UNIVERSITÉ D'ARTOIS

ÉCOLE DOCTORALE SCIENCES TECHNOLOGIES ET SANTÉ N°585

Hard Functions in Knowledge Compilation: from Lower Bounds to Applications

DOCTORAL THESIS

presented and publicly defended on September 26th, 2022

in partial fulfillment of requirements for the degree of

Doctor of Philosophy of Artois University

in the subject of Computer Science

by

Alexis de Colnet

Doctoral Committee

Advisor:	Pierre Marquis	University of Artois
Supervisor:	Stefan Mengel	CNRS
Chair:	Sophie Tison	University of Lille
Reviewers:	Adnan Darwiche	University of California
	Stefan Szeider	Technische Universität Wien
Examiner:	Florent Capelli	University of Lille

CENTRE DE RECHERCHE EN INFORMATIQUE DE LENS – CNRS UMR 8188
Université d'Artois, rue Jean Souvraz, S.P. 18 F-62307, Lens Cedex France
Secrétariat : +33 (0)3 21 79 17 23
http://www.cril.univ-artois.fr

Mise en page avec memcril (B. Mazure, CRIL) et thloria (D. Roegel, LORIA).

Acknowledgements

I would like to thank the reviewers of this thesis: Adnan Darwiche, the father of DNNF, whose many contributions to knowledge compilation are fundamental for many aspects of my research, and Stefan Szeider, who has accepted to take some time to review my thesis despite already having spent many hours to help me write my post-doctoral project. I would also like to thanks the examiners: Florent Capelli and Sophie Tison who has accepted to chair the committee.

I am deeply thankful to my supervisors Pierre Marquis and Stefan Mengel. They have been mentors when I needed to learn from them, and collegues when we were working together. I am grateful that they thought of me to work with them on many research subjects, as well as for the freedom that they let me so that I could explore my own research. I could not have hoped for a best guidance during these three years. I also thank the my master degree supervisor Kuldeep S. Meel who introduced me to research. It is partly thanks to him that I have been accepted for a doctorate at CRIL.

I thank my friends and colleagues at CRIL. This includes the teaching staff, the research staff, the managment staff and, of course, all my fellow PhD students. I give very special thanks to Ryma, Marie and Thanh, who started their PhD at the same time as me, and with whom I have built a strong friendship.

I also give a particular thank you to my family who supported me during all my studies.

Finally, as it is routine to acknowledge the agencies funding our research, I gladly acknowledge one more time that this work has been partly supported by the PING/ACK project of the French National Agency for Research (ANR-18-CE40-0011).

Contents

List of Figures

Résumé en français

En sciences informatiques, un *circuit* est une structure utilisée pour représenter de nombreux types de fonctions, et vers laquelle de nombreuses autres structures peuvent être réduites, dont les diagrames de décision binaires (ou BDD pour « binary decision diagrams »), les arbres de décisions, les formules sous forme normal conjonctive (ou formules CNF pour « conjunctive normal form »), les polynômes, etc. Le calcul d'une fonction par un circuit est divisé en plusieurs étapes correspondant aux portes du circuit, de sorte que le circuit agit comme un programme de calcul pour cette fonction. L'analogie entre circuits et programmes de calcul transparait jusque dans le nom de certains types de cirucits et diagrames, par exemple les programmes de branchement (« branching program », une autre appellation des BDD) et les programmes-$(+, \times)$ (« $(+, \times)$-programs », le nom donné par Valiant aux circuits arithmetiques [Val80]). Ainsi il y a une correspondance entre la taille du plus petit circuit d'un type donné représentant une fonction et la longueur du plus court programme correspondant pour calculer cette fonction.

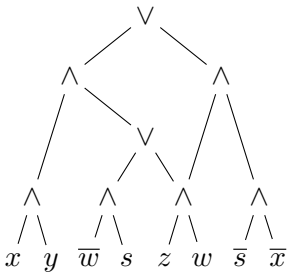
En plus de permettre le calcul de la fonction pour n'importe quelles affectations de variables, certains circuits simplifient la détermination de plusieurs informations sur la fonction, par exemple son nombre de solutions (ou modèles) dans le cas d'une fonction Booléenne. De tels circuits, parfois désignés en anglais par « tractable circuits » [VCL+21, Dar22] (grossièrement traduit par « circuits abordables ») sont au cœur de la *compilation de connaissances*. La compilation de connaissances traite de situations où une base de connaissances, connue en avance, est prétraitée lors d'une phase coûteuse en ressource mais hors-ligne appelée *phase de compilation*, avant son utilisation lors d'une *phase d'exécution*. Dans l'idéal, le coût de la compilation est amorti par les gains de temps obtenus en phase d'exécution pour manipuler et répondre aux différentes requêtes sur la base compilée, en particulier lorsque le nombre de requêtes à traiter est élevé. Dans ce contexte, les classes de circuits abordables forment des *langages de compilation*: quand la base initiale prend la forme d'une fonction, la compilation consiste à trouver un circuit dans cette classe qui calcule la fonction.

Parmi les langages de compilation introduits durant ces dernières vingt années pour les fonctions Booléennes, la classe des circuits sous forme normal négative décomposable, dits circuits en DNNF (pour « Decomposable Negation Normal Form »), occupe une place centrale [Dar01a]. De tous les types de circuits à notre disposition, les circuits DNNF sont parmis les plus généraux permettant un traitement un temps linéaire des requêtes d'implication clausale. Cette requête demande, pour une clause donnée, si cette clause est satisfaite par toutes les solutions du circuit. L'implication clausale est une requête historiquement importante en compilation de connaissances [SK96, Sin02], au point que l'existence d'un algorithme polynomial du test d'implication clausale est, selon certaines sources, une condition nécessaire pour définir les langages de compilation [DM02]. Ainsi la classes de circuits en DNNF, que l'on appellera désormais le langage DNNF, est un des langages de compilation les plus généraux qui ait été étudié. L'intérêt pour la classe des circuits en DNNF s'explique aussi par l'existence de différentes sous-classes utiles pour le comptage de modèles [Dar11, LM17], pour la résolution de problèmes Max-SAT [PD07], pour l'énumération de modèles [ABJM17], pour la résolution de QBF [CM19], etc.

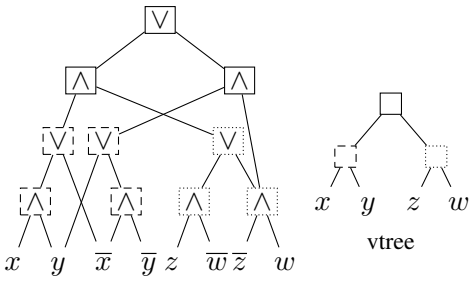
Cette thèse est centrée autour du langage DNNF. Elle peut être vue comme un florilège de l'analyse de circuits en DNNF et de leurs applications, même si la majorité des contributions sont des résultats

« négatifs ». Après une section de préliminaires, la thèse est divisée en trois parties. Dans la première parties, nous étudions la complexité de la compilation dans le langage DNNF en prouvant de nouvelles bornes inférieures sur la taille minimale de circuits en DNNF représentant exactement, ou de manière approchée, des fonctions particulières. Ces bornes inférieures dépendent de manière exponentielle du nombre de variables de la fonction ou d'un paramètre de la fonction, ce sont donc des résultats négatifs et inconditionels pour la compilation des fonctions étudiées en circuits DNNF. Les résultats négatifs ne sont pas toujours une fin en soi et, dans la seconde partie de la thèse, nous donnons des applications de nos bornes inférieures à la fois pour la compilation de connaissances et pour des domaines en apparence sans connexion avec la compilation de connaissances. Enfin, dans la troisième partie, nous explorons de nouveaux horizons pour la compilation de connaissances. Cela passe par l'étude de nouvelles requêtes pour les sous-langages de DNNF et par l'étude de langages de compilations pour représenter des fonctions non-Booléennes. Dans le reste de cette section, nous décrivons plus en détails le langage DNNF et ses sous-langages. Puis nous présentons les sujets étudiés et nos contributions pour chaque sujet.

DNNF et ses sous-langages. Un circuit sous forme normale négative, ou circuit en NNF (pour « Negation Normal Form ») est un graphe acyclique orienté dont les nœuds sont étiquetés par des opérateurs $\vee$ ou $\wedge$, et dont les feuilles sont étiquetées soit par des constantes Booléennes 0 (*faux*) ou 1 (*vrai*), soit par des litéraux sur des variables Booléennes. Un circuit sous forme normale négative décomposable, ou circuit en DNNF (pour « Decomposable NNF »), est un circuit NNF dont les nœuds $\wedge$ respectent la propriété dite de *décomposabilité*, qui est que les ensembles de variables apparaissant sous les fils d'un même nœud $\wedge$ sont deux-à-deux disjoints [Dar01a]. Les deux circuits représentés sur la figure suivante sont des exemples de circuits en DNNF.



Un circuit en DNNF



Un circuit en str-DNNF avec son vtree.

Le langage DNNF est la classes des circuits en DNNF. Ce langage contient plusieurs autres circuits et formules connus – dont les formules DNF – et est *complet* dans le sens où toute fonction Booléenne sur un nombre fini de variables admet une représentation en un circuit en DNNF. La décomposabilité est une restriction sur la structure du circuit grâce à laquelle plusieurs requêtes, pour lesquelles il est peu probable qu'il existe un algorithme fonctionnant en temps polynomial pour n'importe quel circuit, deviennent faisable en temps polynomial. En particulier, décider la satisfiabilité d'un circuit en DNNF et déterminer si ses modèles satisfont une clause donnée sont deux requêtes faisables en temps linéaire en la taille du circuit (le nombre d'arêtes du graphe). En plus de la décomposabilité, d'autres restrictions sur la structure des circuits ont été inventées, notamment la *decomposabilité structurée* et le *déterminisme*, que nous utilisons à plusieurs reprises dans la thèse.

La décomposabilité structurée est une variante de la décomposabilité qui ajoute une contrainte sur la manière selon laquelle l'ensemble des variables d'un nœud $\wedge$ est partitionné entre ses fils [PD08]. Pour l'instant nous en omettons la définition formelle. Nous nous contentons de mentionner que l'agencement des variables dans circuits en NNF respectant la décomposabilité structurée, ou circuits en str-DNNF

(pour « structured DNNF »), est décrit par un *vtree*: un arbre binaire muni d'une bijection entre ses feuilles et les variables du circuit. Un exemple de vtree est visible pour le deuxième circuit de la figure précédente. La classe des circuits en str-DNNF est le langage str-DNNF. Comparée à la simple décomposabilité, la décomposabilité structurée rend plus de manipulations des circuits faisables en temps polynomial, notamment les opérations entre deux circuits respectant le même vtree.

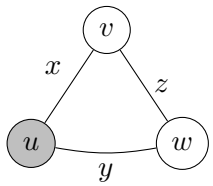
Le déterminisme est une propriété sur les nœuds $\vee$ des circuits. Un nœud $\vee$ est dit déterministe quand il n'existe aucune affectation de variables pouvant satisfaire un même temps les fonctions représentées par n'importe quels deux de ses fils [Dar01a]. Les circuits en DNNF dont les nœuds $\vee$ sont déterministes, ou circuits en d-DNNF (pour « deterministic DNNF »), forment le langage d-DNNF. La combinaison de la décomposabilité et du détermisme rend possible le calcul du nombre de solutions d'un circuit en d-DNNF en temps polynomial, ce qui fait de d-DNNF l'un des langages les plus usités. Une manière simple d'assurer le déterminisme des nœuds $\vee$ est faire en sorte qu'ils calculent des *décompositions de Shannon*, c'est-à-dire que chaque nœud $\vee$ soit de la forme $(\overline{x} \wedge \cdot) \vee (x \wedge \cdot)$ avec x une variable. Les circuits en d-DNNF dont les nœuds $\vee$ sont de cette forme sont appelés circuits en dec-DNNF (pour « decision DNNF ») et la classse de ces circuits est le langage dec-DNNF.

Nous finissons cette brève présentation des langages de compilation en évoquant le cas des diagrammes de décision. De nombreuses type diagrammes de décision, notamment les diagrammes de decision binaire ordonnés (OBDD) [Bry86, Weg00], les diagrammes de decision binaire dits « read-once » (FBDD) [SW95] et les diagrammes de décision sentencielle (SDD) [Dar11] peuvent être ré-écrit en circuits en d-DNNF en temps linéaire. Ainsi les classes de ces diagrammes (en particulier OBDD, FBDD et SDD) sont souvent vues comme des sous-langages de d-DNNF.

La taille minimale de fonctions dans DNNF. Étant donné L un langage de compilation, la *taille minimale* d'une fonction dans L est la taille de la plus petite representation de la fonction dans L. L'existence de classes entières de fonctions dont la taille minimale dans L évolue raisonablement selon le nombre de variable est un argument en faveur de l'utilisation de L comme langage de compilation en pratique. Par exemple on définit pour les formules CNF un graphe dit primal, dont les sommets sont les variables de la formule et tel qu'une arête relie deux variables si et seulement si elles apparaissent ensemble dans une même clause. Or, pour une constante fixée c, les formules CNF dont le graphe primal a une treewidth bornée par c ont une taille minimal dans DNNF qui est polynomiale en le nombre de variables (la treewidth d'un graphe étant un paramètre bien connu dont nous omettons la définition) [Dar01a]. Il est également utile d'avoir à se disposition un certains nombres de fonctions dont la taille minimale dans L dépend plus que polynomialement du nombre de variables. De telles fonctions permettent notamment de comparer L avec d'autres langages en termes d'*efficacité spatiale*: quand une famille infinie de fonctions est telles que leur taille minimale dans L est exponentiellement plus grande que leur taille minimale dans un autre langage L', on peut dire que L' permet une représentation plus efficace, car plus concise, de ces fonctions que L. À titre d'exemple, Wegener donne dans les chapitres 4 et 6 de [Weg00] plusieurs familles de fonctions dont la taille minimale dans OBDD est exponentiellement plus grande que la taille minimale dans FBDD. Plus généralement, la comparaison de circuits en termes d'efficacité spatiale est un des trois axes utilisés pour comparer les langages dans la *carte de compilation de connaissances*, un compendium de langages de compilation introduit par Darwiche et Marquis [DM02].

Le premier résultat négatif sur la taille minimale de fonctions dans DNNF précède, en un sens, la création du langage puisque Selman et Kautz ont montré dans les années 90 qu'à moins que NP $\subseteq$ P/poly, s'il existe un test d'implication clausal fonctionnant en temps polynomial pour des fonctions représentées dans un certain langage de compilation (ce qui est le cas pour DNNF), alors la taille minimale de toute formule Booléenne dans ce langage ne peut pas être bornée par un polynôme en la taille

de la formule [SK91, SK96]. C'est un résultat valable pour tous les langages de compilation, mais qui a le défaut d'être conditionel (« à moins que NP $\subseteq$ P/poly »). Bien après les travaux de Selman et Kautz, une séparation exponentielle et inconditionelle a été montrée entre la classe des formules CNF et le langage DNNF [BCMS14]. Pour expliquer ce qui rend une formule difficile à compiler vers DNNF, une première direction consiste à déterminer des caractéristiques des fonctions Booléennes en lesquelles la taille minimale dans DNNF dépend exponentiellement [JS12, RP13, STV14, BS17a]. Une autre ligne de recherche consiste à étudier la compilation de fonctions spécifiques vers DNNF. Cette recherche a permi d'identifier des classes de fonctions simples à compiler vers DNNF, par exemple les formules CNF dont la treewidth du graphe primal est bornée par une constante [Dar01a] ou les formules CNF dites variable-convexes [BS17b]. Mais elle a aussi généré des résultats négatifs qui, la plupart du temps, prennent la forme de bornes inférieures exponentielles sur la taille minimale dans DNNF, notamment pour les formules CNF monotones [ACMS20] ou pour des systèmes d'équations linéaires [Men16]. Les résultats que nous présentons dans le chapitre 1 s'inscrivent parmi ces résultats négatifs. Nous montrons des bornes inférieures exponentielles sur la taille minimale de certaines contraintes pseudo-Booléennes dans DNNF [dC20] et sur la taille minimale de formules CNF dites de Tseitin (qui seront définies plus loin) dans DNNF [dCM21a]. Après un état de l'art sur la compilation vers DNNF, deux sections sont dédiées aux preuves de chaque des bornes inférieures. La première borne montre que la compilation vers DNNF de contraintes pseudo-Booléennes de la forme $w_1x_1 + \cdots + w_nx_n \geq \theta$, où $w_1, \ldots, w_n, \theta$ sont des nombres réels et $x_1, \ldots, x_n$ sont des variables Booléennes, peut générer des circuits beaucoup trop grands pour être manipulés en pratique. Pourtant la compilation de telles contraintes en DNNF est utile, par exemple pour la génération d'encodage CNF des contraintes ayant des propriétés désirées en programmation par contraintes [AGMS16, KS19]. La deuxième borne inférieure resulte d'un travail commun avec Stefan Mengel. Elle traite de formules de Tseitin. Ces formules jouent un rôle important dans cette thèse. Ce sont des formules CNF representant des systèmes de contraintes de parité structurés par des graphes. Pour un graphe G non-orienté, on associe à chaque sommet v une valeur 0 ou 1 que l'on notera $c(v)$. Les arêtes de G correspondent aux variables de la formule et chaque sommet v est associé à une contrainte de parité χ_v sur les variables $x_1, \ldots, x_k$ correspondant aux arêtes $e_1, \ldots, e_k$ incidentes à v. La contrainte de parité est $\chi_v : x_1 + \cdots + x_k = c(v) \mod 2$. Elle peut être représentée par une formule CNF contenant 2^{k-1} clauses, toutes de taille k. La formule de Tseitin pour le graphe G et le choix des étiquettes $c(v)$ pour chaque sommet v est la conjonction des formules CNF pour chacun de ses sommets. On la note $T(G, c)$. Un exemple de formule de Tseitin est donné par le graphe G suivant où les sommets blancs sont étiquetés par 1 et les sommets gris par 0.



$$\chi_u : x + y = 0 \mod 2 \equiv (\overline{x} \vee y) \wedge (x \vee \overline{y})$$
$$\chi_w : y + z = 1 \mod 2 \equiv (y \vee z) \wedge (\overline{y} \vee \overline{z})$$
$$\chi_v : x + z = 1 \mod 2 \equiv (x \vee z) \wedge (\overline{x} \vee \overline{z})$$
$$T(G, c) = (\overline{x} \vee y) \wedge (x \vee \overline{y}) \wedge (y \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (x \vee z) \wedge (\overline{x} \vee \overline{z})$$

Nous montrons dans le chapitre 1 que, pour toute constante Δ, la taille minimale de telles formules dans DNNF, quand les graphes sont de degré maximum au plus Δ, est exponentielle en la treewidth du graphe. Des applications pour cette borne inférieure sont données dans les chapitres 2, 3 et 4.

La taille minimale d'approximations dans d-DNNF. Quand une fonction a une taille minimale dans DNNF trop grande pour que la compilation soit viable, il est parfois envisageable de compiler une approximation de la fonction dans l'espoir que sa taille minimale dans DNNF soit plus petite. Avec Stefan Mengel, nous avons étudié la compilation de connaissances approchée vers le langage d-DNNF [dCM20]. Le principal attrait de d-DNNF comparé à DNNF est que le nombre de modèles

de circuits en d-DNNF est calculable en temps polynomial [DM02]. En contrepartie, compiler vers d-DNNF est généralement beaucoup plus coûteux que compiler vers DNNF, puisque la taille minimale des fonctions dans d-DNNF est généralement exponentiellement plus grande que leur taille minimale dans DNNF [BCMS16]. Il existe plusieurs stratégie pour la compilation approchée vers différents langages [SK96, Dar01a, BSW02, PD07] qui, à l'exception de celle décrite dans [BSW02], ne permettent pas de mesurer précisement de la qualité de l'approximation. Nous avons considéré deux notions d'approximation offrant différentes garanties sur l'erreur d'approximation. Nous appelons ces notions *l'approximation faible* et *l'approximation forte*. L'approximation faible a été étudié par Bollig, Sauerhoff et Wegener pour la compilation vers le langage OBDD [BSW02]. On peut résumer l'approximation faible de la manière suivante: pour une fonction f, la fonction $\tilde{f}$ ayant les même variables est appelée une ε-approximation faible de f quand la probabilité que $f(a)\tilde{f}(a)$, pour a une affectation aléatoire des variables, est inférieure à ε. Grace aux travaux de Bollig et al. [BSW02], il est connu qu'il existe des fonctions dont les approximations faibles ont toutes une taille minimale dans OBDD qui est exponentielle en leur nombre de variables. Dans le chapitre 2, nous présentons nos contributions pour la compilation de connaissances approchée. Après une section d'état de l'art sur la compilation approchée, nous montrons que les résultats négatifs prouvés par Bollig et al. pour le langage OBDD s'étendent au langage d-DNNF, ce qui n'est pas trivial puisque les circuits en d-DNNF sont generalement exponentiellement plus petits que n'importe quels OBDD représentant la même fonction. Nous expliquons ensuite que l'approximation faible s'avère insuffisante de nombreux cas, puisqu'elle permet d'approximer des fonctions par la fonction qui vaut tout le temps zéro. Une telle approximation étant inutile dans beaucoup de situation, par exemple pour le comptage de modèles approché. Dans une troisième section, nous définissons l'approximation forte pour résoudre ce problème. Celle-ci a été conçue pour le comptage de modèles approché, elle garantie que le nombre de modèle de l'approximation est bornée de chaque côté par le nombre de modèles de la fonction initiale fois un facteur constant. Nous montrons qu'il existe des classes de fonctions qui peuvent être faiblement approximées par la fonction zéro, mais dont la taille minimale dans d-DNNF de leurs approximations fortes est toujours exponentielle en le nombre de variables. Vers la fin du chapitre 2, nous utilisons les bornes inférieures sur la taille minimale de formules de Tseitin dans DNNF obtenues au chapitre 1 pour prouver que notre résultat s'applique notamment à certaines classes de formules CNF.

