to the other algorithms of this chapter, we should use repetitions to decrease the error in the implementation of *exist\_monomial*. In this case the number of calls is multiplied by  $n$  but the evaluation points are of size  $O(nD)$  only.

### 3.6.3 Comparison of the complexity of several interpolation methods

The complexity of KS algorithm and of the interpolation algorithm obtained from Algorithm 8 come essentially from the number of calls they need to build a new a monomial. Indeed, recall that both algorithms find a monomial of  $P - Q$ , where  $Q$  is the sum of the monomials of  $P$  found by the algorithm at this point. Since  $Q$  is given explicitly and may be exponential in size, its evaluation dominates any other part of these algorithms.

To find a monomial of a polynomial with  $n$  variables, a degree  $d$  and a total degree  $D$ , the KS algorithm performs  $n^7 D^4$  calls to the oracle, while Algorithm 8 performs  $O(n^2 D^d)$  calls. Since  $D < nd$ , Algorithm 8 has a better delay than the KS algorithm for polynomial of degree  $d$  less than 9.

**Open question:** is it possible to turn Algorithm 8 into a fixed parameter algorithm? That is to reduce the number of calls to  $O(n^a D^b f(d))$  with  $a$  and  $b$

small but  $f$  increasing exponentially fast or worst. In this case, the interpolation algorithm obtained would be better than KS for any fixed  $d$ .

We now compare Algorithm 6 to classical algorithms *restricted to multilinear polynomials*. In the table,  $T$  is a bound on  $t$  the number of monomials that Ben-Or Tiwari and Zippel algorithms need to know before interpolation. In the row labeled Enumeration is written the kind of enumeration algorithm the interpolation method gives when the polynomial is polynomially computable.

	Ben-Or Tiwari [BO88]	Zippel [Zip90]	KS [KS01]	Algorithm 6
Algorithm type	Deterministic	Probabilistic	Probabilistic	Probabilistic
Number of calls	$2T$	$tnD$	$tn^7 D^4$	$tnD(n + \log(\epsilon^{-1}))$
Total time	Quadratic in $T$	Quadratic in $t$	Quadratic in $t$	Linear in $t$
Enumeration	Exponential	<b>TotalPP</b>	<b>IncPP</b>	<b>DelayPP</b>
Size of points	$T \log(n)$	$\log(nT^2\epsilon^{-1})$	$\log(nD\epsilon^{-1})$	$\log(D)$

Figure 3.1: Comparison of interpolation algorithms on multilinear polynomials

Assume one have to interpolate a polynomial, we would like to use the most efficient algorithm to do the interpolation. Since the choice of the best interpolation algorithm depends on the degree of the polynomial, one needs a way to find its degree quickly and with good probability.

**Proposition 3.28.** *Let  $P$  be a polynomial with  $n$  variables and of total degree  $D$ . We can find the degree of  $P$  with probability  $\epsilon$  in time polynomial in  $n$ ,  $D$ , the size of the coefficients of  $P$  and  $\log(\epsilon^{-1})$ .*

*Proof.* For each  $i$ , we use the Algorithm of Proposition 3.8 to find the degree of  $X_i$  in  $P$  with probability  $\frac{2}{3}$ . One repeats this algorithm  $O(\log(n\epsilon^{-1}))$  and takes the majority of the results: it is the degree of the variable  $X_i$  with probability of error less than  $\epsilon n^{-1}$  by the Chernoff's bound. One returns the maximum of the found degrees, the probability that there is an error in the degree of  $P$  is less than the sum of the errors, that is  $\epsilon$ .  $\square$

The general problem of finding the degree of a polynomial is an interesting and hard question. If the polynomial is given by a circuit, the interpolation technique of [GS09] enables to solve it in a time polynomial in the size of the circuit and the number of monomials.

### 3.7 Modest improvements

In this part we give several methods to improve the complexity of the algorithms of this chapter, especially Algorithm 5. A worthwhile goal is to transform this incremental algorithm into a polynomial delay one.

**Open question:** on what class of polynomials could we store  $Q$ , the polynomials containing the already produced monomials, in a way such that we can



evaluate it in polynomial time? Could modular computation help to solve this problem?

### 3.7.1 Finite fields

Over a finite field  $\mathbb{F}$ , the Schwarz-Zippel lemma holds, but to make it interesting we must find a set of elements of  $\mathbb{F}$  of cardinal greater than  $D$  the total degree of the tested polynomial.

Both studied algorithms use evaluation points in  $[2D]$  to interpolate polynomials of total degree  $D$ . Therefore they can be extended to work for polynomials over fields with  $2D$  elements or more. It is good in comparison with other classical algorithms, for instance the one of [Zip90], which needs exponentially bigger points.

Assume now that we want to interpolate a polynomial  $P$  over  $\mathbb{Z}$  and that its coefficients are small, less than its total degree  $D$ . Remark that it is the case in every example given in 3.5, where all polynomials have coefficients 1 or  $-1$ . There is a prime  $p$  between  $D$  and  $2D$ , and we consider  $P$  modulo  $p$ : it has the same monomials as  $P$ .

All computations are then done in the finite field  $\mathbb{F}_p$ . It is especially useful to speed up the computation of *subtract* in Algorithm 5. Indeed, one needs to evaluate a polynomial given explicitly, and arithmetic operations are quicker in a small finite field. In particular, the result of the evaluation of a monomial in  $\mathbb{F}_p$ , is always of size  $O(\log(D))$ , while its value in  $\mathbb{Z}$  is of size  $O(D \log(D))$ .

Moreover, if we replace a black box call by an actual computation, like in 3.5, it may be more efficient to do this computation over a finite field. For instance, the computation of a determinant can be done in  $O(n^{2,376})$  arithmetic operations [CW90, KV05], which are done in time  $O(\log(D))$  over  $\mathbb{F}_p$ .

### 3.7.2 A method to decrease the degree

We have seen that the complexity of Algorithm 8 is highly dependent on the degree of the interpolated polynomial. We propose here a simple technique to decrease the degree of the polynomial by one, which makes Algorithm 8 competitive for degree 10 polynomials.

Let  $P$  be a polynomial of degree  $d$ . One uses a procedure similar to the one of Algorithm 4 on it. It finds, with probability  $1 - \epsilon$ , a minimal set  $L$  such that  $P_L$  is not zero, with  $n \log(\epsilon^{-1})$  calls to the oracle. In this case, it does not give a monomial but we can write  $P_L(\vec{X}) = \vec{X}^L Q(\vec{X})$  and  $Q$  is of degree  $d - 1$ .

We may simulate an oracle call to  $Q(\vec{X})$  by a call to the oracle giving  $P_L$  and a division by the value of  $\vec{X}^L$  as long as no  $X_i$  is chosen to be 0. Moreover the monomials of  $Q$  are in bijection with those of  $P_L$  by multiplication by  $\vec{X}^L$ . Therefore to find a monomial of  $P$  we only have to find a monomial of  $Q$ .

Algorithm 8 does not evaluate the polynomial to 0, hence we can run it on  $Q$ , that we simulate with a low overhead in time, and one call to  $P$  for each



evaluation. Since the polynomial  $Q$  have degree  $d - 1$ , the computation of one of its monomials requires only  $nD^{d-1}$  calls to the oracle plus the  $n$  calls used to find  $L$ .

### 3.7.3 Reduction of error and number of monomials

In both Algorithm 5 and 6, one imposes an exponentially low error to the base procedure, which is expensive. We now show how to improve the situation in Algorithm 5, but the same technique works for the other one. The error bound of the procedure *find\_monomial* is  $\frac{\epsilon}{2^{n+1}}$  in Algorithm 5 because we call it  $t$  times, and  $t$  is bounded by  $2^n$ .

If we know  $t$  a priori, an error bound of  $\frac{\epsilon}{t}$  would be sufficient, but we cannot hope so. There is nonetheless a way to bound the error only by something which is of the same order as  $t$  and which thus may be less than  $2^n$ .

We add a variable  $i$  to count the number of calls to *find\_monomial* in Algorithm 5. We write  $K = \sum_{j=1}^{\infty} \frac{1}{j^2}$  and replace *find\_monomial*(*subtract*( $P, Q$ ),  $\frac{\epsilon}{2^{n+1}}$ ) by *find\_monomial*(*subtract*( $P, Q$ ),  $\frac{\epsilon}{Ki^2}$ ).

Since there is  $t$  calls of *find\_monomial*, the error bound of the whole algorithm is bounded by  $\sum_{i=1}^t \frac{\epsilon}{Ki^2}$  which is less than  $\frac{\epsilon}{K} \sum_{i=0}^{\infty} \frac{1}{i^2} = \epsilon$ . Therefore the modified algorithm still have a probability  $1 - \epsilon$  to correctly interpolate the polynomial.

The first advantage is obviously that the factor  $n + \log(\epsilon^{-1})$  in the delay, the total time and the number of calls to the oracle of Algorithm 5 becomes  $\log(t) + \log(\epsilon^{-1})$ . It is better when  $\log(t)$  is a  $o(n)$ .

Assume now that one want to broaden the class of polynomial, that one can interpolate with an incremental delay. One needs another implementation of the procedure *find\_monomial*( $P, \epsilon$ ). If we use this new implementation of Algorithm 5, *find\_monomial* has only to be polynomial in  $\epsilon^{-1}$  rather than in  $\log(\epsilon^{-1})$ .

### 3.7.4 Derandomization

Currently, a lot of efforts are done to find deterministic algorithms solving the POLYNOMIAL IDENTITY TESTING problem (PIT) for classes of circuits. We would like to transfer those results to the enumeration of the monomials of a polynomial. It is a generalization of the remark that if the polynomials are monotone then both Algorithms 5 and 6 can be made deterministic.

First remark that if we are able to solve ENUM·POLY in polynomial total time over a class of polynomial, then we can also solve PIT in deterministic polynomial time. Therefore we cannot hope to derandomize algorithms for ENUM·POLY on larger classes of polynomials than PIT.

The good news is that we can derandomize the algorithms of this chapter on any class of circuits on which there is a deterministic algorithm for PIT. In Algorithm 5, the only randomized step is the call to the procedure *not\_zero* which solves PIT. Therefore a deterministic implementation of this algorithm makes Algorithm 5 deterministic.

We cannot obtain the same result for Algorithm 6 over any classes of polynomial. Indeed, we need to use the solution to MONOMIAL-FACTOR on multilinear circuits. Even if it relies on solving PIT, it does so on a modified circuit, which could eventually not be in the class on which PIT is derandomized. However we can adapt the best derandomization so far over circuit of bounded depth, since the transformation does not change the depth.

Here the nodes of the circuits have unbounded fan-in but a limited depth. A  $\Sigma\Pi\Sigma\Pi(k)$  is a circuit of depth 4, such that the top gate is a sum gate of fan-in at most  $k$ .

**Theorem 3.29** ([KMSV10]). *Let  $k, n, s$  be integers. There is an explicit set  $\mathcal{H}$  of size polynomial in  $n, s$  and exponential in  $k$  that can be constructed in a time linear in its size such that the following holds. Let  $P$  be a non-zero polynomial computed by a multilinear  $\Sigma\Pi\Sigma\Pi(k)$  circuit of size  $s$  on  $n$  variables. Then there is some  $\alpha \in \mathcal{H}$  such that  $P(\alpha) = 0$ .*

**Corollary 3.30.** *The problem ENUM-POLY restricted to multilinear polynomials represented by  $\Sigma\Pi\Sigma\Pi(k)$  circuits is in DelayP.*

*Proof.* In Algorithm 6, the only randomized procedure is the one solving MONOMIAL-FACTOR. Proposition 3.11 gives a construction which reduces MONOMIAL-FACTOR to polynomial identity testing. Moreover, remark that the transformation does not increase the number of alternating gates, when we allow an unbounded fanin. Indeed, the product gates are replaced by several product gates and the binary tree to sum them can be merged into the sum gates of the next level. Moreover, the fanin of the top gate is not increased, because it is a sum gate. Therefore, the construction transforms a  $\Sigma\Pi\Sigma\Pi(k)$  circuit into a  $\Sigma\Pi\Sigma\Pi(k)$  circuit. We have thus a polynomial time deterministic algorithm for MONOMIAL-FACTOR and that concludes the proof.  $\square$

### 3.8 Hard questions for easy to compute polynomials

In this section, we assume that all polynomials are given by circuits. We describe four (families of) polynomials, representable by circuits. Their sizes and formal degree are polynomial in the number of their variables therefore the represented polynomials can be evaluated in polynomial time. We prove that we can encode hard combinatorial questions in these polynomials such as the MONOMIAL FACTOR problem or one of the two following problems:

NON-ZERO-MONOMIAL

*Input:* a polynomial given as a circuit and a term  $\vec{X}^{\vec{e}}$

*Output:* accept if  $\vec{X}^{\vec{e}}$  has a coefficient different from zero in the polynomial

MONOMIAL-COEFFICIENT

*Input:* a polynomial given as a circuit and a term  $\vec{X}^{\vec{e}}$

*Output:* return the coefficient of  $\vec{X}^{\vec{e}}$  in the polynomial

**Proposition 3.31.** *If the problem MONOMIAL-FACTOR on a class of polynomials can be solved in probabilistic polynomial time (in the number of variables and the total degree of the polynomial), then NON-ZERO-MONOMIAL can also be solved in probabilistic polynomial time.*

*Proof.* Let  $P$  be a polynomial of total degree  $D$  with  $n$  variables and  $\vec{X}^{\vec{e}}$  a term. Let  $k$  be the total degree of  $\vec{X}^{\vec{e}}$ , we denote by  $P_k$  the sum of monomials of  $P$  of total degree  $k$ . Let us remark that  $(P, \vec{X}^{\vec{e}}) \in \text{NON-ZERO-MONOMIAL}$  if and only if  $(P_k, \vec{X}^{\vec{e}}) \in \text{MONOMIAL-FACTOR}$ .

To prove the proposition, one has only to show that one can evaluate  $P_k$  from a polynomial number of calls to  $P$ . To do this, we use the method of Proposition 3.8. One substitutes to each variable  $X_i$  of  $P$  the product  $X_i X_{n+1}$  to obtain the polynomial  $\tilde{P}$ . One then interpolates the univariate polynomial  $\tilde{P}(\vec{x}, X_{n+1})$  where  $\vec{x}$  is the point on which one wants to evaluate  $P_k$ . The coefficient of the term  $X_{n+1}^k$  is  $P_k(\vec{x})$ . We have obtained it with  $D$  calls to  $P$  and in polynomial time in  $n$  and  $D$ .  $\square$

The converse does not seem to hold, that is MONOMIAL-FACTOR may be harder than NON-ZERO-MONOMIAL. The problem MONOMIAL-COEFFICIENT is the search version of NON-ZERO-MONOMIAL and is thus also harder. Therefore, the best result we can achieve is to prove that NON-ZERO-MONOMIAL is a hard problem on a family of polynomials.

Moreover MONOMIAL-FACTOR is exactly the problem solved for multilinear polynomials by the procedure *not\_zero\_improved* and MONOMIAL-COEFFICIENT the problem solved by *coefficient*. In what follows, we prove that these problems are not likely to be solvable in probabilistic polynomial time, even restricted to polynomials representable by small circuits of small formal degree.

As a consequence, Algorithm 6, which is based on repeatedly solving MONOMIAL-FACTOR, cannot be generalized to polynomials of degree 2 unless  $\text{RP} = \text{NP}$ . Therefore new methods must be devised to find a polynomial delay algorithm for higher degree polynomials.

### 3.8.1 Polynomials of unbounded degree

We define here the polynomial  $Q$  with  $n^2 + n$  variables and degree  $n$ , which has been introduced by Valiant (see [VZG87]):

$$Q(X, Y) = \prod_{i=1}^n \left( \sum_{j=1}^n X_{i,j} Y_j \right)$$

If we see  $Q$  as a polynomial in the variables  $Y_j$  only, the term  $T = \prod_{j=1}^n Y_j$  has

$\sum_{\sigma \in \Sigma_n} \prod_{i=1}^n X_{i, \sigma(i)}$  for coefficient, which is the Permanent in the variables  $X_{i,j}$ .

**Proposition 3.32.** *The problem MONOMIAL-COEFFICIENT is #P-hard.*

*Proof.* One can reduce the problem MONOMIAL-COEFFICIENT in polynomial time to the computation of the Permanent. Assume one wants to compute the Permanent of the  $n^2$  values  $x_{i,j}$ . One builds the polynomial  $Q(\vec{x}, \vec{Y})$  where  $x_{i,j}$  has been substituted to  $X_{i,j}$ . This polynomial can be given by a circuit of polynomial size in  $n$ . Thus the coefficient of  $T$  in  $Q(\vec{x}, \vec{Y})$  is the Permanent of the  $x_{i,j}$ 's and it is also the solution of MONOMIAL-COEFFICIENT when given  $Q(\vec{x}, \vec{Y})$  and  $T$  as input. Since the computation of the Permanent is #P-complete, the problem MONOMIAL-COEFFICIENT is #P hard.  $\square$

### 3.8.2 Degree 3 polynomials

We now prove a hardness result for degree 3 polynomials and the problem NON-ZERO-MONOMIAL. We improve it in the next subsection but the result presented here is still interesting because the polynomial we use has a much simpler form.

**Proposition 3.33.** *The problem NON-ZERO-MONOMIAL restricted to degree 3 polynomials is NP-hard.*

*Proof.* Let  $C$  be a collection of three-elements subsets of  $[n]$  (3-uniform hypergraphs). We construct a polynomial from  $C$ , on the variables  $(X_i)_{i \in [n]}$ , as follows. To each subset  $C'$  of  $C$  we associate the monomial  $\chi(C') = \prod_{\{i,j,k\} \in C'} X_i X_j X_k$ . Let

$Q_C$  be the polynomial:

$$\sum_{C' \subseteq C} \chi(C')$$

It can be represented by a circuit polynomial in the size of  $C$ , since it is equal to:

$$\prod_{\{i,j,k\} \in C} (X_i X_j X_k + 1)$$

The degree of  $Q_C$  is the maximal number of occurrences of an integer in elements of  $C$ . If each integer of  $[n]$  appears in at most three elements of  $C$ ,  $Q_C$  is of degree 3 and the problem of finding an exact cover of  $C$  is still NP-complete [GJ79].

By definition of  $\chi$ , a subset  $C'$  is an exact cover of  $[n]$  if and only if  $\chi(C') = \prod_{i \in [n]} X_i$ . Therefore to decide if  $C'$  has an exact cover, we only have to decide if  $\prod_{i \in [n]} X_i$  has a coefficient different from zero. It proves that NON-ZERO-MONOMIAL is NP-hard over circuits representing degree 3 polynomials.  $\square$

### 3.8.3 Degree 2 polynomials

Here we give hardness result for the problems MONOMIAL-FACTOR and NON-ZERO-MONOMIAL restricted to degree 2 polynomials. Again the second result is stronger but the first one is based on a simple polynomial and does not rely on sophisticated theorems.

**Proposition 3.34.** *The problem MONOMIAL-FACTOR restricted to degree 2 polynomials is NP-hard.*

*Proof.* Let  $\phi$  be a 2-CNF formula, it is a conjunction of  $n$  clauses  $C_i$  and each of them is the disjunction of two literals, which are either a variable or the negation of a variable. We note  $V$  the set of variables of  $\phi$  and  $v \in C_i$  if  $v$  is one of the literal of  $C_i$ . We build the polynomial  $Q_\phi$  from  $\phi$ . It has  $n$  variables  $X_i$  which represent the clauses  $C_i$  and one special variable  $Y$ .

$$Q_\phi(\vec{X}, Y) = \prod_{v \in V} ((Y \prod_{\neg v \in C_j} X_j) + (\prod_{v \in C_i} X_i)) \quad (3.1)$$

Any empty product in the equation is 1.

Remark that each clause has at most two literals, thus any variable  $X_i$  appears in at most two factors of the outermost product. Therefore the polynomial is of degree 2 in the variables  $X_i$ . We rewrite  $Q_\phi$  by expanding the product over  $V$ . In the next equation, the function  $d$  can be seen as a distribution of truth values or as a choice in each factor of  $Q_\phi$  of the left or right part of the sum. The integer  $\alpha(d)$  is the number of  $j$  such that  $d(j) = 0$  and  $\beta(d, i)$  is the number of literals in  $C_i$  made true by  $d$ .

$$Q_\phi(\vec{X}, Y) = \sum_{d \in 2^{|V|}} Y^{\alpha(d)} \prod_i X_i^{\beta(d, i)}$$

We write  $Q_\phi(\vec{X}, Y) = \sum_{k=1}^{|V|} Y^k Q_k(\vec{X})$ . Equation 3.1 allows us to build a circuit of size and formal degree polynomial in  $\phi$ , which represents  $Q_\phi$ . By homogenization, as in Proposition 3.11, one builds from the circuit representing  $Q_\phi$ , a circuit  $k^2$  times larger which represents  $Q_k(\vec{X})$ .

Finally, a monomial comes from a truth value assignment which satisfies  $\phi$  if and only if  $T = \prod_{i=1}^n X_i$  divides the monomial. In addition, a monomial represents an assignment of Hamming weight  $k$  if and only if it is in  $Q_k$ . The problem MONOMIAL-FACTOR for the polynomial  $Q_k$  and the term  $T$  is hence equivalent to the problem of deciding if  $\phi$  has a satisfying assignment of Hamming weight equal to  $k$ . This latter problem is NP-complete over 2-CNF formulas [FG06]. Therefore, MONOMIAL-FACTOR is NP-hard over degree 2 polynomials.  $\square$

**Proposition 3.35.** *The problem NON-ZERO-MONOMIAL restricted to degree 2 polynomials is NP-hard.*

*Proof.* Let  $G$  be a directed graphs on  $n$  vertices, the Laplace matrix  $L(G)$  is defined by  $L(G)_{i,j} = -X_{i,j}$  when  $(i, j) \in E(G)$ ,  $L(G)_{i,i} = \sum_{(i,j) \in E(G)} X_{i,j}$  and 0 otherwise.

Let  $\mathcal{T}_s$  be the set of spanning trees of  $G$ , rooted in  $s$  and such that all edges of a

spanning tree is oriented away from  $s$ . Let  $L(G)_{s,t}$  be the minor of  $L(G)$  where the row  $s$  and the column  $t$  have been deleted.

The Matrix-Tree theorem (see [Aig07] for more details) is the following equality:

$$\det(L(G)_{s,t})(-1)^{s+t} = \sum_{T \in \mathcal{T}_s} \prod_{(i,j) \in T} X_{i,j}$$

We substitute to  $X_{i,j}$  the product of variables  $Y_i Z_j$  in  $\det(L(G)_{s,t})$  which makes it a polynomial in  $2n$  variables. This polynomial is derived from a Determinant and can thus be computed in polynomial time. Every monomial represents a spanning tree whose maximum outdegree is the degree of the monomial. We assume that every vertex of  $G$  has indegree and outdegree less or equal to 2 therefore  $\det(L(G)_{s,t})$  is of degree 2.

Remark now that a spanning tree, all of whose vertices have outdegree and indegree less or equal to 1 is an Hamiltonian path. Therefore  $G$  has an Hamiltonian path beginning by  $s$  and finishing by a vertex  $v$  if and only if  $\det(L(G)_{s,t})$  contains the monomial  $Y_s Z_v \prod_{i \notin \{s,v\}} Y_i Z_i$ . To decide whether  $G$  has an Hamiltonian path, one has only to solve NON-ZERO-MONOMIAL on the polynomial  $\det(L(G)_{s,t})$  and the term  $Y_s Z_v \prod_{i \notin \{s,v\}} Y_i Z_i$ , for all pairs  $(s, v)$  which are in polynomial number. The Hamiltonian path problem restricted to directed graphs of outdegree and indegree at most 2 is NP-complete [Ple79]. Therefore NON-ZERO-MONOMIAL is NP-hard over degree 2 polynomials.  $\square$

**Remark 3.36.** Assume there is a polynomial delay algorithm solving ENUM·POLY over degree 2 polynomials in a way similar to Algorithm 6. It would solve NON-ZERO-MONOMIAL for black box polynomials in probabilistic polynomial time. If the polynomial can be evaluated in polynomial time, like in the previous proposition, then one has a RP algorithm to decide NON-ZERO-MONOMIAL. Therefore, such an algorithm would imply  $\text{RP} = \text{NP}$ .

The previous hardness results suggest that, if there is an algorithm in polynomial delay which solves ENUM·POLY over degree 2 polynomials, it cannot produce the monomials in any order. A related question is asked in [Kay10]: what is the complexity of computing the leading monomial of a depth three circuit?

### 3.8.4 Hardness regardless of the degree

If one considers only the degree of the polynomials, then the strongest result is Proposition 3.35. Nonetheless the polynomials used to prove the others results have a very simple form: a formula with three alternations of operands. They can be represented by circuits with gates of unbounded fanin and depth 3, whereas the last is a determinant which can “efficiently” simulate a large class of polynomials representable by a polynomial size circuit (see [Tod, MP08]).

The first three results can thus be seen as a proof of hardness of some problems over low degree polynomials represented by depth 3 circuits. One may wonder if these problems are still hard for depth 2 circuits, without degree restriction. For

instance, POLYNOMIAL IDENTITY TESTING is solvable in deterministic polynomial time on circuits of depth 2.

A polynomial defined by a polynomial  $\Sigma\Pi$  circuit (a layer of  $\times$  gates followed by a  $+$  gate) has only a polynomial number of monomial, therefore it can be interpolated in polynomial time. From this, one solves the problems NON-ZERO-MONOMIAL, MONOMIAL-COEFFICIENT and MONOMIAL-FACTOR.

**Proposition 3.37.** *The problem NON-ZERO-MONOMIAL is in P over depth 2 circuits.*

*Proof.* We only have to deal with the case of a polynomial  $P$  represented by a  $\Pi\Sigma$  circuit (a layer of  $+$  gates followed by a  $\times$  gate) with  $n$  variables  $X_1, \dots, X_n$ .

We have  $P = \prod_{i=1}^k T_i$  where  $T_i$  is a sum of variables with small positive coefficients.

We reduce NON-ZERO-MONOMIAL to the problem of deciding if there is a perfect matching in a graph. To  $P$  and the term  $\vec{X}^{\vec{e}}$ , we associate the bipartite graph  $G = (V, E)$  defined by:

1.  $V = \{u_1, \dots, u_k\} \cup \{v_j^l | j \in [n] \text{ and } l \leq e_j\}$
2.  $E = \{(u_i, v_j^l) | X_j \text{ has a non-zero coefficient in } T_i\}$

A vertex  $u_i$  represents the linear form  $T_i$ , while the vertices  $v_j^1, \dots, v_j^{e_j}$  represent the variable  $X_j$  ( $e_j$  is its degree in  $\vec{X}^{\vec{e}}$ ). There is an edge  $(u_i, v_j^l)$  if  $X_j$  has a coefficient different from zero in  $T_i$ . A perfect matching  $M$  of  $G$  correspond to the choice of one variable  $X_{\alpha(i)}$  for each  $T_i$ . If we expand the product in the definition of  $P$ , we obtain the term  $\prod_i X_{\alpha(i)}$ . Since all terms obtained by expansion have positive coefficients, they do not cancel and  $\prod_i X_{\alpha(i)}$  has a coefficient different from zero in  $P$ . All vertices  $v_j^l$  are saturated by  $M$ , because it is a perfect matching. It means that  $\prod_i X_{\alpha(i)} = \vec{X}^{\vec{e}}$ .