Analyser la compilation ascendante vers str-DNNF. Même quand il existe un petit circuit représentant une fonction donnée dans un langage, générer un tel circuit peut être extrêmement coûteux selon la méthode de compilation utilisée. Dans le chapitre 3, nous utilisons nos résultats du chapitre 1 sur la taille minimale de formules de Tseitin dans DNNF pour construire des exemples de fonctions exhibant ce comportement pour la *compilation ascendante* (ou compilation « bottom-up »). La compilation ascendante est un des deux paradigmes majeurs pour la compilation avec la *compilation descendante* (ou « top-down »). Dans une compilation descendante, un circuit représentant la fonction initiale est construit comme la trace d'un algorithme explorant la quasi-intégralité de l'espace des modèles de la fonctions [HD05]. En compilation ascendante, on suppose souvent que la fonction initiale est un système de constraintes, par exemple une formule CNF où les contraintes sont des clauses. L'approche se schématise en deux étapes : premièrement, chaque contrainte est individuellement compilée dans le langage, et deuxièmement, les circuits ainsi obtenus sont combinés deux-à-deux jusqu'à obtenir un circuit représentant la fonction initiale. La deuxième étape requière que la conjonction de deux circuits du langage soit faisable en temps polynomial, ce qui n'est generalement possible que quand les circuits en question sont des circuit en str-DNNF respectant le même vtree [PD08]. Ainsi la compilation ascendante est essentiellement une méthode pour compiler vers le langage str-DNNF qui, rappelons-le, généralise les langages OBDD et

SDD [Dar11]. La figure suivante représente une compilation ascendante dans le langage OBDD de la formule CNF $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3)$. Cette compilation utilise trois instructions : Compile pour compiler une clause en OBDD en spécifiant l'ordre π d'apparition des variables, Apply qui permet de réaliser la conjonction de deux OBDD, et Restructure qui permet de modifier un OBDD sans changer la fonction qu'il représente.

$B_1 = \mathsf{Compile}(x_1 \vee \overline{x_2}, \pi)$

$B_2 = \mathsf{Compile}(\overline{x_1} \vee x_2, \pi)$

$B_3 = \mathsf{Apply}(B_1, B_2, \wedge)$

$B_4 = \mathsf{Compile}(x_1 \vee x_2 \vee x_3, \pi')$

$B_5 = \mathsf{Compile}(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}, \pi')$

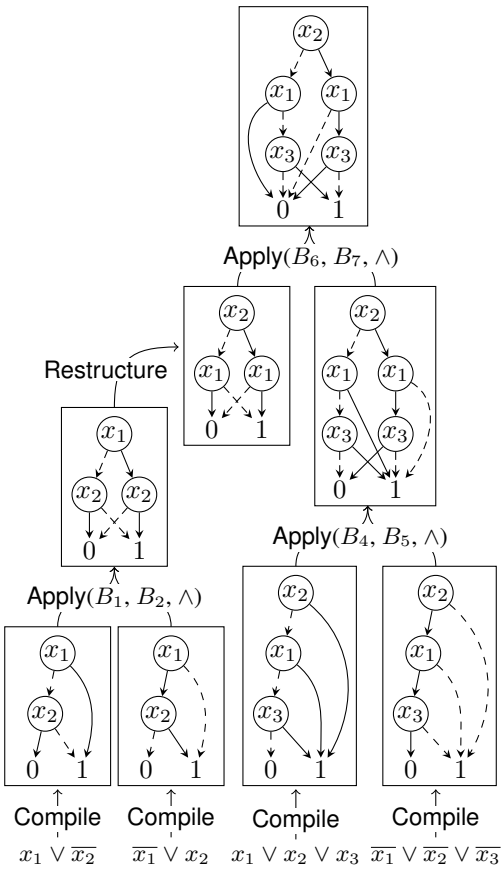
$B_6 = \mathsf{Apply}(B_4, B_5, \wedge)$

$B_7 = \mathsf{Restructure}(B_3, \pi')$

$B_8 = \mathsf{Apply}(B_6, B_7, \wedge)$

où $\pi(x_1) < \pi(x_2) < \pi(x_3)$

et $\pi'(x_2) < \pi'(x_1) < \pi'(x_3)$



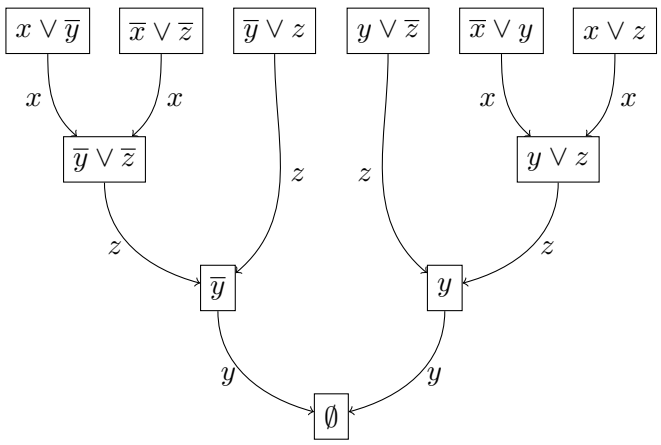
Une compilation ascendante dans le langage OBDD

Comme décrit précédemment, la compilation ascendante génère des circuits intermédiaires, lesquels peuvent être beaucoup plus grands que le système initial et que le circuit obtenu à l'issue de la compilation. Ce phénomène a été observé en pratique [NW07] et prouvé pour la compilation ascendante vers OBDD de différentes formules CNF [Kra08, Seg08, FX13, IKRS20]. Le chapitre 3 de la thèse décrit nos contributions dans l'analyse de la compilation ascendante vers str-DNNF [dCM22b]. Dans les deux premières sections, nous formalisons la compilation ascendante et nous dressons un état de l'art de la compilation ascendante en théorie et en pratique. Dans le reste du chapitre, il est montré comment utiliser la borne inférieure obtenue dans le chapitre 1 sur la taille minimale dans DNNF des formules de Tseitin pour construire des formules CNF de taille polynomiale en le nombre de variables, et dont la taille minimale dans str-DNNF est constante, mais telles que toute compilation ascendante vers str-DNNF génère des circuits intermédiaires de taille exponentielle en la treewidth de leur graphe primal. Le formalisme que nous utilisons pour décrire la compilation ascendante permet que des modifications arbitraires soient apportées aux circuits intermédiaires à certains moments de la compilation, tant que ces modifications préservent les fonctions représentées (c'est le rôle de la méthode Restructure dans l'exemple de la figure). Grace à cette particularité, le fonctionnement des compilateurs utilisant l'approche ascendante [Rud93, CD13]

s'inscrit dans notre modèle et nos résultats s'appliquent à ces compilateurs. À notre connaissance, seules les bornes inférieures prouvées par Itsykson, Knop, Romashchenko et Sokolov pour les OBDD utilisent un tel modèle de compilation [IKRS20]. Nous commençons par prouver notre résultat pour la compilation ascendante de formules de Tseitin non-satisfiables, puis nous étendons le résultat à des formules CNF satisfiables construites sur ces formules de Tseitin. Avec notre formalisme, la compilation ascendante de formules non-satisfiable génère une séquence de circuits qui constitue une preuve de la non-satisfiabilité de la formule, ou une *réfutation* de la formuler. Ainsi notre principale contribution est l'obtention de bornes inférieures sur la taille des réfutations de formules de Tseitin non-satisfiables dans des systèmes de preuve à base de circuits en str-DNNF (ce qui inclue les systèmes de preuve à base d'OBDD et de SDD). Itsykson et al. ont également étudié des formules de Tseitin pour obtenir les résultats publiés dans [IKRS20]. Nos résultats généralisent les leurs de deux manières : premièrement, nous considérons des circuits en str-DNNF, lesquels peuvent être considérablement plus petits que les OBDD équivalents (mais pas l'inverse), et deuxièmement, leurs résultats ne s'appliquent qu'à des formules de Tseitin dont les graphes sont restreints à une classe particulière, alors que les nôtres sont des résultats paramétrés mais qui s'appliquent aux formules de Tseitin pour n'importe quel graphe.

La compilation de connaissances rencontre la complexité de preuves. Une dernière application de notre borne inférieure sur la taille minimale des formules de Tseitin dans DNNF est donnée dans le chapitre 4. Cette application fait le lien entre la compilation de connaissances et la complexité de preuves, qui est le domaine historiquement lié aux formules de Tseitin. Les formules de Tseitin non-satisfiables ont été introduites par Grigori Tseitin dans les années 60 comme des exemples de formules difficiles à réfuter dans le système de preuve par résolution [Tse68, Tse83]. Ce système de preuve est basé sur une seule règle d'inférence appelée la règle de *résolution*, selon laquelle une formule qui satisfait deux clauses $C \vee x$ et $C' \vee \overline{x}$ satisfait également la clause $C \vee C'$. La clause $C \vee C'$ est appelée la *résolvante* de $C \vee x$ et $C' \vee \overline{x}$ et est notée $C \vee C' = \mathsf{Resolution}(\mathsf{C} \vee \mathsf{x}, \mathsf{C}' \vee \overline{\mathsf{x}}, \mathsf{x})$. Toute formule CNF non-satisfiable a une réfutation dans le système de preuve par résolution, c'est-à-dire, une séquence de clauses dont la dernière est la clause vide (signe que la formule est non-satisfiable) et telle que chaque clause soit vient directement de la formule, soit est la résolvante de deux clauses apparaissant précédemment dans la séquence. Par exemple la figure ci-après montre une résolution par réfutation de la formule $(x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{z}) \wedge (x \vee z) \wedge (\overline{y} \vee z) \wedge (y \vee \overline{z})$, avec sa représentation sous forme de graphe acyclique orienté.

$C_1 = x \vee \overline{y}$
$C_2 = \overline{x} \vee \overline{z}$
$C_3 = \overline{y} \vee z$
$C_4 = y \vee \overline{z}$
$C_5 = \overline{x} \vee y$
$C_6 = x \vee z$
$C_7 = \texttt{Resolution}(C_1, C_2, x) = \overline{y} \vee \overline{z}$
$C_8 = \texttt{Resolution}(C_5, C_6, x) = y \vee z$
$C_9 = \texttt{Resolution}(C_3, C_7, z) = \overline{y}$
$C_{10} = \texttt{Resolution}(C_4, C_8, z) = y$
$C_{11} = \texttt{Resolution}(C_9, C_{10}, y) = \emptyset$



L'intuition de Tseitin était que les réfutations de formules de Tseitin dans le système de preuve par résolution ne peuvent être constituées d'un nombre polynomial de clauses. Cette intuition a été confirmée par Urquhart pour des formues de Tseitin non-satisfiables dont les graphes sont des *expanseur* (en anglais « expander graphs ») [Urq87]. Caractériser les formules de Tseitin non-satisfiables dont les réfutations par résolution sont de taille plus que polynomiale en le nombre de variables reste un problème ouvert, en dépit de plusieurs avancées sur la question [AR11, IO13, GTT20]. En particulier, pour le système de preuve plus restrictif dit par resolution *régulière*, Alekhnovic et Razborov ont montré que les formules de Tseitin non-satisfiables peuvent être réfutées en utilisant au plus $2^{O(tw(G))}poly(n)$ clauses ou n est le nombre de variables et $tw(G)$ est la treewidth du graphe [AR11]. On comparaison, Itsykson et al. ont montré que toute réfutation par résolution régulière de ces même formules, quand leurs graphes sont de degré maximal borné par une constante, nécessitent au moins $2^{\Omega(tw(G)/\log(n)))}$ clauses [IRSS21] . Notre principale contribution, avec Stefan Mengel, a été d'utiliser notre borne inférieure sur la taille minimale des formules de Tseitin satisfiables dans DNNF pour améliorer la borne inférieure de [IRSS21]. Notre borne montre que, quand les graphes sont de degré maximal borné par une constante, les réfutations par résolution régulière de formules de Tseitin non-satisfiable génèrent au moins $2^{\Omega(tw(G))}poly(1/n)$ clauses. Pour ces formules, nous atteignons donc la borne supérieur d'Alekhnovic et Razborov. Par conséquent, le nombre minimal de clauses dans une réfutation par résolution régulière de formules de Tseitin non-satisfiables dont les graphes sont de dégré bornés, est polynomial en le nombre de variables n si et seulement si la treewidth du graphe est en $O(\log(n))$ [dCM21a]. La preuve de ce résultat est l'objet du chapitre 4. La première section du chapitre donne une description formelle de la résolution par réfutation régulière. S'ensuit une section où nous rappelons les bornes existantes sur le nombre minimal de clauses nécessaires pour réfuter une formule de Tseitin dans ce système de preuve. Les sections suivantes détaillent une réduction pour passer du poblème consistant à déterminer la taille minimale d'une réfutation par résolution régulière de formules de Tseitin non-satisfiables, au problème dans lequel l'on cherche à déterminer la taille minimale d'une formule de Tseitin satisfiable dans DNNF.

Des requêtes d'énumération pour les circuits en dec-DNNF. Le processus de compilation n'est qu'une partie de la compilation de connaissances, la seconde partie étant l'utilisation de la forme compilée pour répondre aux requêtes pendant la phase d'exécution. Bien que la thèse soit principalement axée autour du processus de compilation, des résultats ont été obtenus sur la complexité des requêtes d'énumération sur des circuits des langages de compilation. Ces résultats proviennent de travaux réalisés avec Pierre Marquis [dCM22a]. Les requêtes d'énumération diffèrent des requêtes de décision (comme le test d'implication clausale) et des requêtes fonctionnelles (comme le comptage de modèles) en ce que la requête n'a pas une seule réponse mais une liste de solutions. Par exemple, lister toutes les affectations de variables satisfaisant un circuit est une requête d'énumération qui a déjà été étudiée pour des circuits de langages de compilation [DM02, ABJM17, CS21]. Un petit circuit peut être satisfait par un nombre exponentiel d'affectations de ses variables, cet exemple montre que pour analyser l'efficacité d'un algorithme d'énumération, il convient de prendre en compte à la taille de l'entrée (ici, la taille du circuit) et la taille de la sortie. La théorie de la complexité d'énumération a été conçue spécialement pour catégoriser les problèmes d'énumération selon leur difficulté [Str19]. Cette théorie est présentée dans la première section du chapitre 5. Pour notre étude, nous nous sommes limités aux classes de complexité OutputP et IncP. OutputP regroupe les problèmes d'énumération pour lesquels l'ensemble des solutions peut être obtenu en temps polynomial en la taille de l'entrée plus la taille de la sortie (donc la taille cumulée de toutes les solutions). IncP est une sous-classe de OutputP qui contient exactement les problèmes d'énumération pour lesquels il existe un algorithme permettant de lister un nombre donné k de solutions en temps polynomial en la taille de l'entrée et k. L'avis général est qu'il existe des problèmes dans Out-

putP qui ne sont pas dans IncP, mais il n'existe à notre connaissance aucun problème « naturel » pour lequel cela ait été prouvé [Str19]. Dans la seconde et la troisième section du chapitre 5, nous étudions des requêtes d'énumération qui consistent à lister les impliquants premiers et les impliqués premiers de circuits en dec-DNNF. En particulier, nous montrons qu'énumérer tous les impliquants premiers (respectivement les impliqués premiers) de tels circuits est un problème dans OutputP, et même dans IncP. Dans la quatrième section, nous considérons l'énumération d'impliquants premiers particuliers appelés *raisons suffisantes* et *explications abductives minimales pour l'inclusion*, deux notions utilisées en intelligence artificielle (IA), et plus particulièrement en IA explicable [SL90, EG95, SCD18b, DH20, INM19]. Alors que nous avons pu trouver des résultats positifs concernant l'énumération de tous les impliquants premiers de circuits en dec-DNNF, nous montrons qu'il est peu probable qu'énumérer les impliquants particuliers précdémment cités soit dans OutputP même quand on restreint les circuits aux OBDD et aux DT.

La compilation de connaissances au-delà du cas Booléen. La compilation de connaissances n'est pas confinée aux fonctions Booléennes. Les langages étudiés dans cette thèse pour les fonctions Booléennes sont élaborés en imposant certaines propriétés aux circuits Booléens (décomposabilité, déterminisme, etc.). Ces propriétés ont des versions analogues pour des circuits représentant des fonctions non-Booléennes, et donc on peut créer des classes de circuits représentant ces nouvelles fonctions sur le modèle de DNNF et de ses sous-langages. Plusieurs travaux on déjà entrepris d'étudier ces propriétés pour des circuits calculant des fonctions réelles. Notamment, la décomposabilité a été étudiée pour les circuits arithmétiques sous le nom de *multilinéarité* [NW97] et, plus généralement la décomposabilité, le déterminisme, etc., ont été largement étudiés pour les circuits probabilistes, c'est-à-dire des circuits calculant des distributions de probabilité [VCL+21]. Comme pour le cas Booléens, différentes combinaisons de propriétés sur les circuits probabilistes permettent de traiter différentes requêtes et transformations sur ces circuits en temps polynomial. De plus, on s'attend généralement à ce que la différence d'efficacité spatiale entre deux classes de circuits probabilistes soit la même que celle entre les classes de circuits Booléens correspondantes (quand elles sont définies). Dans le chapitre 6, nous présentons les travaux réalisés avec Stefan Mengel sur la comparaison en termes d'efficacité spatiale de classes de circuits représentant des fonctions pseudo-Booléennes, c'est-à-dire, des fonctions sur des variables Booléennes qui calculent des valeurs réelles [dCM21b]. Les circuits en question sont des circuits arithmetiques, ou AC (pour « arithmetic circuits »), utilisant les opérations produits et sommes (là où les circuits Booléens utilisent les opérations $\wedge$ et $\vee$). L'appellation « arithmetic circuits » remonte aux articles de Darwiche [Dar02b, Dar03] où ces circuits sont introduits comme représentations de distributions marginales de réseaux Bayésiens. Mais les AC étant des modèles naturels pour représenter des polynômes en général, la recherche sur ces objets précède en réalité les articles de Darwiche. En particulier, Valiant étudiait des objets similaires vingt ans avant sous le nom de programmes-$(+, \times)$ [Val80]. Nous adoptons la définition des AC donnée par Darwiche. Notamment, les entrées de nos AC sont des variables Booléennes (vues comme des variables sur $\{0, 1\}$), leur négations, ou des constantes réelles. Nous apportons une attention particulère aux AC représentant des fonctions non-negatives, auquel cas ils sont appelés des AC *positifs*. L'intérêt pour ces AC positifs vient de leur utilisation comme circuits probabilistes. Nous avons étudié seize classes d'AC positifs construites en imposant différentes combinaisons des cinq propriétés suivantes: le déterminisme, l'homogénéité (« smoothness »), la décomposabilité, une variante de la décomposabilité appelée décomposabilité faible, et la monotonicité. La monotonicité, au contraire des quatres premières propriétés, n'a pas de formulation équivalente pour les circuits Booléens. Un AC est dit *monotone* quand les constantes réelles sur ses feuilles sont non-négatives. D'une certaine manière, un circuit utilisant à la fois des nœuds $+$ et des constantes négatives est capable de réaliser des soustractions, donc un AC monotone est essentiellement un AC positif où la

soustraction n'est pas autorisée. Il est connu depuis les travaux de Valiant que les plus petits AC monotones représentant une fonction peuvent être beaucoup plus grands que certains AC positifs représentant la même fonction [Val80]. Dans le chapitre 6, nous donnons comparons les huit classes d'AC monotones en termes d'efficacité spatiale : pour chacune des vingt-huit paires $(\mathcal{C}_1, \mathcal{C}_2)$ de classes prises parmi les huit classes en question, nous avons déterminé si tout circuit $C_1 \in \mathcal{C}_1$ a un équivalent dans $\mathcal{C}_2$ dont la taille varie est au plus polynomiale en $|C_1|$, et inversement. Nos résultats s'appuient sur une réduction nous permettant de travailler avec des circuits en NNF plutôt qu'avec des AC monotones. Nous avons ensuite démarré une étude similaire pour les huit autres classes d'AC positifs, sans pouvoir l'achever. Afin de motiver la rechercher sur les AC positifs, nous décrivons dans la dernière section des techniques permettant de trouver des bornes inférieures sur la taille d'AC positifs, et nous mettons en application ces techniques dans le cas d'AC respectant la décomposabilité structurée.

Introduction

Circuits in computer science are a model of computation able to represent many kinds of functions, and into which several other models of computation can be translated: binary decision diagrams (BDDs), decision trees, formulas in conjunctive normal form (CNF), polynomials, to only quote a few. A circuit breaks down the computation of a function into a sequence of simple steps that correspond to gates. Thus circuits are essentially *programs* for computing various kinds functions and several types of circuits and computation diagrams are even referred to as programs, like branching programs (another name for BDDs) and $(+, \times)$-programs (a name used by Valiant for arithmetic circuits [Val80]). Studying the size of the smallest circuit from a given family representing a function amounts to studying the hardness of computing this function with respect to a given model of computation. This is an important aspect of *circuit complexity*, a domain of computer science that aims to establish unconditional lower bounds on the computational complexity of selected functions, like the *permanent* or the function detecting k-cliques in graphs.

In addition to computing the value of a function for any assignment to its variables, some circuits allow to easily determine useful information on functions, like the number of satisfying assignments in the case of Boolean functions. Such circuits, sometime called *tractable circuits* (see for instance [VCL+21, Dar22]), are at the core of *knowledge compilation*. Knowledge compilation is a domain of computer science that deals with situations where a knowledge base known in advance is preprocessed, or *compiled*, during a costly *offline phase* called *compilation*, before being queried and manipulated during the *online phase*. Ideally, the cost of compilation is amortized by the time gained working on the processed knowledge base during the online phase, in particular when many queries are answered. In this context, classes of so-called tractable circuits are *compilation languages*, that is, when the knowledge base is a function, compiling the knowledge base consists in finding a tractable circuit computing the corresponding function.

Among the compilation languages that have been introduced during the last two decades for Boolean functions, the class of circuits in decomposable negation normal form (DNNF) introduced in 2001 [Dar01a] is central. Circuits in DNNF are among the most general kind of circuits for which answering the *clausal entailment* query is tractable, a query historically important in knowledge compilation [SK96, Sin02] to the point that its tractability has sometime be regarded as a requirement for compilation languages [DM02]. Moreover circuits in DNNF encompass other classes of circuits useful in model counting [Dar11, LM17], in Max-SAT solving [PD07], in model enumeration [ABJM17], in QBF solving [CM19], etc.

This thesis revolves around the DNNF language, i.e., the class of circuits in DNNF. The best way to describe the thesis is as a "showcase" of the analysis of circuits in DNNF and of their applications (even though most of our results are "negative"). After a preliminary section, the thesis splits into three parts. In the first part we study the hardness of compilation in DNNF by proving new lower bounds on the size of the smallest circuits in DNNF representing particular functions and sometimes approximations of these functions. The lower bounds shown are negative results, but negative results are not always the end of the story. This is the object of the second part of this thesis where we show several applications

of results proved in the first part both inside and outside of the area of knowledge compilation. Finally, the third part of the thesis deals with new queries and new fields for knowledge compilation. In this part, we study new enumeration queries for the enumeration of specific implicants from particular circuits in DNNF. We also go beyond knowledge compilation for Boolean functions and initiate a systematic comparison of compilation languages for non-Boolean functions, more precisely for pseudo-boolean functions. In what follows, we give a more detailed description of the DNNF language and of some of its sublanguages. Then we give a more detailed overview of the subjects that we study in this thesis and present our contributions.

DNNF and its sublanguages. To understand the rest of this introduction, a brief overview of the DNNF language and of some of its sublanguage seems necessary. A circuit in negation normal form (NNF) is a directed acyclic graph whose internal nodes are labelled with $\vee$- or $\wedge$-operations and whose leaves are labelled either by a Boolean constant 0 (*false*) or 1 (*true*), or by literals in Boolean variables. A circuit is in *decomposable* NNF (DNNF) when it is in NNF and when its $\wedge$-nodes respect a property called *decomposability* which enforces that the sets of variables of the children (or inputs) of a $\wedge$-node are pairwise disjoint [Dar01a]. The DNNF language is the class of all circuits in DNNF, it encompasses some well-known kinds of circuits and formulas like DNF formulas, and is *complete* in the sense that every Boolean function over finitely many variables has a representation as a circuit in DNNF. Decomposability is a structural restriction on circuits in NNF thanks to which answering several queries on the circuit becomes tractable. In particular deciding the satisfiability of a circuit in DNNF is feasible in linear time in the size of the circuit (the number of edges of its graph), and so is clausal entailment, that is, given a circuit D in DNNF and a clause γ, determining whether every satisfying assignment, or model, of D satisfies γ is feasible in linear time in the size of D. In addition to decomposability, other structural restrictions have been invented for circuits. For every combination of restrictions, the circuits respecting these restrictions form a new language. In particular, *structured decomposability* and *determinisim* are central restrictions that appear several times in this thesis. Structured decomposability is a refinement of decomposability that puts a constraint on the way the sets of variables of the children of a $\wedge$-node partition the variables of that $\wedge$-node [PD08]. We omit the formal definition for now and only mention that circuits in NNF that respect structured decomposability are said to be in structured decomposable negation normal form (str-DNNF), and that the way variables are arranged in such circuits is described by some kind of combinatorial structure called a *vtree* (whose definition we also omit for now). The class of circuits in str-DNNF is the str-DNNF language. Structured decomposability renders tractable some operations that are intractable over general circuits in DNNF, in particular operations that require manipulating two circuits respecting the same vtree. Another important restriction is *determinism*, which states that the children of every $\vee$-node represent Boolean functions whose sets of satisfying assignments are pairwise disjoint [Dar01a]. Circuits in DNNF respecting determinism are said to be in *deterministic decomposable negation normal form* (d-DNNF) and the class of all these circuits is the d-DNNF language. The combination of decomposability with determinism makes it possible to compute the number of satisfying assignments, or *model count*, of the circuit, which is useful in many settings. One convenient way to ensure that $\vee$-nodes respect determinism is to make them compute *Shannon decompositions*, that is, every $\vee$-node is of the form $(\overline{x} \wedge \cdot) \vee (x \wedge \cdot)$ for some variable x. Circuits in d-DNNF whose $\vee$-nodes are of this form are in *decision DNNF* (dec-DNNF) and the class of all these circuits is the dec-DNNF language. To finish our small presentation of compilation languages, we mention that several classes of decision diagrams, including ordered binary decision diagrams (OBDDs) [Bry86, Weg00], free binary decision diagrams (FBDDs) [SW95] and sentential decision diagrams (SDDs) [Dar11] can be transformated in d-DNNF in polynomial time, so that theses classes (in particular OBDD, FBDD and SDD) are

somtimes seen as sublanguages of d-DNNF.

The DNNF size of selected functions. For a compilation language L, the L *size* of a function is the size of the smallest representation of the function in L. Finding functions whose L size is small is essential to champion L as a viable compilation language. For instance when Darwiche introduced the language DNNF, he promoted the language showing that every CNF formula with bounded primal treewidth is compilable to DNNF in polynomial time [Dar01a]. On the other hand, knowing that many functions have a large L size supports the belief that no compilation language admits small representations for every functions, and is useful to compare L to other languages in terms of space efficiency, or *succinctness*. For instance in [Weg00, Chapters 4 and 6], Wegener give several examples of functions whose OBDD size is exponentially larger than the FBDD size. More generally, comparing circuits in terms of succinctness is one of three methods to compare compilation languages in the *knowledge compilation map*, in framework introduced by Darwiche and Marquis where compilation languages are systematically compared in terms of succinctness and in terms of queries and transformations that are tractable on their circuits [DM02]. The first negative results for the compilation to DNNF predate, in a sense, the creation of the language since Selman and Kautz have shown in the nineties that there is no compilation language that renders clausal entailment tractable (as does DNNF) and in which all formulas have a representation of polynomial size (in the number of variables), unless NP $\subseteq$ P/poly [SK91, SK96]. Since DNNF has been introduced, unconditional separations between the class of CNF formulas and DNNF have been found [BCMS14], thus confirming the conditional negative results of Selman and Kautz in the case of DNNF. To explain what makes a function hard to compile in DNNF, a line of research has developed with the objective of determining parameters of *general* Boolean functions that, when too high, guarantee that the DNNF size is exponential in the number of variables [JS12, RP13, STV14, BS17a]. In another direction, the compilation of *specific* classes of functions in DNNF has been studied with both positive results, for instance for variable-convex CNF formulas [BS17b], and negative results taking the form of exponential lower bounds on the DNNF size, like for monotone CNF formulas [ACMS20] or systems of linear equations [Men16]. The results that we present in Chapter 1 fall within the negative case. We show exponential lower bounds on the DNNF size of particular pseudo-Boolean constraints [dC20] and exponential lower bounds on the DNNF size of particular CNF formulas called Tseitin formulas [dCM21a]. After a section on the state of the art of compilation in DNNF, two sections are dedicated to the proof of each lower bound respectively. The first bound shows that compiling in DNNF pseudo-Boolean constraints of the form $w_1x_1 + \cdots + w_nx_n \geq \theta$, where $w_1, \ldots, w_n, \theta$ are real numbers and $x_1, \ldots, x_n$ are Boolean variables, may create circuits that are too large to work on. Yet compilation to DNNF of such constraints is useful, for instance for generating CNF encoding of the constraint that have nice properties desired in constraint programming [AGMS16, KS19]. The second lower bound has been shown with Stefan Mengel. It deals with Tseitin formulas, that are CNF formulas representing systems of parity constraints structured by a graph. We show that the DNNF size of these formulas, when the underlying graph has maximum degree bounded by a constant, is exponential in the *treewidth* of the graph. Applications of this bound are given in Chapters 2, 3 and 4.

The d-DNNF size of approximations. If a function has a DNNF size too large to be practical, then one may want to compile an approximation of that function in the hope that it is computed by a more compact circuit in DNNF. With Stefan Mengel, we have studied *approximate knowledge compilation* in the d-DNNF language [dCM20]. The major incentive for compiling in d-DNNF is that model counting is tractable on circuits in d-DNNF [DM02]. Unfortunately, compiling to d-DNNF is generally much

more expensive that compiling to DNNF since the d-DNNF size of a function is generally exponentially larger than its DNNF size [BCMS16]. There exist several approaches for approximate knowledge compilation in different languages [SK96, Dar01a, BSW02, PD07] that, with the exception of [BSW02], lack a precise measure on the quality of the approximation. We have considered two notions of approximation offering different guarantees on the approximation error. We call them *weak approximations* and *strong approximations*. Weak approximations have already been studied by Bollig, Sauerhoff and Wegener for the compilation in the language OBDD [BSW02], and it is known that there are functions whose weak approximations all have exponential OBDD size. Chapter 2 of this thesis presents our contributions to approximate knowledge compilation. After a first section on the state of the art of approximate knowledge compilation, the second section shows how to extend the result of Bollig et al. from OBDD to d-DNNF, which is not insignificant since circuits in d-DNNF are in general exponentially smaller than the smallest equivalent OBDDs. Then we explain that weak approximation is not ideal for approximate knowledge compilation to d-DNNF, in particular to perform approximate model counting, as it allows the approximation by the function that is uniformly zero in many situations. In the third section we define the notion of strong approximation to circumvent this problem. Strong approximation has been designed with approximate model counting in mind. Indeed, a strong approximation of a function has a number of models that is within a constant factor of the number of models of the initial function. We show that there exist classes of functions that are weakly approximated by the function that is uniformly zero, but whose strong approximations all have exponential d-DNNF size. We end Chapter 2 by using the lower bounds obtained in Chapter 1 on the DNNF size of particular Tseitin formulas to prove that the latter result holds even for some classes of CNF formulas.

Analyzing bottom-up compilation to str-DNNF. Even if a function can be computed by a small circuit in a compilation language, it might still take exponential space and time to compile in that language depending on the compilation algorithm. In Chapter 3, we use the results of Chapter 1 on the DNNF size of Tseitin formulas to construct examples of such functions for *bottom-up compilation* algorithms. Bottom-up compilation is one of the two main paradigms along with *top-down compilation*. In top-down compilation, a circuit computing the initial function is essentially constructed as the trace of an algorithm exploring the space of assignments to the variables of the function [HD05]. In bottom-up compilation, one often assumes that the initial function is a system of constraints (for instance a CNF formula, where the constraints are clauses). The paradigm then consists of two steps: first, compiling every constraint to the language, and second, combining the resulting circuits until obtaining a circuit that computes the solutions of the initial system. The second step requires that the conjunction of two circuits from the language is tractable, which is generally possible only if the two circuits are in str-DNNF and are similarly structured (that is, respect the same vtree) [PD08]. Thus bottom-up compilation is a method for compiling in the language str-DNNF, which includes OBDD and SDD [Dar11]. As described before, bottom-up compilation creates intermediate circuits, and these can be much larger than both the input system of contraints and the output circuit, as observed in practice [NW07] and proved for particular functions [Kra08, Seg08, FX13, IKRS20] when compiling in OBDD. Chapter 3 describes the contributions made with Stefan Mengel to the analysis of bottom-up compilation to str-DNNF [dCM22b]. The chapter starts with a section defining a formal model of bottom-up compilation, followed by a section on the state of the art of bottom-up compilation, both in practice and in theory. In the remaining sections of the chapter, it is shown how to use the lower bound obtained in Chapter 1 on the DNNF size of Tseitin formulas to construct CNF formulas whose size is polynomial in the number of variables, whose str-DNNF size is constant, but for which bottom-up compilation to str-DNNF creates intermediate circuits whose size is exponential in the treewidth of the primal graphs of the initial formulas. The lower bounds

that are obtained on intermediate circuits take into account arbitrary modifications of intermediate circuits during compilation as long as these modifications preserve the function computed by the circuit (for instance circuit minimization). Thus the framework for our bounds captures the behaviour of practical bottom-up compilers, see for instance [Rud93, CD13]. To the best of our knowledge, the bounds shown by Itsykson, Knop, Romashchenko and Sokolov for OBDDs were the only ones proved in comparable settings [IKRS20]. We start by proving our result for bottom-up compilation of *unsatisfiable* Tseitin formulas and then lift it up to satisfiable CNF formulas built upon those. So we obtain as a first step an exponential lower bound on the size of every refutation of unsatisfiable Tseitin formulas in str-DNNF-based proof systems, including OBDD-based proof systems. Itsykson et al. also study Tseitin formulas to prove bounds in [IKRS20]. Our bounds generalize theirs in two ways: on the one hand we study circuits in str-DNNF, which are generally exponentially smaller than OBDD, on the other hand their bounds are restricted to Tseitin formulas over a particular class of graphs while ours are parameterized and, as such, hold for Tseitin formulas for every graphs.

Using knowledge compilation in proof complexity. Chapter 4 gives a last application of the lower bound on the DNNF size of Tseitin formulas proved in Chapter 1. This application makes the connection with the historic purpose of Tseitin formulas. Unsatisfiable Tseitin formulas have been introduced by Grigori S. Tseitin in the sixities as hard formulas to refute in the resolution proof system [Tse68, Tse83]. The resolution proof system is based on a single inference rule called the *resolution rule*, which states that a formula that satisfies two clauses $C \vee x$ and $C' \vee \overline{x}$ also satisfies $C \vee C'$. Every unsatisfiable CNF formula has a refutation in the resolution proof system, that is, a sequence of clauses either taken from the formula or obtained by resolution rule over previous clauses, that ends with the empty clause. The intuition of Tseitin was that exponentially many clauses would be required in resolution refutations of Tseitin formulas. This intuition was proved to be correct since Urquhart later showed that for unsatisfiable Tseitin formulas whose underlying graphs are expander graphs, the shortest resolution refutations have exponential length [Urq87]. Characterizing unsatisfiable Tseitin formulas whose resolution refutation length is not polynomial in the number of variables remains an open problem, despite several advances on the subject [AR11, IO13, GTT20]. In particular, for the more restrictive proof system of *regular* resolution, the upper bounds shown in [AR11] and the lower bounds shown in [IRSS21] hint on the fact that, when the graph of the Tseitin formula has maximum degree bounded by a constant, the length of the shortest refutation is exponential in the treewidth $tw(G)$ of the graph. The $2^{\Omega(tw(G)/\log(n)))}$ lower bound in [IRSS21] almost matches the $2^{O(tw(G))}poly(n)$ upper bound of [AR11], where n is the number of variables of the formula. Our main contribution, shown with Stefan Mengel, is to use the lower bounds proved in Chapter 1 on the DNNF size of Tseitin formulas to improve the lower bounds and match the upper bound, thus showing that the length of regular resolution refutations of unsatisfiable Tseitin formulas whose graphs have bounded degree is polynomial in the number of variables n if and only if the treewidth of the graph is in $O(\log(n))$ [dCM21a]. This is the subject of Chapter 4. The chapter starts with a section where a formal description of regular resolution is given, followed by a section where we recall existing bounds on the length of refutations of unsatisfiable Tseitin formulas in this proof system. The remaining sections develop a reduction that brings us from the state where we study the length of regular resolution refutations of unsatisfiable Tseitin formulas, to the state where we study the DNNF size of satisfiable Tseitin formulas over the same graphs, and can thus use the lower bounds shown in Chapter 1.

Enumeration queries on circuits in dec-DNNF. The compilation process is only one part of knowledge compilation, the second one being online reasoning on the compiled circuit. While this thesis is mostly centered around the compilation process, the author has also contributed with Pierre Marquis to the research on enumeration queries over circuits in compilation languages [dCM22a]. Enumeration queries differ from decision queries, like clausal entailment, and from function queries, like model counting, in that the answer is not a single value but a list of many solutions. As an example, listing all satisfying assignments of a circuit is an enumeration query that has been studied for circuits in compilation languages [DM02, ABJM17, CS21]. Since the output of enumeration queries can be exponentially larger than the input, they have to be studied under the special framework of enumeration complexity [Str19]. This framework is presented in the first section of Chapter 5. For our contributions, we limited ourselves to studying enumeration queries on circuits in dec-DNNF with respect to the complexity classes OutputP and IncP. The former encompasses enumeration problems for which all solutions can be obtained in time polynomial in the size of the input plus the size of the output (so all solutions). The latter contains problems for which there exists an algorithm to list a given number k of solutions in time polynomial in the size of the input and k. In the second and third section of Chapter 5, we study enumeration queries that consists in enumerating prime implicants and prime implicates of circuits in dec-DNNF. In particular, we show that enumerating all prime implicants (resp. implicates) of such circuits is in OutputP and even in IncP. In the fourth section, we turn to the enumeration of specific prime implicants: sufficient reasons and subset-minimal abductive explanations, two notions used in AI, and especially in explainable AI, in recent and in not so recent years [SL90, EG95, SCD18b, DH20, INM19]. While positive results can be obtained for enumerating prime implicants from circuits in dec-DNNF we show that it is unlikely that enumerating these particular implicants from the same circuits is in OutputP.

Knowledge compilation beyond the Boolean case. Knowledge compilation is not confined to Boolean functions. The languages studied in this thesis for Boolean functions are defined by the structural restrictions that are put on the circuits. But structural restrictions for Boolean circuits can be defined analogously for circuits representing other types of function, and thus classes of circuits for representing these functions can be defined similarly to DNNF and its sublanguages. In particular, decomposability has been studied for arithmetic circuits under the name *multilinearity* [NW97] and, decomposability, determinism, etc. have been studied extensively for probabilistic circuits that, roughly put, are circuits computing probability distributions [VCL+21]. Similarly to the Boolean case, different combinations of structural properties on probabilistic circuits render different queries and transformations tractable. Moreover the succinctness relationship between two classes of probabilistic circuits is generally expected to be the same as the one between the corresponding Boolean classes (when they are defined). In Chapter 6, we present the study conducted with Stefan Mengel on the succinctness relationships between classes of arithmetic circuits (ACs) using product and sum operations to compute pseudo-Boolean functions, that is, functions over Boolean variables and whose outputs are real numbers [dCM21b]. Since arithmetic circuits are natural model of computation for polynomials, research on these data structures predates the articles of Darwiche where they are introduced as representations for marginal distributions of Bayesian network [Dar02b, Dar03]. In particular we have to mention the work done by Valiant on $(+, \times)$-programs [Val80]. We adopt the definition of ACs given by Darwiche. The inputs of the ACs in this chapter are Boolean variables (seen as variables over $\{0, 1\}$), negated Boolean variables, and real constants. ACs can be used as particular probabilistic circuits when they compute non-negative functions, in which case they are called *positive* ACs. We consider sixteen classes of positive ACs by imposing combinations of five structural properties on the circuits: determinism, smoothness, decomposability, weak decomposability (a variant of decomposability), and monotonicity. Monotonicity has no direct equivalence for Boolean circuits. It states that all constants labelling leaves of the circuit are

non-negative. In a sense, a circuit that uses $+$-operation and negative constants can do subtractions, so a *monotone* AC is essentially a positive AC where subtraction is forbidden. It is known that monotone ACs can be much larger than equivalent positive ACs [Val80]. Several sections in Chapter 6 are dedicated to finding *unconditional* succinctness relationships between eight classes of monotone ACs. We obtain all relationships using a reduction from monotone ACs to circuits in negation normal form. Next we initiate a similar study for positive ACs but can only show a couple of results, including that deterministic positive ACs and deterministic monotone ACs are essentially the same objects. In an effort to boost the research on positive ACs, in a last section we describe techniques to bound the size of positive ACs from below and show their usage in the case of structured decomposable positive ACs.