Conversely, and for the same reasons, the term  $\vec{X}^{\vec{e}}$  has a non zero coefficient in  $P$ , if there is a perfect matching in  $G$ . This proves that NON-ZERO-MONOMIAL is reducible to the problem of finding a perfect matching in a bipartite graph, which is in P.  $\square$

We now see that a slight generalization of the class of considered polynomials makes the problem NON-ZERO-MONOMIAL hard.

**Proposition 3.38.** *The problem NON-ZERO-MONOMIAL is NP-hard over the polynomials of the form  $\prod_{i=1}^k T_i$  where  $T_i$  is a sum of variables with coefficients  $-1, 0, 1, 2, 3$ .*

*Proof.* Let  $M$  be a  $n \times n$  matrix with coefficients in  $-1, 0, 1, 2, 3$ . We define the polynomial  $P_M$  to be the product of the  $T_i = \sum_{j \in [n]} M_{i,j} X_j$  for all  $i \in [n]$ . The

coefficient of the term  $\prod_{i=1}^n X_i$  in  $P_M$  is the Permanent of the matrix  $M$ . To decide if a matrix with coefficient in  $-1, 0, 1, 2, 3$  has a permanent 0 is NP-hard [Val79]. Thus NON-ZERO-MONOMIAL is also NP-hard.  $\square$

Finally, the evaluation of the Permanent is #P-complete for Turing reduction, even for matrices with coefficients in  $\{0, 1\}$ , which yields the next proposition.

**Proposition 3.39.** *The problem MONOMIAL-COEFFICIENT is #P-complete over depth 2 circuits.*



## Chapter 4

# Polynomials and Hypergraphs

In this chapter we solve two problems related to acyclicity and hypergraphs thanks to a generalization of the Matrix Tree theorem [BR91] and to the various algorithms on polynomials presented in Chapter 3. In all this chapter, acyclicity means Berge-acyclicity unless it is explicitly stated otherwise. The last section of this chapter is part of a joint work with David Duris accepted at WALCOM 2011 [DS11].

### 4.1 Introduction to the Pfaffian Tree theorem

We present a family of polynomials  $Z_H$ , where each  $Z_H$  is associated to a 3-uniform hypergraph  $H$ . The monomials of  $Z_H$  are in bijection with the spanning hypertrees of  $H$ , whose set is denoted by  $\mathcal{T}(H)$ .

**Definition 4.1.**

$$Z_H = \sum_{T \in \mathcal{T}(H)} \epsilon(T) \prod_{e \in E(T)} w_e$$

where  $\epsilon(T) \in \{-1, 1\}$ .

The function  $\epsilon(T)$  has a precise definition, see [MV02], but it is not used here. This polynomial has exactly one variable  $w_e$  for each hyperedge  $e$  of  $H$ . We also write  $w_{\{i,j,k\}}$  the variable associated to the hyperedge which contains the vertices  $i$ ,  $j$  and  $k$ .

**Definition 4.2.** Let  $H$  be a 3-uniform hypergraph,  $\Lambda(H)$  is the Laplacian matrix defined by  $\Lambda(H)_{i,j} = \sum_{i \neq k, j} \epsilon_{ijk} w_{\{i,j,k\}}$ .

$\epsilon_{i,j,k}$  is 0 when  $\{ijk\} \notin E(H)$ , otherwise  $\epsilon_{ijk} \in \{-1, 1\}$ .

The coefficient  $\epsilon_{ijk}$  is equal to 1 when  $i < j < k$  or any other cyclic permutation and is equal to  $-1$  when  $i < k < j$  or any other cyclic permutation. Thus  $\epsilon_{ijk}$  is computable in polynomial time in the size of  $i$ ,  $j$  and  $k$ . We may relate to  $Z_H$ ,

the Pfaffian of the Laplacian matrix which is of interest since it is computable in polynomial time. The following theorem is inspired by a similar theorem for graphs, called the Matrix-Tree theorem.

**Theorem 4.3** (Pfaffian-Hypertree (cf. [MV02])). *Let  $\Lambda(i)$  be the minor of  $\Lambda(G)$  where the column and the line of index  $i$  have been removed.*

$$Z_H = (-1)^{i-1} Pf(\Lambda(i))$$

For  $H$  a hypergraph with  $n$  vertices and  $m$  hyperedges,  $Z_H$  is a multilinear polynomial with  $m$  variables, and the size of its coefficients is one. Moreover, it is of total degree  $\frac{n-1}{2}$  because a spanning hypertree of a 3-uniform hypergraph has  $\frac{n-1}{2}$  hyperedges.

This theorem allows to build a simple RP algorithm to the following problem (cf. [CMSS08]).

SPANNING HYPERTREE

*Input:* a hypergraph  $\mathcal{H}$

*Output:* is there a spanning hypertree of  $\mathcal{H}$

## 4.2 Enumeration of the spanning hypertrees

The notion of spanning tree of a graph admits several interesting generalizations to a hypergraph. The most well-known notions of acyclicity are Berge,  $\gamma$ ,  $\beta$  and  $\alpha$ -acyclicity. For the three notions  $\gamma$ ,  $\beta$  and  $\alpha$ -acyclicity and 3-uniform hypergraphs, SPANNING HYPERTREE is NP-complete [Dur09]. On the other hand, SPANNING HYPERTREE is NP-complete for Berge-acyclicity and 4-uniform hypergraphs but not when restricted to 3-uniform hypergraphs. Indeed, one can adapt Lovász matching algorithm in linear polymatroids [Lov80] to solve SPANNING HYPERTREE for 3-uniform hypergraphs. This algorithm is very complicated and not designed for this particular case hence it seems hard to extend it into an efficient enumeration algorithm of the spanning hypertrees. Nonetheless it is easy to give a randomized enumeration algorithm by using Algorithm 6 of the last chapter.

**Proposition 4.4.** *The problem ENUM·SPANNING HYPERTREE for 3-uniform hypergraphs is in DelayPP. More precisely, there is an algorithm solving the problem with delay  $O(mn^{3.7}(n \log(n) + \log(\epsilon^{-1})))$  where  $m$  is the number of hyperedges,  $n$  is the number of vertices and  $\epsilon$  is a bound on the probability of error.*

*Proof.* Let  $H$  be a hypergraph with  $n$  vertices, the degree of  $Z_H$  is  $\frac{n-1}{2}$  (or 0). There is always a prime number  $p$  between  $\frac{n-1}{2}$  and  $n$ , we consider  $Z_H$  as a polynomial over  $\mathbb{F}_p$ .

We execute Algorithm 6 on the polynomial  $Z_H$ . We evaluate first the time taken by the the calls to the oracle, here the computation of the polynomial  $Z_H$ . The first step to compute  $Z_H$  is to build the  $(n-1) * (n-1)$  matrix  $\Lambda(i)$  and

is negligible with regard to the time to compute its Pfaffian. One needs the same time as to compute a Determinant that is to say  $(n^\eta)^{1+o(1)}$  field operations [KV05] where  $\eta$  is a parameter related to  $\omega$ , the exponent of matrix multiplication. Currently, we know that  $\omega < 2.376$  and  $\eta < 2.7$ . A field operation has a complexity  $O(\log(n))$ , therefore the evaluation of  $Z_H$  on any points of  $\mathbb{F}_p$  has a complexity  $(n^\eta)^{1+o(1)}$ .

By Theorem 3.12 there are  $O(mn(m + \log(\epsilon^{-1})))$  oracle calls between two solutions. By the remark of Section 3.7 on the error bound, we can improve it to  $O(mn(\log(t) + \log(\frac{1}{\epsilon})))$  where  $t$  is the number of spanning hypertrees that is bounded by  $n^n$ . In conclusion, the contribution to the delay of the evaluations of  $Z_H$  is  $O((mn^{\eta+1}(n + \log(\epsilon^{-1})))^{1+o(1)})$ .

One must take into account the cost of the univariate interpolation that the procedure *not\_zero\_improved* realizes. Since  $Z_H$  is defined over a finite field, the size of the evaluations of the polynomial are  $O(\log(n))$ . Furthermore, its total degree is  $\frac{n-1}{2}$ , hence the time to do the interpolation is  $O(n^2 \log(n))$ . The contribution to the delay is negligible, since for one interpolation, there are  $n$  evaluations of  $Z_H$ , each of them taking more time than the interpolation.  $\square$

Since the size of a 3-uniform hypergraph is typically  $m = O(n^3)$ , the delay is quite good, less than cubic. We could yet improve the complexity of the described algorithm, by using a probabilistic algorithm to compute the Determinant.

We can consider the “3-Pfaffian” hypergraphs (cf. [GdM10]). They are a generalization of the graphs with a “Pfaffian” orientation, like in the algorithm to count perfect matchings in planar graph [Kas61]. Their associated polynomial  $Z(H)$  is monotone, thus the previous algorithm can be made deterministic on this class.

### 4.3 Parallelism

The computation of the determinant and the Algorithm 6 are parallelizable. We can then think that the previous enumeration algorithm is parallelizable, but there are no definition of parallelism for enumeration. We now propose (without being too formal about the model of parallel computation) two possible ways of defining parallelism in enumeration algorithm. We then explain how good the previous algorithm is with regard to these notions.

**Total time** The first notion is related to the total time and the number of solutions.

**Definition 4.5.** A problem  $\text{ENUM-}A$  can be solved in parallel polynomial total time if there is an algorithm which, on an instance  $x$ , uses a total time polynomial in  $|x|$  and a number of processors polynomial in  $|A(x)|$ .

Algorithm 6 is obviously solvable in probabilistic parallel polynomial total time since, at each recursive call, we can use two processors to compute them at the

same time. In fact, one simply does a parallel traversal of the tree described in Sec 3.4. The enumeration of spanning hypertree has thus a parallel total time bounded by a polynomial in the number of hyperedges denoted by  $m$ . This time is the product of the depth of the tree which is equal to  $m$  and of the cost of doing one step in the tree which is also polynomial in  $m$ .

**Delay** The second notion is related to the delay.

**Definition 4.6.** A problem  $\text{ENUM}\cdot A$  can be solved with a highly parallel delay if there is an algorithm solving  $\text{ENUM}\cdot A$  on an instance  $x$  with a delay  $\log(|x|)^{O(1)}$  and it uses  $|x|^{O(1)}$  processors.

Note that a lot of interesting enumeration problems have a corresponding decision problem, which is  $P$ -complete like  $\text{ENUM}\cdot\text{HORNSAT}$ . They cannot have a highly parallel delay algorithm or every problem in  $P$  would be in  $\text{NC}$ , which is widely believed to be false.

The previous problem may not have a highly parallel delay. However, one can decrease substantially its delay by using a polynomial number of processors. First, the step of reduction of the error is only a repetition of the same algorithm, which can be trivially done in parallel. Second, the evaluation of the polynomial  $Z_H$  is the computation of a determinant and is thus in  $\text{NC}$ . The transformation on the variables of the determinant to obtain  $Z_H$  are only a few multiplications and additions, which can be done in  $\text{NC}$  also. Finally, the univariate interpolation is done through the resolution of a linear system which can be done in  $\text{NC}$ . Therefore only the depth first search in the tree is inherently sequential. The parallel delay of the algorithm, when given a polynomial number of processors to be executed on, is  $O(m^{1+o(1)})$  (almost linear).

Inspired by the definition of the class  $\text{SDelayP}$ , we can imagine a somewhat nicer way to parametrize the delay by a complexity classe.

**Definition 4.7.** A problem  $\text{ENUM}\cdot A$  has a  $\mathcal{C}$  delay if for every instance  $x$  there is a total order  $<_x$  on  $A(x)$  such that the following problems are in  $\mathcal{C}$ :

1. given  $x$ , output the first element of  $A(x)$  for  $<_x$
2. given  $x$  and  $y \in A(x)$  output the next element of  $A(x)$  for  $<_x$  or a special value if there is none

To define a class of problem with a “parallelizable delay”, one replaces  $\mathcal{C}$  by its favorite parallel complexity class, for instance  $\text{NC}$ .

## 4.4 Maximal spanning hypertree

We are interested in the following problem:

MAXIMAL ACYCLIC SUBHYPERGRAPH

*Input:* a hypergraph  $\mathcal{H}$  and an integer  $n$

*Output :* is there a subhypergraph of  $\mathcal{H}$  of size  $n$ , i.e. with  $n$  hyperedges?

Restricted to graphs, the problem is trivial since there is always a spanning tree, while for 4-uniform hypergraphs it is already NP-complete. Moreover, for the three other notions of acyclicity ( $\gamma$ ,  $\beta$  and  $\alpha$ ) it is NP-complete even for 3-uniform hypergraphs [DS11]. We study here the last case –3-uniform hypergraphs and Berge acyclicity– and gives an efficient randomized algorithm to solve it. The strategy is to relate this question to one on spanning hypertrees that we can solve thanks to the Pfaffian Hypertree theorem and Proposition 3.8.

**Proposition 4.8.** *Let  $H_n$  be the complete 3-uniform hypergraph over  $n$  elements and  $H$  one of its acyclic subhypergraphs. We can extend  $H$  into a spanning acyclic subhypergraph of  $H_n$ ,  $H_{n+1}$  or  $H_{n+2}$ .*

*Proof.* Let  $H$  be an acyclic hypergraph, let  $C_0, \dots, C_t$  be its connected components and  $v_0, \dots, v_t$  be vertices such that  $v_i \in C_i$ . The hypergraph  $H'$  is the union of the hyperedges of  $H$  and of  $\{v_{2i}, v_{2i+1}, v_{2i+2}\}$  for  $0 \leq i \leq \lceil \frac{n-t}{2} \rceil$ . If  $v_{t+1}$  appears, it is a new vertex and  $H'$  is a subhypergraph of  $H_{n+1}$ , otherwise it is a subhypergraph of  $H_n$ . Since we have connected by a path all connected components of  $H$  which is acyclic,  $H'$  is both acyclic and connected: it is a hypertree.

Let now  $H$  be a subhypertree on the vertices  $1, \dots, k$  of the hypergraph  $H_n$ . The hypergraph  $H''$  is the union of the hyperedges of  $H$  and the hyperedges  $(2i+k, 2i+k+1, 2i+k+2)$  for  $0 \leq i \leq \frac{n-k}{2}$ . Again we may introduce a new vertex labeled  $n+1$ , which makes  $H''$  a subhypergraph of  $H_{n+1}$  instead of  $H_n$ . The edges added to  $H$  form a path which covers all points not in  $H$ , therefore  $H''$  spans either  $H_n$  or  $H_{n+1}$ . Since this path has only one point in common with  $H$  which is acyclic,  $H''$  is also acyclic.

Combining the two constructions proves the result.  $\square$

**Theorem 4.9.** *The problem MAXIMAL ACYCLIC SUBHYPERGRAPH is in RP.*

*Proof.* Let  $H$  be a hypergraph on  $n$  vertices and  $k$  an integer, we want to decide if there is an acyclic subhypergraph of size  $k$  in  $H$ . Consider the polynomial  $Z(H_n)$ , its monomials are in bijection with the spanning hypertrees of the complete hypergraph  $H_n$ . We note  $S$  the set of indices of variables of  $Z(H_n)$  which represent the edges of  $H$ .

By Proposition 4.8, an acyclic hypergraph  $H'$  can be extended into a spanning hypertree, say w.l.o.g. of  $H_n$ , which is hence represented by a monomial of  $Z(H_n)$ . This monomial have a total degree with regard to  $S$  equal to the size of  $H'$ . Conversely, when a monomial of  $Z(H_n)$  is of degree  $l$  with regard to  $S$ , it means that the restriction of the corresponding hypertree to  $S$ , which is an acyclic hypergraph, is of size  $l$ . Therefore the maximum of the degrees of  $Z(H_n)$ ,  $Z(H_{n+1})$  and  $Z(H_{n+2})$  with regard to  $S$  is the maximal size of an acyclic subhypergraph of  $H$ .

We now use the algorithm of Proposition 3.8 to find the total degrees of  $Z(H_n)$ ,  $Z(H_{n+1})$  and  $Z(H_{n+1})$  with regard to  $S$ . Those polynomials have  $n$  variables, total degree less than  $n$  and coefficients bounded in size by 1, hence the algorithm is in time polynomial in  $n$  and uses less than  $n$  evaluations on points of size less than  $\log n^2$ . Thanks to Theorem 4.3, the evaluations can also be done in time polynomial in  $n$ , thus we find the degrees in polynomial time with probability  $\frac{1}{2}$ . If the maximum of the degree is more or equal to  $k$ , we are sure that there is an acyclic subhypergraph of size  $k$ , and if not there is none with probability  $\frac{1}{2}$ , which proves that the problem is in RP.  $\square$

**Remark 4.10.** The algorithm which is proposed here is in fact in RNC, the class of problem solved by randomized NC algorithms. Indeed, both the evaluation of the polynomial and the interpolation in the algorithm of Proposition 3.8 can be done in NC, as it is explained in the previous section.

The problem MAXIMAL ACYCLIC SUBHYPERGRAPH can naturally be seen as a problem parametrized by the size of the acyclic hypergraph one looks for. A hypergraph is Berge-acyclic if and only if there is an order on the edges such that for all edges  $E$ , the union of all edges greater than  $E$  has an intersection with  $E$  of size less than one. One may express the fact that a hypergraph represented by an incidence structure, equipped with an order, satisfies this condition by means of a  $\Pi_1$  formula. This means that MAXIMAL BERGE-ACYCLIC SUBHYPERGRAPH is in the class  $W[1]$  (for the definition of parametrized complexity and of the class  $W[1]$  see [FG06]). **Open question:** is MAXIMAL BERGE-ACYCLIC SUBHYPERGRAPH  $W[1]$ -complete or FPT?

**Part II**

**Logic**





## Chapter 5

# Monadic Second-Order Logic

In this chapter, we present the logic  $MSO$  over several structures and give some illustrations of its expressive power. We also give some background informations on graph and matroid decompositions. We present several parameters such as the tree-width or the branch-width. One of the main interest of these parameters is that, if we fix them many NP-complete problems become solvable in polynomial time. In particular, the model-checking of  $MSO$  is easy for bounded tree-width. For a very complete view on these subjects (and many others) see the book Graph Structure and Monadic Second-Order Logic by Courcelle [Cou] and also the Phd thesis of Kanté [Kan08].

### 5.1 Terms and trees

Recall that a tree is an acyclic graph. One considers *rooted trees* which have a distinguished vertex called the *root*. The vertices of a tree are called *nodes*, those of degree 1 are called *leaves*, whereas the others are called *inner nodes*. A *labeled tree* is a tree together with a function from the nodes of this tree to a set  $S$ , which most of the time is finite or is a set of words on a finite alphabet.

A functional signature is a pair  $(F, A)$ , where  $F$  is a finite set of function symbols of positive arity and  $A$  is a finite set of constants. We denote by  $T(F, A)$  the set of terms built over  $(F, A)$ . Note that a term can be seen as a ranked tree of bounded degree: each internal node is labeled by an element of  $F$ , each leaf by an element of  $A$ . In this thesis all the terms/trees are binary.

The terms of  $T(F, A)$  are represented by a relational structure whose domain is the set of nodes of the term. The structure has the binary relations  $lchild(x, y)$  and  $rchild(x, y)$  which are true when  $y$  is the left child, respectively the right child, of  $x$ . We also have one unary relation for each symbol in  $F$  and  $A$ , denoted by  $label(s) = e$ , which holds when  $e$  is the label of the node  $s$ .

We recall the definition of monadic second-order logic, here given over terms, i.e. the atoms are made from the relations of the structure which represents a term. The particularity of this logic is to use two types of variables. A first order

variable (in lower case) represents an element of the domain, whereas a second-order variables (in upper case) represents a subset of elements of the domain.

**Definition 5.1.** One builds atomic formulas from first and second-order variables and from the relations  $=$ ,  $\in$ ,  $rchild(x, y)$ ,  $lchild(x, y)$  and  $label(s) = e$  for all  $e$  of  $A \cup F$ . The set of **Monadic Second Order** formulas, denoted by  $MSO$ , is the closure of these atomic formulas by the usual quantifiers  $\exists$ ,  $\forall$  and the logical connectives  $\wedge$ ,  $\vee$  and  $\neg$ .

The equality is the equality over the elements of the domain, but we extend it to sets, since it is definable by a simple formula. The relation  $x \in X$  means that the element denoted by  $x$  is a member of the set denoted by  $X$ . We also use freely  $\neq$  and  $\subseteq$  over elements and sets since they are easily definable in  $MSO$ . We can express by a  $FO$  formula the fact to be the root or a leaf:

$$root(s) \equiv \forall x \neg (lchild(x, s) \vee rchild(x, s))$$

$$leaf(s) \equiv \forall x \neg (lchild(s, x) \vee rchild(s, x))$$

The structure we have described is not necessarily a binary tree. However, one can express this fact by a  $MSO$  formula, which states that the structure is connected (see the formula given in the preliminaries), has no cycles and each node is of degree 3 at most.

One can decide if an  $MSO$  formula holds over a term by building an appropriate tree automaton and running it on the term. This yields the following classical theorem.

**Theorem 5.2** (Thatcher and Wright [TW68]). *The model-checking of  $MSO$  formulas over terms is solvable by a fixed parameter linear algorithm, the parameter is the sum of the size of the formula and the size of the functional signature on which the terms are defined.*

## 5.2 Decomposition: the different notions of width

Since many NP-hard problems defined on graphs become easy on trees, a lot of effort have been put into understanding how a graph can be represented by a tree. There exist a lot of decomposition of graphs into trees, with an associated notion of width, which characterizes how far the graph is from being a tree. We describe here several of this decompositions, among them the branch-width is of special interest for the next chapter.

### 5.2.1 Tree-width

The tree-width that we now present is the first width notion that has been defined. Along with the path-width, it is the first concept introduced by Robertson and Seymour in their graph minor project [RS83, RS86].

Let  $G = (V, E)$  be a graph, we say that  $T$  is a tree decomposition of  $G$  if:

1. each node of  $T$  is labeled by a subset of  $V$
2. for every edge  $(u, v) \in E$ , there is a node of  $T$  labeled by a set  $V'$  such that  $u$  and  $v$  are in  $V'$
3. for every element  $u \in V$ , the set of nodes of  $T$  whose label contains  $u$  is connected

The *width* of the tree decomposition is the maximum size of the sets labeling  $T$  minus one. The *tree-width* of the graph is the minimum of the width over all its tree decompositions. If one additionally requires the tree  $T$  to be a path, one defines *path decomposition* and *path-width*.

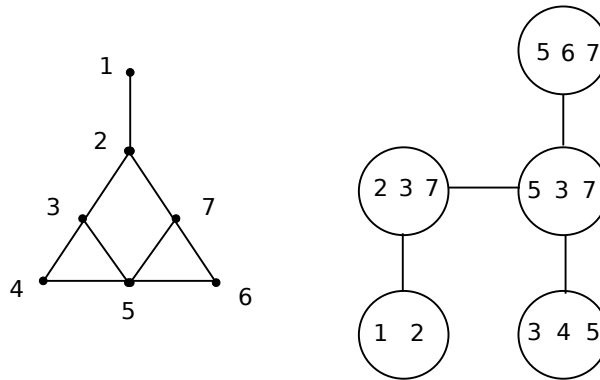


Figure 5.1: a graph and one of its tree-decomposition of width 2

**Example 5.3.** In Figure 5.1, we give an example of a graph and of one of its tree decomposition of width 2. The graphs of tree-width 1 are the trees (or the forests when they are not connected). Therefore the decomposition given in Figure 5.1 is optimal.

At most, the tree-width of a graph on  $n$  vertices is  $n - 1$ . This value is attained for the complete graph on  $n$  vertices.

The problem of finding the tree-width of a graph is NP-hard [ACP87]. However, there are plenty of polynomial time algorithms to find a good tree decomposition of a graph if the tree-width is less than a fixed  $k$ .

**Theorem 5.4** ([Bod93]). *There exists an algorithm which finds an optimal tree decomposition of a graph in time  $f(k)n$ , where  $n$  is the size of the graph,  $k$  its tree-width and  $f$  a computable function.*

One alternative to decompositions like tree-width is the use of algebraic operation to define a class of graphs. Let us give an example of a set of simple graph operations, that we will extend to matroids in the next chapter. A graph with two distinguished vertices, named respectively  $s$ -vertex and  $t$ -vertex, is called a 2-terminal graph. We define two operations on 2-terminal graphs:

1. let  $G_1 // G_2$  be the 2-terminal graph obtained by identifying the  $s$  and  $t$ -vertices of  $G_1$  with the ones of  $G_2$ .
2. let  $G_1 \bullet G_2$  the 2-terminal graph obtained by identifying the  $t$ -vertex of  $G_1$  with the  $s$ -vertex of  $G_2$ . The  $s$  vertex of  $G_1 \bullet G_2$  is the  $s$ -vertex of  $G_1$  and its  $t$ -vertex is the one of  $G_2$ .

*Series-parallel graphs* are represented by the terms on the function  $//$ ,  $\bullet$  and the constant  $\mathbf{a}$  which represents a graph with the two distinguished vertices  $s$  and  $t$ , and the edge  $(s, t)$ . The series-parallel graphs are exactly the graphs of tree-width 2.

### 5.2.2 Branch-width

In this part we define the *branch-width*, thanks to the general notion of connectivity function, which allows to define several decompositions. We follow the presentation of [Gro08].

Let  $S$  be a finite set and  $\kappa : 2^S \rightarrow \mathbb{N}$ . The function  $\kappa$  is symmetric if  $\kappa(B) = \kappa(S \setminus B)$  for all  $B \subseteq S$ . The function  $\kappa$  is submodular if for all  $B, C \subseteq S$

$$\kappa(B) + \kappa(C) \geq \kappa(B \cup C) + \kappa(B \cap C).$$

If  $\kappa$  is symmetric and submodular, it is a *connectivity function*.

A branch decomposition of  $(S, \kappa)$  is a pair  $(T, l)$  where  $T$  is a binary tree and  $l$  is a one to one labeling of the leaves of  $T$  by the elements of  $S$ . We define the mapping  $\tilde{l}$ , from the vertices of the graph to the sets of  $S$  recursively:

$$\tilde{l}(t) = \begin{cases} \{l(t)\} & \text{if } t \text{ is a leaf} \\ \tilde{l}(t_1) \cup \tilde{l}(t_2) & \text{if } t \text{ is an inner node with children } t_1, t_2 \end{cases}$$

The width of the branch decomposition of  $(S, \kappa)$  is defined by

$$\text{width}(T, \tilde{l}) = \max \left\{ \kappa(\tilde{l}(t)) \mid t \in V(T) \right\}$$

The branch-width of  $(S, \kappa)$  is the minimum of the width over all branch decompositions. Thanks to a result of Iwata, Fleischer and Fujishige [IFF01] about minimalization of a submodular function, we know that we can find an almost optimal branch decomposition with a fixed parameter tractable algorithm.