Preliminaries

1 Notions on Graphs

Unless stated otherwise, the graphs considered in this thesis are undirected, have no parallel edges (so no multigraphs) and no self loops. For a graph G whose set of vertices and edges are not specified, we denote by $V(G)$ its set of vertices and by $E(G)$ its set of edges. The edge between two vertices u and v is denoted by uv or vu. For $v \in V(G)$, $E_G(v)$ denotes the set of edges incident to v in G. The set of vertices that are neighbors of v in G is denoted as $N_G(v)$, more formally $N_G(v) := \{u \mid uv \in E(G)\}$. Note that, since G has no self loop, v is not in $N_G(v)$. We drop the subscript from the notations $E_G(v)$ and $N_G(v)$ when the graph is clear from the context. We write $\deg(v) = |N(v)|$ and we denote by $\Delta(G) = \max_{v \in V(G)} \deg(v)$. For a subset V' of $V(G)$ we denote by $G[V']$ the subgraph of G induced by V', that is, the graph whose set of vertices is V' and whose set of edges is $E(G) \cap \{uv \mid u \in V', v \in V'\}$.

A graph is *connected* if for every $u, v \in V(G)$, there is a path from u to v in G, that is, there are edges $u_1u_2, u_2u_3, \ldots, u_{k-1}u_k$ in $E(G)$ with $u_1 = u$ and $u_k = v$. A separator S in a connected graph G is a vertex set such that $S \subseteq V(G)$ and $G[V(G) \setminus S]$ is and not connected. A graph G is called k-connected if and only if it has at least $k+1$ vertices and, for every $S \subseteq V(G)$, $|S| \leq k-1$, the graph $G[V(G) \setminus S]$ is connected.

An *edge separator* of G is a set of edges whose removal strictly increases the number of connected components of G. Given a partition (A, B) of $V(G)$ (that is, $A \cup B = V(G)$ and $A \cap B = \emptyset$), we denote by $E(A, B)$ the set of edges that have one endpoint in A and the other in B. Note that if G is connected then $E(A, B)$ is an edge separator of G.

Treewidth and Pathwidth. Treewidth is a well-known graph measure that has several equivalent definitions. We use the definition based on tree decomposition. A *tree decomposition* of G is a rooted tree T such that every node $t \in T$ is associated with a set $B_t \subseteq V(G)$ called a *bag* and such that

- $V(G) = \bigcup_{t \in T} B_t$
- for every $uv \in E(G)$, there is $t \in T$ such that $\{u, v\} \subseteq B_t$
- for every $u \in V(G)$ the subgraph of T restricted the nodes whose bags contain u is connected (so its a tree)

When T is a path $(t_1, \ldots, t_N)$, it is called a *path decomposition* of G. For path decompositions, the third rule may be rephrased as follow:

- for every $1 \leq i \leq j \leq k \leq N$, $B_{t_j} \subseteq B_{t_i} \cap B_{t_k}$

Definition 1. *Let T be a tree decomposition of G, the* width *of the decomposition is $\max_{t \in T} |B_t| - 1$. The* treewidth *of G, denoted by $tw(G)$, is the smallest width of a tree decomposition of G. More formally,*

$$tw(G) = \min_{T} \max_{t \in T} |B_t| - 1$$

where $\min_T$ is over all tree decompositions of G.

Definition 2. *The* pathwidth *of G, denoted by $pw(G)$, is the smallest width of a path decomposition of G. More formally,*

$$pw(G) = \min_{P} \max_{t \in P} |B_t| - 1$$

where $\min_P$ is over all path decompositions of G.

Branchwidth. A binary tree whose leaves are in bijection with the edges of G is called a *branch decomposition* of G. Branch decompositions are often defined using unrooted trees, however our definition is equivalent and is more convenient in our setting. The removal of any edge e of a branch decomposition T splits T into two connected components. The sets of edges of G that label the leaves of each component form a bipartition of $E(G)$. The number of vertices of G that are incident to edges in both parts of this partition is called *the order of* e, denoted by $order(e, T)$. A branch decomposition is called *linear* when every internal node of the tree has at least one child that is a leaf. Linear branch decompositions represent orderings, or permutations, of $E(G)$. Writing $E(G) = \{e_1, \ldots, e_m\}$, an ordering of $E(G)$ is a sequence $(e_{\pi(1)}, \ldots, e_{\pi(m)})$ with π a permutation of $\{1, \ldots, m\}$. A linear branch decomposition T represents the permutation defined as $\pi(i) < \pi(j)$ if and only if e_i is closer to T's root than e_j or, if e_i and e_j are the children of the same node and e_i is the right child.

Example 1. Consider the graph G and the branch decomposition T below: The branch decomposition

is linear and represents the ordering (e_1, e_4, e_3, e_2). Let e be the dashed edge of T. Removing e from T yields one tree over $\{e_1, e_4\}$ and another over $\{e_2, e_3\}$. Exactly two vertices of G are incident to edges in both $\{e_1, e_4\}$ and $\{e_2, e_3\}$, namely b and d, so $order(e, T) = 2$. ◀

Definition 3. *The* branchwidth *of graph G, denoted by $bw(G)$, is defined as*

$$bw(G) = \min_{T} \max_{e \in E(T)} order(e, T)$$

where $\min_T$ is over all branch decompositions of G.

Definition 4. *The* linear branchwidth *of graph G, denoted by $bw_\ell(G)$, is defined as*

$$bw_\ell(G) = \min_{T_\ell} \max_{e \in E(T_\ell)} order(e, T_\ell)$$

where $\min_{T_\ell}$ is over all linear branch decompositions of G.

While it is sometimes convenient to work with branchwidth or linear branchwidth, we nearly always give our results using the more well-known treewidth or pathwidth. This is justified by the two following lemmas connecting the four measures, and whose proofs can be found in [HW17, Lemma 12], and in the master thesis [Nor17, Theorem 6.1], respectively. For the convenience of the reader, we reproduce the proof of the second lemma .

Lemma 1. *If $bw(G) \geq 2$, then $bw(G) - 1 \leq tw(G) \leq \frac{3}{2} bw(G)$*

Lemma 2. *If $bw_\ell(G) \geq 2$, then $bw_\ell(G) - 1 \leq pw(G) \leq bw_\ell(G) + 1$.*

Proof. Removing isolated vertices in the graph does not impact the pathwidth or the linear branchwidth, so without loss of generality we assume that G has no such vertices.

First we prove that $pw(G) - 1 \leq bw_\ell(G)$. Assume G has linear branchwidth k and let $E(G) = \{e_1, \ldots, e_m\}$. Let π be a permutation of $\{1, \ldots, m\}$. For i ranging from 1 to m define B_i to be the vertex set containing both vertices of $e_{\pi(i)}$ plus all vertices that are incident to some edge in $\{e_{\pi(1)}, \ldots, e_{\pi(i)}\}$ and to some edge in $\{e_{\pi(i+1)}, \ldots, e_{\pi(m)}\}$. Observe that if there is $i' < i < i''$ such that u is incident to

$e_{\pi(i')}$ and to $e_{\pi(i'')}$ then u is in B_i. From the definition of B_i we deduce that $|B_i| \leq order(i,\pi)+2$ (the $+2$ comes from the vertices of $e_{\pi(i)}$) so if $B_1,\ldots,B_m$ is a path decomposition then its width is at most $\max_i order(i,\pi)+1$. We claim that $B_1,\ldots,B_m$ is a path decomposition. From the first two conditions and since G has no isolated vertices, we obtain that the union of $B_1,\ldots,B_m$ equals $V(G)$ and that every edge e_j of G is at least contained in $B_{\pi^{-1}(j)}$. Now for the third condition, take $1 \leq i < j < k \leq m$ and consider $u \in B_i \cap B_k$. By definition there are $i' \leq i \leq i''$ and $k' \leq k \leq k''$ such that u is a vertex of $e_{\pi(i')}$, $e_{\pi(i'')}$, $e_{\pi(k')}$ and $e_{\pi(k'')}$. Since but then $i' < j < k''$ holds and therefore u is in B_j.

Now we prove the other direction, so $bw_\ell(G) \leq pw(G)+1$. Consider a path decomposition $\mathcal{P} = (B_1,\ldots,B_N)$. We construct π as follows. Let $c=1$ and start with all edges of G unmarked. We visit the bags in order. When the vertices of an unmarked edge e_i are both in the current bag, then set $\pi(c)$ to i, mark e_i and increment c. If there are several such edges, we apply an arbitrary tie-breaking and treat all of them in any order. At the end of the procedure all the edges are marked (since each edge is contained in some bag), so π is permutation of $\{1,\ldots,m\}$. Now for each $1 \leq i \leq m$, let i^* be the smallest index of a bag containing e_i. By construction if $\pi^{-1}(i) \leq \pi^{-1}(j)$ then $i^* \leq j^*$. Now take $1 \leq k < m$, and let u be a vertex incident to an edge e_i in $\{e_{\pi(1)},\ldots,e_{\pi(\pi^{-1}(k))}\}$ and to an edge e_j in $\{e_{\pi(\pi^{-1}(k)+1)},\ldots,e_{\pi(m)}\}$. Since $\pi^{-1}(i) \leq \pi^{-1}(k) < \pi^{-1}(j)$ we have $i^* \leq k^* \leq j^*$. Moreover, since $e_i \subseteq B_{i^*}$ and $e_j \subseteq B_{j^*}$ we have $u \in B_{i^*} \cap B_{j^*} \subseteq B_{k^*}$. This shows that $order(\pi^{-1}(k),\pi) \leq |B_{k^*}|$. So if $\mathcal{P}$ is a path decomposition of width $pw(G)$ the corresponding ordering π verifies $\max_k order(k,\pi) \leq pw(G)+1$. □

2 Notions from Propositional Logic

A Boolean variable x is a variable that takes value 1 (*true*) or 0 (*false*). The negated variable $\overline{x}$ takes value 1 (resp. 0) when x takes value 0 (resp. 1). A *literal* in the variable x is either x or $\overline{x}$. We often use the letter ℓ for literals and ℓ_x to denote any literal in the variable x. We use the usual symbols $\wedge$, $\vee$ and $\neg$ representing, respectively, the conjunction, disjunction and negation operators. We sometimes use the $\overline{*}$ notation to write negation, for instance we may write $\overline{x \vee y} = \overline{x} \wedge \overline{y}$ or $\overline{\overline{x}} = x$.

Truth assignments. Given a set X of Boolean variable, a *truth assignment*, or just an *assignment*, to X is a mapping a from X to $\{0,1\}$. An assignment to a strict subset Y of X is called a *partial assignment* to X. When X is not specified, we denote by $var(a)$ the set of variables that are given a value by a (so $var(a) = a^{-1}(0) \cup a^{-1}(1)$). Equivalently, an assignment a to X is represented as a set of literals $\{\ell_x \mid x \in X\}$ or as a term $\bigwedge_{x\in X} \ell_x$ such that $\ell_x = \overline{x}$ if $a(x)=0$ and $\ell_x = 1$ if $a(x)=1$. The set representation allows us to define the *empty assignment* $a_\emptyset$ as the unique assignment to $\emptyset$ (whose set representation is thus $\{\}$). An assignment a is said to *extend* the assignment a' if, using the set representations of a and a', we have $a' \subseteq a$. Given two assignments a and a' that do not contain contradicting literals, that is, if $\ell_x \in a$ then $\overline{\ell_x} \notin a'$ and vice versa, we denote by $a \cup a'$ the assignment to $var(a) \cup var(a')$ whose set representation is the union of the set representations of a and a'. We denote by $\{0,1\}^X$ the set of all assignments to X. For two assignments $a, a' \in \{0,1\}^X$, we say that a' *dominates* a, written $a \leq a'$, if $a'(x)=1$ for every $x \in X$ such that $a(x)=1$. We write $a < a'$ when $a \leq a'$ and $a \neq a'$.

Boolean functions. A *Boolean function* f over a set of Boolean variables X is a mapping from $\{0,1\}^X$ to $\{0,1\}$. Given χ a constraint over X (for instance $x+y+z \geq 2$ or $x \in S$ for some set S), we denote by $\mathbb{1}_\chi : \{0,1\}^X \to \{0,1\}$ the indicator function of that constraint defined by $\mathbb{1}_\chi(a)=1$ if and only if a satisfies χ. When X is not specified we write $var(f)$ to denote the set of variables of X. An assignment a' to $Y \subseteq var(f)$ *satisfies* f, or is *accepted* by f, if $f(a)=1$ for every assignment a to $var(f)$ that extends a'. The assignment a' *falsifies* f if $f(a)=0$ for every assignment a to $var(f)$ that extends of

a'. Note that a' may neither satisfy nor falsify f, but note also that an assignment a to $var(f)$ either satisfies or falsifies f. An assignment to $var(f)$ that satisfies f is called a *model* of f. We often write $sat(f) = f^{-1}(1)$ for the set of models of f. For a partial assignment a, we denote by $f|a$ the Boolean function over $var(f) \setminus var(a)$ whose models are exactly the models of f restricted to $var(f) \setminus var(a)$. Given a set $X \subseteq var(f)$ we denote by $\exists X.f$ the function $\bigvee_{a \in \{0,1\}^X} f|a$. Going from f to $\exists X.f$ is called *forgetting X from f*. A function f is called *satisfiable* when $sat(f) \neq \emptyset$, otherwise it is called *unsatisfiable*. f is called *valid* when $sat(f) = \{0,1\}^{var(f)}$, that is, when it accepts every assignment to its variables. Given another function g, we write $f \models g$ when every assignment that satisfies f also satisfies g. When $var(f) = var(g)$ we may sometimes write $f \leq g$ for $f \models g$. We write $f < g$ when $f \leq g$ and $f \neq g$.

Boolean formulas. A *Boolean formula* over a set of Boolean variables X is any expression E generated by the grammar $E ::= 0 \mid 1 \mid \ell \mid \neg E_1 \mid E_1 \otimes \cdots \otimes E_k$ where ℓ is any literal in a variable in X and $\otimes$ is any associative binary Boolean operator. Note that not every variable of X must appear in a formula over X. The set of literals appearing in ϕ is denoted by $lit(\phi)$. The value of a formula ϕ on an assignment a is the Boolean value (0 or 1) obtained by replacing each occurrence of x (resp. $\overline{x}$) in ϕ by $a(x)$ (resp. $1 - a(x)$) and by following the computations given by the expression. Thus each formula over X is associated with (or computes) the unique Boolean function over X whose models are the assignments that make the formula evaluate to 1. In this thesis, formulas are often directly seen as Boolean functions. Given two Boolean formulas ϕ and ψ over X, we say that ϕ is equivalent to ψ, denoted by $\phi \equiv \psi$, when ϕ and ψ compute the same Boolean function, for instance $x \wedge (y \vee \overline{y}) \equiv x$. We try, as much as possible, to write $\phi = \psi$ only when ϕ and ψ are the same expression. A *term* is a formula whose expression is a conjunction of literals and a *clause* is a disjunction of literals. A *CNF formula* is a conjunction of clauses and a *DNF formula* is a disjunction of terms. We denote by $clause(\phi)$ (resp. $term(\phi)$) the set of clauses (resp. of terms) of the CNF (resp. DNF) formula ϕ. Terms and clauses are equivalently represented by their respective sets of literals. We define a term (resp. a clause) called the *empty term $t_\emptyset$* (resp. the *empty clause $c_\emptyset$*) whose set of literals is empty and that is equivalent to 1 (resp. to 0).

CNF and DNF formulas are associated with several kind of graphs (and hypergraphs). The *primal graph* of a CNF (resp. DNF) formula ϕ is the graph $G = (V, E)$ where $V = \{v_x \mid x \in var(\phi)\}$ and such that, for every $x \neq y$, the edge $v_x v_y$ is in E if and only if there is a clause C of ϕ containing literals for both x and y. The *incidence graph* of ϕ is a bipartite graph such that the vertices are $V = \{v_x \mid x \in var(\phi)\}$ on one side and $U = \{u_c \mid c \in clause(\phi)\}$ on the other, and such that there is an edge between v_x and u_c if and only if $x \in var(c)$.

3 Notions on Circuits

3.1 Computational Circuits

Definition 5. *A* computational circuit*, or just a* circuit*, over a set S is a directed acyclic graph (DAG) G whose nodes are labelled as followed: for all nodes v of G*

- *if v has no children, then v is labelled by a constant in S or by a variable taking value in S,*
- *if v has $m > 0$ children, then v is labelled by an operation from S^m to S.*

The nodes labelled by variables are the inputs *of the circuit. The nodes with no parent are the* outputs *of the circuit. A node labelled by an operation $\mathtt{op}$ is called an* $\mathtt{op}$-node.

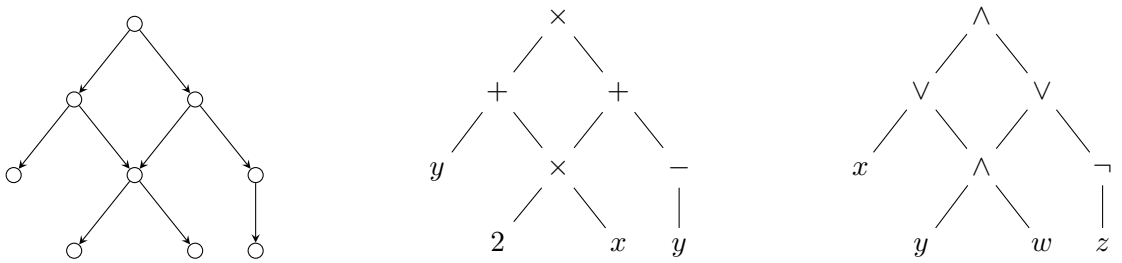


Figure 4: Computation circuits and underlying graphs.

When a circuit has a single output, the corresponding node is called the *root* of the circuit. Let C be a circuit whose underlying graph is G. Each node v of C is the root of a circuit C_v whose underlying DAG is the subgraph of G rooted at v. The set $var(C)$ of variables of C is the set of variables labelling nodes of C. We sometimes use the notation $var(v)$ instead of $var(C_v)$. The size of C, denoted by $|C|$ is the number of edges of the underlying graph.

Assume C as a single output and let a be an assignment to a superset of $var(C)$. For every node v in C we define $C_v(a)$ as follows:

- if v is an input for the variable x then $C_v(a) = a(x)$,
- if v is labelled by a constant c then $C_v(a) = c$,
- if v is an $\mathtt{op}$-node with m children $v_1, \ldots, v_m$, then $C_v(a) = \mathtt{op}(C_{v_1}(a), \ldots, C_{v_m}(a))$ which we will often be able to write $C_v(a) = C_{v_1}(a)\,\mathtt{op}\ldots\mathtt{op}\,C_{v_m}(a)$.

Let r be the root of C. Then we note $C(a) = C_r(a)$. We say that C *computes*, or *represents*, the function that maps every assignment a to $var(C)$ to the value $C(a)$.

Example 2. Figure 4 shows two circuits whose underlying graph is the DAG represented on the left. The circuit in the middle computes the function $x, y \mapsto 4x^2 - y^2$ over $\mathbb{Z}$ and the circuit at the right computes the Boolean function $w, x, y, z \mapsto w \wedge y \wedge (x \vee \neg z)$. ◀

In this thesis we focus exclusively on circuits that have a single output and represent functions over a finite number of variables.

A *Boolean circuit* is a circuit that computes a Boolean function. Any Boolean formula ϕ can be represented as a Boolean circuit: if $\phi = \ell$ or $\phi = 0$ or $\phi = 1$ then the circuit is a single node labelled by ℓ, 0 or 1, and if $\phi = \phi_1 \otimes \cdots \otimes \phi_k$ then the root of the circuit is a k-ary $\otimes$-node whose children are the root of the circuits for $\phi_1, \ldots, \phi_k$. The underlying graph of the resulting Boolean circuit is a tree.

Definition 6. *A class $\mathcal{C}$ of Boolean circuits is called* complete *when, for every finite set X of variables and every Boolean function f over X, there is a circuit $C \in \mathcal{C}$ computing f. Otherwise $\mathcal{C}$ is called* incomplete.

Examples of incomplete classes of circuits include the class of Horn CNF formulas and the class of k-CNF formulas for any fixed k. In contrast the class of CNF formulas is complete. The following two sections introduce the classes of circuits NNF, DNNF, d-DNNF, dec-DNNF, str-DNNF, FBDD, OBDD, DT, nFBDD, nOBDD, SDD, etc. that are all complete.

Definition 7. *Let f be a Boolean function and let $\mathcal{C}$ be a complete class of Boolean circuits. The $\mathcal{C}$* size *of f is the smallest size of a circuit in $\mathcal{C}$ computing f.*

In a Boolean circuit made of $\vee$-nodes, $\wedge$-nodes and unary $\neg$-nodes, we say that we *propagate (away)* the constants when we iteratively apply the following simplifications (in that order) while it is possible:

1) replace every $\vee$-node (resp. $\wedge$-node) whose children are all labelled by 0 (resp. 1) by a node labelled by 0 (resp. 1)
2) replace every $\neg$-node whose child is labelled by 0 (resp. 1) by a node labelled by 1 (resp. 0)
3) replace every $\vee$-node (resp. $\wedge$-node) that has a child labelled by 1 (resp. 0) by a node labelled by 1 (resp. 0)
4) disconnect the remaining $\vee$-nodes (resp. $\wedge$-nodes) from their children labelled by 0 (resp. 1)
5) for each $\vee$-node (resp. $\wedge$-node) that has a single child, connect its parents to its child and remove the node

Once constants have been propagated away, the Boolean circuit is either a single node labelled by 0 or 1, or it has no node labelled by 0 or 1.