**Theorem 5.5** (Oum and Seymour [OS06]). *Let  $\kappa$  be a polynomial time computable connectivity function, symmetric, submodular and such that  $\kappa(\{v\}) \leq 1$ . For all  $k \geq 0$ , there is a polynomial-time algorithm which outputs a branch decomposition of  $(S, \kappa)$  of width at most  $3k$  if  $\text{bw}(S, \kappa) \leq k$ . If  $\text{bw}(S, \kappa) > k$ , the algorithm outputs a certificate of this fact.*

We introduce two notions of graph decomposition, which are branch decompositions of the sets of edges and of the set of vertices.

**Definition 5.6** (Branch-width of a graph). Let  $G = (V, E)$  be a graph. Let  $X$  be a set of edges, we note  $T_X$  the sets of vertices incident to an edge of  $X$ . For all  $X \subseteq E$ , we let  $\eta(X) = |\{T_X \cap T_{E \setminus X}\}|$ . The branch-width of  $G$ , denoted by  $bw(G)$ , is the branch-width of  $(E, \eta)$ .

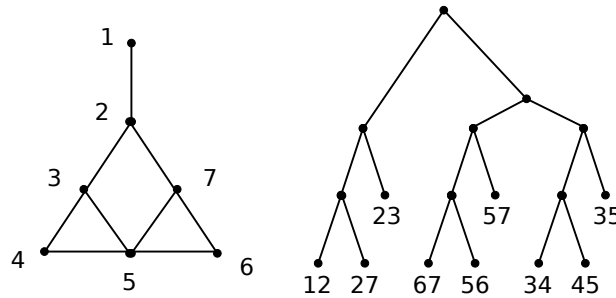


Figure 5.2: a branch-decomposition of width 2 of the graph of Example 5.1

The branch-width of a graph is closely related to tree-width: they are always within a constant factor of each other.

**Theorem 5.7** ([RS91]). *Let  $G$  be a graph, if  $bw(G) > 1$  then we have the inequalities:*

$$bw(G) - 1 \leq tw(G) \leq \lfloor \frac{3}{2}bw(G) \rfloor - 1.$$

We will also introduce the branch-width of matroids in this chapter, which is closely related to the branch-width of graphs, but also to the the rank-width of graphs, that we now define.

**Definition 5.8** (Rank-width). Let  $G = (V, E)$  be a graph. Let  $X$  a set of vertices and  $\bar{X}$  its complementary. The matrix  $(A_G)_{\bar{X}}^{\bar{X}}$  is the adjacency matrix restricted to the rows indexed by elements of  $X$  and the columns indexed by the elements of  $\bar{X}$ . For all  $X \subseteq V$ , we let  $\rho = rk((A_G)_{\bar{X}}^{\bar{X}})$ . The rank-width of  $G$ , denoted by  $rw(G)$ , is the branch-width of  $(V, \rho)$ .

Note that both functions are symmetric and submodular. We now give a theorem which shows that these widths are closely related.

**Theorem 5.9** ([Oum08]). *For a graph  $G$ ,  $I(G)$  is the incidence graph built from  $G$ . Then  $rw(I(G))$  is equal to  $bw(G) - 1$  or  $bw(G)$  unless the maximum degree of  $G$  is 0 or 1. If the maximum degree of  $G$  is 0 or 1, then  $rw(I(G)) \leq 1$  and  $bw(G) = 0$ .*

### 5.2.3 Clique-width

We describe the clique-width of a graph (introduced in [CER93]), which is a nice example of a decomposition measure given by a graph grammar. The definition is taken from [CO00]. Let  $\mathcal{L}$  be a set of labels, a labeled graph is a pair  $(G, \gamma)$  where  $\gamma$  is a function from  $V$  to  $\mathcal{L}$ . Let us consider the following set of graph operations:

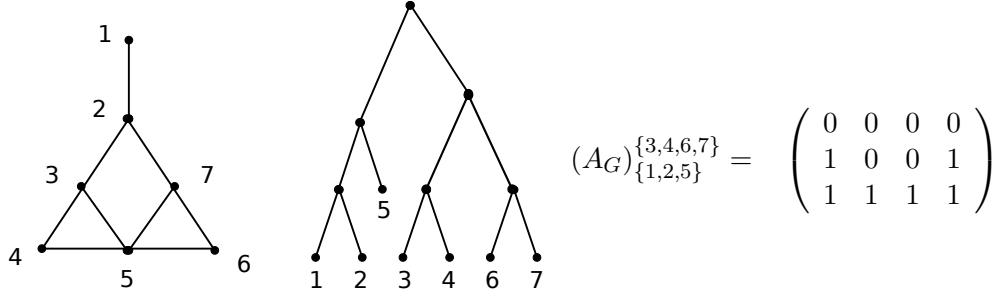


Figure 5.3: a rank-decomposition of width 2 of the graph of Example 5.1

- The disjoint union of two labeled graphs is denoted by  $\oplus$ .
- For all  $a, b \in \mathcal{L}$ , let  $\rho_{a \rightarrow b}$  be the function which renames every vertex labeled by  $a$  into  $b$ .
- For all  $a, b \in \mathcal{L}$ , let  $\eta_{a,b}$  be the function which adds all edges between the vertices labeled  $a$  and those labeled  $b$ .

The set of the functions foredefined is noted  $F_{\mathcal{L}}$ . For all  $a \in \mathcal{L}$ , let  $G_a$  be the graph with one vertex labeled by  $a$  and  $C_{\mathcal{L}} = \{G_a | a \in \mathcal{L}\}$ . The set  $T(F_{\mathcal{L}}, C_{\mathcal{L}})$  is a set of terms and it can also be seen as the set of labeled graphs it defines.

**Definition 5.10.** The clique-width of the graph  $G$ , denoted by  $cw(G)$ , is the minimum of the  $n \in \mathbb{N}$  such that  $\exists \gamma, (G, \gamma) \in T(F_{[n]}, C_{[n]})$ . A term of  $T(F_{[n]}, C_{[n]})$  is called a  $n$ -expression.

The class of graphs with a bounded clique-width is more general than the class of graphs with a bounded tree-width. Indeed, the next theorem states that the clique-width can be bounded by the tree-width. On the other hand, the converse is not true: the complete graphs have clique-width 2, while their tree-width is unbounded. The problem of finding the clique-width of a graph is NP-complete [FRRS06]. Moreover, there is no known polynomial-time algorithm for the recognition of graphs of clique-width at most  $k$  for  $k > 3$ .

**Theorem 5.11** ([CR05]). *Let  $G$  be a graph, then  $cw(G) \leq 3 \cdot 2^{tw(G)-1}$ .*

On the other hand, the clique-width and the rank-width are equivalent. In fact one can compute the rank-width to approximate the clique-width.

**Theorem 5.12** ([OS06]). *Let  $G$  be a graph, then  $rw(G) \leq cw(G) \leq 2^{rw(G)+1} - 1$ .*

### 5.3 The logic MSO on graphs

There are several versions of the MSO logic over graphs. More precisely, there are different ways of representing a graph by a structure and thus several MSO logic associated to these structures.

In the introduction, we say that a graph is represented by a structure whose domain is the set of its vertices and a binary relation  $E$  such that  $E(x, y)$  holds if  $(x, y)$  is an edge. The MSO logic over this structure is sometimes called  $MS_1$ .

Let  $G = (V, E)$  be a graph. We represent it by the structure  $(V \cup E, inc)$  where  $inc = \{(u, e) | u \in V, e \in E \text{ and } u \in e\}$ . One can represent any structure by such an *incidence structure* where each tuple of a relation is represented by a point of the domain. We write  $MS_2$  for the MSO logic interpreted over this representation of graphs.

One can define more graph properties in  $MS_2$  than in  $MS_1$ . For instance, the property that a graph has a perfect matching is easy to express in  $MS_2$ . One writes  $\exists X \phi(X)$  where  $\phi(X)$  holds if  $X$  is a set of edges and a perfect matching. However, there is no such  $MS_1$  formula [Cou].

The following theorem relate the notion of graph decomposition to the problem of the model checking of MSO. For a proof of the theorem and to find background and references to numerous similar results one may look in one of [HOSG08, Gro08, Cou].

**Theorem 5.13.** *There exist two functions  $f$  and  $g$  such that:*

1. *The model-checking problem for  $MS_1$  is decidable in time  $n^3 f(t, l)$ , where  $t$  is the clique-width of the graph and  $l$  the size of the formula.*
2. *The model-checking problem for  $MS_2$  is decidable in time  $ng(t, l)$ , where  $t$  is the tree-width of the graph and  $l$  the size of the formula.*

### 5.4 The logic MSO on higher order structures

#### 5.4.1 Hypergraphs

We describe here the MSO logic adapted to hypergraph. A hypergraph  $H$  is represented by the relational structure  $(V, E)$  where  $V$  is the set of vertices of  $H$  and  $E(X)$  is a second-order predicate which holds when  $X$  is a hyperedge. It is somewhat similar to the logic  $MS_2$ . However, it is less general since we cannot quantify over edges.

We can express the fact that a hypergraph is a clutter, that is no edge is a subset of another edge:

$$\forall X \forall Y (E(X) \wedge E(Y) \Rightarrow ((X \subseteq Y) \Rightarrow (X = Y)))$$

### 5.4.2 Matroids

Since matroids are hypergraphs, which enjoy some properties, they can be represented by the same structure as hypergraph. The relation  $E$  is noted *indep* and  $indep(X)$  means that  $X$  is an independent set. We could equivalently represent a matroid by the hypergraph of its circuits, bases, flats ...

The  $MSO$  logic over the structures representing matroids is denoted by  $MSO_M$ . Most important objects of matroids are definable in  $MSO_M$ , for instance  $X$  is a circuit if and only if it satisfies:

$$circuit(X) \equiv \neg indep(X) \wedge \forall Y (Y \not\subseteq X \vee X = Y \vee indep(Y))$$

We give now some examples of properties expressible in  $MSO_M$ . For more details and examples, one may read [Hli03]. We can express that a matroid is connected, meaning that every pair of elements is in a circuit, a notion similar to 2-connectivity in graphs:

$$\forall x, y \exists X x \in X \wedge y \in X \wedge circuit(X)$$

Matroid axioms given in Chapter 1 for the circuits are also expressible in  $MSO_M$ :

- $\neg Circuit(\emptyset)$
- $\forall C_1, C_2 (Circuit(C_1) \wedge Circuit(C_2) \wedge C_1 \subseteq C_2) \rightarrow C_1 = C_2$
- $\forall C_1, C_2, e (Circuit(C_1) \wedge Circuit(C_2) \wedge C_1 \neq C_2 \wedge C_1(e) \wedge C_2(e)) \rightarrow$   
 $(\exists C Circuit(C) \wedge C \subseteq C_1 \cup C_2 \wedge \neg C(e))$

One defines the notion of minor of a matroid by using the restriction presented in the introduction and an operation of contraction. For any matroid  $N$ , one can write a formula  $\psi_N$  of  $MSO_M$  which is true on a matroid  $M$  if and only if  $N$  is a minor of the matroid  $M$  (see [Hli03]). Therefore all classes of matroids defined by excluded minors, such as the binary matroids, are also definable by an  $MSO_M$  formula.

One can express some properties about a graph by a formula over its cycle matroid. For instance, one can check that a graph is Hamiltonian if and only if it has a cycle containing a spanning tree. This can be stated by the next formula, where  $basis(X)$  is a formula which holds if and only if  $X$  is a basis:

$$\exists C circuit(C) \wedge \exists x basis(C \setminus \{x\})$$

In fact, it has been proven in [Hli06] that any sentence about a loopless 3-connected graph  $G$  in  $MS_2$  can be expressed as a sentence about its cycle matroid in  $MSO_M$ . This property can be generalized to any graph, by considering the cycle matroid of  $G \uplus K_3$  which is a disjoint union of  $G$  and  $K_3$  with all edges between the two graphs.



**Matroid branch-width** Let  $M$  be a finite matroid with ground set  $S$  and let  $B$  be a set of elements of  $M$ . We define the connectivity function by

$$\kappa(B) = \text{rank}(B) + \text{rank}(S \setminus B) - \text{rank}(S).$$

The function  $\kappa$  is symmetric by construction and submodular because the rank function is submodular. It generalizes the branch-width of graphs in the following way:

**Theorem 5.14** ([HMJ07, MT07]). *The branch-width of a 2-connected graph  $G$  is equal to the branch-width of its cycle matroid.*

The classes of matroids of fixed low branch-width have been well studied. The matroid of branch-width 1 are direct sums of loops and coloops while the matroids of branch-width 2 is the class of direct sums of series-parallel networks. These classes can all be defined by exclusion of a few minors [RS91, HOSW02], whose list is explicitly known.

It is interesting to note that the notion of tree-width is hard to generalize to matroids. However, Hliněný and Whittle have proposed a definition and proved in [HW06] that a matroid is of bounded branch-width if and only if it is of bounded tree-width.

Theorem [OS06] holds in the case of the branch-width of matroids. Therefore one can find a near optimal branch decomposition in a time  $f(k)P(n)$ , where  $n$  is the size of the matroid,  $k$  its branch-width,  $f$  a computable function and  $P$  a polynomial. However, in the case of representable matroids over a fixed finite field, we have the following better result.

**Theorem 5.15** ([HO08]). *Let  $k$  be a fixed integer and let  $\mathbb{F}$  be a fixed finite field. We assume that a vector matroid  $M$  of size  $n$  is given as an input. There is an algorithm which outputs in time  $O(n^3)$  (parametrized by  $k$  and  $|\mathbb{F}|$ ), a branch-decomposition of the matroid  $M$  of width at most  $k$ , or confirms that  $\text{bw}(M) > k$ .*

The rank-decomposition of a graph is the branch-decomposition of some binary matroid. Thus, we can use the previous theorem to compute an exact rank-decomposition of a graph in cubic time for a fixed rank-width. Moreover, from this theorem and the inequality given by Theorem 5.12, we derive the following corollary. It is the only known way to obtain an approximate  $k$ -expression of a graph.

**Corollary 5.16.** *For a given  $k$ , there is an algorithm that, with input a graph  $G$ , either concludes that  $\text{cw}(G) > k$  or outputs a  $2^{k+2} - 1$ -expression of  $G$  in time polynomial in the size of the graph.*

Finally, one can decide  $\text{MSO}_M$  over matroids of fixed branch-width representable over finite fields. The first objective of the next chapter will be to give an alternate proof of this result. Please remark that this result holds only for matroids representable over *finite* fields. Over matroids representable over  $\mathbb{Q}$  of branch-width 3, the model-checking of  $\text{MSO}_M$  is NP-hard [HW06].

**Theorem 5.17** ([HW06]). *Let  $k$  be a fixed integer and let  $\mathbb{F}$  be a fixed finite field and let  $\phi \in MSO_M$ . We assume that a vector matroid  $M$  of size  $n$  is given as an input. There is an algorithm which decides whether  $M \models \phi$  in time  $O(n^3)$  with parameters  $k$ ,  $|\mathbb{F}|$  and  $|\phi|$ .*

## Chapter 6

# Monadic Second-Order Model-checking on Decomposable Matroids

### 6.1 Introduction

The model-checking of monadic second-order formulas is a natural and extensively studied problem that is relevant to many fields of computer science such as verification or database theory. This problem is hard in general (since *MSO* can express NP-complete properties like 3-colorability) but it has been proved tractable on various structures. For example, it is decidable in linear time on trees [TW68] thanks to automata techniques. It also remains linear time decidable [Cou91] on the widely studied class of graphs of bounded tree-width. Since then, a lot of similar results have been found, either with similar notions of width, like clique-width and rank-width, or for extensions of *MSO*, for instance by counting predicates (see [Gro08, HOSG08]).

In this chapter, we study the model-checking of monadic second order sentences on matroids and especially on representable matroids, which are a natural generalization of both graphs and matrices. Natural notions of decomposition such as tree-width or branch-width can be adapted in this context. It is also interesting to note that tree-width and branch-width on matroids are generalizations of the same notions on graphs. That is to say, the width of the cycle matroid is the same as the width of the graph, if it is simple and connected [Hli06], therefore all theorems on matroids can be specialized to graphs. The monadic second-order logic on matroids, denoted by  $MSO_M$ , enables to express many interesting matroids properties (see [Hli03] and the references therein) such as the connectivity or the representability over  $\mathbb{F}_2$  or  $\mathbb{F}_3$ .

Recently, the model-checking of *MSO* formulas on representable matroids of bounded branch-width has been studied and it has been proved to be decidable in a time linear in the size of the matroid [Hli06]. This result has been subsequently

extended in [Kra09] to a broader but more abstract class of matroids. The first contribution of this chapter is to introduce an alternative method to study these matroids, by an appropriate decomposition into labeled trees, called *enhanced trees* and a translation of  $MSO_M$  into  $MSO$ . For this purpose, we introduce the notion of *signature* over decomposed matroids which appears to be a useful general tool to study several classes of matroids. Signatures can be seen as the states of a nondeterministic bottom-up automaton which checks the dependence of a set of a matroid represented by a set of leaves of an enhanced tree.

As a corollary of this method, we give a new proof of the linear time model-checking of  $MSO_M$  formulas on representable matroids of bounded branch-width, and also an enumeration algorithm of all tuples satisfying a  $MSO_M$  query with a linear delay. We apply this result to the problem A-CIRCUIT that we have presented in Chapter 2. We obtain better algorithms for this problem and its enumeration version, in the case of  $\mathbb{F}$ -matroids of bounded branch-width. We also remark that the generalized spectrum (sizes of the sets satisfying a formula with free set variables) of a  $MSO_M$  formula is semi-linear over the class of  $\mathbb{F}$ -matroids of bounded branch-width.

From this starting result, we derive a general way to build matroid grammars, inspired by the parse tree of [Hli06]. We first introduce a grammar for matrices, which is similar to the one for representable matroids introduced in [Hli06]. We show why it is more appropriate to see this grammar as a matrix rather than a matroid one. Thanks to the connection with enhanced trees, we easily prove that it describes the representable matroids of bounded branch-width. We then build the class of matroid  $\mathcal{T}_k$  by means of series-parallel operations. As a decomposition measure, it appears to be distinct from the notion of branch-width. We give some useful insights about the structures of matroids in  $\mathcal{T}_k$  and its relations with the branch-width. Using the same approach as for the matroids of bounded branch-width, we build a  $MSO$  formula expressing the dependence relation over terms representing a matroid of  $\mathcal{T}_k$ . We prove that the model-checking of  $MSO_M$  formulas is decidable in linear time on them. To our knowledge, it is the first such result that applies to non necessarily representable matroids.

## 6.2 Matroid decomposition

### 6.2.1 Matroid branch-width

Let  $M$  be a finite matroid with ground set  $S$  and let  $B$  be a set of elements of  $M$ . Recall that the connectivity function in a matroid is defined by

$$\kappa(B) = \text{rank}(B) + \text{rank}(S \setminus B) - \text{rank}(S).$$

The function  $\kappa$  is symmetric by construction and submodular because the rank function is submodular. This connectivity function gives us a notion of branch-decomposition and branch-width, as explained in the Chapter 5.

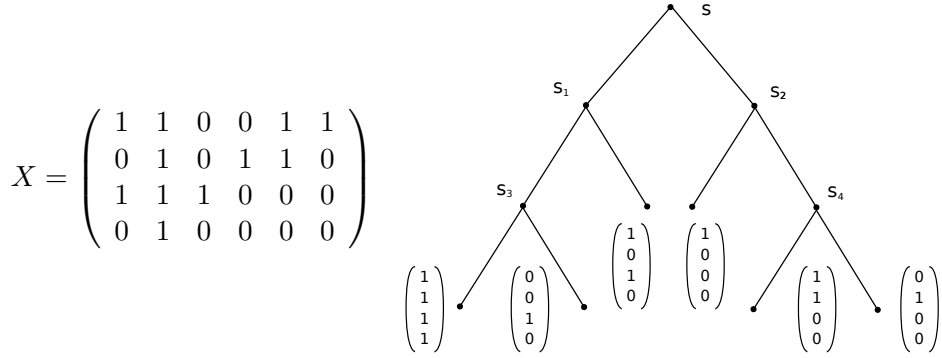


Figure 6.1: A matrix  $X$  and one of its branch decomposition of width 1

In this chapter, we restrict our study of branch-width to representable matroids. It means that  $M$  is given as a matrix  $A$  over a field  $\mathbb{F}$ . In this case, the rank in the matroid  $M$  is the same as the rank function in linear algebra. Moreover, the rank of a family of column vectors  $B$  is the dimension of the vector subspace it generates, denoted by  $\langle B \rangle$ .

It holds:

$$\dim(\langle S \rangle) = \dim(\langle B \rangle \cup \langle S \setminus B \rangle).$$

Therefore, by expanding the union, we get:

$$\dim(\langle S \rangle) = \dim(\langle B \rangle) + \dim(\langle S \setminus B \rangle) - \dim(\langle B \rangle \cap \langle S \setminus B \rangle).$$

We replace  $\dim(\langle S \rangle)$  by this expression in the definition of  $\kappa$  to obtain:

$$\kappa(B) = \dim(\langle B \rangle \cap \langle S \setminus B \rangle).$$

Let  $(T, l)$  be a branch decomposition of width  $t$  of  $M$  and let  $s$  be a node of  $T$ . In this article, we note  $T_s$  the subtree of  $T$  rooted in  $s$  and  $E_s$  the vector subspace generated by  $\tilde{l}(s)$ , that is to say the set of leaves of  $T_s$ . Let  $E_s^c$  be the subspace generated by  $S \setminus \tilde{l}(s)$  i.e. the leaves which do not belong to  $T_s$ . Let  $B_s$  be the subspace  $E_s \cap E_s^c$ , it is the boundary between what is described inside and outside of  $T_s$ .

**Remark 6.1.** We have seen that  $\kappa(\tilde{l}(s)) = \dim(E_s \cap E_s^c)$  which is equal by definition to  $\dim(B_s)$ . If  $t$  is the width of the branch decomposition  $(T, l)$ , for all nodes  $s$  of  $T$ ,  $\dim B_s \leq t$ .

**Example 6.2.** To illustrate this notion, we compute  $E_{s_1}$  and  $E_{s_1}^c$  to find  $B_{s_1}$  in the tree of Fig. 6.1. Notice that, when  $s$  is a leaf, the subspace  $E_s$  is generated by the single vector  $l(s)$ . Therefore  $B_s = E_s \cap E_s^c$  is either equal to  $E_s$  or trivial, i.e. equal to the zero vector, as in the case of the left child of  $s_3$  in Fig. 6.1.

$$E_{s_1} = \left\langle \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\rangle; E_{s_1}^c = \left\langle \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \right\rangle; B_{s_1} = \left\langle \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right\rangle$$

A near optimal branch-decomposition tree may be found for any submodular function, thanks to Theorem 5.5 given in the previous chapter. In the case of a  $\mathbb{F}$ -matroid, we recall that there is an algorithm which finds an exact branch-decomposition:

**Theorem 6.3** (Hliněný and Oum [HO08]). *Let  $k$  be a fixed integer and let  $\mathbb{F}$  be a fixed finite field. We assume that a vector matroid  $M$  of size  $n$  is given as an input. There is an algorithm which outputs in time  $O(n^3)$  (parametrized by  $k$  and  $|\mathbb{F}|$ ), a branch-decomposition of the matroid  $M$  of width at most  $k$ , or confirms that  $\text{bw}(M) > k$ .*

## 6.2.2 Enhanced branch decomposition Tree

From now on, all matroids will be representable over a fixed finite field  $\mathbb{F}$ . The results of the next part are false if  $\mathbb{F}$  is not finite, see [Hli06]. As we do not know how to decide in polynomial time if a matroid is representable, when we say it is, we assume that it has been given as a matrix. Furthermore, to simplify the presentation, we assume that the matroids have no loops, but this condition could easily be lifted.

Let  $t$  be a fixed parameter representing the maximal branch-width of the considered matroids. Let  $M$  be a matroid represented by the matrix  $A$  over  $\mathbb{F}$  and  $(T, l)$  a branch decomposition of width at most  $t$ . We will often not distinguish a leaf  $v$  of  $T$  from the column vector  $l(v)$  it represents. Let  $E$  be the vector space generated by the column vectors of  $A$ , we suppose that its dimension is the same as the length of the columns of  $A$  and we denote it by  $n$ .

We now build, for each node  $s$ , a matrix  $C_s$ . The construction is bottom-up, that is from leaves to root. The column vectors of this matrix are elements of  $E$  and they are partitioned in three parts which are bases of subspaces of  $E$ . If  $s$  is a leaf,  $C_s$  is a base vector of the subspace  $B_s$ . If  $s$  has two children  $s_1$  and  $s_2$ , the matrix  $C_s$  is divided in three parts  $(C_1|C_2|C_3)$  where  $C_1$ ,  $C_2$  and  $C_3$  are bases of  $B_{s_1}$ ,  $B_{s_2}$  and  $B_s$  respectively. By induction hypothesis, one already knows the bases of  $B_{s_1}$  and  $B_{s_2}$  used to build  $C_{s_1}$  and  $C_{s_2}$  and we choose them for  $C_1$  and  $C_2$ . We then choose any base of  $B_s$  for  $C_3$ .

Matrices  $C_s$  are of size  $n * t_1$ , with  $t_1 \leq 3t$ , because of Remark 6.1 on the dimension of boundary subspaces. A *characteristic matrix* at  $s$  is obtained by selecting a maximal independent set of rows of  $C_s$ . It can be done by Gaussian elimination in cubic time. The result is a matrix  $N_s = (N_1|N_2|N_3)$  of dimension  $t_2 * t_1$  with  $t_2 \leq 3t$ .

The vectors in  $N_s$  still represent the bases of  $B_{s_1}$ ,  $B_{s_2}$  and  $B_s$  in the same order but they only carry the dependence information. In fact, any linear dependence relation between the columns of  $C_s$  is a linear dependence relation between the

same columns of  $N_s$  with the same coefficients, and conversely. Note that the characteristic matrix at a node is not unique. It depends on the choice of bases used to represent the  $B_s$  subspaces and on the rows which have been removed by Gaussian elimination.

**Definition 6.4** (Enhanced branch decomposition tree). Let  $M$  be a  $\mathbb{F}$ -matroid and  $(T, l)$  one of its branch decomposition of width  $t$ . Let  $\tilde{T}$  be the tree  $T$  labeled at each node by a characteristic matrix obtained by the previous construction. We say that  $\tilde{T}$  is an enhanced branch decomposition tree of  $M$  of width  $t$  (enhanced tree for short).

Each label can be represented by a word of size polynomial in  $t$ . This is the reason why the matrix  $N$  has been chosen instead of  $C$  which is of size linear in the matroid. Indeed, the labels later appear in a formula whose size has to depend only in  $t$ . Note also that the leaves of the enhanced tree are in bijection with the elements of the matroid, by the same function  $l$  as for the branch decomposition tree.

Remark that, given a matrix of size  $n * m$  and a branch decomposition tree of width  $t$ , one can transform this tree into an enhanced tree in cubic time. The transformation of  $C$  into  $N$  only takes a linear time. To build a matrix  $C$  we have to build a base of the boundary space  $B_s$  for each node  $s$  of the tree. This can be done in quadratic time if the matrix  $A$  we are working with has been given in the normal form  $(Id|X)$ . If it is not given in such a way, it is always possible to compute such a normal form in cubic time.

**Example 6.5.** Figure 6.2 represents an enhanced tree constructed from the branch decomposition tree of Figure 6.1. Some of the intermediate computations needed to find it are also given for illustration. One may check that each label  $N_s$  of the tree is obtained by Gaussian elimination from  $C_s$ . Remark that, as it is a decomposition of branch-width 1, the subspaces  $B_s$  are of dimension 1 and are thus represented here by one vector.

## 6.3 Decision on an enhanced tree

### 6.3.1 Signature

We show how dependent sets of a matroid of bounded branch-width can be characterized using its enhanced tree. This will later allow us to define the dependence predicate by a formula in  $MSO$ .

We define the signature, an object which characterizes a set of a matroid with respect to a given enhanced tree. More precisely, we want to represent, for a node  $s$  of an enhanced tree, the elements of  $B_s$  which can be generated by a given set of elements of the matroid. We choose to represent one element of  $B_s$ , instead of all its elements or one of its basis. This will lead to shorter formulas in Sec.6.3.2 and the proofs are almost the same in the three cases.

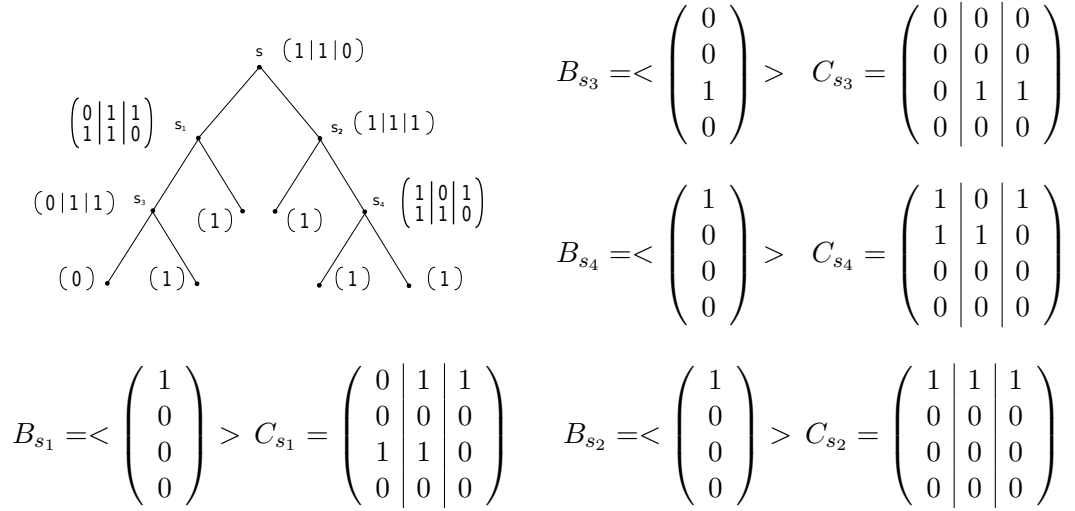


Figure 6.2: An enhanced tree built from the branch decomposition tree of Figure 6.1.

**Definition 6.6** (Signature). A signature is a finite sequence of elements of  $\mathbb{F}$ , denoted by  $\lambda = (\lambda_1, \dots, \lambda_l)$  or by  $\emptyset$  when it is of length 0.

**Definition 6.7** (Signatures of a set). Let  $A$  be a matrix representing a matroid and  $T$  one of its enhanced tree. Let  $s$  be a node of  $T$  and let  $X$  be a subset of the leaves of  $T_s$  which are seen as columns of  $A$ . Let  $v$  be an element of  $B_s$ , obtained by a nontrivial<sup>1</sup> linear combination of elements of  $X$ . Let  $c_1, \dots, c_l$  denote the column vectors of the third part of  $C_s$ . They form a base of  $B_s$ . Thus there is a signature  $\lambda = (\lambda_1, \dots, \lambda_l)$  such that  $v = \sum_{i=1}^l \lambda_i c_i$ . We say that  $X$  admits the signature  $\lambda$  at  $s$ . The set  $X$  also always admits the signature  $\emptyset$  at  $s$ .

The size  $l$  of a signature at a node  $s$  is the dimension of  $B_s$ , thus it is at most  $t$ , the width of the branch decomposition used to build  $T$ . Notice also that a set admits a lot of different signatures, in fact they form a vector subspace of  $\mathbb{F}^l$ .

**Example 6.8.** We illustrate the previous definition in the case of a leaf  $s$ .  
Case 1: the set  $X$  is empty, therefore there is no combination of its elements and the only signature it admits at  $s$  is  $\emptyset$ .  
Case 2: the set  $X = \{x\}$  and the label of  $s$  is the matrix  $(0)$ . The space  $B_s$  is the zero vector and there is no nontrivial combination of  $x$  equals to 0, therefore the only signature  $X$  admits at  $s$  is  $\emptyset$ .  
Case 3: the set  $X = \{x\}$  and the label of  $s$  is the matrix  $(\alpha)$  with  $\alpha \neq 0$ . The set  $B_s$  is hence generated by  $x$  and the set  $X$  admits the signatures  $(\lambda)$  for all  $\lambda \neq 0 \in \mathbb{F}$  and  $\emptyset$ .

<sup>1</sup>at least one of the coefficient of the linear combination is not zero



We now define a relation which describes how the signature a node admits is related to the signatures its children admit.

**Definition 6.9.** Let  $N$  be a matrix over  $\mathbb{F}$  divided in three parts  $(N_1|N_2|N_3)$ , and let  $\lambda, \mu, \delta$  be three signatures over  $\mathbb{F}$ . The submatrix  $N_i$  has  $l_i$  columns, and its  $j^{\text{th}}$  vector is denoted by  $N_i^j$ . The relation  $R(N, \lambda, \mu, \delta)$  is true if:

- $\lambda = \mu = \delta = \emptyset$  or
- $\lambda$  and at least one of  $\mu, \delta$  are not  $\emptyset$  and the following equation holds

$$\sum_{i=1}^{l_1} \mu_i N_1^i + \sum_{j=1}^{l_2} \delta_j N_2^j = \sum_{k=1}^{l_3} \lambda_k N_3^k \quad (6.1)$$

If a signature is  $\emptyset$ , the corresponding sum in Eq. 6.1 is replaced by 0.

**Lemma 6.10.** Let  $T$  be an enhanced tree,  $s$  one of its nodes with children  $s_1, s_2$  and  $N$  the label of  $s$ . Let  $X_1$  and  $X_2$  be two sets of leaves chosen amongst the leaves of  $T_{s_1}$  and  $T_{s_2}$  respectively. If  $X_1$  admits  $\mu$  at  $s_1$ ,  $X_2$  admits  $\delta$  at  $s_2$  and  $R(N, \lambda, \mu, \delta)$  holds then  $X = X_1 \cup X_2$  admits  $\lambda$  at  $s$ .

*Proof.* The case  $\lambda = \mu = \delta = \emptyset$  is obvious. By construction of  $N$ , we know that  $N_1, N_2$  and  $N_3$  represent the bases  $C_1, C_2$  and  $C_3$  of  $B_{s_1}, B_{s_2}$  and  $B_s$  respectively, meaning that they satisfy the same linear dependence relations. Then Equation 6.1 implies

$$\sum_{i=1}^{l_1} \mu_i C_1^i + \sum_{j=1}^{l_2} \delta_j C_2^j = \sum_{k=1}^{l_3} \lambda_k C_3^k$$

We assume without loss of generality that  $X_1$  admits a signature  $\mu \neq \emptyset$ . Therefore, there is a nontrivial linear combination of elements of  $X_1$  equals to  $\sum_{i=1}^{l_1} \mu_i C_1^i$ . The set  $X_2$  admits the signature  $\delta$ , thus there is a linear combination of elements of  $X_2$  equal to  $\sum_{j=1}^{l_2} \delta_j C_2^j$ . By summing the two linear combinations, we obtain a

nontrivial linear combination of elements of  $X_1 \cup X_2$  equal to  $\sum_{i=1}^{l_1} \mu_i C_1^i + \sum_{j=1}^{l_2} \delta_j C_2^j$

which is equal to  $\sum_{k=1}^{l_3} \lambda_k C_3^k$  by the previous equality.  $\square$

**Lemma 6.11.** Let  $T$  be an enhanced tree,  $s$  one of its nodes with children  $s_1, s_2$  and  $N$  the label of  $s$ . Let  $X_1$  and  $X_2$  be two sets of leaves chosen amongst the leaves of  $T_{s_1}$  and  $T_{s_2}$  respectively. If  $X = X_1 \cup X_2$  admits  $\lambda$  at  $s$ , then there are two signatures  $\mu$  and  $\delta$  such that  $R(N, \lambda, \mu, \delta)$  holds,  $X_1$  admits  $\mu$  at  $s_1$  and  $X_2$  admits  $\delta$  at  $s_2$ .

*Proof.* If  $\lambda = \emptyset$ , then the choice of  $\mu = \delta = \emptyset$  proves the lemma. Assume now that  $X$  admits  $\lambda \neq \emptyset$ , hence there is a nontrivial linear combination of elements in  $X$  equal to  $v = \sum_{k=1}^{l_3} \lambda_k C_3^k$ . We can divide the linear combination of elements in  $X$  into a sum of element in  $X_1$  equal to  $v_1$  and a sum of elements in  $X_2$  equal to  $v_2$  with  $v = v_1 + v_2$ . At least one of those combinations is nontrivial, we assume it is the one equal to  $v_1$ .

Since  $v_1 = v - v_2$  and  $v \in B_s$ , we have  $v_1 \in \langle E_{s_2} \cup B_s \rangle$ . Moreover  $B_s \subseteq E_s^c \subseteq E_{s_1}^c$  and  $E_{s_2} \subseteq E_{s_1}^c$  then  $v_1 \in E_{s_1}^c$ . Hence we have proven that  $v_1$  is in  $E_{s_1} \cap E_{s_1}^c = B_{s_1}$ . The vectors  $C_1^i$  are a base of  $B_{s_1}$ , so that there is  $\mu = (\mu_1, \dots, \mu_{l_1}) \neq \emptyset$  such that  $v_1 = \sum_{i=1}^{l_1} \mu_i C_1^i$ . It means that  $X_1$  admits the signature  $\mu$  at  $s_1$ . Since  $X_2$  plays a symmetric role, the same demonstration proves that it admits the signature  $\delta = (\delta_1, \dots, \delta_{l_2})$  at  $s_2$  such that  $v_2 = \sum_{j=1}^{l_2} \delta_j C_2^j$ .

Finally we have  $\sum_{i=1}^{l_1} \mu_i C_1^i + \sum_{j=1}^{l_2} \delta_j C_2^j = \sum_{k=1}^{l_3} \lambda_k C_3^k$  and we can replace the columns of  $C_i$  by those of  $N_i$ , which proves that  $R(N, \lambda, \mu, \delta)$  holds.  $\square$

We then derive a global result on enhanced trees and signatures.

**Lemma 6.12.** *Let  $A$  be a matrix representing a matroid,  $T$  one of its enhanced tree and  $X$  a set of columns of  $A$ . The set  $X$  admits the signature  $\lambda$  at  $u$  if and only if there exists a signature  $\lambda_s$  for each node  $s$  of the tree  $T$  such that:*

1. *for every node  $s$  labeled by  $N$  with children  $s_1$  and  $s_2$ ,  $R(N, \lambda_s, \lambda_{s_1}, \lambda_{s_2})$  holds.*
2. *for every leaf  $s$ ,  $\lambda_s \neq \emptyset$  only if  $s$  is in  $X$  and  $s$  is labeled by the matrix  $(\alpha)$  with  $\alpha \neq 0$ .*
3.  $\lambda_u = \lambda$ .

*Proof.* The proof is by induction on the height of  $s$  in  $T$ . If  $u$  is a leaf of  $T$ , the equivalence is true because of the second condition and Example 6.8.

Assume now that  $u$  is an internal node labeled by  $N$  and with children  $s_1$  and  $s_2$ . The induction hypothesis and the conditions 1 and 3 enable us to use the Lemmas 6.10 and 6.11 to prove both sides of the equivalence.  $\square$

The following theorem is the key to the next part, it shows that testing dependence of a set can be done by checking local constraints on signatures.

**Theorem 6.13** (Characterization of dependence). *Let  $A$  be a matrix representing a matroid  $M$ ,  $T$  one of its enhanced tree and  $X$  a set of columns of  $A$ . The set  $X$  is dependent if and only if there exists a signature  $\lambda_s$  for each node  $s$  of the tree  $T$  such that:*

1. *for every node  $s$  labeled by  $N$  with children  $s_1$  and  $s_2$ ,  $R(N, \lambda_s, \lambda_{s_1}, \lambda_{s_2})$  holds.*
2. *for every leaf  $s$ ,  $\lambda_s \neq \emptyset$  only if  $s$  is in  $X$  and  $s$  is labeled by the matrix  $(\alpha)$  with  $\alpha \neq 0$ .*
3. *the signature at the root is  $(0, \dots, 0)$*

*Proof.* If a set  $X$  admits the signature  $(0, \dots, 0)$  at the root, it means that there is a nontrivial linear combination of its elements equal to 0. It is therefore equivalent for  $X$  to be a dependent set of  $M$  and to admit signature  $(0, \dots, 0)$  at the root of  $T$ . The proof of the theorem follows from this remark and Lemma 6.12 applied at the root.  $\square$

### 6.3.2 From matroids to trees

The aim of this section is to translate  $MSO_M$  formulas over a matroid into  $MSO$  formulas over its enhanced tree. These two formalisms have been presented in Chapter 5. The main difficulty is to express the predicate *indep* in  $MSO$ . To achieve that, we use Theorem 6.13 which reduces this property to an easily checkable condition on a signature at each node of the enhanced tree. This can be seen as finding an accepting run of a non deterministic automaton whose states are signatures.

The formula is defined for enhanced trees of width less than  $t$  over a field  $\mathbb{F}$  of size  $k$ . We have to encode in  $MSO$  a signature  $\lambda$  of size at most  $t$  at each node of an enhanced tree. These signatures are represented by the set  $\vec{X}$  of set variables  $X_\lambda$  indexed by all signatures  $\lambda$  of size at most  $t$ . The number of such variables is bounded by  $(k+1)^t$ , a constant because both the field and the branch-width are fixed.

The relation  $X_\lambda(s)$  holds if and only if  $\lambda$  is the signature at  $s$ . The following formula states that there is one and only one value for the signature at each  $s$ .

$$\Omega(\vec{X}) \equiv \forall s \bigvee_{\lambda} \left( X_{\lambda}(s) \bigwedge_{\lambda' \neq \lambda} \neg X_{\lambda'}(s) \right)$$

---

**Discussion:** We could have defined the signature of a set as the union of all the signature it admits, it would have then be unique. But in this case, we would have dealt with  $2^{k^t}$  possible signatures which is still bounded if  $k$  and  $t$  are fixed, but is much larger and further decreases the practical interest of the algorithm we provide.

If we want to be more efficient and use less variables, we may encode in binary the value of each element  $\lambda_i \in \mathbb{F}$  of the signature  $\lambda$ . We only need  $\log(k)t$  variables to do so and it also spares us the formula  $\Omega$  but it would obfuscate the presentation.

---

The formula  $dep(Y)$  that represents the negation of the relation *indep* is now built in three steps corresponding to the three conditions of Theorem 6.13.

1. The formula  $\Psi_1$  ensures that the relation  $R$  holds at every internal node. It is a conjunction on all possible characteristic matrices  $N$  and all signatures  $\lambda$ , thus there are less than  $(k+1)^{9t^2+3t}$  terms in this conjunction, which is a constant.

$$\Psi_1(\vec{X}) \equiv \forall s \neg leaf(s) \Rightarrow [\exists s_1 s_2 lchild(s, s_1) \wedge rchild(s, s_2) \wedge \bigwedge_{\lambda_1, \lambda_2, \lambda, N} ((label(s) = N \wedge X_{\lambda_1}(s_1) \wedge X_{\lambda_2}(s_2) \wedge X_{\lambda}(s)) \Rightarrow R(N, \lambda, \lambda_1, \lambda_2))]$$

2. We define the formula  $\Psi_2(Y, \vec{X})$  which means that a leaf with a signature different from  $\emptyset$  is in  $Y$  and has a label different from the matrix  $(0)$ .

$$\Psi_2(Y, \vec{X}) \equiv \forall s (leaf(s) \wedge \neg X_{\emptyset}(s)) \Rightarrow (Y(s) \wedge label(s) \neq (0))$$

3.  $\Psi_3(\vec{X})$  states that the signature at the root is  $(0, \dots, 0)$ .

$$\Psi_3(\vec{X}) \equiv \exists s root(s) \wedge X_{(0, \dots, 0)}(s)$$

Thanks to Theorem 6.13 we know that the following formula is true on an enhanced tree  $T$  of a matroid  $M$  if and only if  $Y$  is a set of leaves of  $T$  in bijection with a dependent set of  $M$ .

$$dep(Y) \equiv \exists \vec{X} \Omega(\vec{X}) \wedge \Psi_1(\vec{X}) \wedge \Psi_2(Y, \vec{X}) \wedge \Psi_3(\vec{X})$$

Recall that we note  $l$  the bijection between the leaves of an enhanced tree and the elements of the matroid they represent. We define by induction a formula  $F(\phi(\vec{x}))$  of  $MSO$  from the formula  $\phi(\vec{x}) \in MSO_M$ , by relativization to the leaves.

- if  $\phi(\vec{x})$  is the relation  $x = y$  or  $x \in X$ ,  $F(\phi(\vec{x}))$  is the same relation
- if  $\phi(\vec{x})$  is the relation  $\text{indep}(X)$ ,  $F(\phi(\vec{x}))$  is the negation of the formula  $\text{dep}(X)$  we have just defined
- if  $\phi(\vec{x})$  is the formula  $\psi(\vec{x}) \wedge \chi(\vec{x})$ ,  $F(\phi(\vec{x}))$  is the formula  $F(\psi(\vec{x})) \wedge F(\chi(\vec{x}))$
- if  $\phi(\vec{x})$  is the formula  $\exists y\psi(y)$ ,  $F(\phi(\vec{x}))$  is the formula  $\exists y(\text{leaf}(y) \wedge F(\psi(y)))$
- if  $\phi(\vec{x})$  is the formula  $\exists Y\psi(Y)$ ,  $F(\phi(\vec{x}))$  is the formula  $\exists Y[\forall y(y \in Y \Rightarrow \text{leaf}(y)) \wedge F(\psi(Y))]$

Moreover, for every free first-order variable  $y$  and every free second-order variable  $Y$ , we take the conjunction of the relativized formula above with:

- $\text{leaf}(y)$
- $\forall y(y \in Y \Rightarrow \text{leaf}(y))$

We can now state the main theorem:

**Theorem 6.14.** *Let  $M$  be a matroid of branch-width less than  $t$ ,  $T$  one of its enhanced tree and  $\phi(\vec{x})$  a  $MSO_M$  formula with free variables  $\vec{x}$ , we have*

$$(M, \vec{a}) \models \phi(\vec{x}) \Leftrightarrow (T, l(\vec{a})) \models F(\phi(\vec{x}))$$

*Proof.* The demonstration is done by induction, every case is trivial except the translation of the predicate  $\text{indep}$  whose correctness is given by Theorem 6.13.  $\square$

Suppose we have a formula  $\phi$  of  $MSO_M$  and a representable matroid  $M$  of branch-width  $t$ . We know that we can find a branch-width decomposition of width less than  $3t$  in cubic time [HO08]. Furthermore, we can build from it an enhanced tree of  $M$  in cubic time. By Theorem 6.14, we know that we need only to decide the formula  $F(\phi)$  on the enhanced tree to decide  $\phi$  on  $M$ , which is done in linear time by Theorem 5.2. We have, as a corollary, the main result of [Hli06].

**Corollary 6.15** (Hliněný). *The model-checking problem of  $MSO_M$  formulas is decidable in time  $f(t, k, l) \times n^3$  over the set of  $\mathbb{F}$ -matroids given by a matrix, where  $n$  is the number of elements in the matroid,  $t$  is its branch-width,  $k$  is the size of  $\mathbb{F}$ ,  $l$  is the size of the formula and  $f$  is a computable function.*

Since we can decide dependence in a represented matroid of bounded branch-width in linear time by only using one of its enhanced tree, the enhanced trees are a way to describe completely a matroid and then to represent it. Moreover, this representation is compact, since the size of an enhanced tree is  $O(t^2 \times n)$ , where  $n$  is the size of the ground set of the matroid, while the matrix which usually defines it, is of size  $O(n^2)$ .

## 6.4 Extensions and applications

In this section, we present applications and generalizations of the result of the previous section, by an extension of the model or of the language. In particular, we use it to give a new algorithm for the problem  $\text{ENUM}\cdot A - \text{Circuit}$  studied in chapter 2.

### 6.4.1 Logic extension

**Colored matroids** We can work with colored matroids, meaning that we add a finite number of unary predicates to the language which are interpreted by subsets of the ground set. Theorem 6.14 still holds for colored matroids except that we now have colored trees, on which the decision problem for  $MSO$  is still in linear time.

The problem  $A\text{-CIRCUIT}$  studied in Chapter 2 is easily expressible in  $MSO_M$  over a colored matroid with one color, i.e. a unary second-order predicate denoted by  $A$ :

$$A - \text{Circuit}(X) \equiv A \subseteq X \wedge \text{Circuit}(X).$$

Thus  $A\text{-CIRCUIT}$  is decidable in polynomial time for matroids of branch-width  $t$ . It is an example of a NP-complete problem over matroids which is made tractable for bounded branch-width.

**Counting  $MSO$**  The second generalization is to add to the language a finite number of second-order predicates  $\text{Mod}_{p,q}(X)$  which mean that  $X$  is of size  $p$  modulo  $q$ . We obtain the logic called  $CMSO_M$  for counting monadic second-order. In this logic, we can express the fact that a set is a circuit of even cardinality, which is not possible in  $MSO_M$ . Theorem 6.14 also holds for  $CMSO_M$  except that the translated formula is now in  $CMSO$ . This is interesting since the model-checking of  $CMSO$  is solvable in linear time over trees [Cou92].

We could also adapt Theorem 6.14 to  $MSO_M$  problems with optimization constraints, that is finding the minimal or maximal size of a set which satisfies a formula. This kind of problem has been introduced in [ALS91] for graphs under the name of  $EMSO$ . These problems are solvable in linear time for graphs of bounded tree-width. For instance, using the formula  $A - \text{Circuit}(X)$ , we can find the size of the minimum circuit which extends a set  $A$ . When the matroid is binary and  $|A| = 1$ , it is equivalent to the problem of finding the minimum weight of a solution of an affine formula, which is NP-complete [BMVT78].

### 6.4.2 Spectra of $MSO_M$ formulas

In this subsection, Theorem 6.14 is used to prove that the generalized spectra of  $MSO_M$  formulas are semi-linear.

**Definition 6.16** (Spectrum). The spectrum of a formula  $\phi$  is the set  $\text{spec}(\phi) = \{n \mid M \models \phi \text{ and } |M| = n\}$ .

**Definition 6.17** (Ultimately periodic). A set  $X$  of integers is said to be *ultimately periodic* if there are two integers  $a$  and  $b$  such that, for  $n > a$  in  $X$  we have  $n = a + k \times b$ .

This kind of result has been proved for various restrictions of the second order logic, of the vocabulary or of the set of allowed models. The result holds for first order logic, finitely many unary relations and one unary function, cf. [DFL97]. Then it has been generalized to second order logic with the same vocabulary in [GS03]. Fischer and Makowsky obtained similar results for different notions of width on graphs in [FM04] by reduction to labeled trees. We use the same kind of method since matroids of bounded branch-width are reducible to enhanced trees by Theorem 6.14.

The following theorem is one of the simplest variant of the spectrum theorems. It comes from the pumping lemma on trees and the fact that recognizability and definability in  $MSO$  are equivalent for a set of trees (see [CDG<sup>+</sup>07]).

**Theorem 6.18.** *Let  $\phi$  be a  $MSO$  formula on labeled trees, then  $\text{spec}(\phi)$  is ultimately periodic.*

We give our corollary for a generalization of the notion of spectra and ultimately periodic sets taken from [Cou95].

**Definition 6.19.** A subset of  $\mathbb{N}^k$  is *linear* if it is of the form  $\{f_1(\vec{x}), \dots, f_k(\vec{x}) \mid \vec{x} \in \mathbb{N}^d\}$  where the  $f_i$  are affine functions. We say that a set is *semi-linear* if it is a finite union of linear sets.

**Definition 6.20.** Let  $K$  be a set of relational structures and let  $\phi(X_1, \dots, X_k)$  be a monadic second order formula where  $X_1, \dots, X_k$  are free variables. The generalized spectrum of  $\phi$  over  $K$  is the set  $\{|X_1|, \dots, |X_k| \mid S \in K, (S, X_1, \dots, X_k) \models \phi\}$ .

We can now state the main theorem of this part, which is a corollary of Theorem 6.14 and its generalizations to other logics.

**Theorem 6.21.** *Let  $\phi$  a formula of  $MSO_M$ ,  $CMSO_M$  or  $MSO_M$  over matroids or colored matroids, then the generalized spectrum of  $\phi$  over  $\mathbb{F}$ -matroids of branch-width  $t$  is semi-linear.*

*Proof.* By Theorem 6.14, we have a formula  $F(\phi) \in MSO$  such that for each enhanced tree  $T$  of width  $t$  representing a matroid  $M$ :

$$(M, \vec{A}) \models \phi(\vec{X}) \Leftrightarrow (T, f(\vec{A})) \models F(\phi(\vec{X}))$$

The generalized spectra of a  $MSO$  formula over terms is semi-linear (see Theorem 3.2 of [Cou95]). We first restrict the terms to the enhanced trees of branch-width  $t$  by a  $MSO$  formula. Therefore it holds that the generalized spectrum of  $F(\phi)$  is semi-linear over the enhanced trees. If we consider  $(T, f(A_1), \dots, f(A_k)) \models F(\phi(\vec{X}))$ , we have that  $|f(A_i)| = |A_i|$  since the free variables of  $F(\phi)$  are relativized to the leaves and in bijection with the elements of

$M$  through  $f$ . Therefore the generalized spectrum of  $\phi$  is semi-linear over the  $\mathbb{F}$ -matroids. The proof for  $CMSO_M$  or  $MSO_M$  over matroids with additional unary predicates is the same.  $\square$

### 6.4.3 Enumeration

The following theorem on enumeration and  $MSO$  over terms is proved in [Cou09] along with generalizations to other structures by  $MSO$  reduction to trees. For instance, we have a similar theorem for graph of bounded tree-width [Bag06].