3.2 Classes of Circuits in Negation Normal Form

Definition 8 (Circuits in NNF)**.** *A circuit in* Negation Normal Form *(NNF), is a Boolean circuit whose internal nodes are restricted to $\vee$-, $\wedge$-nodes and to unary $\neg$-nodes and such that the child of every $\neg$-node is a variable input. Equivalently, circuits in NNF can be seen as computational circuits whose internal nodes are restricted to binary $\vee$-, $\wedge$-nodes and whose inputs are labelled by literals rather than variables.*

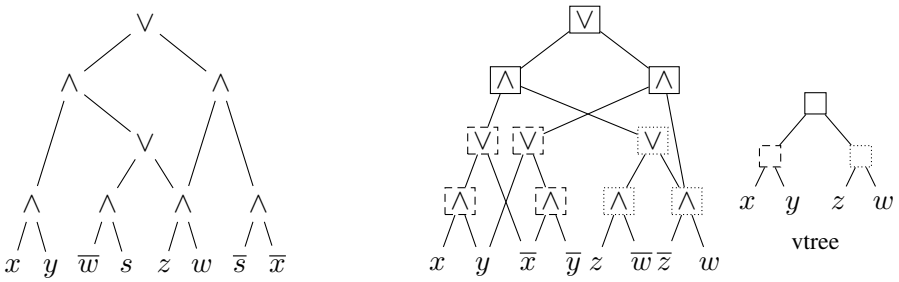
By de Morgan's laws, every Boolean circuit whose internal nodes are $\vee$-, $\wedge$-, and $\neg$-nodes can be transformed in an equivalent circuit in NNF in polynomial time. We denote by NNF the class of all circuits in NNF. CNF formulas can be seen as specific circuits in NNF. We denote by CNF the class of all CNF formulas, thus CNF $\subset$ NNF.

Definition 9 (Circuits in DNNF [Dar01a])**.** *A circuit D in* Decomposable Negation Normal Form *(DNNF), is a circuit in NNF whose $\wedge$-nodes are* decomposable*, that is, for every $\wedge$-node with children $u_1, \ldots, u_m$, it holds that $var(D_{u_i}) \cap var(D_{u_j}) = \emptyset$ for all $i \neq j$.*

Definition 10 (Circuits in d-DNNF [DM02])**.** *A circuit D in* Deterministic Decomposable Negation Normal Form *(d-DNNF), is a circuit in DNNF whose $\vee$-nodes are* deterministic*, that is, for every $\vee$-node with children $u_1, \ldots, u_m$, it holds that $D_{u_i} \wedge D_{u_j} \equiv 0$ for all $i \neq j$.*

We denote by DNNF the class of all circuits in DNNF and by d-DNNF the class of all circuits in d-DNNF. DNF formulas can be seen as specific circuits in DNNF. We denote by DNF the class of all DNF formulas, thus DNF $\subset$ DNNF $\subset$ NNF. In this thesis we often make the assumption that $\wedge$-nodes and $\vee$-nodes are binary, that is, that they each have exactly two children. This assumption is made without loss of generality given the following transformation on circuits in NNF: first remove all unary $\wedge$-nodes and $\vee$-nodes after connecting their respective parents to the unique child, then replace every $\vee$-node (resp. $\wedge$-node) that has $m > 2$ children by a succession of binary $\vee$-nodes (resp. $\wedge$-node). The transformation preserves decomposability and determinism and linearly increases the size of the circuit. So the size of such a circuit can be seen as its number of nodes up to a constant factor.

Definition 11 (Circuits in str-DNNF [PD08])**.** *A circuit D in* Structured-Decomposable Negation Normal Form *(str-DNNF), is a circuit in DNNF whose $\wedge$-nodes have exactly two children and which is equipped with a pair (T, λ) where T is a binary tree whose leaves are in bijection with a superset of $var(D)$, and where λ is mapping from the nodes of D to the nodes of T such that for all node u of D*



(a) A circuit in DNNF

(b) A circuit in str-DNNF and its vtree.

Figure 5: Circuits in DNNF and circuits in str-DNNF.

- $var(D_u) \subseteq var(\lambda(u))$ *holds and,*
- *if* u *is an* $\vee$*-node with children* $u_1, \ldots, u_m$*, then* $\lambda(u) = \lambda(u_1) = \cdots = \lambda(u_m)$
- *if* u *is an* $\wedge$*-node with children* v *and* w*, let* $t = \lambda(u)$*. Either* t *is a leaf of* T *labelled by a variable* x *and* x *is the unique variable appearing under* u*, or* t *an internal node of* T *with children* t_r *and* t_l *there is a node* t' *under its first child and a node* t'' *under its second child such that* $\lambda(v) = t'$ *and* $\lambda(w) = t''$*.*

T *is called a* vtree *(variable tree) over* X*.* D *is said to* respect T*, or to be* structured *by* T*.*

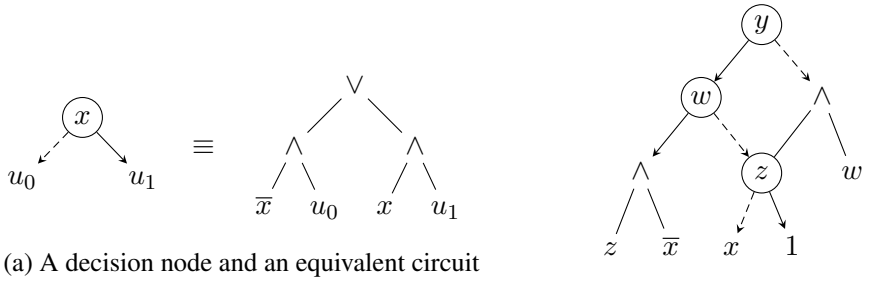
Example 3. Figure 5b shows a circuit in str-DNNF with its vtree. In the circuit shown in the figure, each literal input is mapped to the leaf of the vtree for the corresponding variable. The mapping of internal nodes of the circuit to nodes of the circuit is rendered visible by the different boxes.

Figure 5a, on the other hand, shows a circuit in DNNF which is not structured by any vtree. Indeed assume, towards a contradiction, that the circuit represented is structured by a vtree T and let λ be the mapping from the circuit's nodes to T's nodes. Let t be the node of T corresponding to the root node of the circuit. t can not be a leaf of T so let t_0 and t_1 be its children. The root of the circuit is a $\vee$-node so its children are also mapped to t by λ. Both children of the root are $\wedge$-nodes. Looking at the $\wedge$-node at the right we see that by definition there should be $\{z, w\} \subseteq var(t_j)$ and $\{s, x\} \subseteq var(t_{1-j})$ for some $j \in \{0, 1\}$. Looking at the $\wedge$-node at the left we see that by definition there should be $\{x, y\} \subseteq var(t_i)$ and $\{s, w, z\} \subseteq var(t_{1-i})$ for some $i \in \{0, 1\}$. In the former case x and s are both under t_0 or both under t_1, in the latter case x and s are not both under t_0 or t_1, this is as contradiction, thus the circuit is not structured by any vtree. ◀

Given a vtree T over a finite set of variables X, we denote by $\mathsf{str\text{-}DNNF}_T$ the class of circuits in str-DNNF structured by T. Note that since X is finite, the circuits in this class can only compute functions over X or subset of X. Consequently $\mathsf{str\text{-}DNNF}_T$ contains a finite number of circuits. We denote $\mathsf{str\text{-}DNNF} = \bigcup_T \mathsf{str\text{-}DNNF}_T$ the class of circuits in DNNF structured by some vtree.

Definition 12 (Circuits in dec-DNNF [HD07])**.** *A circuit* C *in* Decision Decomposable Negation Normal Form *(dec-DNNF), is a circuit in DNNF such that every* $\vee$*-node has exactly two children that are both* $\wedge$*-nodes, and such for every* $\vee$*-node* u *with children* v *and* w*, there is a variable* x *such that a child of* v *is labelled by* x *and a child of* w *is labelled by* $\overline{x}$ *(or vice-versa).*

We denote by $\mathsf{dec\text{-}DNNF}$ the class of all circuits in dec-DNNF. It is readily verified that the $\vee$-nodes of a circuit in dec-DNNF are deterministic, thus $\mathsf{dec\text{-}DNNF} \subset \mathsf{d\text{-}DNNF}$. The $\vee$-nodes of a circuit in



(a) A decision node and an equivalent circuit representation using $\vee$-nodes and $\wedge$-nodes.

(b) A circuit in dec-DNNF.

Figure 6: Circuits in DNNF with decision nodes.

dec-DNNF are always represented by *decision nodes*, as shown in Figure 6a. A decision node is labelled by a variable and has two children called the 0-child and the 1-child. If u is a decision node labelled by x, whose 0-child is u_0 and whose 1-child is u_1, then we have that $C_u \equiv (\overline{x} \wedge C_{u_0}) \vee (x \wedge C_{u_1})$. An example of circuit in dec-DNNF is shown Figure 6b.

Definition 13 (Smooth NNF circuits)**.** *An circuit D in NNF is called* smooth *when, for every $\vee$-node v of D, calling $v_1, \ldots, v_m$ its children, we have that $var(D_v) = var(D_{v_1}) = \cdots = var(D_{v_m})$.*

Smoothness is a property that can be enforced on any circuit in DNNF, d-DNNF, str-DNNF or dec-DNNF in polynomial time, see for instance [DM02] for the case of circuits in DNNF or d-DNNF.

3.3 Classes of Binary Decision Diagrams

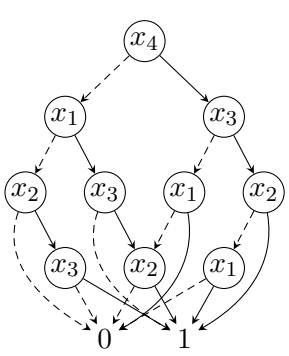
Let X be a set of Boolean variables and let S be any set (for instance $Y = \{0, 1\}$, or $Y = \mathbb{N}$). A binary decision diagram, or BDD, over X and with domain Y, is a directed acyclic graph with the following properties:

- the circuit has a single node without parent called the *root* or the *source*,
- the nodes with no children, called the *sinks*, are labelled by elements of Y,
- internal nodes are decision nodes labelled by variables in X (as defined above for circuits in dec-DNNF).

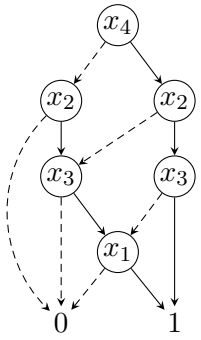
A path from the source to a sink is called a *computation path*. Let B be a BDD over X and with domain Y and let a be an assignment to X. We construct a computation path corresponding to a. Start from the source of B and repeat the following until reaching a sink: if the current node is a decision node u for a variable x then follow the 0-child of u if $a(x) = 0$ and follow the 1-child of u if $a(x) = 1$. We denote by $B(a)$ the element labelling the sink reached. Note that a computation path may correspond to several assignments. We say that B *computes* the function from $\{0, 1\}^X$ to Y that maps $a \in \{0, 1\}^X$ to $B(a)$.

Definition 14 (FBDD)**.** *Let X be a finite set of Boolean variables. A* free binary decision diagram *(FBDD) over X, is a BDD over X with domain $\{0, 1\}$ where each variable appears at most once on every computation path.*

An example of FBDD is shown Figure 7a. We denote by FBDD the class of all FBDDs. A notable subclass of FBDD is the class DT of *decision trees*, that is, FBDDs whose underlying graphs are trees.



(a) An FBDD.



(b) An OBDD with the variable ordering $\pi(x_4) < \pi(x_2) < \pi(x_3) < \pi(x_1)$.

Figure 7: Free BDD and ordered BDD

Definition 15 (OBDD)**.** *Let X be a finite set of Boolean variables and let π be a bijection from X to $\{1, \ldots, |X|\}$. An* π-ordered binary decision diagram *(π-OBDD) over X, is an FBDD over X such that for each computation path P and for every two decision nodes u and v in P for the variables x_u and x_v, it holds that $\pi(x_u) \leq \pi(x_v)$ if and only if u appears before v in P. We say that the computation path are* consistent *with π. When π is not specified, we just called the circuit an OBDD. π is called the* variable ordering *of the OBDD.*

An example of OBDD is shown Figure 7b. We denote by OBDD the class of all OBDDs. For a given variable-ordering $\pi : X \to \{1, \ldots, |X|\}$, we denote by π-OBDD the class of all π-OBDDs. Since X is finite, the class π-OBDD is finite. Rewriting the decision nodes of an OBDD as shown Figure 6a yields a circuit in dec-DNNF that is structured by a *linear vtree*, that is, a vtree where every internal node has at least one child that is a leaf. So OBDD can be seen as a subclass of dec-DNNF $\cap$ str-DNNF. It is clear that OBDD $\subset$ FBDD. It is also easily verified that FBDD contains exactly the circuits of dec-DNNF that have no $\wedge$-nodes and whose leaves are labelled by 1 or 0. Thus OBDD $\subset$ FBDD $\subset$ dec-DNNF $\subset$ d-DNNF $\subset$ DNNF.

Non-deterministic BDDs (nBDDs for short) are BDDs that have unlabelled internal nodes called "guess nodes" along with decision nodes. A computation path for an assignment a in an nBDD is defined like a computation path for in a BDD, except that when a guess node is in the path, the following node is any of its children. Thus there can be several computation path for a. For an nBDD B over X with domain Y, $B(a)$ denotes the set of elements in Y that can be reached by a computation path for a. When $Y = \{0, 1\}$, B computes the Boolean function over X whose models are the assignments a such that $1 \in B(a)$.

Definition 16 (nFBDD and nOBDD)**.** *An nFBDD is an nBDD with domain $\{0, 1\}$ where no path contains two decision nodes labelled by the same variable. An nOBDD is an nFBDD where the order of the variables along any paths is consistent with a fixed variable ordering.*

We call nFBDD the class of nFBDDs and nOBDD the class of nOBDDs. It is easy to see that OBDD $\subset$ nOBDD and FBDD $\subset$ nFBDD. The transformation of FBDDs into circuits in DNNF can be extended to nFBDDs by replacing simply replacing guess nodes by $\vee$-nodes (note that the resulting circuit in DNNF may not be deterministic). It follows that nFBDD is a subclass of DNNF.

To finish this section on binary decision diagrams, we present a generalization of OBDDs called sentential decision diagrams (SDDs for short) and introduced in [Dar11]. Since SDDs are not directly studied in this thesis we omit their formal definition. SDDs can be interpreted as specific deterministic

circuits in str-DNNF, so the class SDD of all SDDs can be seen as contained in str-DNNF ∩ d-DNNF. Several compilers to SDD have been developed, see for instance [CD13, OD15]. It is known that some functions can only be represented by large OBDDs (OBDDs whose size is exponential in the number of variables) but have small representations as SDDs (SDDs whose size is polynomial in the number of variables) [Bov16].

4 Knowledge Compilation

Knowledge compilation is a paradigm for reasoning on a knowledge base that is known in advance and that may thus be preprocessed during a phase called *compilation*. In our case the knowledge base is a function (often a Boolean function). Functions can be represented in different ways, for instance a Boolean function on finitely many variables can be represented by a circuit in DNNF or by a BDD. We call *language* a class of representations for a class of functions. So for each object C of a language $\mathcal{C}$, there is a unique interpretation of C as a function. Furthermore we assume that the size of C, denoted by $|C|$ is defined for every $C \in \mathcal{C}$. A compilation from a language $\mathcal{C}_0$ to the language $\mathcal{C}_1$ is a procedure that, given the representation of a function in $\mathcal{C}_0$, returns an equivalent representation in $\mathcal{C}_1$ (if it exists). Note that the running time of a compilation from $\mathcal{C}_0$ to $\mathcal{C}_1$ is irrelevant. $\mathcal{C}_0$ is called the *initial language* and $\mathcal{C}_1$ is called the *compilation language*, or the *target language*. We will use the classes of circuits and BDDs defined in the previous section as languages. For instance we will often write "the DNNF language" or "the SDD language".

Definition 17. *Let $\mathcal{C}_1$ and $\mathcal{C}_0$ be two languages.$\mathcal{C}_1$ is called* as expressive *as $\mathcal{C}_0$ if, for every $C_0 \in \mathcal{C}_0$, there exists $C_1 \in \mathcal{C}_1$ that represents the same function.*

Definition 18. *Let $\mathcal{C}_1$ and $\mathcal{C}_0$ be two languages such that $\mathcal{C}_1$ is as expressive as $\mathcal{C}_0$. $\mathcal{C}_1$ is* more succinct *than $\mathcal{C}_0$, denoted by $\mathcal{C}_1 \leq \mathcal{C}_0$, when there exists a polynomial p such that, for every $C_0 \in \mathcal{C}_0$, there exists $C_1 \in \mathcal{C}_1$ such that $|C_1| \leq p(|C_0|)$. $\mathcal{C}_1$ is* strictly more succinct *than $\mathcal{C}_0$, denoted by $\mathcal{C}_1 < \mathcal{C}_0$, when $\mathcal{C}_1 \leq \mathcal{C}_0$ and $\mathcal{C}_0 \not\leq \mathcal{C}_1$. $\mathcal{C}_1$ and $\mathcal{C}_0$ are* equally succinct*, denoted by $\mathcal{C}_1 \simeq \mathcal{C}_0$ when when $\mathcal{C}_1 \leq \mathcal{C}_0$ and $\mathcal{C}_0 \leq \mathcal{C}_1$.*

Definition 19. *Let $\mathcal{C}_1$ and $\mathcal{C}_0$ be two languages such that $\mathcal{C}_1$ is as expressive as $\mathcal{C}_0$. A polynomial-size compilation from $\mathcal{C}_0$ to $\mathcal{C}_1$ is a computable function $c : \mathcal{C}_0 \to \mathcal{C}_1$ such that*

- *for every $C \in \mathcal{C}_0$, C and $c(C)$ represent the same function and*
- *there exists a polynomial p such that for every $C \in \mathcal{C}_0$ we have $|c(C)| \leq p(|C|)$*

It is readily verified that a polynomial-size compilation from $\mathcal{C}_0$ to $\mathcal{C}_1$ exists only if $\mathcal{C}_1 \leq \mathcal{C}_0$.

The *knowledge compilation map* is a framework introduced by Darwiche and Marquis in [DM02] to compare compilation languages according to two standards. On the one hand, compilation languages are pairwise compared in terms of succinctness so that we know which languages are more likely to yield the smallest compiled forms. On the other hand the languages are compared in terms of their efficiency for answering queries and performing transformations. *Queries* aim at extracting valuable information from the circuit, while *transformations* aim at modifying the circuit while staying within the language. Darwiche and Marquis's map includes eight queries among which *satisfiability testing* (is the circuit satisfiable?), *model counting* (how many models does the circuit have?) and *clausal entailment* (does the circuit entails a given clause?). It also includes eight transformations including *conditioning* (given a circuit C in L and an assignment a find a circuit in L computing $C|a$) or *bounded conjunction* (given two circuits in the language L find another circuit in L that compute their conjunction). A language L is said to *support* a query (or a transformation) if there is a polynomial-time algorithm answering that query

(or performing the transformation) in L. Compilation from an initial language L_i to a target language L_t is useful only if certain queries and transformations are easier to perform in L_t than in L_i. Ideally, the most important queries and transformations for the user should be *supported* by the compilation language, that is, they should be tractable in the language. Several queries and transformations have studied for several sublanguages of NNF in [DM02]. One of the take-home message of [DM02] is that the languages supporting the most queries and transformations are DNNF and its sublanguages (although other languages distinct from DNNF have since been considered, see e.g. [KLMT13b]). As a result, in practice, target languages for compiling Boolean functions are sublanguages of DNNF such as that described in previous sections, see for instance the compilers described in [Dar02a, Dar04, OD15, LM17].

5 Rectangle Covers for the Analysis of Circuits in DNNF

5.1 Rectangle Covers

Definition 20. *Let X be a finite set of variables. A* bipartition *of X is a pair (X_1, X_2) such that $X_1 \cup X_2 = X$ and $X_1 \cap X_2 = \emptyset$. The partition is called* balanced *when $\frac{|X|}{3} \leq |X_1| \leq \frac{2|X|}{3}$.*

We often talk of a *balanced partition* when it is clear from context that it is a balanced bipartition.

Definition 21. *Let X be a finite set of Boolean variables and let $\Pi = (X_1, X_2)$ be a partition of X. A set of S assignments to X is called a (combinatorial)* rectangle with respect to Π *when there are a set A of assignments to X_1 and a set B of assignments to X_2 such that $S = \{a \cup b \mid a \in A, b \in B\} := A \times B$.*

Every set of assignments S to X can be seen as a rectangle for the trivial partition $(X, \emptyset)$ with $A = S$ and $B = \{a_\emptyset\}$. But we will not be interested in such rectangles.