**Theorem 6.22** (Courcelle [Cou09]). *Let  $\phi(X_1, \dots, X_m)$  be an  $MSO$  formula, there exists an enumeration algorithm which given a term  $T$  of size  $n$  and of depth  $d$  enumerate the  $m$ -tuples  $B_1, \dots, B_m$  such that  $T \models \phi(B_1, \dots, B_m)$  with a linear delay and a preprocessing time  $O(n \times d)$ .*

The next corollary is a direct consequence of Theorem 6.22 and of Theorem 6.14, which allows us to interpret  $MSO_M$  logic over matroids of branch-width  $t$  into  $MSO$  logic over their enhanced trees.

**Corollary 6.23.** *Let  $\phi(X_1, \dots, X_m)$  be an  $MSO_M$  formula, let  $t$  be an integer and let  $\mathbb{F}$  be a field. There is an algorithm, which given a  $\mathbb{F}$ -matroid  $M$  of branch-width less than  $t$ , enumerates the  $m$ -tuples  $B_1, \dots, B_m$  such that  $M \models \phi(B_1, \dots, B_m)$  with a linear delay after a cubic preprocessing time.*

*Proof.* Let  $\phi(X_1, \dots, X_m)$  be an  $MSO_M$  formula, we compute the formula  $F(\phi(X_1, \dots, X_m))$  for matroids of branch-width  $t$  in linear time. Then, given a matroid of branch-width  $t$ , we compute its enhanced tree in cubic time. We run the enumeration algorithm given by Theorem 6.22 on this enhanced tree and the formula  $F(\phi(X_1, \dots, X_m))$ . Each time we find a  $m$ -tuple satisfying the formula, we output its image by the bijection between the leaves of the enhanced tree and the elements of the matroid. This algorithm gives the solutions of  $\phi(X_1, \dots, X_m)$  with a linear delay and a cubic preprocessing time.  $\square$

Corollary 6.23 can be adapted to  $MSO_M$  over colored matroids and thus applied to the formula  $A - \text{Circuit}(X)$ . We obtain an algorithm in linear delay, which solves  $\text{ENUM} \cdot A - \text{Circuit}$  on matroids representable over a finite field and of branch-width  $t$ . In addition to its good delay, this algorithm is the first which solves the problem for an unbounded  $A$ . Moreover, the time it takes to output all solutions is linear in the number of solutions, while the incremental algorithm of [KBE<sup>+</sup>05] needs a time cubic in this number.

We have seen in Chapter 2 that this enumeration problem is also in **DelayP** for very dense representable matroids. The class of  $\mathbb{F}$ -matroid of bounded branch-width being very different from the later, it could be interesting to find classes of representable matroids inbetween, on which the enumeration of circuits is still in **DelayP**.



## 6.5 Matroid operations

In this section, we give two different ways to build matroids by means of some well chosen operations. We then prove that the model-checking of  $MSO_M$  is decidable in linear time on these classes of matroids. Definitions and notations are taken from [Hli06] but they are slightly more general, since we will build two different matroid grammars.

### 6.5.1 Amalgam of boundaried matrices

**Definition 6.24** (Boundaried matroid). A pair  $(M, \gamma)$  is called a  $t$  boundaried matroid if  $M$  is a matroid and  $\gamma$  is an injective function from  $[1, t]$  to  $M$  whose image is an independent set. The elements of the image of  $\gamma$  are called boundary elements and the others are called internal elements.

The restriction of  $M$  to its ground set minus the elements of the boundary is called the internal matroid of  $(M, \gamma)$ . We need an operation  $\oplus$ , which associates a matroid  $N_1 \oplus N_2$  to two  $t$  boundaried matroids  $N_1$  and  $N_2$ . By means of this operation, we try to properly define a set of terms similar to those introduced in [Hli06]. Hereafter, we explain how these terms are related to enhanced trees. The same technique will be used with a different operation in the next section.

A  $t$  boundaried matrix is a matrix  $A$  and an injective function  $\gamma$  from  $[1, t]$  to  $A$  whose image is an independent set of columns. Boundaried matrices represent boundaried matroids in the obvious way. In fact, we define the operation  $\oplus$  on boundaried matrices and not on the boundaried matroids they represent.

We want to define  $\oplus$  as the pushout (or amalgam) of the two boundaried matrices. It would then generalize the construction of decomposition trees for graphs of bounded branch-width, also obtained by a pushout in the category of graphs. By amalgam, we mean an operation such that  $A_1$  and  $A_2$  can be injected in  $A_1 \oplus A_2$  by the functions  $i_1$  and  $i_2$  respectively and such that  $i_1(\gamma_1(j)) = i_2(\gamma_2(j))$  for all  $j$ . We present a way to define such an amalgam between two vector spaces, which yields an operation on boundaried matrices.

Let  $(A_1, \gamma_1)$  and  $(A_2, \gamma_2)$  be two  $t$  boundaried matrices. We see  $A_i$  as a set of vectors in the vector space  $E_i$ . Let  $E_1 \times E_2$  be the direct product of the two vector spaces and let  $B$  be its subspace generated by the elements  $(\gamma_1(j), -\gamma_2(j))$  for all  $j$ .

**Definition 6.25.** Let  $E$  be the quotient space of  $(E_1 \times E_2)$  by  $B$ . There are natural injections from  $A_1$  and  $A_2$  into  $E_1 \times E_2$  and then in  $E$ . We write  $(A_1, \gamma_1) \oplus (A_2, \gamma_2)$  the set of vectors in  $E$  of the form  $(a_1, 0)$  with  $a_1 \in A_1 \setminus \gamma_1([1, t])$  and  $(0, a_2)$  with  $a_2 \in A_2 \setminus \gamma_2([1, t])$ .

Remark that  $(A_1, \gamma_1) \oplus (A_2, \gamma_2)$  defines a (non boundaried) matroid. To have a more specific idea of the action of  $\oplus$  and give examples, we must explain how to unambiguously represent  $(A_1, \gamma_1) \oplus (A_2, \gamma_2)$  by a matrix. Since, once a base

is chosen, a set of vectors and a matrix are the same objects, we only have to give an algorithm to build a base of  $E$ . We build a base  $B$  of  $E$  from  $C$  and  $D$ , the canonical bases of  $E_1$  and  $E_2$ . Let  $i$  be the injection from  $E_1 \cup E_2$  to  $E$ . Let  $B_0 = i(C)$  and  $B_{j+1} = B_j \cup \{i(D_{j+1})\}$  if this set is independent, otherwise  $B_{j+1} = B_j$ . Let  $n$  be the size of  $C_2$ , then  $B$  is  $B_n$ , which is by construction a base of  $E$ .

**Example 6.26.**

$$\begin{pmatrix} 1 & 0 & | & 1 \\ 0 & 1 & | & 1 \end{pmatrix} \oplus \begin{pmatrix} 1 & 0 & | & 1 & 0 \\ 0 & 1 & | & 1 & 0 \\ 0 & 1 & | & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & -1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & | & 1 \\ 0 & 1 & | & 1 \end{pmatrix} \oplus \begin{pmatrix} 1 & 0 & | & 2 & 0 \\ 0 & 1 & | & 1 & 0 \\ 0 & 1 & | & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & -1 \end{pmatrix}$$

The matrices of the example can be seen as defined over  $\mathbb{F}_3$  or any larger field. The boundary elements are the two first columns of the matrices, separated from the others by the symbol  $|$  for clarity. The image of the canonical base of  $E_1$  in  $E$  is  $\{e_1, e_2\}$  and the image of  $E_2$  is  $\{e_3, e_4, e_5\}$ . By identification of the first and second columns, we have  $e_1 = e_3$  and  $e_2 = e_4 + e_5$ . The basis built by the algorithm is thus  $\{e_1, e_2, e_4\}$ .

The column  $(1, 1, 1)^t$  of the second matrix in the left hand side of the first equation is represented in the right hand side by  $(1, 1, 0)^t$ . Indeed, once injected in  $E$ , this vector is equal to  $e_3 + e_4 + e_5$  which is equal to  $e_1 + e_2$ , the sum of the two first vectors of the base we have built.

Notice that the columns 1 and 2 of the result in the first equation form a dependent set but not in the result of the second, thus the two matrices obtained represent distinct matroids. Yet the matrices we combine by  $\oplus$ , although different, represent the same matroid in both equations.

Example 6.26 shows that  $\oplus$  cannot be seen as an operation on matroids because the result depends on the way the matroids are represented. We could also make this kind of construction by representing matroids by projective spaces, as it is done in [Hli06]. Unfortunately, we would define essentially the same operation, which would still be defined over the projective spaces and not the matroids. Nevertheless, if we restrict  $\oplus$  to matrices over  $\mathbb{F}_2$ , it properly defines an operation on the matroids they represent.

**Proposition 6.27.** *Let  $(M_1, \gamma_1)$  and  $(M_2, \gamma_2)$  be two bounded  $\mathbb{F}_2$ -matroids. For all matrices  $A_1$  and  $A_2$  which represents these matroids, the matroid represented by  $A_1 \oplus A_2$  is the same.*

*Proof.* We are going to show that the fact to be a circuit of  $A_1 \oplus A_2$  depends only on  $M_1$  and  $M_2$ . Since a matroid is entirely determined by its set of circuit, it will prove the proposition.

A circuit of  $A_1 \oplus A_2$  is the union of internal elements of  $A_1$  and  $A_2$  denoted by  $X$  and  $Y$  such that  $\sum_{x \in X} (x, 0) + \sum_{y \in Y} (0, y) \in \langle \{(\gamma_1(j), -\gamma_2(j))\}_{j \leq t} \rangle$  and  $X \cup Y$  is minimal for this property. Equivalently, there is a set  $S \subseteq [1, t]$  such that the two following relation hold:

- $\sum_{x \in X} x + \sum_{i \in S} \gamma_1(i) = 0$
- $\sum_{y \in Y} y + \sum_{i \in S} \gamma_2(i) = 0$

This is true because, the matrices  $A_1$  and  $A_2$  are defined over  $\mathbb{F}_2$ , therefore all coefficients different from zero have to be one. It is equivalent to:  $X \cup \gamma_1(S)$  is a circuit of  $A_1$ , thus of  $M_1$  and  $Y \cup \gamma_2(S)$  is a circuit of  $A_2$  thus of  $M_2$   $\square$

Behind this proof is hidden the notion of the signature of a set in a boundaried matroid that we are going to use afterwards. We now want to build matroids from successive applications of the operation  $\oplus$ .

**Definition 6.28.** Let  $A$  be a matrix and let  $\gamma_i^A$  for  $i = 1, 2, 3$  be three injective functions from  $[1, t_i]$  to the columns of  $A$ . If the sets  $\gamma_i^A([1, t_i])$  are independent and form a partition of the columns of  $A$ , then  $(A, \{\gamma_i^A\}_{i=1,2,3})$  is called a *3-partitioned matrix*.

Let  $M$  be a matroid and let  $\gamma_i^M$  for  $i = 1, 2, 3$  be three injective functions from  $[1, t_i]$  to the ground set of  $M$ . If the sets  $\gamma_i^M([1, t_i])$  are independent and form a partition of the columns of  $M$ , then  $(M, \{\gamma_i^M\}_{i=1,2,3})$  is called a *3-partitioned matroid*.

The characteristic matrices used to build the enhanced trees may be seen as 3-partitioned matrices. From  $\oplus$  and  $A$  a 3-partitioned matrix we define an operator  $\odot_A$  which associates a boundaried matrix to two boundaried matrices. It is defined by two successive uses of  $\oplus$  on the boundaries  $\gamma_1^A$  and  $\gamma_2^A$ .

**Definition 6.29.** Let  $\overline{A_1} = (A_1, \gamma_1)$  and  $\overline{A_2} = (A_2, \gamma_2)$  be respectively a  $t_1$  and a  $t_2$  boundaried matrix and let  $A$  be a 3-partitioned matrix. We call  $\overline{A_1} \odot_A \overline{A_2}$  the  $t_3$  boundaried matrix defined by  $(\overline{A_1} \oplus (A, \gamma_1^A), \gamma_2^A) \oplus \overline{A_2}$  with boundary  $\gamma_3^A$ .

The operation  $\oplus$  is ‘‘associative’’ meaning that  $\overline{A_1} \odot_A \overline{A_2}$  can also be defined by  $\overline{A_1} \oplus ((A, \gamma_2^A) \oplus \overline{A_2}, \gamma_1^A)$  with boundary  $\gamma_3^A$ .

Let  $\Upsilon$  be the set containing the two following 1-boundaried matrices:

- $\Upsilon_0$  is the matrix  $\left( \begin{array}{c|c} 1 & 0 \\ 0 & 1 \end{array} \right)$ .
- $\Upsilon_1$  is the matrix  $( 1 \mid 1 )$ .

**Definition 6.30.** Let  $\mathcal{M}_t^{\mathbb{F}}$  be the set of terms which are inductively defined by:

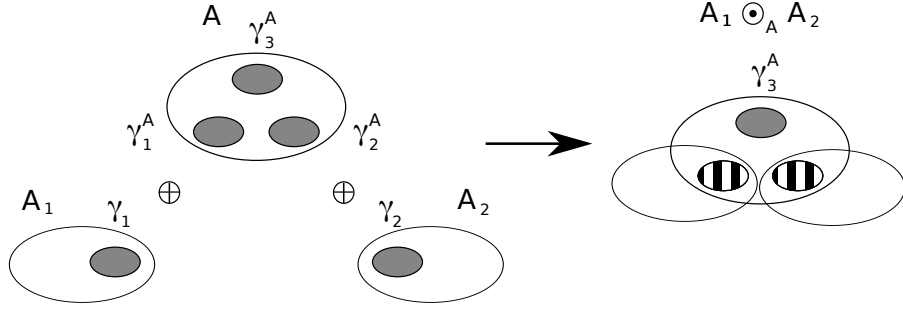


Figure 6.3: Representation of the operation  $\odot$ , boundaries represented in grey and removed parts hatched

- An element of  $\mathcal{Y}$  is a term of  $\mathcal{M}_t^{\mathbb{F}}$ .
- Let  $T_1$  and  $T_2$  be two terms of value  $A_1$  and  $A_2$  which are a  $t_1$  and a  $t_2$  boundaried matrix. Let  $A$  be a 3-partitioned matrix, its three parts being of cardinality  $t_1$ ,  $t_2$  and  $t_3$ , all less than or equal to  $t$ . Then  $A_1 \odot_A A_2$  is a term of  $\mathcal{M}_t^{\mathbb{F}}$  whose value is a  $t_3$  boundaried matrix.

The value of a term of  $\mathcal{M}_t^{\mathbb{F}}$  is a matrix with a boundary. We will not distinguish a term from its value and the matroid it represents when we remove the boundary. To study the matroids represented by these terms, we now need to define the signature of a set  $X$  in exactly the same way as for enhanced trees.

**Definition 6.31.** Let  $T$  be a term of  $\mathcal{M}_t^{\mathbb{F}}$  and let  $(A, \gamma^A)$  the boundaried matrix defined by  $T$ . We write  $l$  for the size of the boundary of  $A$ . Let  $X$  be a subset of internal elements of  $A$ . We say that  $X$  admits the signature  $\lambda = (\lambda_1, \dots, \lambda_l)$  in  $T$  if there is a nontrivial linear combination of its elements equal to  $\sum_{i \leq l} \lambda_i \gamma^A(i)$ .

The set  $X$  always admits the signature  $\emptyset$ .

We now show that the signatures in a term of  $\mathcal{M}_t^{\mathbb{F}}$  satisfy the relation  $R$  given in Definition 6.9. To this aim, we prove two lemmas similar to Lemmas 6.10 and 6.11 in which we use the following notations:

- Let  $T$  be a term of  $\mathcal{M}_t^{\mathbb{F}}$  equal to  $T_1 \odot_A T_2$ , where  $A$  is a 3-partitioned matrices.
- The terms  $T_1$ ,  $T_2$  and  $T$  represent the  $t_1$ ,  $t_2$  and  $t_3$  boundaried matrices  $(N_1, \gamma_1)$ ,  $(N_2, \gamma_2)$  and  $(N, \gamma_3)$ .
- Let  $E_1$ ,  $E_2$  and  $E_3$  be the vector spaces generated by the columns of  $N_1$ ,  $N_2$  and  $A$ .
- Let  $V$  be the vector space  $E_1 \times E_2 \times E_3$ .
- Let  $B$  be  $\langle \{(\gamma_1(j), 0, 0) - (0, 0, \gamma_1^A(j))\} \rangle$  and  $C$  be  $\langle \{(0, \gamma_2(j), 0) - (0, 0, \gamma_2^A(j))\} \rangle$ .

- Let  $E$  be the quotient of  $V$  by  $B$  and then by  $C$ , it is the vector space which is used to define  $(N, \gamma_3)$ .
- Let  $\phi_1, \phi_2$  and  $\phi_3$  be the injection of  $E_1, E_2$  and  $E_3$  in  $E$ .

**Lemma 6.32.** *Let  $X_1$  and  $X_2$  be two sets of internal elements of  $N_1$  and  $N_2$ . If  $X_1$  admits  $\mu$  in  $T_1$ ,  $X_2$  admits  $\delta$  in  $T_2$  and  $R(A, \lambda, \mu, \delta)$  holds then  $X = X_1 \cup X_2$  admits  $\lambda$  in  $T$ .*

*Proof.* By definition of the signature, we know that a nontrivial combination of internal elements of  $X_1$  (respectively of  $X_2$ ) is equal to  $\sum_{1 \leq i \leq t_1} \mu_i \gamma_1(i)$  (respectively to  $\sum_{1 \leq j \leq t_2} \delta_j \gamma_2(j)$ ). Therefore, there is a combination of elements of  $X_1 \cup X_2$  seen as elements of  $E$  which we write  $v$  and which satisfies:

$$v = \phi_1 \left( \sum_{1 \leq i \leq t_1} \mu_i \gamma_1(i) \right) + \phi_2 \left( \sum_{1 \leq j \leq t_2} \delta_j \gamma_2(j) \right)$$

Since  $\phi_1(\gamma_1(i)) = \phi_3(\gamma_1^A(i))$  and  $\phi_2(\gamma_2(j)) = \phi_3(\gamma_2^A(j))$  for all  $i, j$ ,

$$v = \phi_3 \left( \sum_{1 \leq i \leq t_1} \mu_i \gamma_1^A(i) \right) + \phi_3 \left( \sum_{1 \leq j \leq t_2} \delta_j \gamma_2^A(j) \right)$$

Moreover,  $\phi_3$  is a linear function, therefore we have:

$$v = \phi_3 \left( \sum_{1 \leq i \leq t_1} \mu_i \gamma_1^A(i) + \sum_{1 \leq j \leq t_2} \delta_j \gamma_2^A(j) \right)$$

Because  $R(A, \lambda, \mu, \delta)$  holds, we have the equality

$$\sum_{1 \leq k \leq t_3} \lambda_k \gamma_3^A(k) = \sum_{1 \leq i \leq t_1} \mu_i \gamma_1^A(i) + \sum_{1 \leq j \leq t_2} \delta_j \gamma_2^A(j)$$

This equality yields

$$v = \phi_3 \left( \sum_{1 \leq k \leq t_3} \lambda_k \gamma_3^A(k) \right) = \sum_{1 \leq k \leq t_3} \lambda_k \gamma_3(k)$$

It means that  $X = X_1 \cup X_2$  admits the signature  $\lambda$  in  $T$ , since  $\gamma_3$  is the boundary of  $N$ .  $\square$

**Lemma 6.33.** *Let  $X_1$  and  $X_2$  be two sets of internal elements of  $N_1$  and  $N_2$ . If  $X = X_1 \cup X_2$  admits  $\lambda$  in  $T$ , then there are two signatures  $\mu$  and  $\delta$  such that  $R(A, \lambda, \mu, \delta)$  holds,  $X_1$  admits  $\mu$  in  $T_1$  and  $X_2$  admits  $\delta$  in  $T_2$ .*

*Proof.* Since  $X$  admits  $\lambda$  in  $T$ , there is a linear combination of elements of  $X$  equal to  $\phi_3 \left( \sum_{1 \leq k \leq t_3} \lambda_k \gamma_3(k) \right)$ . It is equivalent to say that we have the following equality in  $V$ :

$$(v_1, 0, 0) + (0, v_2, 0) + (b_1, 0, b_2) + (0, c_1, c_2) = \sum_{1 \leq k \leq t_3} (0, 0, \lambda_k \gamma_3^A(k)), \quad (6.2)$$

where  $(v_1, 0, 0)$  is a combination of elements of  $X_1$  injected in  $V$ ,  $(0, v_2, 0)$  is a combination of elements of  $X_2$  injected in  $V$ ,  $(b_1, 0, b_2) \in B$  and  $(0, c_1, c_2) \in C$ . Since  $(b_1, 0, b_2)$  is in  $B$ , there is a signature  $\mu$  such that it is equal to:

$$\sum_{1 \leq i \leq t_1} (\mu_i \gamma_1(i), 0, -\mu_i \gamma_1^A(i))$$

In the same way, there is a signature  $\delta$  such that  $(0, c_1, c_2)$  is equal to:

$$\sum_{1 \leq j \leq t_2} (0, \delta_j \gamma_1(j), -\delta_j \gamma_1^A(j))$$

Equation 6.2 implies that  $v_1 = -b_1$  and  $v_2 = -c_1$ , therefore  $X_1$  is of signature  $\mu$  in  $T_1$  and  $X_2$  is of signature  $\delta$  in  $T_2$ . We also deduce from Equation 6.2:

$$b_2 + c_2 = \sum_{1 \leq k \leq t_3} (0, 0, \lambda_k \gamma_3^A(k))$$

Therefore  $R(A, \lambda, \mu, \delta)$  holds.  $\square$

By means of these two lemmas, we can prove that enhanced trees of width  $t$  and  $\mathcal{M}_t^{\mathbb{F}}$  are the same object. Let  $g$  be the afterdefined bijection between the enhanced trees of width  $t$  and  $\mathcal{M}_t^{\mathbb{F}}$ . Let  $T$  be an enhanced tree, one replaces  $A$  on each internal node by  $\odot_A$  (a characteristic matrix is a 3-partitioned matrix). The images of the leaves labeled (0) and (1) are the constants  $\mathcal{T}_0$  and  $\mathcal{T}_1$  respectively .

**Theorem 6.34.** *Let  $M$  be a  $\mathbb{F}$ -matroid, then  $T$  is one of its enhanced tree of width  $t$  if and only if  $M$  is the value of the term  $g(T)$ .*

*Proof.* One can prove a theorem of characterization of dependent sets by the signatures on the terms of  $\mathcal{M}_t^{\mathbb{F}}$  identical to Theorem 6.13, using Lemmas 6.32 and 6.33. Therefore  $T$  and  $g(T)$  define the same matroid.  $\square$

**Example 6.35.** We give here the matrices, with their boundary on the left side, which are constructed when evaluating the term of Fig. 6.4

$$\begin{aligned} M_{s_3} &= \left( \begin{array}{c|cc} 0 & 1 & 0 \\ 1 & 0 & 1 \end{array} \right) & M_{s_4} &= \left( \begin{array}{c|cc} 1 & 1 & 0 \\ 0 & 1 & 1 \end{array} \right) \\ M_{s_1} &= \left( \begin{array}{c|ccc} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{array} \right) & M_{s_2} &= \left( \begin{array}{c|ccc} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array} \right) \end{aligned}$$

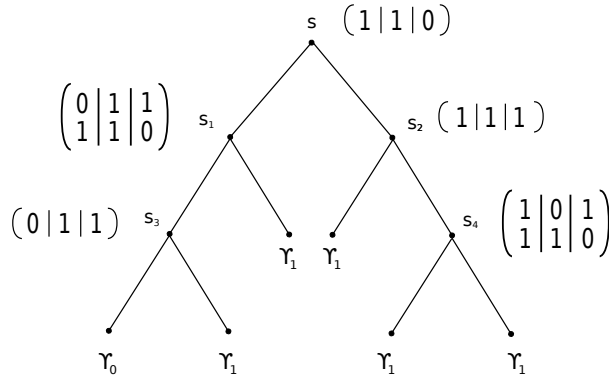


Figure 6.4: The term associated to the enhanced tree of Fig. 6.2

$$M_s = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

The matrix  $M_s$  represents the same matroid as the matrix  $X$  of Fig. 6.1 which was used to find an enhanced tree and then a term as explained in the proof of the previous theorem.

### 6.5.2 Series and parallel connections

In this subsection we consider two of the most simple operations on matroids, called the series and parallel connections. They extend well-known graph operations, which are used to characterize the graphs of tree-width 2 [Bod98]. By means of these operations, we describe a class of matroids, which are not all representable, using the methods introduced in the previous subsection. The following definition and theorem are taken from [Oxl92].

**Definition 6.36.** Let  $M_1$  and  $M_2$  be two 1 boundaried matroids of ground set  $S_1$  and  $S_2$ . Their respective boundaries are  $\{p_1\}$  and  $\{p_2\}$ . We denote by  $\mathcal{C}(M)$  the collection of circuits of the matroid  $M$ . Let  $E$  be the set  $S_1 \cup S_2 \cup \{p\} \setminus \{p_1, p_2\}$ . We define two collections of subsets of  $E$ :

$$C_S = \left\{ \begin{array}{l} \mathcal{C}(M_1 \setminus \{p_1\}) \cup \mathcal{C}(M_2 \setminus \{p_2\}) \\ \cup \{C_1 \setminus \{p_1\} \cup C_2 \setminus \{p_2\} \cup \{p\} \mid p_i \in C_i \in \mathcal{C}(M_i)\} \end{array} \right.$$

$$C_P = \left\{ \begin{array}{l} \mathcal{C}(M_1 \setminus \{p_1\}) \cup \mathcal{C}(M_2 \setminus \{p_2\}) \\ \cup_{i=1,2} \{C_i \setminus \{p_i\} \cup \{p\} \mid p_i \in C_i \in \mathcal{C}(M_i)\} \\ \cup \{C_1 \setminus \{p_1\} \cup C_2 \setminus \{p_2\} \mid p_i \in C_i \in \mathcal{C}(M_i)\} \end{array} \right.$$

**Theorem 6.37.** The sets  $C_S$  and  $C_P$  are collections of circuits of a matroid on  $E$ .

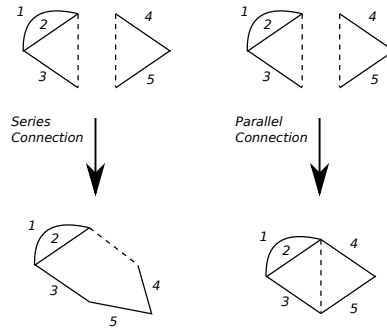


Figure 6.5: Example of series and parallel connections over graphs with boundaries represented by a dotted line

The matroid defined by  $C_P$  is called the parallel connection of  $M_1$  and  $M_2$  while the one defined by  $C_S$  is the series connection of  $M_1$  and  $M_2$ .

**Definition 6.38.** We write  $M_1 \oplus_p M_2$  for the parallel connection of  $M_1$  and  $M_2$  restricted to the ground set  $S_1 \cup S_2 \setminus \{p_1, p_2\}$  (one removes the boundary  $\{p\}$ ).

The operator  $\oplus_p$  is known under the name of *2 sum* (see [Oxl92]). We could also consider an operator  $\oplus_s$ , but it is only the direct sum of two matroids and it will not enlarge the class of matroids we are about to define. We now consider the operation  $\odot$  defined as in Definition 6.29, except that  $\oplus$  is replaced by  $\oplus_p$ .

**Definition 6.39.** Let  $\mathcal{L}_k$  be the set of 1 boundaried matroids of size at most  $k$  and let  $\mathcal{M}$  be the set of 3-partitioned matroids of size 3. We write  $\mathcal{T}_k$  for the set of terms  $T(\mathcal{L}_k, \mathcal{M})$ .

A term of  $\mathcal{T}_k$  has for value a 1 boundaried matroid. Notice that the class of boundaried matroids of size  $k$  closed by the series parallel operation is strictly larger than  $\mathcal{T}_k$ . Indeed, when one builds a term, the position of the boundary is imposed. It could be interesting to extend the result of this section to this broader class.

A term of  $\mathcal{T}_k$  can have a non representable matroid for value, since the constants at the leaves are arbitrary matroids. Therefore the matroids represented by elements of  $\mathcal{T}_k$  and elements of  $\mathcal{M}_t^{\mathbb{F}}$  are different. Nevertheless there is a relation between the operations  $\oplus_p$  and  $\oplus$  as illustrated by the next proposition.

**Proposition 6.40.** Let  $M_1$  (resp.  $M_2$ ) be a matroid of boundary  $\{p_1\}$  (resp.  $\{p_2\}$ ) represented by the sets of vectors  $A_1$  (resp.  $A_2$ ). Then  $A_1 \oplus A_2$  represents the matroid  $M_1 \oplus_p M_2$ .

*Proof.* We prove that the dependent sets of  $A_1 \oplus A_2$  are the same as the dependent sets of  $M_1 \oplus_p M_2$ . In fact, we only show that a dependent set  $D$  of  $M_1 \oplus_p M_2$  is a dependent set of  $A_1 \oplus A_2$ . The converse is easy and left to the reader. By the definition of  $\oplus_p$ , the dependent set  $D$  can be of two different kinds. It may be the



image of a dependent set of  $M_1 \setminus \{p_1\}$  or  $M_2 \setminus \{p_2\}$ , it is then trivially a dependent set of  $A_1 \oplus A_2$ .

Assume now that  $D = D_1 \setminus \{p_1\} \cup D_2 \setminus \{p_2\}$ , where  $D_1$  is a dependent set of  $M_1$  containing  $\{p_1\}$  and  $D_2$  a dependent set of  $M_2$  containing  $\{p_2\}$ . Since  $M_1$  and  $M_2$  are represented by  $A_1$  and  $A_2$ , we have the following linear dependence relations of their columns in bijection with  $D_1$  and  $D_2$ :

$$\lambda_1 p_1 + \sum \alpha_i A_1^i = 0 \text{ and } \lambda_2 p_2 + \sum \beta_i A_2^i = 0$$

By linear combination of the two previous equalities we get:

$$\lambda_1(p_1 - p_2) + \sum \alpha_i A_1^i + \sum -\lambda_1 \lambda_2^{-1} \beta_i A_2^i = 0$$

In  $A_1 \oplus A_2$ , we have  $p_1 = p_2$  therefore, the equation becomes:

$$\sum \alpha_i A_1^i + \sum -\lambda_1 \lambda_2^{-1} \beta_i A_2^i = 0$$

This last equation proves that  $D$  is dependent in  $A_1 \oplus A_2$ . □

It seems that the previous lemma would fail for generalizations of  $\oplus_p$  to a boundary larger than one. Indeed,  $\oplus$  is not an operation on matroids as seen in Example 6.26 with a boundary of size two. In fact, one of the natural generalizations of  $\oplus_p$  to boundary of size  $k$  is the  $k$  sum (see [Ox192]) which is defined on binary matroids only.

**Corollary 6.41.** *A matroid defined by a term of  $\mathcal{T}_k$  whose constants are  $\mathbb{F}$ -matroids is an  $\mathbb{F}$ -matroid of branch-width at most  $k$ .*

*Proof.* By structural induction on the terms of  $\mathcal{T}_k$  whose constants are  $\mathbb{F}$ -matroids. The constants are matroids of size  $k$  because they are in  $\mathcal{T}_k$  and they are representable by hypothesis, hence they are of branch-width at most  $k$ . Assume now that  $T = T_1 \odot_M T_2$ , where the values of  $T_1$  and  $T_2$  are matroid of branch-width  $k$  represented by  $A_1$  and  $A_2$  respectively. All matroids of size 3 are cycle matroids and hence are representable in any field. Therefore  $M$  is represented by the 3-partitioned matrix  $A$  over  $\mathbb{F}$ . Using the previous proposition, we have that  $A_1 \odot_A A_2$  represents the same boundaried matroid as  $T = T_1 \odot_M T_2$ . Finally, Theorem 6.34 proves that  $A_1 \odot_A A_2$  is of branch-width  $k$ , which completes the proof. □

We now define a very general notion of signature to use the previously introduced technique and illustrate it in this setting. A signature describes which sets of elements of the boundary make a set of internal elements dependent. Notice that, contrary to the representable matroid case, the signature of a set is unique. We could use this notion of signature for other operations than  $\oplus_p$ , over matroids of boundary bigger than 1.

**Definition 6.42** (Signature). Let  $T$  be a term whose value is a boundaried matroid  $M$  and let  $X$  be a set of internal elements of  $M$ . The signature of the set  $X$  in  $T$  is the set of all the subsets  $S$  of the boundary such that  $X \cup S$  is a dependent set in  $M$ .

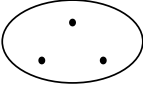
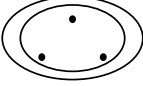
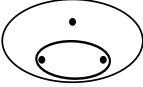
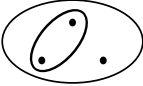
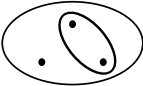
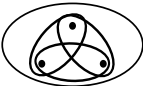
In our setting, the term is in  $\mathcal{T}_k$ , thus there is only one boundary element denoted by 1. We have only three different signatures:

1. if  $X$  is dependent then it is of signature  $\{\{\}, \{1\}\}$  that we denote by **2**
2. if  $X$  is dependent only when we add the boundary element then it is of signature  $\{\{1\}\}$  which we denote by **1**
3. if  $X$  is independent even with the boundary element then it is of signature  $\emptyset$  which we denote by **0**

Note that an empty set is of signature **0**, because the boundary is an independent set. We now prove in this context a result similar to Lemma 6.10. We will not need an equivalent of Lemma 6.11, since here the signatures are unique.

**Lemma 6.43.** *There is a relation  $R_p(\mu, \delta, \lambda, N)$ , where the first three arguments are signatures and  $N$  is a 3-partitioned matroid of size 3, such that the following holds. Let  $T = T_1 \odot_N T_2$  be a term of  $\mathcal{T}_k$ , let  $X_1$  and  $X_2$  be sets of internal elements of the boundaried matroids represented by respectively  $T_1$  and  $T_2$ . If the set  $X_1$  is of signature  $\mu$  in  $T_1$ , the set  $X_2$  is of signature  $\delta$  in  $T_2$  and  $R_p(\mu, \delta, \lambda, N)$  holds, then the set  $X_1 \cup X_2$  is of signature  $\lambda$  in  $T$ .*

*Proof.* There are six 3-partitioned matroids of size 3, which we denote by  $N_i$  for  $i = 1, \dots, 6$ . We represent each of them by three points in an ellipse. The bottom left point is  $\gamma_1^{N_i}(1)$ , the bottom right one is  $\gamma_2^{N_i}(1)$  and the top one is  $\gamma_3^{N_i}(1)$ . The smaller ellipses enclosing points represent the circuits of the matroid. We give here the value of the relation  $R_p$  for each  $N_i$ . One may then easily check that the proposition holds.

$N_1$		$R_p(\cdot, \cdot, \mathbf{2}, N_1) = \{(\mathbf{0}, \mathbf{2}), (\mathbf{2}, \mathbf{0}), (\mathbf{1}, \mathbf{2}), (\mathbf{2}, \mathbf{1}), (\mathbf{2}, \mathbf{2})\}$ $R_p(\cdot, \cdot, \mathbf{1}, N_1) = \{\}$
$N_2$		$R_p(\cdot, \cdot, \mathbf{2}, N_2) = \{(\mathbf{0}, \mathbf{2}), (\mathbf{2}, \mathbf{0}), (\mathbf{1}, \mathbf{2}), (\mathbf{2}, \mathbf{1}), (\mathbf{2}, \mathbf{2})\}$ $R_p(\cdot, \cdot, \mathbf{1}, N_2) = \{(\mathbf{1}, \mathbf{1})\}$
$N_3$		$R_p(\cdot, \cdot, \mathbf{2}, N_3) = \{(\mathbf{0}, \mathbf{2}), (\mathbf{2}, \mathbf{0}), (\mathbf{1}, \mathbf{2}), (\mathbf{2}, \mathbf{1}), (\mathbf{2}, \mathbf{2}), (\mathbf{1}, \mathbf{1})\}$ $R_p(\cdot, \cdot, \mathbf{1}, N_3) = \{\}$
$N_4$		$R_p(\cdot, \cdot, \mathbf{2}, N_4) = \{(\mathbf{0}, \mathbf{2}), (\mathbf{2}, \mathbf{0}), (\mathbf{1}, \mathbf{2}), (\mathbf{2}, \mathbf{1}), (\mathbf{2}, \mathbf{2})\}$ $R_p(\cdot, \cdot, \mathbf{1}, N_4) = \{(\mathbf{1}, \mathbf{1}), (\mathbf{1}, \mathbf{0})\}$
$N_5$		$R_p(\cdot, \cdot, \mathbf{2}, N_5) = \{(\mathbf{0}, \mathbf{2}), (\mathbf{2}, \mathbf{0}), (\mathbf{1}, \mathbf{2}), (\mathbf{2}, \mathbf{1}), (\mathbf{2}, \mathbf{2})\}$ $R_p(\cdot, \cdot, \mathbf{1}, N_5) = \{(\mathbf{1}, \mathbf{1}), (\mathbf{0}, \mathbf{1})\}$
$N_6$		$R_p(\cdot, \cdot, \mathbf{2}, N_5) = \{(\mathbf{0}, \mathbf{2}), (\mathbf{2}, \mathbf{0}), (\mathbf{1}, \mathbf{2}), (\mathbf{2}, \mathbf{1}), (\mathbf{2}, \mathbf{2}), (\mathbf{1}, \mathbf{1})\}$ $R_p(\cdot, \cdot, \mathbf{1}, N_5) = \{(\mathbf{0}, \mathbf{1}), (\mathbf{1}, \mathbf{0})\}$

□

We give in the proof the value of the relation  $R_p$  which plays the same role as  $R$  in Lemma 6.10. The precise value of  $R_p$  is not important for the proof: what matters is that it only depends on  $\mu$ ,  $\delta$ ,  $\lambda$  and  $N$ , but not on  $X_1$ ,  $X_2$  or  $T$ .

A close examination of the operations  $\odot_{N_i}$  in the previous proof shows that we already know three of them:

- $M_1 \odot_{N_1} M_2$  is the matroid given by the direct sum of  $M_1$  and  $M_2$  with boundary  $\gamma_3^{N_1}$ .
- $M_1 \odot_{N_2} M_2$  is the matroid given by the series connection of  $M_1$  and  $M_2$  with boundary  $\gamma_3^{N_2}$ .
- $M_1 \odot_{N_6} M_2$  is the matroid given by the parallel connection of  $M_1$  and  $M_2$  with boundary  $\gamma_3^{N_6}$ .

Observe that a leaf of a term of  $\mathcal{T}_k$  represents a matroid of size less than  $k$ , while a leaf of a term in  $\mathcal{M}_t^{\mathbb{F}}$  represents one element of the matroid it defines. To use our method on terms of  $\mathcal{T}_k$ , it is convenient to modify them. At each leaf labeled by an abstract 1 boundaried matroid  $M$ , we root a binary tree with as many leaves as internal elements in  $M$ . We denote by  $\tilde{\mathcal{T}}_k$ , the sets of terms of  $\mathcal{T}_k$  transformed this way. We now have a bijection between the leaves of a term of  $\tilde{\mathcal{T}}_k$  and the elements of the matroid it represents.

**Theorem 6.44** (Characterization of dependence). *Let  $T$  be a term of  $\tilde{\mathcal{T}}_k$  which represents the matroid  $M$  and let  $X$  be a set of elements of  $M$ . The set  $X$  is*

dependent if and only if there exists a signature  $\lambda_s$  at each node  $s$  of  $T$  seen as a labeled tree:

1. if  $s_1$  and  $s_2$  are the children of  $s$  of label  $\odot_N$  then  $R_p(\lambda_{s_1}, \lambda_{s_2}, \lambda_s, N)$
2. if  $s$  is labeled by an abstract boundaried matroid  $N$ , then  $X \cap N$  is a set of signature  $\lambda_s$  in  $N$
3. the signature at the root is **2**

*Proof.* Let us remark that the set  $X$  is dependent in  $M$  if its signature contains the set  $\{\}$ , i.e. if it is **2**. We thus have to prove by induction on  $T$  that  $\lambda_s$  is the signature of  $X$  in  $T_s$ . The base case is given by the condition 2, while Lemma 6.43 and condition 1 allow us to prove the induction step.  $\square$

The function  $F$  we use in the next theorem is the same as in Section 6.3. It associates a formula of  $MSO$  to a formula of  $MSO_M$  by relativization to the leaves and the use of a formula  $dep$ , whose new definition is given in the proof of the next theorem.

**Theorem 6.45.** *There exists a mapping  $F$  such that if  $T$  is a term of  $\tilde{\mathcal{T}}_k$  which represents the matroid  $M$  and if  $f$  is the bijection between the leaves of  $T$  and the elements of  $M$  then  $M \models \phi(\vec{a}) \Leftrightarrow T \models F(\phi(f(\vec{a})))$ .*

*Proof.* The demonstration is done by the construction of a formula  $dep(Y)$  satisfying the conditions of the characterization theorem. We use the formulas defined in the proof of Theorem 6.14, condition 1 is implemented by the formula  $\Psi_1$  except that  $R$  is now the relation  $R_p$ . In  $\Psi_3$ , we replace  $X_{(0,\dots,0)}$  by  $X_{\mathbf{2}}$  to satisfy condition 3.

Let  $Q(S, N, \lambda)$  be the relation which is true if and only if  $S$  is a subset of the boundaried matroid  $N$  of signature  $\lambda$ . Recall that the set of signatures  $\lambda_s$  is represented by a set of second-order variables  $\vec{X}$ . To enforce condition 2, we define a formula  $\Psi_4(X, \vec{X}, s)$ . It is true if and only if each internal node  $s$  of signature  $\lambda$  is labeled by a boundaried matroid  $N$  and  $\lambda$  is indeed the signature of the intersection of  $Y$  with  $N$ . We write  $Y \cap N = S$  for the fact that the elements of  $Y$  which are leaves of a subtree rooted in a node labeled by the boundaried matroid  $N$  form the subset  $S$  of  $N$ . One may check that it is expressible by a  $MSO$  formula.

$$\Psi_2(Y, \vec{X}, s) = \bigwedge_{(N, S \subseteq N), \lambda} (\text{label}(s) = N \wedge X_\lambda(s) \wedge Y \cap N = S) \Rightarrow Q(S, N, \lambda)$$

This formula is a conjunction on all boundaried matroids  $N$  of size  $k$  and their subsets, which are in number bounded by  $2^{2^k}$ , and on the three possible signatures. We define the formula  $dep$  of size  $O(2^{2^k})$ :

$$dep(Y) \equiv \exists \vec{X} \Omega(\vec{X}) \wedge \Psi_1(\vec{X}) \wedge \Psi_2(Y, \vec{X}) \wedge \Psi_3(\vec{X})$$

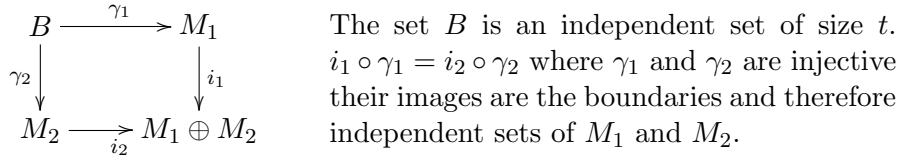


Figure 6.6: The diagram of the amalgam/pushout

The characterization theorem proves that the formula  $dep$  is correct and the theorem is then obtained by a simple induction on the formula.  $\square$

**Corollary 6.46.** *The model-checking problem of  $MSO_M$  is decidable in time  $f(k, l) \times n$  over the set of matroids given by a term of  $\mathcal{T}_k$ , where  $n$  is the number of elements in the matroid,  $l$  is the size of the formula and  $f$  is a computable function.*

## 6.6 Discussion

In this chapter, we have studied the representable matroids of bounded branch-width. We have given a new proof of the fact that model-checking of  $MSO_M$  over them can be done in polynomial time (linear if a suitable representation is given). Moreover we have linked together the notion of enhanced tree, adapted from the branch decomposition, and the terms of  $\mathcal{M}_t^{\mathbb{F}}$ . In both cases, we use the same tools, namely the relation  $R$  and the characterization of dependent sets through signatures and  $R$ .

We have also introduced the set of terms  $\mathcal{T}_k$ , which represent matroids different from  $\mathcal{M}_t^{\mathbb{F}}$ . We have then used the exact same method with signatures and a relation  $R_p$  to characterize the dependent sets in a matroid represented by a term of  $\mathcal{M}_t^{\mathbb{F}}$ . In fact, we could use this method on any term built from an operation  $M_1 \oplus M_2$ , such that  $M_1$  and  $M_2$  are restrictions of  $M_1 \oplus M_2$ . In other words, the operation has to be derived from an amalgam or push-out over a class of matroids. If we rephrase this in the language of category, it means the following. The matroid  $M_1 \oplus M_2$  is the limit of the diagram of Figure 6.6 in the category of finite matroids whose arrows are morphisms preserving the dependence relation. For this limit to exist, we have to restrict the class of finite matroid to one of its proper subclass (see [Ox192] Chapter 12).

**Possible extensions** One could lift the condition that the boundaries are independent sets. We obtain a larger set of terms build from  $\oplus$  on which the model-checking of  $MSO_M$  is still decidable in linear time. The question is: will we really capture more matroids in this way?

One can extend the operation  $\oplus_p$  to a boundary of size  $k$ , into an operation called the generalized parallel connection (see [Ox192]). However, it is defined only

for very few bounded matroids. It could be interesting to study the set of terms obtained from this operation and suitable 3-partitioned matroids.

A very simple generalization, that we do not give here is to mix  $\oplus$  and  $\oplus_p$ . In this way we obtain terms with two natural parameters on which all the techniques introduced in this chapter work easily. The only thing to do is to explain how a signature for  $\oplus$  can be transformed into a signature for  $\oplus_p$ .

Finally, since there are few operations on abstract matroids, we can restrict our attention to matroid subclasses different from cycle matroids or vector matroids. We could for instance try to find grammars to generate bicircular matroids, transversal matroids or gammoids. In the next chapter, we try to develop the method used in this chapter to deal with hypergraphs or structures represented by hypergraphs.

One other open question is to devise a decomposition algorithm, which given a matroid outputs a term of  $\mathcal{T}_k$  with  $k$  optimal or within a constant factor of the optimum.

## Chapter 7

# Decomposition of Hypergraph Like Structures

In this exploratory chapter, we try to give an abstract approach to the method with signatures and matroid grammars we have presented. We provide a way to represent hypergraphs by terms. By means of this method, we can encode essentially any grammar for hypergraphs (and thus for matroids or graphs). We prove that any *MSO* formula over hypergraphs given by these terms can be translated into a *MSO* formula over the terms. Therefore the model-checking problem is linear time decidable for these decomposed hypergraphs. This framework can be useful to simplify proofs that model-checking is easy on a class of objects. To do this, we only need to prove that a particular class of hypergraphs or hypergraph like structures can be represented by the terms we introduce.

### 7.1 Representation of a hypergraph

In this part, we define hypergraphs by terms. In fact, the construction allows us to define any structure which is a hypergraph with additional properties like to be a matroid. We could even generalize the next construction to second-order relation of larger arity.

**Definition 7.1.** Let  $F_t$  be the set of functions from  $[0, t] \times [0, t]$  to  $[0, t]$ . We denote by  $\mathcal{H}_t$  the set of terms  $T(F_t, \{l\})$ .

The constant  $l$  is arbitrary and only indicates that an element is a leaf. Therefore the terms of  $\mathcal{H}_t$  have no well-defined value. We now explain how they represent hypergraphs, by means of a suitable substitution of 0 or 1 to the labels of the leaves.

**Definition 7.2** (Value of a set). Let  $T$  be a term of  $\mathcal{H}_t$ , let  $L$  be the set of leaves of  $T$  and let  $X$  be one of its subsets. The value of the term  $T$  where the labels of the leaves in  $X$  are replaced by 1 and the others are replaced by 0 is called the value of  $X$  in  $T$  and is denoted by  $v(X, T)$ .

A term of  $\mathcal{H}_t$  is meant to represent a hypergraph whose base set is the set of all leaves of the term. The notion of value of a set is used to define the hyperedges.

**Definition 7.3** (Hypergraph represented by a term). Let  $T$  be a term of  $\mathcal{H}_t$  and let  $L$  be the set of leaves of  $T$ . The hypergraph  $H_T$  represented by  $T$  has  $L$  for vertices and its set of hyperedges is  $\{X \subseteq L \mid v(X, T) = 1\}$ .

In fact, a term of  $\mathcal{H}_t$  with  $n$  leaves defines a function from  $[0, t]^n$  to  $[0, t]$ . When restricted on  $\{0, 1\}^n$ , and assuming it takes values in  $\{0, 1\}$ , it can be seen as the sum of the indicator functions of the edges of the represented hypergraph.

**Example 7.4.** The hypergraph  $H_{k,n}$  which has  $n$  vertices and all edges of size  $k$  is represented by a term of  $\mathcal{H}_{k+1}$ . In fact, it is represented by any binary tree with  $n$  leaves, whose nodes are labeled by the function:

$$f(x, y) = \begin{cases} \text{if } x + y \leq k, & x + y \\ \text{otherwise,} & k + 1 \end{cases}$$

and whose root is labeled by  $g$  such that  $g(x, y) = 1$  if and only if  $x + y = k$ .

It is easy to prove that it is not represented by a term of  $\mathcal{H}_k$ .

**Example 7.5.** Here are the tables of the four functions used in the next example:

$f_1$		0		1
0		1		0
1		0		1

$f_2$		0		1
0		0		1
1		0		0

$f_3$		0		1
0		0		0
1		1		0

$f_4$		0		1
0		0		0
1		0		1

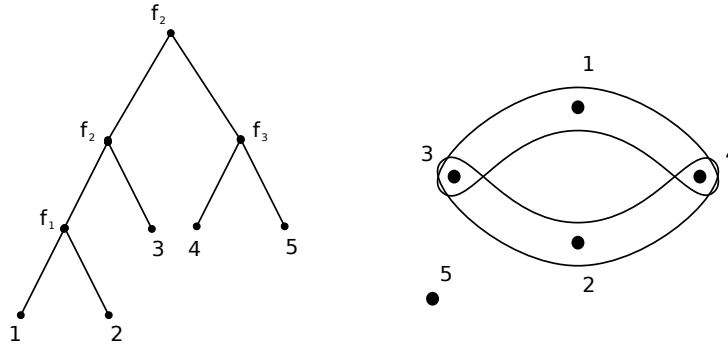


Figure 7.1: A term of  $\mathcal{H}_1$  and its associated hypergraph

**Remark 7.6.** We give here three variations of the definition of  $\mathcal{H}_t$  and of the way it defines hypergraphs. We explain that it does not change the class of hypergraphs represented by terms of  $\mathcal{H}_t$ . We later use these variations as “syntactic sugars“ to simplify some proofs.

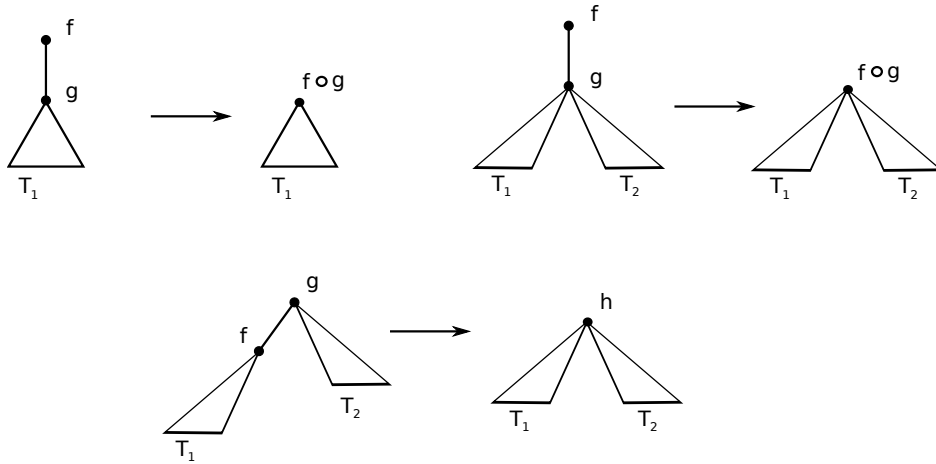
1. Let  $T$  be a term of  $\mathcal{H}_t$  and let  $S$  be a set of integers less than or equal to  $t$ . The hypergraph reresented by  $(T, S)$  has for leaves the set of leaves of  $T$  and



for hyperedges the set  $\{X \subseteq L \mid v(X, T) \in S\}$ . The set of hypergraphs represented in this way is not larger. Let  $\tilde{T}$  be the term  $T$  where one composes the function labeling the root with a function which maps the elements of  $S$  to 1 and the others to 0. The hypergraph represented by  $\tilde{T}$  is isomorphic to the hypergraph represented by  $(T, S)$ .

2. One may want to label the leaves by integers less than or equal to  $t$ . The value of a set  $X$  is then the value of the term  $T$  where the labels of the leaves not in  $X$  are replaced by 0. Again, it does not define a larger class of hypergraphs. Let  $s$  be a node of  $T$  labeled by  $f$ , its left child is the constant  $k$  and its right child is a subterm denoted by  $T_1$ . Let  $\tilde{f}(x, y) = f(g(x), y)$  where  $g$  maps 1 to  $k$  and is the identity otherwise. One replaces  $f(k, T_1)$  by  $\tilde{f}(l, T_1)$ . By doing this transformation for each constant, we obtain a term of  $\mathcal{H}_t$  which represents the same hypergraph as  $T$ .

3. Finally one may allow the set of functions  $F_t$  to also contain the functions from  $[0, t]$  to  $[0, t]$ . Let  $T$  be a term which contains such unary functions. We now give a visual representation of the three situation in which a unary function may appear if the term has at least two leaves. For each situation, we provide a local rule to remove the unary function, which proves that they do not bring any more expressive power. The unary function is denoted by  $f$ ,  $T_1$  and  $T_2$  are subterms and  $h$  is the function defined by  $h(x, y) = g(f(x), y)$ .