Another way to look at rectangles is as Boolean functions. A rectangle r with respect to a partition (X_1, X_2) of x is then a Boolean function $r(X) = \rho_1(X_1) \wedge \rho_2(X_2)$ where $\rho_1 : \{0,1\}^{X_1} \to \{0,1\}$ and $\rho_2 : \{0,1\}^{X_2} \to \{0,1\}$ two Boolean functions over X_1 and X_2 respectively. The link between this definition and that in terms of sets of assignments is made setting $A := \rho_1^{-1}(1)$ and $B := \rho_2^{-1}(1)$. We mostly see rectangles as functions in this thesis.

The name "rectangle" is better understood using a matrix representation. Let $r(X) = \rho_1(X_1) \wedge \rho_2(X_20$ and let M_r be the matrix whose columns are indexed by all assignments to X_1 and whose rows are indexed by all assignments to X_2 and such that the entry of M_r at column a_1 and row a_2 is 1 if $\rho_1(a_1) = 1$ and $\rho_2(a_2) = 1$, and 0 otherwise. Then there are ways to order the rows and columns of M_R such that the 1s of M_R form a rectangle submatrix in M_R.

Example 4. Let $X = \{x_1, x_2, x_3, x_4, x_5\}$ and consider the partition $\Pi = (\{x_1, x_2, x_3\}, \{x_4, x_5\}) = (X_1, X_2)$ of X. Let A be the set that contains exactly the assignments to $\{x_1, x_2, x_3\}$ that map at least two variables to 1 and let B be the set that contains exactly the assignments to $\{x_4, x_5\}$ that map at least one variable to 1. Then $A \times B$ is a rectangle with respect to π. The function representation of $A \times B$ is $r = \mathbb{1}_{[x_1+x_2+x_3 \geq 2]} \wedge \mathbb{1}_{[x_4+x_5 \geq 1]}$.

$$
\begin{array}{c}
\begin{array}{cc} x_1 & 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1 \\ x_2 & 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1 \\ x_3 & 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1 \end{array} \\
\begin{array}{c} x_4x_5 \\ 00 \\ 10 \\ 01 \\ 11 \end{array}
\left(\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \boxed{1} & \boxed{1} & \boxed{1} & \boxed{1} \\
0 & 0 & 0 & 0 & \boxed{1} & \boxed{1} & \boxed{1} & \boxed{1} \\
0 & 0 & 0 & 0 & \boxed{1} & \boxed{1} & \boxed{1} & \boxed{1}
\end{array}\right)
\end{array}
$$

A possible matrix M_r is the one above, where the rectangle submatrix of 1s has been made visible. ◀

Definition 22. *Let X be a finite set of Boolean variables and let $f : \{0,1\}^X \to \{0,1\}$. A* rectangle cover of f *is a set of rectangles $\{r_1, \ldots, r_N\}$, possibly with respect to different bipartitions of X, such that $f^{-1}(1) = r_1 \cup \cdots \cup r_N$ (or $f = r_1 \vee \cdots \vee r_N$ when the rectangles are seen as Boolean functions).*

A rectangle cover of f is called balanced *when all rectangles of the cover are with respect with, possibly different, balanced partitions of X.*

The size *of a rectangle cover is the number of rectangles in the cover.*

Example 5. Let maj_5 be the Boolean function over $\{x_1, x_2, x_3, x_4, x_5\}$ that maps to 1 exactly the assignments to $\{x_1, x_2, x_3, x_4, x_5\}$ that set at least three variables to 1. Consider the balanced partition $\Pi = (\{x_1, x_2, x_3\}, \{x_4, x_5\})$. For $1 \leq i \leq 3$, let A_i be the set of assignments to $\{x_1, x_2, x_3\}$ such that at least $4 - i$ variables are set to 1, and let B_i be the set of assignments to $\{x_4, x_5\}$ such that at least $i - 1$ variables are set to 1. Then $A_1 \times B_1$, $A_2 \times B_2$ and $A_3 \times B_3$ are rectangles with respect to Π whose equivalent representations as Boolean functions are, respectively:

$$r_1 = \mathbb{1}_{[x_1+x_2+x_3=3]}$$

$$r_2 = \mathbb{1}_{[x_1+x_2+x_3\geq 2]} \wedge \mathbb{1}_{[x_4+x_5\geq 1]}$$

$$r_3 = \mathbb{1}_{[x_1+x_2+x_3\geq 1]} \wedge \mathbb{1}_{[x_4+x_5\geq 2]}$$

It is easy to see that $maj_5(x_1, x_2, x_3, x_4, x_5) \equiv r_1 \vee r_2 \vee r_3$, so $\{r_1, r_2, r_3\}$ is a balanced rectangle cover of maj_5. The size of the cover is 3. In this example the rectangles in the cover respect the same variable partition. ◀

Balanced rectangle covers exist for every Boolean function. To see this, let a be an assignment to X, and $\mathbb{1}_a : \{0,1\}^X \to \{0,1\}$ be the Boolean function that maps a to 1 and every other assignment to 0. It is easy to represent $\mathbb{1}_a$ as a rectangle with respect to any partition (X_1, X_2) of X. Simply let a_1 and a_2 be the restriction of a to X_1 and X_2, respectively and there is $\mathbb{1}_a(X) = \mathbb{1}_{a_1}(X_1) \wedge \mathbb{1}_{a_2}(X_2)$, so $\mathbb{1}_a$ is a rectangle with respect to (X_1, X_2). This holds regardless of a and of the partition (X_1, X_2). Consequently for any function $f : \{0,1\}^X \to \{0,1\}$, $\{\mathbb{1}_a \mid a \in f^{-1}(1)\}$ is a balanced rectangle cover of f. Thus the following definition is sound:

Definition 23. *Let X be a finite set of Boolean variables and let $f : \{0,1\}^X \to \{0,1\}$. The minimum number of rectangles in a* balanced *rectangle cover of f is denoted by $R(f)$.*

Since there is at least one *balanced* rectangle cover for every function, $R(f)$ is well-defined for every f. The restriction to balanced rectangle cover is important. Lifting this restriction would allow rectangle with respect to trivial partitions, that are clearly not balanced, and therefore yields $R(f) = 1$ for every f.

Example 6. We show that $R(maj_5) = 3$. For this example we denote by $w(a) = |a^{-1}(1)|$ the weight of a. The minimal models of a monotone function are the satisfying assignments such that setting to 1 a variable previously set to 0 yields a falsifying assignment. In the case of maj_5, the minimal models are the assignments of weight 3. Let r be a rectangle with respect to a balanced partition (X_1, X_2) such that $r \leq maj_5$. Without loss of generality, we may assume that $|X_1| = 3$ and $|X_2| = 2$. Let a and b be minimal models of maj_5 such that $r(a) = r(b) = 1$. Let a_1, b_1 be their restrictions to X_1 and a_2, b_2 be their restrictions to X_2. Note that $w(a_1), w(b_1), w(a_2), w(b_2) \leq 3$. If $w(a_1) \neq w(b_1)$, say $w(a_1) < w(b_1)$, since r is a rectangle we have $r(a_1 \cup b_2) = 1$, but $w(a_1 \cup b_2) = w(a_1) + w(b_2) = w(a_1) + 3 - w(b_1) < 3$ so r would accept an assignment falsifying maj_5. Thus for r, there exists $k_r \in \{1, 2, 3\}$ such that r accepts only the minimal models of maj_5 such that $w(a_1) = k_r$ and $w(a_2) = 3 - k_r$. If

$k_r = 1$ then r accepts at most three minimal models, if $k_r = 2$ then r accepts at most six minimal models and if $k_r = 3$ then r accepts at most one minimal model.

Now assume $maj_5 = r \vee r'$ where r' as another rectangle with respect to a balanced partition (X'_1, X'_2). Without loss of generality we assume that $|X'_1| = 3$ and that $|X'_2| = 2$. The number of minimal models of maj_5 accepted by $r \vee r'$ is at most that accepted by r plus that accepted by r'. Since there are ten such models and since r and r' can each accept more than 5 minimal models only if $k_r = k_{r'} = 2$, the only way for all ten minimal models of maj_5 to be accepted by $r \vee r'$ is if $k_r = k_{r'} = 2$. We consider three cases.

- If $|X_1 \cap X'_1| = 3$ then $X_1 = X'_1$. Thus r and r' can only accept all six same minimal models, which is not enough.
- If $|X_1 \cap X'_1| = 2$ then let a be the assignment that sets the variables of $X_1 \cap X'_1$ to 0 and all other variables to 1. Then a is a minimal model of maj_5 but a is accepted by neither r nor r' as its weight restricted to X_1 or X'_1 is fewer than 2.
- If $|X_1 \cap X'_1| = 1$ then let a be the assignment that set all variables in X_1 to 1 and all other variables to 0. Then a is a minimal model of maj_5 but a is not accepted by r since it has weight 3 on X_1 instead of 2, and it is not accepted by r since it has weight 1 on X'_1 instead of 2.

So in all three cases ($|X_1 \cap X'_1| = 0$ is not possible) we reach a contradiction. This shows that $R(maj_5) > 2$. Example 5 shows a balanced rectangle cover of maj_5 of size 3, thus $R(maj_5) = 3$. ◀

Two rectangles r and r' over X, potentially with respect to different partitions, are called *disjoint* when $r \wedge r' \equiv 0$ (or, equivalently, when $r \cap r' = \emptyset$ when the rectangles are seen as sets of assignments). Given two distinct assignments a and a' to X, it is clear that the rectangle corresponding to $\mathbb{1}_a$ and the rectangle corresponding to $\mathbb{1}_{a'}$ are disjoint. The set $\{\mathbb{1}_a \mid a \in f^{-1}(1)\}$ can be seen as a balanced rectangle cover of f whose rectangles are pairwise disjoint.

Definition 24. *Let X be a finite set of Boolean variables and let $f : \{0,1\}^X \to \{0,1\}$.*

- *The minimum number of rectangles in a balanced rectangle cover of f whose rectangles are pairwise disjoint is denoted by $R_d(f)$.*
- *The minimum number of rectangles in a balanced rectangle cover of f whose rectangles are all with respect to the partition Π is denoted by $R_\Pi(f)$.*
- *The minimum number of rectangles in a balanced rectangle cover of f whose rectangles are pairwise disjoint and all respect the partition Π is denoted by $R_{d,\Pi}(f)$.*

It is readily verified that $R(f) \leq R_d(f) \leq R_{d,\Pi}(f)$ and that $R(f) \leq R_\Pi(f) \leq R_{d,\Pi}(f)$ for every balanced partition Π.

Example 7. The rectangles of the cover for maj_5 shown in Example 5 are not pairwise disjoint. For instance the assignment that set all variables to 1 is accepted by $r_1 = \mathbb{1}_{[x_1+x_2+x_3=3]}$, and by $r_2 = \mathbb{1}_{[x_1+x_2+x_3\geq 2]} \wedge \mathbb{1}_{[x_4+x_5\geq 1]}$, and by $r_3 = \mathbb{1}_{[x_1+x_2+x_3\geq 1]} \wedge \mathbb{1}_{[x_4+x_5\geq 2]}$. However the rectangles can easily be modified in a way that render them pairwise disjoint. Indeed consider the following three rectangles with respect to the balanced partition $\Pi = (\{x_1, x_2, x_3\}, \{x_4, x_5\})$:

$$r'_1 = \mathbb{1}_{[x_1+x_2+x_3=3]}$$

$$r'_2 = \mathbb{1}_{[x_1+x_2+x_3=2]} \wedge \mathbb{1}_{[x_4+x_5\geq 1]}$$

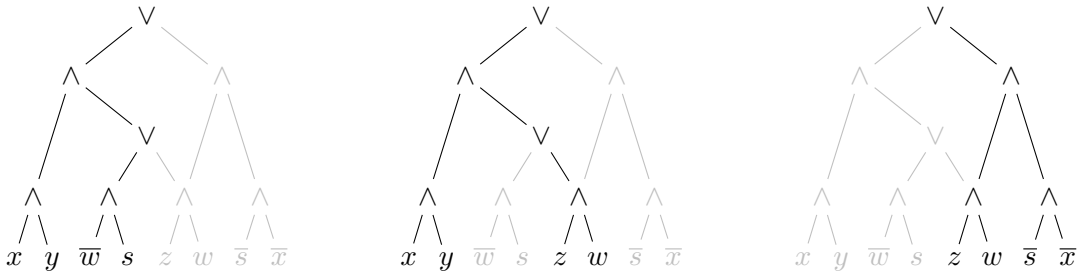


Figure 8: Proof trees of a circuit in DNNF.

$$r'_3 = \mathbb{1}_{[x_1+x_2+x_3=1]} \wedge \mathbb{1}_{[x_4+x_5\geq 2]}$$

It is easy to see that $maj_5 \equiv r'_1 \vee r'_2 \vee r'_3$ and that the three rectangles are pairwise disjoint. Since we have shown in Example 6 that $R(maj_5) = 3$, it follows that $R_d(maj_5) = 3$. Since the three rectangles are with respect to the same partition Π, we also have that $R(maj_5) = R_\Pi(maj_5) = R_{d,\Pi}(maj_5) = 3$. ◀

5.2 Proof Trees

In the next section we will recap known relationships between the size of the smallest rectangle cover of a function and the DNNF, d-DNNF and str-DNNF sizes of that function. We provide the necessary background by introducing the concept of proof trees. This concept has been identified, indeed proof trees correspond to *certificates* in [BCMS16] and to *complete subcircuits* in the context of arithmetic circuits in [CD06].

Definition 25. *Let D be a circuit in DNNF where constants have been propagated away. Let G be the underlying DAG of D and assume G has a single node with no ancestor called the* root *of G (or the root of D). A* proof tree *T of D is any subcircuit of D for a subgraph G' of G constructed as follows. First G' contains only the root of G. Next, while there exists a node $u \in V(G) \cap V(G')$ that has children in G but no children in G'*

- *if u is a $\wedge$-node, then for* every *child v of u in G, add v to $V(G')$ and uv to $E(G')$*
- *if u is a $\vee$-node, then choose* one *child v of u in G, add v to $V(G')$ and uv to $E(G')$*

As the name indicates, the proof trees of a circuit in DNNF are Boolean circuits shaped like trees.

Lemma 3. *Let D be a circuit in DNNF where constants have been propagated away and that has a single root. Let T be a proof tree of D. Then the underlying graph of T is a tree whose edges are directed from the root to the leafs.*

Let T be a proof tree of D. Every node u of T is also a node of D, but not all parents and children of u in D appear T. Since proof trees are circuits in NNF, the notations $var(T)$ and $lit(T)$ are well-defined. Recall that T_u is the subcircuit of T rooted at u. It is readily verified that, if a node u of a circuit D in DNNF is also a node of the proof tree T of D, then T_u is a proof tree of D_u. It is also clear that $var(T) \subseteq var(D)$, thus proof trees of a circuit in DNNF are also circuits in DNNF.

Lemma 4. *Let D be a circuit in DNNF where constants have been propagated away and that has a single root. Any proof tree of D computes a single term over $var(D)$. Moreover if D is smooth, then for every proof tree T of D we have $var(T) = var(D)$ so T computes a term that accepts a single assignment to $var(D)$.*

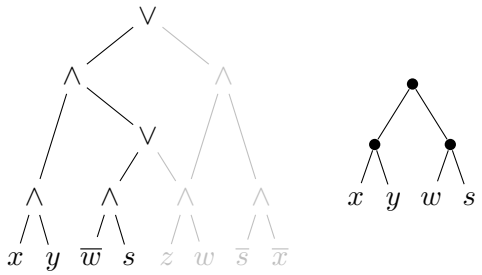


Figure 9: A proof tree and its underlying vtree.

Lemma 5. *Let D be a circuit in DNNF where constants have been propagated away and that has a single root. Let $T_1, \ldots, T_k$ be all the proof trees of D then $D \equiv T_1 \vee \cdots \vee T_k$.*

Example 8. The circuit in DNNF represented Figure 5a has three proof trees that are represented in Figure 8. The proof tree on the left computes the term $x \wedge y \wedge \overline{w} \wedge s$, the proof tree on the middle computes the term $x \wedge y \wedge z \wedge w$, and the proof proof on the right computes the term $z \wedge w \wedge \overline{s} \wedge \overline{x}$. Thus the circuit in DNNF represented Figure 5a computes the function $(x \wedge y \wedge \overline{w} \wedge s) \vee (x \wedge y \wedge z \wedge w) \vee (z \wedge w \wedge \overline{s} \wedge \overline{x})$. ◀

The definition of proof trees clearly applies to circuits in DNNF respecting additional properties like determinism for circuits in d-DNNF and structured decomposability for circuits in str-DNNF. The sets of proof trees generated from circuits in d-DNNF or str-DNNF have additional properties that we describe now.

Lemma 6. *Let D be a circuit in d-DNNF where constants have been propagated away and that has a single root. Let T and T' be distinct proof trees of D, then $T \wedge T' \equiv 0$.*

Recall that proof trees of a circuit in DNNF are circuits in DNNF. Since it holds that $var(T_u) \subseteq var(D_u)$ for every node u appearing in a proof tree T of D, if D is structured by a vtree then T is structured by the same vtree.

Lemma 7. *Let D be a circuit in str-DNNF. Let T be proof tree of D, then T is structured by the same vtree as D.*

5.3 Lower Bounds on the DNNF Size from Rectangle Covers

Proof trees can be gathered into rectangles so as to establish a bound between $R(f)$ and DNNF size of f. The bound in question is proved in [BCMS16].

Lemma 8. [BCMS16] *Let D be a circuit in DNNF. For v a $\wedge$-node in D, let $sat(D, v)$ be the set of assignments to $var(D)$ such that $a \in sat(D, v)$ if and only if a satisfies some proof tree of D containing v. Then $sat(D, v)$ is a rectangle with respect to the partition $(var(D_v), var(D) \setminus var(D_v))$ and $sat(D, v) \subseteq sat(D)$.*

Theorem 1. [BCMS16] *Let D be a smooth circuit in DNNF, then $R(D) \leq |D|$.*

The idea of the proof is essentially to find a node v in D such that $(var(D_v), var(D) \setminus var(D_v))$ is a balanced partition of $var(D)$, to add the rectangle $sat(D, v)$ given by Lemma 8 to the cover, to delete v from D, and to repeat the procedure until D is empty. Removing v from D only remove models of D that have already been added to the cover via the rectangle $sat(D, v)$, in addition, removing v prevents from choosing a node twice in the procedure. If D is a smooth circuit in d-DNNF, then removing v also ensures that the next rectangles in the cover will be disjoint from the ones already in it, thus:

Theorem 2. [BCMS16] *Let D be a smooth circuit in d-DNNF, then $R_d(D) \leq |D|$.*

Finally, if D is a smooth circuit in str-DNNF respecting some vtree, one can show that the node chosen at every iteration of the procedure can be selected so that it always corresponds to the same node in the vtree. This ensures that the partition $(var(D_v), var(D) \setminus var(D_v))$ is consistent throughout the procedure. One can show the following in consequence:

Theorem 3. *Let D be a smooth circuit in str-DNNF, then $R_\Pi(D) \leq |D|$ for Π some balanced partition.*

6 Tseitin Formulas

Tseitin formulas are CNF formulas representing systems of parity constraints structured by a graph $G = (V, E)$. The graph is equipped with a function $c : V \to \{0, 1\}$ which assigns *charges* 0 or 1 to its vertices. Each edge e of G is associated to a Boolean variable x_e. Given a set $E' \subseteq E$, we write $X_{E'} = \{x_e \mid e \in E'\}$. The Tseitin formula encodes the fact that, if we only keep in G the edges whose variables are given value 1, then all vertices with charge 1 have an odd degree and all vertices with charge 0 have an even degree. More formally let $\chi_v(G, c)$ be the constraint

$$\chi_v(G, c) : \sum_{e \in E_G(v)} x_e = c(v) \mod 2$$

Note that, if $E(v) = \emptyset$, that is, v is an isolated vertex in G, then $\chi_v(G, c)$ is simply the constraint $0 = c(v)$. The Tseitin formula $T(G, c)$ is a CNF formula that computes $\bigwedge_{v \in V} \chi_v(G, c)$. Each $\chi_v(G, c)$ is turned into a CNF formula $F_v(G, c)$ as follows. Assume $E(v)$ is not empty, then $F_v(G, c)$ is a CNF formula on variables $X_{E(v)}$ composed of $2^{|E(v)|-1} = 2^{\deg(v)-1}$ clauses of size $\deg(v)$. In the case when $E(v) = \emptyset$, then either $c(v) = 0$ in which case $F_v(G, c)$ is empty (thus valid), or $c(v) = 1$ in which case $F_v(G, c)$ contains only the empty clause (thus is unsatisfiable). The Tseitin formula over G for the charge function c is the CNF formula

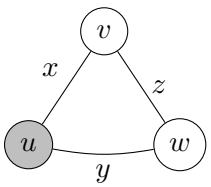
$$T(G, c) = \bigwedge_{v \in V} F_v(G, c)$$

For convenience we often write only $T(G)$, χ_v, or F_v.

For $v \in V$ let $\mathbb{1}_v : V \to \{0, 1\}$ be the function mapping v to 1 and all other vertices to 0. The complement parity constraint to $\chi_v(G, c)$ is $\chi_v(G, c + \mathbb{1}_v \mod 2)$, which we write $\overline{\chi_v(G, c)}$ for convenience.