Remark that one can represent the hypergraph with one vertex and no edge in these three variations, while the original definition did not allow to represent it. In fact in all the previous transformations, we have assumed that there is at least two leaves (or equivalently a function) in the term.

## 7.2 Decomposition-width of a hypergraph and its properties

The grammar  $\mathcal{H}_t$  can be used to define a width parameter, that we call decomposition-width, since it is very similar to the parameter of the same name introduced in [Kra09].

**Definition 7.7.** Let  $H$  be a hypergraph. The decomposition-width of  $H$ , written  $dw(H)$ , is the smallest integer  $t$  such that  $H$  is represented by a term of  $\mathcal{H}_t$ .

The question is now, what are the hypergraphs of decomposition-width  $t$  and which properties do they enjoy?

**Lemma 7.8.** Let  $H$  be a hypergraph with  $n$  vertices and  $k$  edges denoted by  $E_1, \dots, E_k$ , there is a term  $T$  of  $\mathcal{H}_k$  whose leaves are in bijection with the vertices of  $H$  such that  $v(E_i, T) = i$  and if  $X$  is not an edge,  $v(X, T) = 0$ .

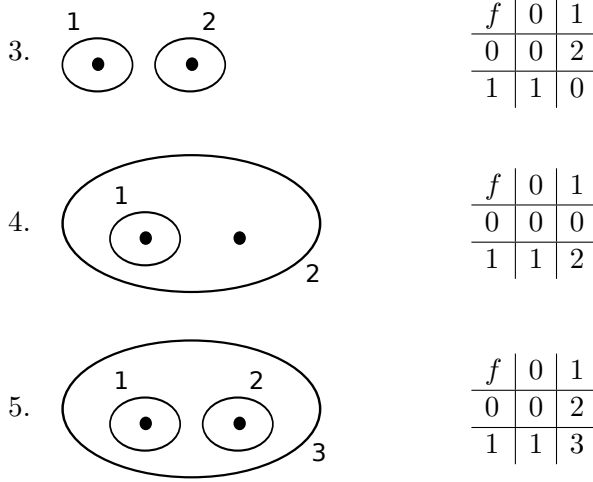
*Proof.* We prove the proposition by induction on  $n$ .

In the base case, that is  $H$  a hypergraph with two vertices, it is easy to check that there is always a term  $f(l, l)$  such that the property is satisfied. We give these terms in Example 7.9.

Let  $H$  be a hypergraph with  $n$  vertices and  $k$  hyperedges  $E_1, \dots, E_k$ . We partition the vertices in  $V_1$  and  $V_2$  such that neither  $V_1$  or  $V_2$  are empty. Let  $H_1$  and  $H_2$  be the subhypergraphs of  $H$  induced by  $V_1$  and  $V_2$  respectively. Since  $H_1$  and  $H_2$  have less than  $n$  vertices and at most  $k$  edges, we can use the induction hypothesis. We obtain two terms  $T_1$  and  $T_2$  which satisfy the proposition for  $T_1$  and  $T_2$  respectively. We now define a function  $f$  from  $[0, k]^2$  to  $[0, k]$ . For all  $m \leq k$ ,  $E_m$  is a hyperedge of  $H$ , thus  $E_m \cap V_1$  is a hyperedge of  $H_1$  indexed by some  $i$  and  $E_m \cap V_2$  is a hyperedge of  $H_2$  indexed by some  $j$ . We set  $f(i, j) = m$  and we set  $f$  to 0 on all the other arguments. Remark now that  $v(E_m, f(T_1, T_2)) = f(v(E_m \cap V_1, T_1), v(E_m \cap V_2, T_2))$ . By definition of  $T_1$  and  $T_2$   $v(E_m \cap V_1, T_1) = i$  and  $v(E_m \cap V_2, T_2) = j$ . Thus, by definition of  $f$ ,  $v(E_m, f(T_1, T_2)) = f(i, j) = m$ . If  $X$  is not a hyperedge of  $H$ , then for the same reasons  $v(X, f(T_1, T_2)) = 0$ . This completes the induction and the proof.  $\square$

**Example 7.9.** Here are presented all hypergraphs on two vertices together with the function  $f$  such that they are represented by the term  $f(l, l)$  and satisfies the property given in the proof of Proposition 7.8.

1.	•      •	<table style="border-collapse: collapse; text-align: center;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;"><math>f</math></td><td style="padding: 2px 5px;">0</td><td style="border-left: 1px solid black; padding: 2px 5px;">1</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="border-left: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">1</td><td style="padding: 2px 5px;">0</td><td style="border-left: 1px solid black; padding: 2px 5px;">0</td></tr> </table>	$f$	0	1	0	0	0	1	0	0		
$f$	0	1											
0	0	0											
1	0	0											
2.	<table style="border-collapse: collapse; vertical-align: middle;"> <tr><td style="padding: 0 5px;">1</td></tr> <tr><td style="border: 1px solid black; border-radius: 50%; padding: 2px 5px; text-align: center;">•</td></tr> </table> •	1	•	<table style="border-collapse: collapse; text-align: center;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;"><math>f</math></td><td style="padding: 2px 5px;">0</td><td style="border-left: 1px solid black; padding: 2px 5px;">1</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">0</td><td style="padding: 2px 5px;">0</td><td style="border-left: 1px solid black; padding: 2px 5px;">0</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">1</td><td style="padding: 2px 5px;">1</td><td style="border-left: 1px solid black; padding: 2px 5px;">0</td></tr> </table>	$f$	0	1	0	0	0	1	1	0
1													
•													
$f$	0	1											
0	0	0											
1	1	0											



**Proposition 7.10.** *Let  $H$  be a hypergraph, the following holds:*

- if  $H$  has  $k$  edges, then  $dw(H) \leq k$
- if  $H$  has  $n$  vertices, then  $dw(H) \leq \mathcal{H}_{2^{\lceil \frac{n}{2} \rceil}}$

*Proof.* By Lemma 7.8, we have a term  $T$  of  $\mathcal{H}_k$  such that  $v(E_i, T) = i$  and if  $X$  is not an edge,  $v(X, T) = 0$ . Let  $S = [1, k]$ ,  $H$  is defined by  $(T, S)$  as in Remark 7.6 and is thus of decomposition-width less than  $k$ .

Assume that  $H$  has  $n$  vertices. Let  $V_1, V_2$  be a partition of the vertices of  $H$  such that their cardinals are at most  $l = \lceil \frac{n}{2} \rceil$ . We consider the induced subhypergraphs  $H_{V_1}$  and  $H_{V_2}$ . They both have at most  $2^l$  edges. We apply Lemma 7.8 to find two terms  $T_1$  and  $T_2$  of  $\mathcal{H}_l$  which represents  $H_{V_1}$  and  $H_{V_2}$ . One defines a function  $f$  as in the proof of Lemma 7.8. Let  $g(0) = 0$  and  $g(x) = 1$  otherwise, the function  $g \circ f$  is a function from  $[0, l]^2$  to  $\{0, 1\}$ . Since  $g \circ f(T_1, T_2)$  represents  $H$ ,  $dw(H) \leq 2^l$ .  $\square$

**Remark 7.11.** Proposition 7.10 proves that all hypergraphs have a finite decomposition-width.

One can prove by counting the terms of  $\mathcal{H}_t$  (and by eliminating some redundancies) that there is at most

$$\frac{(2n)!}{n!(n+1)!2^n} (t+1)^{\lceil \frac{n}{2} \rceil ((t+1)^2+4)}$$

hypergraphs with  $n$  vertices and a decomposition-width less than  $t$ . On the other hand, there are  $2^{2^n}$  hypergraphs with  $n$  vertices. Thus, there must be one hypergraph  $H$  with  $n$  vertices such that  $dw(H) > \frac{2^{\lceil \frac{n}{2} \rceil}}{n}$ . Therefore the decomposition-width of the hypergraphs is unbounded.

**Open question:** the lower and upper bound on the maximal decomposition-width of a  $n$  vertices hypergraph are very close, can we eliminate this gap?

### 7.2.1 Structural properties

We now list some nice properties of the decomposition-width with regard to natural operation on hypergraphs.

**Definition 7.12.** Let  $H$  be a hypergraph, the complement hypergraph of  $H$ , denoted by  $\overline{H}$ , is the hypergraph with the same vertices and such that a set is an edge of  $\overline{H}$  if and only if it is not an edge of  $H$ .

**Proposition 7.13.** Let  $H$  be a hypergraph, the following equality holds:  $dw(H) = dw(\overline{H})$ .

*Proof.* Let  $T$  be a term of  $\mathcal{H}_t$  such that  $H$  is represented by  $T$ . It is easy to see that  $(T, [0, t] \setminus \{1\})$  represents  $\overline{H}$ . By Remark 7.6, it implies that  $dw(\overline{H}) \leq dw(H)$ . Since  $H = \overline{\overline{H}}$  we obtain  $dw(H) = dw(\overline{H})$ .  $\square$

The previous proposition enables us to slightly improve Proposition 7.10. Indeed, when given a  $n$  vertices hypergraph, either itself or its complement has less than  $2^{n-1}$  hyperedges. Using this trick in the proof of Proposition 7.10, we obtain that a hypergraph  $H$  on  $n$  vertices satisfies  $dw(H) \leq 2^{\lceil \frac{n}{2} \rceil - 1}$ .

Let  $H = (V, E)$  be a hypergraph and  $W \subseteq V$ . Recall that the section hypergraph induced by  $W$  is denoted by  $H \times W$  and is equal to  $(W, \{e \subseteq W \mid e \in E\})$ .

**Proposition 7.14.** Let  $H = (V, E)$  be a hypergraph and  $W \subseteq V$  then  $dw(H \times W) \leq dw(H)$ .

*Proof.* Let  $T$  be a term representing  $H$ . Assume that  $W = V \setminus \{a\}$  it is enough to prove the proposition in this case, we then conclude by removing points successively.

Some subterm of  $T$  is of the form  $g(f(a, T_1), T_2)$ . Let  $h(x, y) = g(f(0, x), y)$ . Let  $\tilde{T}$  be the tree  $T$  where this subterm is changed by  $h(T_1, T_2)$ . One may check that the values of all sets without  $a$  remain the same as in  $T$ , therefore  $\tilde{T}$  represents  $H \times W$ . From this  $dw(H \times W) \leq dw(H)$  and that concludes the proof.  $\square$

This proposition is interesting if we use hypergraphs to represent the independent sets of matroids. Indeed for hereditary structures such as the independent sets, the section hypergraph and the induced hypergraph are the same. Moreover, the notion of matroid restriction is nothing but the induced (or section) hypergraph of the independent sets of a matroid.

**Remark 7.15.** The removal of an edge can increase the decomposition-width. Indeed, the complete hypergraph on  $n$  vertices has a decomposition-width of 1, while any hypergraph on  $n$  vertices can be obtained by removing some of its hyperedges.

**Open question:** is decomposition-width decreasing under contraction of an edge? Is decomposition-width decreasing when we consider a subhypergraph induced by a set of vertices?

**Proposition 7.16.** *Let  $H_1$  and  $H_2$  be two hypergraphs and  $H_1 \cup H_2$  their disjoint union (or direct sum) and let  $m = \max(dw(H_1), dw(H_2))$ . The following inequalities hold:  $m \leq dw(H_1 \cup H_2) \leq m + 1$ .*

*Proof.* Let  $T_1$  and  $T_2$  be two terms representing respectively  $H_1$  and  $H_2$  and which belong respectively to  $\mathcal{H}_{dw(H_1)}$  and  $\mathcal{H}_{dw(H_2)}$ . There are two terms  $\tilde{T}_1$  and  $\tilde{T}_2$  of  $\mathcal{H}_{m+1}$  obtained by a slight modification of  $T_1$  and  $T_2$  such that the only set of value 0 in  $\tilde{T}_1$  or  $\tilde{T}_2$  is the empty set. Moreover,  $\tilde{T}_1$  and  $\tilde{T}_2$  represent  $H_1$  and  $H_2$ .

Let  $f$  be the function from  $[0, m+1]^2$  to  $\{0, 1\}$  defined by  $f(1, 0) = f(0, 1) = 1$  and  $f$  is zero on the other couples. The term  $f(\tilde{T}_1, \tilde{T}_2)$  represents  $H_1 \cup H_2$  therefore  $dw(H_1 \cup H_2) \leq m + 1$ .

Let  $V_1$  and  $V_2$  denote the set of vertices of respectively  $H_1$  and  $H_2$ . It holds that  $H_1 = (H_1 \cup H_2) \times V_1$  and  $H_2 = (H_1 \cup H_2) \times V_2$ . By Proposition 7.14, one obtains that  $dw(H_1 \cup H_2) \geq dw(H_1)$  and  $dw(H_1 \cup H_2) \geq dw(H_2)$ , which concludes the proof.  $\square$

**Open Question:** let  $H_1$  and  $H_2$  be two hypergraphs, what is the relation of their decomposition-width with the one of the amalgam of  $H_1$  and  $H_2$  along a set of size  $k$ ?

### 7.3 Model-checking of MSO on hypergraphs represented by a term of $\mathcal{H}_t$

We study the monadic second-order logic over hypergraphs and we first give several examples of what it can express. Recall that a hypergraph is represented by a finite domain and one second-order relation denoted by  $R$  such that  $R(X)$  holds if and only if  $X$  is an edge of the hypergraph.

The formula  $\forall X R(X)$  defines the complete hypergraph. We can also state that  $R$  is hereditary by  $\forall X, Y [(X \subset Y \wedge R(Y)) \Rightarrow R(X)]$ . In fact, as seen in Chapter 6, the three axioms which characterize the circuits of a matroid are expressible in MSO. Therefore, there is a formula *Matroid*, which is true if and only if the hypergraph is the set of circuits of a matroid.

We can characterize interesting objects, like the hitting sets of a hypergraph, by a MSO formula:

$$\text{Hitting} - \text{set}(X) \equiv \forall Y [R(Y) \Rightarrow (X \cap Y \neq \emptyset)]$$

We can also characterize transversals, which is useful since the associated enumeration problem is hard:

$$\text{Transversal}(X) \equiv \text{Hitting} - \text{set}(X) \wedge \forall Y [\text{Hitting} - \text{set}(Y) \Rightarrow \neg(Y \subsetneq X)]$$

A  $k$ -coloring of the hypergraph  $H = (V, E)$  is a function  $f$  from  $V$  to  $[k]$  such that for each edge  $e \in E$  there are  $v_1, v_2 \in e$  such that  $f(v_1) \neq f(v_2)$ . The

property to be  $k$ -colorable can be expressed by the following formula:

$$\begin{aligned} & \exists X_1 \dots \exists X_k \bigwedge_{i \neq j} (X_i \cap X_j = \emptyset) \wedge \forall X R(X) \Rightarrow \\ & [\exists v_1 \exists v_2 (v_1 \in X) \wedge (v_2 \in X) \wedge \bigvee_{i \neq j} (v_1 \in X_i) \wedge (v_2 \in X_j)] \end{aligned}$$

### 7.3.1 Representation of hyperedges over a term

Let  $T$  be a term of  $\mathcal{H}_t$  which represents  $H$ . We explain in this subsection how to translate the hyperedge predicate  $R$  over  $H$  into a  $MSO$  formula over  $T$ . Here the value of a set plays a role similar to the signatures of the previous chapter.

**Remark 7.17.** Let  $T$  be a term of  $\mathcal{H}_t$  and  $H$  is its associated hypergraph. Let  $Y$  be a subset of the leaves of  $T$ . For each node  $s$  of  $T$ , we denote by  $L_s$  the leaves of  $T_s$ . For each node  $s$  of  $T$  labeled by  $f$ , with left child  $s_1$  and right child  $s_2$ , the equality  $v(Y \cap L_s, T_s) = f(v(Y \cap L_{s_1}, T_{s_1}), v(Y \cap L_{s_2}, T_{s_2}))$  holds.

We want to write a formula with one free second order variable  $Y$ , which states that  $Y$  is an edge. We represent the value of  $v(Y \cap L_s, T_s)$  for each  $s$  by the second-order variables  $X_i$  for  $i = 0, \dots, t$ . We will have  $X_i(s)$  if and only if  $v(Y \cap L_s, T_s) = i$ . We write  $\vec{X}$  for the set of variables  $\{X_i\}_{i \leq t}$ . The following formula states that  $\vec{X}$  represents one and only one value for each node  $s$ .

$$\Omega(\vec{X}) = \forall s \bigvee_{i=0}^t \left( X_i(s) \wedge \bigwedge_{j \neq i} \neg X_j(s) \right)$$

The following  $MSO$  formula ensures that the values defined by  $\vec{X}$  satisfy the equation of Remark 7.17:

$$\begin{aligned} \Psi_1(\vec{X}) & \equiv \forall s \neg leaf(s) \Rightarrow \exists s_1, s_2 lchild(s, s_1) \wedge rchild(s, s_2) \wedge \\ & \bigwedge_{f, i, j, k} (label(s) = f \wedge X_i(s_1) \wedge X_j(s_2) \wedge X_k(s)) \Rightarrow f(i, j) = k \end{aligned}$$

We also need a formula which expresses the fact that the value of a leaf of  $T$  is 1 if it is in  $Y$  and is 0 otherwise:

$$\Psi_2(\vec{X}, Y) \equiv \forall s leaf(s) \Rightarrow [(s \in Y \Rightarrow X_1(s)) \wedge (s \notin Y \Rightarrow X_0(s))]$$

The conjunction of the formulas  $\Psi_1$  and  $\Psi_2$  ensures that  $\vec{X}$  represents the function which associates to a node  $s$  of  $T$  the value  $v(Y \cap L_s, T_s)$ . By definition,  $Y$  is a hyperedge of  $T$  if  $v(Y, T) = 1$ , which is expressed in the next formula:

$$\Psi_3(\vec{X}) \equiv \exists r root(r) \wedge X_1(r)$$

The following formula is true if and only if  $Y$  is a hyperedge of  $T$ :

$$\text{hyperedge}(Y) \equiv \exists X_0 \dots \exists X_t \left[ \Psi_1(\vec{X}) \wedge \Psi_2(\vec{X}, Y) \wedge \Psi_3(\vec{X}) \right]$$

Let  $F$  be the function, which associates to the formula  $\phi$  in  $MSO$  over hypergraphs the formula  $F(\phi)$  by relativization to the leaves and replacement of the predicate  $R$  by the formula  $\text{hyperedge}$ . It is defined exactly as in Chapter 6. We call  $f$  the bijection between the leaves of a term of  $\mathcal{H}_t$  and the vertices of its associated hypergraph.

**Theorem 7.18.** *Let  $H$  be a hypergraph associated to the term  $T$  of  $\mathcal{H}_t$  and let  $\phi(\vec{x})$  be a  $MSO$  formula over the hypergraphs with free variables  $\vec{x}$ , we have*

$$(H, \vec{a}) \models \phi(\vec{x}) \Leftrightarrow (T, f(\vec{a})) \models F(\phi(\vec{x})).$$

**Corollary 7.19.** *The model checking problem for  $MSO$  over the hypergraphs given by a term of  $\mathcal{H}_t$  is decidable in time  $f(t, l) \times n$  where  $n$  is the number of vertices of  $H$ ,  $l$  the size of the formula and  $f$  a computable function.*

Deciding if a graph is  $k$ -colorable is NP-complete for  $k \geq 3$ . Since it is a special case of the problem of deciding if a hypergraph is  $k$ -colorable, the latter is also NP-complete for  $k \geq 3$ . We have given a  $MSO$  formula which holds if and only if a hypergraph is  $k$ -colorable, thus this problem is linear time decidable over  $\mathcal{H}_t$ .

We can also use Theorem 6.22 to obtain a corollary similar to Corollary 6.23 on enumeration:

**Corollary 7.20.** *Let  $\phi(X_1, \dots, X_m)$  be an  $MSO$  formula over hypergraphs. The enumeration of the sets satisfying  $\phi$  can be done in linear delay on the hypergraphs associated to a term of  $\mathcal{H}_t$ .*

We have given the formula  $\text{Transversal}(X)$  which is true if and only if  $X$  is a transversal. Therefore, one can enumerate the transversals of a hypergraph associated to a term of  $\mathcal{H}_t$  in linear delay. The best enumeration algorithm for this problem over all hypergraphs is in incremental superpolynomial time [FK96].

To make the model-checking results more interesting, it would be interesting to have an algorithm which given a hypergraph finds a term of good decomposition-width representing the hypergraph.

**Open questions:** prove that computing the decomposition-width is NP-complete. Find a suitable restriction of the hypergraphs (or of  $\mathcal{H}_t$ ) such that there is an approximation algorithm similar to the one for branch-width or tree-width.

## 7.4 Representation of hypergraph like structures

### 7.4.1 Definable subclasses

We want to represent hypergraph like structures by terms of  $\mathcal{H}_t$ . Let say we have a class  $\mathcal{C}$  of structures and a simple encoding of these structures into hypergraphs,

that is an efficiently computable injective function from  $\mathcal{C}$  to the hypergraphs. We need the relations and functions of  $\mathcal{C}$  to be expressible in *MSO* over the hypergraphs which encode them (it is an interpretation).

In addition, we want a *MSO* formula which is satisfied by a hypergraph if and only if it encodes one of the structures we consider. Hence, we can recognize the terms of  $\mathcal{H}_t$  which encode an element of  $\mathcal{C}$  in linear time (for a fixed  $t$ ). It is thus reasonable to define the decomposition-width of an element of  $\mathcal{C}$  as the decomposition-width of its encoding.

**Matroids** We have seen that there is a formula *Matroid* which holds if and only if the term represents the circuits of a matroid. We can then define a notion of decomposition-width for matroids represented by their circuits. In fact, we will see in the next section, the classes of matroids defined in the previous chapter, and they are represented by the hypergraph of their dependent sets. However, it is not hard to derive a *MSO* axiomatization of the dependent sets from the one of the circuits.

**Oriented Matroids** We define the oriented matroids of a given decomposition-width, thanks to a slightly more involved encoding. Let  $M$  be an oriented matroid of ground set  $S = \{s_1, \dots, s_n\}$ . To simplify the presentation,  $M$  has no loops neither parallel elements (dependent sets of size two).

Let  $S_1 = \{s_1^1, \dots, s_n^1\}$  and  $S_2 = \{s_1^2, \dots, s_n^2\}$ ,  $M$  is represented by a hypergraph  $H$  with vertices  $S_1 \cup S_2$ . The element of  $S_1$  represent the points of  $M$  in "negative" position and  $S_2$  the points in "positive" position. The hyperedges represent the oriented circuits of  $M$  in the following way: if  $(C^-, C^+)$  is an oriented circuit then  $\{s_i^1 \mid s_i \in C^-\} \cup \{s_i^2 \mid s_i \in C^+\}$  is a hyperedge of  $H$ . The singletons  $\{s_i^1\}$  and the pairs  $\{s_i^1, s_i^2\}$  are also hyperedges of  $H$  for all  $i$ .

Given any hypergraph, we want to be able to test that it encodes an oriented matroid in the way we have just described. It means that the set of pairs of the hypergraph has to define a bijection from the negative to the positive elements. Let  $\phi$  a *MSO* formula which expresses this condition, it is the conjunction of the following formulas:

- $\forall x R(\{x\}) \Rightarrow \exists! y (x \neq y \wedge R(\{x, y\}))$
- $\forall x \neg R(\{x\}) \Rightarrow \exists! y (x \neq y \wedge R(\{x, y\}))$
- $\forall x y R(\{x, y\}) \Rightarrow (R(\{x\}) \wedge R(\{y\})) \vee (R(\{y\}) \wedge R(\{x\}))$

We now list a set of atomic formulas over the oriented matroid  $M$  and their translation over the hypergraph  $H$  which represents it:

- $C(X) \equiv \exists x_1 x_2 x_3 (\bigwedge_{i \neq j} x_i \neq x_j \wedge \{x_1, x_2, x_3\} \subseteq X) \wedge R(X)$
- $x \in X^+ \equiv x \in X \wedge \neg R(\{x\})$



- $Y = X^+ \equiv \forall x(x \in X^+ \Leftrightarrow x \in Y)$
- $x \in X^- \equiv x \in X \wedge R(\{x\})$
- $Y = X^- \equiv \forall x(x \in X^- \Leftrightarrow x \in Y)$
- $-x \in X \equiv \exists y y \neq x \wedge R(\{x, y\}) \wedge y \in X$
- $Y = -X \equiv \forall x(-x \in X \Leftrightarrow x \in Y)$
- $\underline{X} \subseteq \underline{Y} \equiv \forall x(x \in X^+) \vee (-x \in X^+) \Rightarrow (x \in Y^+) \vee (-x \in Y^+)$

From these atomic formulas, we can express the axioms that an oriented matroid has to satisfy:

- $\forall X \exists Y (C(X) \wedge Y = -X) \Rightarrow C(Y)$
- $\forall X Y (C(X) \wedge C(Y) \wedge \underline{X} \subseteq \underline{Y}) \Rightarrow (X = Y \vee X = -Y)$
- $\forall X Y (X \neq Y \wedge C(X) \wedge C(Y) \wedge (\exists e e \in X^+ \wedge e \in Y^-)) \Rightarrow (\exists Z C(Z) e \notin Z^+ \wedge e \notin Z^- \wedge \forall x (x \in Z^+ \Rightarrow x \in X^+ \vee x \in Y^+) \wedge (x \in Z^- \Rightarrow x \in X^- \vee x \in Y^-))$

Finally, we write  $\Psi$  for the conjunction of these *MSO* formulas with  $\phi$ . The set of hypergraphs represented by a term of  $\mathcal{H}_t$  and satisfying  $\Psi$  encodes the oriented matroids of decomposition-width  $t$ .

### 7.4.2 Encoding other decompositions

One can interpret many decompositions as a subset of the terms of  $\mathcal{H}_t$ . One then automatically obtains the linear time model-checking of *MSO* for the objects represented by the interpreted decomposition. In such a reduction there are two obvious objectives:

- to do the reduction to  $\mathcal{H}_t$  for the smallest possible  $t$
- to reduce to a subset of  $\mathcal{H}_t$  which uses as few different functions of  $F_t$  as possible

**Clique-width** We now explain how to interpret a  $t$ -expression as a term of  $\mathcal{H}_{t+2}$ . This proves that graphs of clique-width  $t$  have a decomposition-width at most  $t + 2$ . The idea to encode a  $t$ -expression into a term  $\tilde{T}$  of  $\mathcal{H}_{t+2}$  is to use  $v(X, \tilde{T})$  to represent the color of the vertices when  $X$  is a singleton, the fact that two vertices are linked by an edge or that  $X$  is not an edge. We use the values from 1 to  $t$  to represent the colors from 1 to  $t$ . The value  $t + 1$  stands for an edge, the value 0 denotes an empty set, while the value  $t + 2$  is a garbage state for the rest.

**Definition 7.21.** Let  $\mathcal{H}_{t+2}^c$  be the set of terms of  $\mathcal{H}_{t+2}$  such that they are built from binary functions  $f$  or unary functions  $g$  (as in Remark 7.6) satisfying the following properties:

- $f(0) = g(0, 0) = 0$
- for  $i \in [1, t]$ ,  $g(0, i), g(i, 0), f(i) \in [1, t]$
- for  $i, j \in [1, t]$ ,  $g(i, j) \in \{t+1, t+2\}$
- $g(0, t+1) = g(0, t+1) = f(t+1) = t+1$
- for all  $i > 0$ ,  $g(i, t+1) = g(t+1, i) = t+2$
- for all  $x, y$ ,  $g(x, t+2) = g(t+2, y) = f(t+2) = t+2$

**Lemma 7.22.** Let  $T \in \mathcal{H}_{t+2}^c$  and  $X$  a set of leaves of  $T$  then the following holds:

- if  $X = \emptyset$  then  $v(X, T) = 0$
- if  $|X| = 1$  then  $v(X, T) \in [1, t]$
- if  $|X| = 2$  then  $v(X, T) = t+1$  or  $v(X, T) = t+2$
- if  $|X| \geq 3$  then  $v(X, T) = t+2$

*Proof.* A trivial structural induction on the terms of  $\mathcal{H}_t^c$  for each size of  $X$  proves the lemma.  $\square$

**Proposition 7.23.** Let  $G$  be a graph, then  $dw(G) \leq cw(G) + 2$ .

*Proof.* In this proof, we use all the flexibility provided by Remark 7.6. Let  $T$  be a  $t$ -expression which represents  $G$ , we inductively build a term  $\tilde{T}$  of  $\mathcal{H}_{t+2}^c$  such that  $(\tilde{T}, \{t+1\})$  also represents  $G$ . We will not distinguish the leaves of the trees  $T$ ,  $\tilde{T}$  and the vertices of  $G$  (the construction of  $\tilde{T}$  gives a simple bijection between them). One adds to the induction hypothesis the following properties:

1. for any leaf  $u$  of  $T$  of color  $i$ , we have  $v(\{u\}, \tilde{T}) = i$
  2. for any  $\{u, w\}$  edge of  $G$ ,  $v(\{u, w\}, \tilde{T}) = t+1$
  3. for any subterm  $T_1$  of  $T$  and  $\{u, w\}$  two leaves of  $\tilde{T}_1$  which are not an edge of  $G$ ,  $v(X, \tilde{T}_1) = t+2$
- Let  $T$  be the constant  $G_i$ , it represents a graph with one vertex of color  $i$ . The term  $\tilde{T}$  is the constant  $i$ , it satisfies the induction hypothesis and is in  $\mathcal{H}_{t+2}^c$ .
  - Let  $T = T_1 \oplus T_2$ , where  $T_1$  and  $T_2$  are subterms. The induction hypothesis gives two terms  $\tilde{T}_1$  and  $\tilde{T}_2$  of  $\mathcal{H}_{t+2}^c$  associated to  $T_1$  and  $T_2$ . Let  $f(x, 0) = x$ ,  $f(0, y) = y$  and  $f(x, y) = t+2$  otherwise. We let  $\tilde{T} = f(T_1, T_2)$ , it is a term of  $\mathcal{H}_{t+2}^c$  and it represents the same graph as  $T = T_1 \oplus T_2$ .

- Let  $T = \rho_{a \rightarrow b} T_1$ , where  $T_1$  is a subterm and  $\tilde{T}_1$  the term of  $\mathcal{H}_{t+2}^c$  given by the induction hypothesis. Let  $f(a) = b$ ,  $f$  is the identity otherwise. We let  $\tilde{T} = f(T_1)$ , it is a term of  $\mathcal{H}_{t+2}^c$  and it represents the same graph as  $T = \rho_{a \rightarrow b} T_1$ .
- Let  $T = \eta_{a,b} T_1$  where  $T_1$  is a subterm and  $\tilde{T}_1$  the term of  $\mathcal{H}_{t+2}^c$  given by the induction hypothesis.

For all leaf  $u$  of color  $a$  and leaf  $w$  of color  $b$  in  $T$ , there is exactly one subterm  $f(\tilde{T}_2, \tilde{T}_3)$  such that  $u$  is a leaf of  $\tilde{T}_2$  and  $w$  is a leaf of  $\tilde{T}_3$ . Let  $\tilde{f}(v(\{u\}, \tilde{T}_2), v(\{w\}, \tilde{T}_3)) = t + 1$  and  $\tilde{f}(x, y) = f(x, y)$  otherwise. We let  $\tilde{T}$  be the term  $\tilde{T}_1$  where all the functions  $f$  defined as previously are replaced by  $\tilde{f}$ . Obviously  $\tilde{T}$  is a term of  $\mathcal{H}_{t+2}^c$ .

We must prove that  $\tilde{T}$  satisfies the induction hypothesis. First remark that  $v(X, \tilde{T}) \neq v(X, \tilde{T}_1)$  only if  $|X| = 2$ . Indeed, the transformation of  $f$  into  $\tilde{f}$  only changes the image of some couples  $(i, j)$  where  $i, j \in [1, t]$ .

Let  $\{u, v\}$  be an edge of the graph represented by  $\tilde{T}_1$ , then  $v(\{u, w\}, \tilde{T}) = t + 1$  by induction hypothesis. Because  $\tilde{T} \in \mathcal{H}_{t+2}^c$ , we obtain that  $v(\{u, w\}, \tilde{T}) = t + 1$ . Assume now that  $u$  is of color  $a$  and  $w$  is of color  $b$  in  $T_1$ . By construction, there is a subterm  $\tilde{T}_4$  of  $\tilde{T}$  such that  $v(\{u, w\}, \tilde{T}_4) = t + 1$ . Because  $\tilde{T} \in \mathcal{H}_{t+2}^c$ , we obtain that  $v(\{u, w\}, \tilde{T}) = t + 1$ .

Finally, we have to check that if  $\{u, w\}$  is not an edge in  $G$ , then  $v(\{u, w\}, \tilde{T}_2) = t + 2$  for all subterms  $\tilde{T}_2$  of  $\tilde{T}$ . Let  $\tilde{f}(\tilde{T}_3, \tilde{T}_4)$  be the subterm such that  $u$  is a leaf of  $\tilde{T}_2$  and  $w$  is a leaf of  $\tilde{T}_3$ . By induction hypothesis,  $v(\{u, w\}, \tilde{f}(\tilde{T}_3, \tilde{T}_4)) = t + 2$ , thus if  $v(\{u, w\}, \tilde{f}(\tilde{T}_3, \tilde{T}_4)) = t + 1$  then there is  $u'$  of color  $a$  and  $w'$  of color  $b$  such that  $v(\{u\}, \tilde{T}_3) = v(\{u'\}, \tilde{T}_3)$  and  $v(\{w\}, \tilde{T}_4) = v(\{w'\}, \tilde{T}_4)$ . Since  $\tilde{T}_1 \in \mathcal{H}_{t+2}^c$ , if two leaves have the same value in a subterm of  $\tilde{T}_1$ , they have the same value in  $\tilde{T}_1$ . Therefore the color of  $u$  is  $a$  and the color of  $w$  is  $b$  and  $\{u, w\}$  is an edge of  $G$ .

□

**Open question:** can we bound the decomposition-width of graphs by the clique-width? By a result of Courcelle and Oum [CO07], it is sufficient to prove that the graphs of bounded decomposition-width have a decidable  $CMS^1$  theory.

**Matroid branch-width** Recall that a matroid  $M$  represented by a matrix  $A$  over the finite field  $\mathbb{F}$  and of branch-width  $t$  can be represented by a term  $T \in \mathcal{M}_t^{\mathbb{F}}$ . We can transform  $T$  into a term of  $\mathcal{H}_k$  for some  $k$  which represents the same matroid as  $T$ . The natural idea is to map each signature to an integer and to see a function  $\odot_A$  of  $T$  as a function on the signatures and thus on their encoding by integers. This function would be  $(x, y) \rightarrow z$  such that  $R(A, x, y, z)$ , but this does not define a function. Hence, we have to change what represents a signature.

<sup>1</sup> $MS_1$  and a predicate expressing that a set has even cardinality

The idea is that the signature of a set of leaves  $X$  in a term representing a bounded matroid denotes the intersection of the boundary space and of the vector subspace generated by  $X$  rather than just one vector of this space. For a proof using this idea in a similar context, see [Kra09]. By following the same proof, we would obtain  $dw(M) \leq \frac{q^{t+1}-q(t+1)+t}{(q-1)^2}$  where  $q = |\mathbb{F}|$ .

**Matroids represented by a term of  $\mathcal{T}_k$**  There is a direct interpretation of the term of  $\mathcal{T}_k$  into terms of  $\mathcal{H}_t$  or some  $t$ . One has to represent any matroid of size  $k$ , by Proposition 7.10 we can do that by a term of  $\mathcal{H}_{2^{\lceil \frac{k}{2} \rceil - 1}}$ . Then we only need to represent the three signatures  $\mathbf{0}$ ,  $\mathbf{1}$  and  $\mathbf{2}$  by the three integers 0, 1 and 2. One may check that the operation  $\odot_M$  of  $\mathcal{T}_k$  realizes the following function on signatures:  $(x, y) \rightarrow z$  such that  $R_p(M, x, y, z)$  (see Lemma 6.43). One just replaces in a term of  $\mathcal{T}_k$  all functions by the ones we have just described and the leaves representing matroids of size  $k$  by the terms given by Proposition 7.10.

Therefore, if  $M$  is a matroid represented by a term of  $\mathcal{T}_k$ , then  $dw(M) \leq \max(2^{\lceil \frac{k}{2} \rceil - 1}, 2)$ .

# Index

- DelayP, 29
- DelayPP, 62
- EnumP, 25
- IncP, 28
- IncPP, 60
- QueryP, 31
- SDelayP, 29
- TotalP, 26
- TotalPP, 60
- ENUM·A-CIRCUIT, 41
- ENUM·POLY, 49
- ENUM·TRANSVERSAL, 27
- $k$ -uniform, 2
- A-CIRCUIT, 41
- ANOTHERSOLUTION<sub>A</sub>, 28
- MONOMIAL-COEFFICIENT, 70
- MONOMIAL-FACTOR, 54
- NON-ZERO-MONOMIAL, 70
- POLYNOMIAL IDENTITY TESTING, 17
  
- ENUM·MAXIS, 25
  
- Berge-acyclicity, 2
- bipartite graph, 2
- BPP, 13
- branch-width, 88
  
- circuit, 3
- clique-width, 90
- complete problem, 11
- contraction, 4
- cycle, 1
- cycle matroid, 4
  
- degree, 16
- degree with regard to a set, 54
- delay, 9
  
- Determinant, 18
  
- first-order logic, 14
- formal degree, 17
  
- graph, 1
- graph minor, 2
  
- Hamiltonian path, 2
- hard problem, 11
- hypergraph, 2
- hypertree, 2
  
- induced subhypergraph, 3
  
- labeled tree, 85
  
- matching, 2
- matroid, 3
- minor, 4
  
- Parsimonious reduction, 11
- Permanent, 19
- Pfaffian, 18
- planar, 2
- polynomially balanced predicate, 7
  
- RAM machine, 7
- RAM machine with oracle, 10
- rank, 5
- restriction, 4
- RNC, 14
- RP, 13
  
- SAT, 12
- second-order logic, 15
- section hypergraph, 3
- semi-linear, 107

series-parallel graphs, 88  
spanning tree, 2  
support, 49

term, 16  
total degree, 16  
total time, 9  
transversal, 3  
tree, 1  
tree-width, 87

vector matroid, 4

# Bibliography

- [AB09] S. Arora and B. Barak. Computational Complexity: A Modern Approach. 2009. [6](#), [60](#)
- [ACP87] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Algebraic Discrete Methods*, 8(2):277–284, 1987. [87](#)
- [AF96] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996. [23](#), [30](#)
- [Aig07] M. Aigner. *A course in enumeration*. Springer Verlag, 2007. [48](#), [74](#)
- [AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. [32](#)
- [ALS91] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. [106](#)
- [AMAM09] V. Arvind, P. Mukhopadhyay, S. Albers, and J.Y. Marion. Quantum query complexity of multilinear identity testing. In *Symposium on Theoretical Aspects of Computer Science*, volume 3, pages 87–98, 2009. [47](#)
- [ASY] AV Aho, TG Szymanski, and M. Yannakakis. Sorting the Cartesian Product. In *Proc. 1980 Princeton Conf. on Information Sciences and Systems*, pages 557–560. [24](#)
- [Bag] G. Bagan. *Algorithmes et Complexité des Problèmes d'Énumération pour l'Évaluation de Requêtes Logiques*. PhD thesis, Université de Caen, 2009. [7](#)
- [Bag06] G. Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Computer Science Logic*, pages 167–181. Springer, 2006. [108](#)

- [BDGO08] G. Bagan, A. Durand, E. Grandjean, and F. Olive. Computing the  $j$ th solution of a first-order query. *RAIRO Theor. Inf. Appl.*, 42:147–164, 2008. [23](#), [31](#)
- [Ber84] S.J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147–150, 1984. [18](#)
- [BMVT78] E.R. Berlekamp, R.J. McEliece, and H.C.A. Van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978. [106](#)
- [BO88] M. Ben-Or. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 301–309. ACM New York, NY, USA, 1988. [47](#), [67](#)
- [Bod93] H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, page 234. ACM, 1993. [87](#)
- [Bod98] H.L. Bodlaender. A partial  $k$ -arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998. [115](#)
- [BR91] R.A. Brualdi and H.J. Ryser. *Combinatorial matrix theory*. Cambridge Univ Pr, 1991. [19](#), [77](#)
- [CDG<sup>+</sup>07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: <http://www.grappa.univ-lille3.fr/tata>, 2007. release October, 12th 2007. [107](#)
- [CER93] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993. [89](#)
- [CH97] N. Creignou and JJ Hébrard. On generating all solutions of generalized satisfiability problems. *RAIRO Theoretical Informatics and Applications*, 31(6), 1997. [30](#)
- [CKS01] N. Creignou, S. Khanna, and M. Sudan. *Complexity classifications of boolean constraint satisfaction problems*. Society for Industrial Mathematics, 2001. [40](#)
- [CMSS08] S. Caracciolo, G. Masbaum, A.D. Sokal, and A. Sportiello. A randomized polynomial-time algorithm for the Spanning Hypertree Problem on 3-uniform hypergraphs. *Arxiv preprint arXiv:0812.3593*, 2008. [78](#)



- [CO00] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. **89**
- [CO07] B. Courcelle and S. Oum. Vertex-minors, monadic second-order logic, and a conjecture by Seese. *Journal of Combinatorial Theory, Series B*, 97(1):91–126, 2007. **135**
- [Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, page 158. ACM, 1971. **ix, xv, 12**
- [Cou] B. Courcelle. *Graph algebras and monadic second-order logic*. To be published by Oxford University Press. **85, 91**
- [Cou91] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 193–242. 1991. **xiii, xix, 95**
- [Cou92] B. Courcelle. The monadic second-order logic of graphs. III: Tree-decompositions, minors and complexity issues. *Informatique théorique et applications*, 26(3):257–286, 1992. **106**
- [Cou95] B. Courcelle. Structural properties of context-free sets of graphs generated by vertex replacement. *Information and Computation*, 116(2):275–293, 1995. **107**
- [Cou09] B. Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009. **23, 29, 31, 108**
- [CR05] D. Corneil and U. Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. **90**
- [CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990. **68**
- [DFL97] A. Durand, R. Fagin, and B. Loescher. Spectra with only unary function symbols. In *Computer Science Logic*, pages 189–202. Springer, 1997. **107**
- [DG07] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4), 2007. **23**
- [DH03] A. Durand and M. Hermann. The inference problem for propositional circumscription of affine formulas Is coNP-complete. *STACS 2003: 20th Annual Symposium on Theoretical Aspects of Computer Science*, pages 451–462, 2003. **41**

- [DH09] T. Daigo and K. Hirata. On generating all maximal acyclic subhypergraphs with polynomial delay. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science*, pages 181–192. Springer, 2009. [23](#), [24](#), [37](#)
- [DHK05] A. Durand, M. Hermann, and P.G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science*, 340(3):496–513, 2005. [25](#), [39](#)
- [Die05] R. Diestel. Graph theory, volume 173 of Graduate Texts in Mathematics. Springer, Heidelberg, 91:92, 2005. [1](#)
- [DS11] D. Duris and Y. Strozecki. The complexity of acyclic subhypergraph problems. *Workshop on Algorithms and Computation*, 2011. [77](#), [81](#)
- [Dur09] D. Duris. *Acyclicité des hypergraphes et liens avec la logique sur les structures relationnelles finies*. PhD thesis, Université Paris Diderot - Paris 7, 2009. [2](#), [78](#)
- [Edm65] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965. [x](#), [xv](#), [11](#)
- [EG] T. Eiter and G. Gottlob. Hypergraph transversal computation and related problems in logic and AI. [3](#), [23](#)
- [EG95] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal of Computing*, 24(6):1278–1304, 1995. [3](#), [23](#), [27](#)
- [Fag74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *American Mathematical Society*, pages 43–74, 1974. [15](#)
- [FG06] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer-Verlag New York Inc, 2006. [73](#), [82](#)
- [FK96] M.L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996. [3](#), [23](#), [27](#), [131](#)
- [FM04] E. Fischer and J.A. Makowsky. On spectra of sentences of monadic second order logic with counting. *Journal of Symbolic Logic*, 69(3):617–640, 2004. [107](#)
- [FRRS06] M.R. Fellows, F.A. Rosamond, U. Rotics, and S. Szeider. Clique-width minimization is NP-hard. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, page 362. ACM, 2006. [90](#)

- [GdM10] A. Goodall and A. de Mier. Spanning trees of 3-uniform hypergraphs. *Arxiv preprint arXiv:1002.3331*, 2010. 79
- [GHR95] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, USA, 1995. 14
- [GJ79] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to NP-completeness*, 1979. 72
- [GO04] E. Grandjean and F. Olive. Graph properties checkable in linear time in the number of vertices. *Journal of Computer and System Sciences*, 68(3):546–597, 2004. 7
- [Gra96] E. Grandjean. Sorting, linear time and the satisfiability problem. *Annals of Mathematics and Artificial Intelligence*, 16(1):183–236, 1996. 7
- [Gro08] M. Grohe. Logic, graphs, and algorithms. In *Logic and Automata: History and Perspectives*, pages 357–422. Amsterdam Univ Pr, 2008. 88, 91, 95
- [GS02] E. Grandjean and T. Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. *SIAM Journal on Computing*, 32:196, 2002. 7
- [GS03] Y. Gurevich and S. Shelah. Spectra of monadic second-order formulas with one unary function. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 291–300. IEEE Computer Society, 2003. 107
- [GS09] S. Garg and É. Schost. Interpolation of polynomials given by straight-line programs. *Theoretical Computer Science*, 410(27-29):2659–2662, 2009. 47, 60, 67
- [Hli03] P. Hliněný. On matroid properties definable in the MSO logic. *Mathematical Foundations of Computer Science*, pages 470–479, 2003. 92, 95
- [Hli06] P. Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. *Journal of Combinatorial Theory, Series B*, 96(3):325–351, 2006. xiii, xix, 92, 95, 96, 98, 105, 109, 110
- [HMJ07] I.V. Hicks and N.B. McMurray Jr. The branchwidth of graphs and their cycle matroids. *Journal of Combinatorial Theory, Series B*, 97(5):681–692, 2007. 93

- [HO08] P. Hliněný and S. Oum. Finding Branch-Decompositions and Rank-Decompositions. *SIAM Journal on Computing*, 38:1012, 2008. [93](#), [98](#), [105](#)
- [HOSG08] P. Hliněný, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51(3):326, 2008. [91](#), [95](#)
- [HOSW02] R. Hall, J. Oxley, C. Semple, and G. Whittle. On matroids of branch-width three. *Journal of Combinatorial Theory, Series B*, 86(1):148–171, 2002. [93](#)
- [HW06] P. Hliněný and G. Whittle. Matroid tree-width. *European Journal of Combinatorics*, 27(7):1117–1128, 2006. [93](#), [94](#)
- [IFF01] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48(4):761–777, 2001. [88](#)
- [IW97] R. Impagliazzo and A. Wigderson. P= BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 220–229. ACM, 1997. [xi](#), [xvii](#)
- [Jer03] M. Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Birkhäuser, 2003. [18](#)
- [JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988. [9](#), [23](#), [24](#), [26](#), [29](#), [34](#)
- [Kan08] M. Kanté. *Graph Structurings: Some Algorithmic Applications*. PhD thesis, Université Bordeaux 1, 2008. [85](#)
- [Kas61] P.W. Kasteleyn. The statistics of dimers on a lattice. *Physica*, 27:1209–1225, 1961. [19](#), [79](#)
- [Kay10] N. Kayal. Algorithms for Arithmetic Circuits. *ECCC Report TR10-073*, 2010. [74](#)
- [KBE<sup>+</sup>05] L. Khachiyan, E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, 2005. [10](#), [23](#), [28](#), [33](#), [41](#), [108](#)
- [KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1):1–46, 2004. [xi](#), [xvii](#)

- [KLL00] E. Kaltofen, W. Lee, and A.A. Lobo. Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. In *Proceedings of the 2000 international symposium on Symbolic and algebraic computation*, pages 192–201. ACM New York, NY, USA, 2000. 47, 60
- [KMSV10] Zohar S. Karnin, Partha Mukhopadhyay, Amir Shpilka, and Ilya Volkovich. Deterministic identity testing of depth-4 multilinear circuits with bounded top fan-in. In *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*, pages 649–658, New York, NY, USA, 2010. ACM. xi, xvii, 47, 70
- [Kra09] D. Kral. Decomposition width—a new width parameter for matroids. *Arxiv preprint arXiv:0904.2785*, 2009. 96, 126, 136
- [KS93] D. Kavvadias and M. Sideri. On Horn envelopes and hypergraph transversals. 1993. 23
- [KS01] A.R. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 216–223. ACM New York, NY, USA, 2001. xii, xvii, 47, 48, 60, 63, 67
- [KS08] Z.S. Karnin and A. Shpilka. Black box polynomial identity testing of generalized depth-3 arithmetic circuits with bounded top fan-in. In *Conference on Computational Complexity*, pages 280–291, 2008. 47
- [KSS00] D.J. Kavvadias, M. Sideri, and E.C. Stavropoulos. Generating all maximal models of a Boolean expression. *Information Processing Letters*, 74(3-4):157–162, 2000. 23, 27, 30, 41
- [Kur30] K. Kuratowski. Sur le probleme des courbes gauches en topologie. *Fund. Math*, 15(271-283):79, 1930. 2
- [KV91] S. Khuller and V.V. Vazirani. Planar graph coloring is not self-reducible, assuming  $P \neq NP$ . *Theoretical Computer Science*, 88(1):183, 1991. 30
- [KV05] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13(3):91–130, 2005. 18, 68, 79
- [LFK92] C. Lund, L. Fortnow, and H. Karloff. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):868, 1992. 53
- [Lib04] L. Libkin. *Elements of finite model theory*. Springer Verlag, 2004. 14, 15

- [Lov80] L. Lovász. Matroid matching and some applications. *J. Combin. Theory Ser. B*, 28(2):208–236, 1980. [xiii](#), [xix](#), [78](#)
- [McC80] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial intelligence*, 13(1-2):27–39, 1980. [41](#)
- [MP08] G. Malod and N. Portier. Characterizing Valiant’s algebraic complexity classes. *Journal of complexity*, 24(1):16–38, 2008. [74](#)
- [MT07] F. Mazoit and S. Thomassé. Branchwidth of graphic matroids. *Surveys in combinatorics*, 346:275–286, 2007. [93](#)
- [MV02] G. Masbaum and A. Vaintrob. A new matrix-tree theorem. *International Mathematics Research Notices*, 2002(27):1397, 2002. [77](#), [78](#)
- [OS06] S. Oum and P. Seymour. Approximating clique-width and branchwidth. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006. [88](#), [90](#), [93](#)
- [Oum08] S. Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008. [89](#)
- [Oxl92] J.G. Oxley. *Matroid Theory*. Oxford University Press, 1992. [3](#), [115](#), [116](#), [117](#), [121](#)
- [Ple79] Ján Plesník. The np-completeness of the hamiltonian cycle problem in planar digraphs with degree bound two. *Inf. Process. Lett.*, 8(4):199–201, 1979. [74](#)
- [PP94] C.H. Papadimitriou and CH Papadimitriou. *Computational complexity*. Addison-Wesley Reading, MA, 1994. [6](#)
- [PR94] G. Pruesse and F. Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2):373–386, 1994. [23](#)
- [Rot01] G. Rote. Division-free algorithms for the determinant and the pfaffian: algebraic and combinatorial approaches. *Lecture Notes in Computer Science*, pages 119–135, 2001. [18](#)
- [RS83] N. Robertson and P.D. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. [xiii](#), [xviii](#), [86](#)
- [RS86] N. Robertson and P.D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986. [xiii](#), [xviii](#), [86](#)
- [RS91] N. Robertson and P.D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991. [xiii](#), [xviii](#), [89](#), [93](#)

- [RS95] N. Robertson and PD Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. [xiii](#), [xviii](#), [41](#)
- [RT75] RC Read and RE Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975. [41](#)
- [Sax09] N. Saxena. Progress on Polynomial Identity Testing. *Bull. EATCS*, 99:49–79, 2009. [47](#)
- [Sch80] JT Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):717, 1980. [xi](#), [xvii](#), [50](#)
- [Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 330–335. ACM, 1983. [13](#)
- [SS10] N. Saxena and C. Seshadhri. From sylvester-gallai configurations to rank bounds: Improved black-box identity test for depth-3 circuits. 2010. [xi](#), [xvii](#)
- [Str10] Y. Strozecki. Enumeration of the monomials of a polynomial and related complexity classes. *Mathematical Foundations of Computer Science*, pages 629–640, 2010. [48](#)
- [TIAS77] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6:505, 1977. [29](#)
- [Tod] S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Transactions on Information and Systems, E*, 75:116–124. [74](#)
- [Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):877, 1991. [13](#)
- [TW68] JW Thatcher and JB Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Theory of Computing Systems*, 2(1):57–81, 1968. [xiii](#), [xviii](#), [86](#), [95](#)
- [Uno97] T. Uno. Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. *Algorithms and Computation*, pages 92–101, 1997. [x](#), [xvi](#), [11](#), [23](#), [30](#), [33](#), [62](#)
- [Val79] L.G. Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979. [x](#), [11](#), [19](#), [76](#)

- [vEB91] P. van Emde Boas. Machine models and simulations, Handbook of theoretical computer science (vol. A): algorithms and complexity, 1991. 8
- [VZG87] J. Von Zur Gathen. Feasible arithmetic computations: Valiant's hypothesis. *Journal of Symbolic Computation*, 4(2):137–172, 1987. 71
- [Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. *Symbolic and algebraic computation*, pages 216–226, 1979. xi, xvii
- [Zip90] R. Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9(3):375–403, 1990. 47, 50, 60, 67, 68