We use the notation $clause(\chi_v(G, c)) = clause(F_v(G, c)$. Thus $clause(T(G, c)) = \bigcup_{v \in V(G)} clause(\chi_v(G, c))$.

Example 9. Let G be the graph



where charges are indicated by the color code: gray vertices all have charge 0 and white vertices all have

charge 1. We have:

$$\chi_u : x + y = 0 \mod 2$$
$$\chi_w : y + z = 1 \mod 2$$
$$\chi_v : x + z = 1 \mod 2$$
$$F_u = (\overline{x} \vee y) \wedge (x \vee \overline{y})$$
$$F_w = (y \vee z) \wedge (\overline{y} \vee \overline{z})$$
$$F_v = (x \vee z) \wedge (\overline{x} \vee \overline{z})$$
$$T(G) = (\overline{x} \vee y) \wedge (x \vee \overline{y}) \wedge (y \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (x \vee z) \wedge (\overline{x} \vee \overline{z})$$

◀

There is a simple criterion for the satisfiability of Tseitin formulas and a simple expression to compute their number of models.

Lemma 9. [Urq87] *$T(G, c)$ is satisfiable if and only if $\sum_{v \in U} c(v) = 0 \mod 2$ holds for all connected components $G' = (U, E')$ of G.*

Lemma 10. [GI17] *Let K be the number of connected components of G. If $T(G, c)$ is satisfiable, then it has $2^{|E(G)|-|V(G)|+K}$ models.*

Example 10. In Example 9, the graph G has three vertices u, v and w, three edges corresponding to the variables x, y and z, and only one connected component. There is $c(u) + c(v) + c(w) = 2 = 0 \mod 2$, so $T(G, c)$ has $2^{3-3+1} = 2$ satisfying assignments. One can verify that these assignments are $\{\overline{x}, \overline{y}, z\}$ and $\{x, y, \overline{z}\}$. ◀

After conditioning a Tseitin formula $T(G, c)$ on a partial assignment a to its variables, the resulting CNF formula is also a Tseitin formula. Let $E_a \subseteq E(G)$ be the set of edges corresponding to the variables assigned by a and define $G_a = (V(G), E(G) \setminus E_a)$. Let $E_a(v) = E_a \cap E(v)$ and let $c_a : V(G) \setminus V_a \to \{0, 1\}$ be the following charge function:

$c_a(v) = c(v)$ if the number of edges of $E_a(v)$ whose variables are assigned value 1 by a is even,
$c_a(v) = c(v) + 1 \mod 2$ otherwise

Note that $E_a \cap E(v)$ may be empty (then $c_a(v) = c(v)$). The assignments satisfying $F_v(G, c)$ are the solutions to $\chi_v(G, c) : \sum_{e \in E(v)} x_e = c(v) \mod 2$, the assignments satisfying $F_v(G, c)|a$ are the solutions to $\sum_{e \in E(v) \setminus E_a(v)} x_e = c(v) - \sum_{e \in E_a(v)} a(x_e) = c_a(v) \mod 2$, which is exactly the constraint $\chi_v(G_a, c_a)$. Thus we have that

$$T(G, c)|a = T(G_a, c_a).$$

This is not just an logical equivalence but a syntactic equality: $T(G, c)|a$ and $T(G_a, c_a)$ contains exactly the same clauses (modulo the removal of duplicate clauses in each formula). Especially if there is a vertex v such that $E(v) \subseteq E_a$, then either a falsifies $\chi_v(G, c)$, in which case the empty clause is in $T(G_a, c_a)$, or a satisfies $\chi_v(G, c)$, in which case removing v from G_a has no impact on the above equality. A particularly interesting case occurs when G is connected and E_a is an edge separator of G.

Lemma 11. *Let G be a connected graph and let $T(G, c)$ be a Tseitin formula. Let (A, B) be a partition of $V(G)$ such that both $G[A]$ and $G[B]$ are connected. Let a be an assignment to $\{x_e \mid e \in E(A, B)\}$. Then there are two charge functions $c_a^A : A \to \{0, 1\}$ and $c_a^B : B \to \{0, 1\}$ such that*

$$T(G, c)|a = T(G[A], c_a^A) \wedge T(G[B], c_a^B)$$

If $T(G,c)$ is unsatisfiable, then either $T(G[A], c_a^A)$ or $T(G[B], c_a^B)$ is unsatisfiable, but not both. Moreover the parity of a characterizes which of the two subformulas is satisfiable, i.e., there is $\Gamma \in \{A, B\}$ such that $T(G[\Gamma], c_a^\Gamma)$ is satisfiable if and only if a assigns an odd number of variables to 1.

Given a Tseitin formula $T(G,c)$, the primal graph of the formula is easily obtained from G. One just has to start from isolated vertices corresponding to $var(T(G,c))$, and for each vertex $v \in V(G)$, to connect the vertices corresponding to the variables $var(\chi_v)$ in a clique. Thus there is the following relation between the primal treewidth of $T(G,c)$ and $tw(G)$.

Lemma 12. *Let $T(G,c)$ be a Tseitin formula whose underlying graph G has maximum degree bounded by Δ, then the primal treewidth of $T(G,c)$ is at most $tw(G) \times \Delta$.*

Proof. We use a tree decomposition of G to construct a tree decomposition of the primal graph G_p of $T(G,c)$.

Let T be a tree decomposition of G. For $v \in V(G)$ recall that $|var(\chi_v)| = |E(v)| \leq \Delta$ and that the CNF encoding F_v of χ_v is made of $2^{|var(\chi_v)|-1}$ clauses that contain each all variables of χ_v. We define $B'_t := \bigcup_{v \in B_t} var(\chi_v)$. We claim that the tree T' which is a clone of T except that every node t associated with B_t in T is now associated with B'_t in T', is a tree decomposition of G_p.

First, we have that $\bigcup_{t \in T} B'_t = \bigcup_{v \in V(G)} var(\chi_v) = var(T(G,c)) = V(G_p)$.

Second, let x and y be variables connected by an edge in G_p. Then there is a clause C such that $x, y \in var(C)$. C belongs to $clause(F_v)$ for some vertex $v \in V(G)$. But then $var(C) = var(\chi_v)$ and x and y appear together in any B'_t such that $v \in B_t$.

Third, we show that for every x, the subgraph of T' restricted to nodes t such that $x \in B'_t$ is connected. Recall that x corresponds to an edge $uv \in E(G)$ and that the constraints whose sets of variable contain x are exactly χ_u and χ_v. So $\{t \mid x \in B'_t\} = \{t \mid u \in B_t\} \cup \{t \mid v \in B_t\}$. Since T is a tree decomposition of T, restricting T to nodes $\{t \mid u \in B_t\}$ yields a connected subtree T_u. And restricting T to nodes $\{t \mid v \in B_t\}$ yields another connected subtree T_v. Moreover, since $uv \in E(G)$, the intersection $\{t \mid u \in B_t\} \cap \{t \mid v \in B_t\}$ is not empty. So $T_{u,v}$ obtained T by restricting to nodes $\{t \mid u \in B_t\} \cup \{t \mid v \in B_t\}$ is a connected subtree of T. Restricting T' to the nodes $\{t \mid x \in B'_t\}$ yields a connected subtree isomorphic to $T_{u,v}$.

So T' is a tree decomposition of G_p, and by construction its width is

$$\max_{t \in T} |B'_t| \leq \max_{t \in T} \sum_{v \in B_t} |var(\chi_v)| \leq \max_{t \in T} |B_t| \times \Delta$$

□

Part I

Lower Bounds on the Size of Representations in Compilation Languages

Chapter 1

Lower Bounds on the DNNF Size of Specific Functions

A complete language L in which barely any function has a small representation is intuitively a poor choice of compilation language, even if it supports many transformations and queries in polynomial-time. Thus the existence of classes of "interesting" functions whose L size is polynomial in the number of variables is an argument to champion L as a compilation language. Classes of functions whose L size is not polynomial are equally important as they allow us to compare L to other languages in terms of succinctness. In this chapter we present new lower bounds on the DNNF size of specific Boolean functions. We will first show that the DNNF size of functions representing pseudo-Boolean inequalities can be exponential in the number of variables, thus showing an exponential separation between the language DNNF and the (incomplete) language of pseudo-Boolean inequalities. In a second contribution we show that the DNNF size of satisfiable Tseitin formulas – CNF formulas representing systems of parity constraints structured by a graph – is exponential in the treewidth of the underlying graph when the maximum degree of the graph is bounded by a constant. This lower bound is the basis of other contributions of this thesis shown in later chapters.

We begin this chapter by a section where we review what is known on compilation to DNNF and also position our contributions vis-à-vis the state-of-the-art. In the second section we prove our first contribution on the exponential separation between DNNF and the language of pseudo-Boolean inequalities. In the third and last section, we prove our parameterized lower bound on the DNNF size of Tseitin formulas.

1.1 The State of Compilation to DNNF and sublanguages

The oldest negative result regarding compilation to DNNF indirectly appears in the seminal work of Selman and Kautz on knowledge compilation [SK91, SK96]. They show that there exist no languages that both support polynomial-time clausal entailment and allow polynomial-size compilation of Boolean formulas unless $\mathsf{NP} = \mathsf{P/poly}$ (consequently, unless the polynomial hierarchy collapses at the second level). This applies to DNNF as the language supports linear-time clausal entailment. Since then, *unconditional* separations have been shown for the compilation from several languages to DNNF, including an exponential separation between monotone 2-CNF formulas and circuits in DNNF [BCMS14, Cap16]. Nevertheless, the many interesting properties of DNNF and of its sublanguages encourage the knowledge compilation community to develop compilers to DNNF (and to its sublanguages), see for instance [Dar02a, HD04, PD10, OD15, LM17]. The general idea is that the benefits of compiling are such that, one may as well try to compile a given instance with the hope that a compiled form of reason-

able size exists (and sometimes it does!). In parallel to the development of new compilers, another line of research aims to characterize Boolean functions that have small DNNF size, or small L size, for L a sublanguage of DNNF. A first objective of this research is to determine parameters of Boolean functions that can be used in lower bounds and (or) upper bounds on the L size. In a slightly different direction, one can focus on particular classes of Boolean functions and formulas and identify, first whether all functions of the class have polynomial-size compiled form in L, and second what parameters characterize the L size of these functions (so just like before, but restricted to particular classes of functions). We briefly summarize the progress that has been made in these two directions. But first, we recall the upper bound on the DNNF size of CNF formulas that has fuelled a lot of the research on the DNNF size of general and selected Boolean functions.

DNNF size of CNF formulas and treewidth. The language DNNF has been introduced by Darwiche [Dar01a, Dar01b] along with algorithms to compile CNF formulas to DNNF. One of the arguments invoked by Darwiche to champion DNNF as a target language for knowledge compilation was that every CNF formula whose primal treewidth is bounded can be compiled in polynomial time in DNNF. More precisely, given a CNF formula ϕ and a dtree (decomposition tree) of ϕ of width w, there is an algorithm that compiles ϕ in DNNF that takes $O(|var(\phi)|w2^w)$ time and space. A dtree of ϕ is a rooted binary tree whose leaves are in bijection with the clauses of ϕ and whose width, whose formal definition we omit, essentially measures how well the bipartitions of the clauses of ϕ induced at the dtree nodes separate the variables. Given a tree decomposition T of the primal graph of ϕ, one can construct in linear time a dtree whose width is at most the width of T. Since finding a tree decomposition of a graph is fixed-parameter tractable in the treewidth of the graph [Bod96], it follows that compiling a CNF formula to DNNF is also fixed-parameter tractable in the primal treewidth of the formula. In [Dar01b], Darwiche introduced the d-DNNF language and show that the compilation algorithms in [Dar01a] actually generate circuits in d-DNNF, so the upper bound holds for the d-DNNF language as well. With the introduction of new sublanguages of d-DNNF like SDD, it was shown that for every CNF formula ϕ there exists an SDD of size $O(|var(\phi)|2^{tw_p(\phi)})$ that computes ϕ, where $tw_p(\phi)$ is the primal treewidth of ϕ [Dar11]. As a side note, if we restrict the target language to OBDD – which is a sublanguage of SDD – then there exists an OBDD equivalent to ϕ of size $O(|var(\phi)|2^{pw_p(\phi)})$, where $pw_p(\phi)$ is the primal pathwidth of ϕ. Compared to the upper bound for SDDs, it is known that $pw(G) = O(\log(|V(G)|)tw(G))$ for every graph G [Bod98], so we have $O(|var(\phi)|2^{pw_p(\phi)}) = O(|var(\phi)|^{tw_p(\phi)+1})$.

DNNF size of every Boolean functions. Since every Boolean function over a set of variables X can be represented by a CNF formula over X, the SDD size of any function is at most exponential in the smallest primal treewidth of a CNF formula representing the function. This upper bound is loose. For instance $\phi = x_1 \vee \cdots \vee x_n$ has primal treewidth $n-1$ but is trivial to represent by small circuits in DNNF or d-DNNF, and by small SDDs and OBDDs. To remedy this problem, one can modify the upper bound so that it holds for incidence treewidth, but the resulting upper bound would still be too loose as there are CNF formulas that have large incidence treewidth but small DNNF size (unsatisfiable formulas for instance). In the same vein, other width measures for CNF formulas that generalizes treewidth, such as *subterm width* [BS17b] or *PS-width* [STV14, Cap16] have been introduced to find upper bound on the L size, for different sublanguages L of DNNF, where the dependence in the measure is again at most single exponential (so in 2^w where a is the measure). To avoid the requirement that the function is given as a CNF formula, we turn to the notion of *expression treewidth (resp. pathwidth)* introduced by Jha and Suciu [JS12], and later called *circuit treewidth (resp. pathwidth)* [RP13, BS17a]. The circuit treewidth $ctw(f)$ (resp. pathwidth $cpw(f)$) of a Boolean function f is the minimum treewidth (resp. pathwidth) over all Boolean circuits that compute f using only binary nodes and unary $\neg$-nodes, and where each variables labels at most one leaf. By treewidth (resp. pathwidth) of a Boolean circuit, we mean the

treewidth (resp. pathwidth) of its underlying DAG. The circuit treewidth (resp. pathwidth) of a CNF formula is upper-bounded by its incidence treewidth (resp. pathwidth), and thus by its primal treewidth (resp. pathwidth). Moreover the circuit treewidth (resp. pathwidth) of an unsatisfiable formula is trivially 1. Jha and Suciu showed that the OBDD size of a Boolean function f on n variables is bounded by $n^{O(g(ctw(f)))}$ and by $O(g(cpw(f))poly(n))$ where $g(k)$ is doubly exponential in k. The upper bound for circuit treewidth is completed by an $n^{\Omega(ctw(f))}$ lower bound by Razgon [Raz14]. As for circuit pathwidth, Jha and Suciu showed that OBDDs of width k compute functions of circuit pathwidth $O(k)$, thus proving that a function has OBDD size linear in its number of variables if and only if its circuit pathwidth is bounded by a constant. Bova and Szeider have shown that a similar connection exists between the SDD size and circuit treewidth: a function has SDD size linear in its number of variables if and only if its circuit treewidth is bounded by a constant [BS17a]. More formally they show that the SDD size of a Boolean function f on n variables is at most $O(g(ctw(f))poly(n))$ where $g(k)$ is triple exponential in k (so in $2^{2^{2^k}}$), and that every Boolean function of SDD width k has circuit treewidth $O(k)$. The triple exponential dependence on circuit treewidth in the upper bound could be quite loose. Amarilli et al. have shown that the d-str-DNNF size of f is $O(g(ctw(f))poly(n))$ where f is *single* exponential and d-str-DNNF $=$ str-DNNF $\cap$ d-DNNF is a language that contains SDD, so the upper bound for SDD size can perhaps be improved [ACMS20].

DNNF size of selected Boolean functions. Research has also focused on particular classes of formulas, not necessarily complete, to prove sharp lower bounds on their L size. This is the case for several classes of "graph-based" CNF formulas. We can define such formulas as being (almost) completely determined by one of their underlying graphs (primal graph, incidence graph, etc.) or hypergraphs. For instance, Bova and Slivovsky [BS17b] and Razgon [Raz21] looked at the class of monotone 2-CNF formulas, that are completely determined by their underlying hypergraphs (which are actually graphs). Bova and Slivosky show that the OBDD size of such formulas is at least single exponential in their subterm width, while Razgon shows that it is at least single exponential in another parameter that he called *linear upper maximum induced matching width*. Instead of introducing new width parameters, Amarilli et al. proved several lower bounds on the L size of general monotone CNF formulas, for different sublanguages L of DNNF, using only the treewidth, the degree and the arity of their underlying hypergraph [ACMS20]. Their result matches the upper bound of [Dar11] when the arity and the degree are bounded by a constant. For other classes of CNF formulas, positive compilability results could even be found. Of course by [Dar11] we already have one such class with CNF formulas of bounded treewidth. But we can give other examples! For instance the class of CNF formulas whose underlying hypergraph is β-acyclic has polynomial size compilation to dec-DNNF [Cap16] – a language that was identified by Huang and Darwiche in [HD07] – and the class of OCNF (ordered CNF) and the more general class of variable-convex CNF formulas have polynomial-time compilations to OBDD [CBLM05, BS17b].

Contributions. We show new exponential lower bounds on the DNNF size of specific Boolean functions. First we consider pseudo-Boolean constraints in the form $w_1x_1 + \cdots + w_nx_n \geq \theta$ where $x_1, \ldots, x_n$ are $0/1$ Boolean variables, $w_1, \ldots, w_n$ are real-valued weights and θ is a real-valued threshold. We find a class of pseudo-Boolean constraints whose DNNF size is exponential in $\sqrt{n}$. This result generalizes a result of Hosaka et al. [HTKY97] on the OBDD size of specific pseudo-Boolean constraints (since circuits in DNNF can be exponentially smaller than equivalent OBDDs but not the converse), and a result of Le Berre et al. [LMMW18] on sets of pseudo-Boolean constraints such that the whole *set* (so a conjunction of constraints) has exponential DNNF size. We note that the pseudo-Boolean constraints used in our results do not seem easy to represent as CNF formulas without additional encoding variables, so our result is one of the few on the compilation to DNNF of functions not represented as CNF formulas.

This first contribution has been presented by the author in [dC20]. Our second contribution concerns specific formulas called Tseitin formulas. These formulas have been introduced in the preliminaries, we simply recall here that they represent systems of parity constraints structured by simple undirected graphs (the edges are the variables and each vertex corresponds to a parity constraints). We show that satisfiable Tseitin formulas whose graphs have bounded degree have DNNF size polynomial in n, the number of variables, if and only if the treewidth of the graph is at most $O(\log(n))$. Later in this thesis we will show two applications of this result. For now we only mention that our result contrasts with the state-of-the-art in that it is, to the best of our knowledge, the only characterization of the DNNF size of non-monotone graph-based CNF formulas. This second contribution appears in the paper [dCM21a] co-authored with Stefan Mengel.

1.2 Pseudo-Boolean Constraints that have Exponential DNNF Size

Many types of constraints are easily compilable to DNNF and to its sublanguages. For instance it is easy to find an OBDD computing a clause, a term, a parity constraint (i.e., a constraint of the form $x_1 + \cdots + x_n = 0 \mod 2$ or $x_1 + \cdots + x_n = 1 \mod 2$) or a cardinality constraint (i.e., a constraint of the form $x_1 + \cdots + x_n \geq k$). One is often interested in compiling Boolean functions given as systems of constraints, like CNF formulas, that are systems of clauses. But in some situations, one needs to compile constraints of the system individually. One setting where this happens is when one tries to compile a system of constraint in a bottom-up manner, that is, one first compiles each constraint individually in the target language – in practice the languages SDD or OBDD – and successively aggregates the compiled form of the constraints by means of a procedure called the `Apply` procedure, until computing a circuit in the target language that computes the whole system, see Example 22 in Chapter of a bottom-up compilation of a CNF formula to OBDD. Another situation that involves compiling single constraints arises when trying to solve a constraint satisfaction problem by translating the constraints into a CNF formula and feeding it to a SAT solver. Two major considerations here are the size of the CNF encoding and its propagation strength. One wants, on the one hand, to avoid the size of the encoding to explode, and on the other hand, to guarantee a good behaviour of the SAT instance under unit propagation – a technique at the very core of SAT solving. Desired propagation strength properties are, for instance, generalized arc consistency (GAC) [Bac07] or propagation completeness (PC) [BM12]. Several encodings follow the same two-step method: first, each constraint is compiled to BDD or DNNF. Second, the compiled forms are turned into CNF formulas using different transformations depending on the propagation strength wanted for the formula [AGMS16, KS19]. For both bottom-up compilation and for obtaining good constraint encodings, it is key that the size of the circuit in DNNF representing the constraint be of reasonable size. In this section we show that it is not the case for pseudo-Boolean constraints. It is worth mentioning that there are GAC encodings of pseudo-Boolean constraints into polynomial size CNFs that do not follow the two-step method [BBR09]. However no similar result is known for PC encodings. PC encodings are more restrictive that GAC encodings and may be obtained via techniques requiring compilation to DNNF [KS19].

Pseudo-Boolean (PB) constraints are inequalities the form $\sum_{i=1}^{n} w_i x_i \ \mathtt{op}\ \theta$ where the x_i are $0/1$ Boolean variables, the w_i and θ are integers, and $\mathtt{op}$ is one of the comparison operator $<$, $\leq$, $>$ or $\geq$. PB-constraints have been studied extensively under different names (e.g. threshold functions [HTKY97], knapsack constraints [GKM+11]) and occur in many problems in AI [Ivă65, Pap77, BHMR99, BLS02, ARMS02]. A PB-constraint is associated with a Boolean function whose satisfying assignments are exactly the assignments to $\{x_1, \ldots, x_n\}$ that satisfy the inequality. For simplicity we directly consider PB-constraints as Boolean functions – although the same function may be represented by different constraints, for instance $x_1 + x_2 \geq 1$ and $2x_1 + x_2 \geq 1$ are equivalent – while keeping the term "constraints"

when referring to them. We restrict our attention to PB-constraints where $\mathtt{op}$ is $\geq$ and all weights are positive integers. Note that with this choice of operator and the restriction to positive integers, PB-constraints are monotone Boolean functions. Given a sequence of positive integer weights $W = (w_1, \ldots, w_n)$ and an integer threshold θ, we define the function $w : \{0,1\}^X \to \mathbb{N}$ that maps any assignment to its weight by $w(a) = \sum_{i=1}^{n} w_i a(x_i)$. With this notation, a PB-constraint over X for a given pair (W, θ) is a Boolean function whose satisfying assignments are exactly the a such that $w(a) \geq \theta$.

Example 11. Let $n = 5$, $W = (1, 2, 3, 4, 5)$ and $\theta = 9$. The PB-constraint for (W, θ) is the Boolean function whose models are the assignments such that $x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \geq 9$. The assignment defined by $\{\overline{x_1}, x_2, x_3, \overline{x_4}, x_5\}$ (so the assignment a such that $a(x_1) = a(x_4) = 0$ and $a(x_2) = a(x_3) = a(x_5) = 1$), is a satisfying assignment of the constraint. Its weight is $w(a) = 2 + 3 + 5 = 10$. ◀

The main result of this section is the following theorem.

Theorem 4. *There is a class of PB-constraints $\mathcal{F}$ such that for any constraint $f \in \mathcal{F}$ on n^2 variables, any circuit in DNNF computing f has size $2^{\Omega(n)}$.*

Theorem 4 generalises a similar result written in [HTKY97, ANO+12] for OBDDs. Since there exist infinitely many functions that have polynomial DNNF size but exponential OBDD size (in the number of variables), our result is strictly stronger. The class $\mathcal{F}$ is similar to that used in [HTKY97, ANO+12], actually the only difference is the choice of the threshold for the PB-constraints. Yet, adapting the proofs given in [HTKY97, ANO+12] for OBDDs to circuits in DNNF is not straightforward, thus our proof of Theorem 4 bears very little resemblance. Our result it also related to what has been done in [LMMW18] where it is shown that there exist sets of PB-constraints such that the whole *set* (so a conjunction of PB-constraints) is computed only by circuits in DNNF of exponential size. Our result is a generalisation to *single* PB-constraints.

1.2.1 Restriction to Threshold Models of PB-Constraints

The strategy to prove Theorem 4 is to find a PB-constraint f over n variables such that $R(f)$ is exponential in $\sqrt{n}$ and then use Theorem 1. We first show that we can restrict our attention to covering particular models of f with rectangles rather than the whole function. In this section X is a set of n Boolean variables and f is a PB-constraint over X. Recall that we only consider constraints of the form $\sum_{i=1}^{n} w_i x_i \geq \theta$ where the w_i and θ are *positive* integers.

Definition 26. *Let f be a PB constraint $\sum_{i=1}^{n} w_i x_i \geq \theta$ over $X = \{x_1, \ldots, x_n\}$. The* threshold models *of f are the assignments a to X such that $w(a) = \theta$.*

Threshold models should not be confused with minimal models.

Definition 27. *A* minimal model *of f is a model a of f such that no assignment a' to $var(f)$ such that $a' < a$ satisfies f.*

For a monotone PB-constraint, minimal models are the assignments whose weights are above or equal to the threshold but drop below the threshold when re-assigning any variable from 1 to 0. Any threshold model is a minimal model, but not all minimal models are threshold models. There even exist constraints with no threshold models (for instance when the weights are even integers but the threshold is an odd integer) while there always are minimal models for satisfiable constraints.

Example 12. The minimal models of the PB-constraint from Example 11 are $\{\overline{x_1}, \overline{x_2}, \overline{x_3}, x_4, x_5\}$, $\{\overline{x_1}, x_2, x_3, x_4, \overline{x_5}\}$, $\{x_1, \overline{x_2}, x_3, \overline{x_4}, x_5\}$ and $\{\overline{x_1}, x_2, x_3, \overline{x_4}, x_5)$. The first three are threshold models. ◀

Lemma 13. *Let f be a PB-constraint whose weights are positive and let f^* be the Boolean function whose models are exactly the threshold models of f. Then $R(f) \geq R(f^*)$.*

Proof. Let $X = var(f)$ and let $r = \rho_1 \wedge \rho_2$ with respect to a balanced partition $\Pi = (X_1, X_2)$ of X such that $r \leq f$. Assume that r accepts some threshold models. For a an assignment to X, we denote by a_1 (resp. a_2) its restriction to X_1 (resp. X_2). We claim that there exist two integers θ_1 and θ_2 such that $\theta_1 + \theta_2 = \theta$ and, for any threshold model a in r, we have $w(a_1) = \theta_1$ and $w(a_2) = \theta_2$. To see this, assume by contradiction that there exists $(\theta'_1, \theta'_2) \neq (\theta_1, \theta_2)$ such that $\theta = \theta'_1 + \theta'_2$ such that some other threshold model b with $w(b_1) = \theta'_1$ and $w(b_2) = \theta'_2$ is accepted by r. Then either $w(a_1) + w(b_2) < \theta$ or $w(b_1) + w(a_2) < \theta$, but since $a_1 \cup b_2$ and $b_1 \cup a_2$ are also accepted by r, the rectangle r would accept assignments falsifying f, which is forbidden. Now let ρ_1^* (resp. ρ_2^*) be the Boolean function over X_1 (resp. X_2) whose satisfying assignments are exactly the satisfying assignments of ρ_1 (resp. ρ_2) of weight θ_1 (resp. θ_2). Then $r^* = \rho_1^* \wedge \rho_2^*$ is a rectangle with respect to Π that accepts exactly the threshold models in r.

Now consider a balanced rectangle cover of f of size $R(f)$. For each rectangle r of the cover, if r contains no threshold model then discard it, otherwise construct r^*. The disjunction of these new rectangles is a balanced rectangle cover of f^* of size at most $R(f)$. Therefore $R(f) \geq R(f^*)$. □

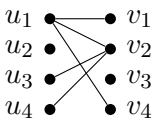
1.2.2 Reduction to Covering Maximal Matchings of $K_{n,n}$

We define the class of hard PB-constraints for Theorem 4 in this section. We will show, using Lemma 13, that the problem can be reduced to that of covering all maximal matchings of the complete $n \times n$ bipartite graph $K_{n,n}$ with rectangles. In this section, X is a set of n^2 Boolean variables. To simplify the presentation, assignments to X are written as $n \times n$ matrices. Each variable $x_{i,j}$ has the weight $w_{i,j} = (2^i + 2^{j+n})/2$. Define the matrix of weights $W = (w_{i,j} : 1 \leq i, j \leq n)$ and the threshold $\theta = 2^{2n} - 1$. The PB-constraint f for the pair (W, θ) is such that $f(a) = 1$ if and only if a satisfies

$$\sum_{1 \leq i,j \leq n} \left(\frac{2^i + 2^{j+n}}{2} \right) x_{i,j} \geq 2^{2n} - 1 \tag{PB1}$$

Constraints of this form constitute the class of hard constraints of Theorem 4. One may find it easier to picture f writing the weights and threshold as binary numbers over $2n$ bits. Bits of indices 1 to n form the *lower part* of the number and those of indices $n+1$ to $2n$ form the *upper part*. The weight $w_{i,j}$ is the binary number where the only bits set to 1 are the ith bit of the lower part and the jth bit of the upper part. Thus when a variable $x_{i,j}$ is set to 1, exactly one bit of value 1 is added to each part of the binary number of the sum.

Assignments to X uniquely encode subgraphs of $K_{n,n}$. We call $U = \{u_1, \ldots, u_n\}$ the set of nodes on the left side of $K_{n,n}$ and $V = \{v_1, \ldots, v_n\}$ the set of nodes on the right side of $K_{n,n}$. The bipartite graph encoded by a is such that there is an edge between the u_i and v_j if and only if $a(x_{i,j}) = 1$.

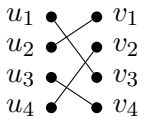
Example 13. Take $n = 4$. The assignment $a = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$ encodes 

◀

Definition 28. *A* maximal matching assignment *is an assignment a to $X = \{x_{i,j} \mid 1 \leq i, j \leq n\}$ such that*

- *for any $i \in [n]$, there is exactly one k such that $a(x_{i,k}) = 1$*
- *for any $j \in [n]$, there is exactly one k such that $a(x_{k,j}) = 1$*

As the name suggests, the maximal matching assignments are those encoding bipartite graphs whose edges form a maximal matching of $K_{n,n}$ (i.e., a maximum cardinality matching). One can also see them as encodings for permutations of $[n]$.

Example 14. $a = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$ is a maximal matching assignment that encodes 

◀

For an assignment a, let $var_k(a)$ be defined as

- $var_k(a) = \{j \mid a(x_{k,j}) = 1\}$ when $1 \leq k \leq n$
- $var_k(a) = \{i \mid a(x_{i,k-n}) = 1\}$ when $n+1 \leq k \leq 2n$.

$var_k(a)$ stores the index of variables in a that directly add 1 to the kth bit of $w(a)$. Note that a maximal matching assignment is an assignment a such that $|var_k(a)| = 1$ for all k. It is easy to see that maximal matching assignments are threshold models of f: seeing weights as binary numbers of $2n$ bits, for every bit of the resulting sum of weights, there is exactly one weight that has value 1 at that bit, so exactly the first $2n$ bits of the sum are set to 1, which gives us that the decimal value of the sum is θ. We note that not all threshold models of f are maximal matching assignments.

Example 15. Consider the assignment a represented in Example 13. a does not encode a maximal matching and its weight for the PB-constraint (PB1) with $n = 4$ is: $\frac{2+2^{n+1}}{2} + \frac{2+2^{n+2}}{2} + \frac{2+2^{n+4}}{2} + \frac{2^3+2^{n+2}}{2} + \frac{2^4+2^{n+2}}{2} = 17 + 33 + 129 + 36 + 40 = 255$. Since the threshold of (PB1) is $2^{2n} - 1 = 2^8 - 1 = 255$, the assignment a is a threshold model. ◀

Lemma 14. *Let $\Pi = (X_1, X_2)$ be a partition of X. Let a and b be maximal matching assignments and, for $i \in \{1, 2\}$, denote by a_i (resp. b_i) the restriction of a (resp. b) to X_i. If $a_1 \cup b_2$ and $b_1 \cup a_2$ both have weight $\theta = 2^{2n} - 1$ then both are maximal matching assignments.*

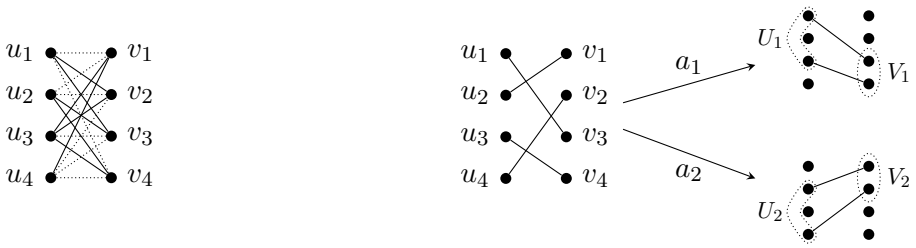
Proof. It is sufficient to show that $|var_k(a_1 \cup b_2)| = 1$ and $|var_k(b_1 \cup a_2)| = 1$ for all $1 \leq k \leq 2n$. We prove it for $a_1 \cup b_2$ by induction on k. First observe that since $|var_k(a)| = 1$ and $|var_k(b)| = 1$ for all $1 \leq k \leq 2n$, the only possibilities for $|var_k(a_1 \cup b_2)| = 1$ are 0, 1 or 2.

- For the base case $k = 1$, if $|var_k(a_1 \cup b_2)|$ is even then the first bit of $w(a_1) + w(b_2)$ is 0 and the weight of $a_1 \cup b_2$ is not θ. So $|var_1(a_1 \cup b_2)| = 1$.
- For the general case $1 < k \leq 2n$, assume it holds that $|var_1(a_1 \cup b_2)| = \cdots = |var_{k-1}(a_1 \cup b_2)| = 1$. So the kth bit of $w(a_1) + w(b_2)$ depends only on the parity of $|var_k(a_1 \cup b_2)|$: the kth bit is 0 if $|var_k(a_1 \cup b_2)|$ is even and it is 1 otherwise. $a_1 \cup b_2$ has weight θ so $|var_k(a_1 \cup b_2)| = 1$.

The argument applies to $b_1 \cup a_2$ analogously. □

Lemma 15. *Let f be the PB-constraint (PB1) and let $\hat{f}$ be the function whose satisfying assignments are exactly the maximal matching assignments. Then $R(f) \geq R(\hat{f})$.*

Proof. Let f^* be the function whose satisfying assignments are the threshold models of f. By Lemma 13, it is sufficient to prove that $R(f^*) \geq R(\hat{f})$. We already know that $\hat{f} \leq f^*$. Let $r = \rho_1 \wedge \rho_2$ be a rectangle with respect to the balanced partition $\Pi = (X_1, X_2)$ such that $r \leq f^*$, and assume that r accepts some maximal matching assignment. Let $\hat{\rho}_1$ (resp. $\hat{\rho}_2$) be the Boolean function over X_1 (resp. X_2) whose satisfying assignments are the a_1 (resp. a_2) such that there is a maximal matching assignment $a_1 \cup a_2$ accepted by r. We claim that the rectangle $\hat{r} = \hat{\rho}_1 \wedge \hat{\rho}_2$ contains exactly the maximal matching



(a) Balanced partition Π of $K_{4,4}$

(b) Partition of a maximal matching w.r.t. Π

Figure 1.1: Partition of maximal matching

assignments in r. On the one hand, it is clear that all maximal matching assignments accepted by r are also accepted by $\hat{r}$. On the other hand, $\hat{r}$ contains only threshold models of f of the form $a_1 \cup b_2$, where $a_1 \cup a_2$ and $b_1 \cup b_2$ encode maximal matchings, so by Lemma 14, $\hat{r}$ contains only maximal matching assignments in r.

Now consider a balanced rectangle cover of f^* of size $R(f^*)$. For each rectangle r of the cover, if r contains no maximal matching assignment then discard it, otherwise construct $\hat{r}$. The disjunction of these new rectangles is a balanced rectangle cover of $\hat{f}$ of size at most $R(f^*)$. Therefore $R(f^*) \geq R(\hat{f})$. □

1.2.3 Proof of Theorem 4

We now prove Theorem 4

Theorem 4. *There is a class of PB-constraints $\mathcal{F}$ such that for any constraint $f \in \mathcal{F}$ on n^2 variables, any circuit in DNNF computing f has size $2^{\Omega(n)}$.*

$\mathcal{F}$ is the class of constraints of the form (PB1). Thanks to Theorem 1 and Lemma 15, the proof boils down to finding exponential lower bounds on $R(\hat{f})$, where $\hat{f}$ is the Boolean function on n^2 variables whose models encode exactly the maximal matchings of $K_{n,n}$ (or equivalently, the permutations of $[n]$). $\hat{f}$ has $n!$ models. The idea is now to prove that rectangles covering $\hat{f}$ must be relatively small, so that covering the whole function takes many rectangles.

Lemma 16. *Let $\Pi = (X_1, X_2)$ be a balanced partition of X and let $\alpha = \sqrt{2/3}$. Let r be a rectangle with respect to Π with $r \leq \hat{f}$. Then $|r^{-1}(1)| \leq \frac{n!}{\binom{n}{\alpha n}}$.*

The function $\hat{f}$ has already been studied extensively in the literature, often under the name PERM_n (for *permutations on* $[n]$), see for instance [Weg00, Chapter 4] or [MW19, Section 6.2] where a statement similar to Lemma 16 is established. With Lemma 16 we can give the proof of Theorem 4.

Theorem 4. Let $\bigvee_{k=1}^{R(\hat{f})} r_k$ be a balanced rectangle cover of $\hat{f}$. We have $\sum_{k=1}^{R(\hat{f})} |r_k^{-1}(1)| \geq |sat(\hat{f})| = n!$. Lemma 16 gives us $R(\hat{f}) \frac{n!}{\binom{n}{\alpha n}} \geq n!$, thus

$$R(\hat{f}) \geq \binom{n}{\alpha n} \geq \left(\frac{n}{\alpha n}\right)^{\alpha n} = \left(\frac{3}{2}\right)^{\frac{\alpha n}{2}} \geq 2^{\frac{\alpha n}{4}} = 2^{\Omega(n)}$$

where we have used $\binom{a}{b} \geq (a/b)^b$ and $3/2 \geq \sqrt{2}$. Using Lemma 15 we get that $R(f) \geq R(\hat{f}) \geq 2^{\Omega(n)}$. Theorem 1 allows us to conclude. □

It only remains to prove Lemma 16.

Lemma 16. Let $r = \rho_1 \wedge \rho_2$ and $\Pi = (X_1, X_2)$. Recall that $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$ are the nodes on the left and right part of $K_{n,n}$ respectively. Define

$$U_1 = \{u_i \mid \text{there exists } x_{i,l} \in X_1 \text{ such that } a(x_{i,l}) = 1 \text{ for some } a \in \rho_1^{-1}(1)\}$$
$$V_1 = \{v_j \mid \text{there exists } x_{l,j} \in X_1 \text{ such that } a(x_{l,j}) = 1 \text{ for some } a \in \rho_1^{-1}(1)\}$$

Define U_2 and V_2 analogously (this time using X_2 and ρ_2). Figure 1.1 illustrates the construction of these sets: Figure 1.1a shows a partition Π of the edges of $K_{4,4}$ (full edges in X_1, dotted edges in X_2) and Figure 1.1b shows the contribution of an assignment accepted by r to U_1, V_1, U_2, and V_2 after partition according to Π.

Assignments in $\rho_1^{-1}(1)$ encode matchings of $K_{n,n}$, more precisely each such assignment encodes a matching between U_1 and V_1. We claim that such matchings between U_1 and V_1 are maximal. To verify this, observe that $U_1 \cap U_2 = \emptyset$ and $V_1 \cap V_2 = \emptyset$ since otherwise r accepts an assignment that does not correspond to a matching. Thus if ρ_1 were satisfied by a non-maximal matching between U_1 and V_1 then r would accept a non-maximal matching between U and V. So ρ_1 is satisfied only by maximal matchings between U_1 and V_1, consequently $|U_1| = |V_1|$. The argument applies symmetrically for V_2 and U_2. We note $k = |U_1|$. It stands that $U_1 \cup U_2 = U$ and $V_1 \cup V_2 = V$ as otherwise r accepts matchings that are not maximal. So $|U_2| = |V_2| = n - k$. We now have $|\rho_1^{-1}(1)| \leq k!$ and $|\rho_2^{-1}(1)| \leq (n-k)!$, leading to $|r^{-1}(1)| \leq k!(n-k)! = n!/\binom{n}{k}$.

Up to k^2 edges may be used to construct matchings between U_1 and V_1. Since R is balanced we obtain $k^2 \leq 2n^2/3$. Applying the same argument to U_2 and V_2 gives us $(n-k)^2 \leq 2n^2/3$, so $n(1 - \sqrt{2/3}) \leq k \leq n\sqrt{2/3}$. Finally, the function $k \mapsto n!/\binom{n}{k}$, when restricted to some interval $[\![n(1-\beta), \beta n]\!]$, reaches its maximum at $k = \beta n$, hence the upper bound $|r^{-1}(1)| \leq n!/\binom{n}{n\sqrt{2/3}}$. □

1.3 Tseitin Formulas that have Exponential DNNF Size

Going back to formulas and systems of constraints, in this section we study a class of "graph-based" CNF formulas and, contrary to PB-constraints, we give a characterization for when these formulas are computed by small circuits in DNNF. Graph-based formulas are intuitively almost entirely determined by their underlying graph (primal graph, hypergraph, or other). For example, monotone CNF formulas are uniquely defined by their hypergraphs so they qualify as graph-based formulas. The L size of graph-based CNF formulas, for a sublanguage L of DNNF, seems easier to analyse than general CNF formulas in that lower bounds that almost match the known upper bounds on the L size can be proved for them (see for instance [BCMS14, BS17b, ACMS20, Raz21]). In this section, we study a kind of graph-based CNF formulas introduced in the preliminaries: Tseitin formulas. We show that, when a Tseitin formula based on a graph G of bounded degree is satisfiable, its DNNF size is polynomial in its number n of variables if and only if $tw(G)/\Delta(G) = O(\log(n))$. Formally, our main result for this section is the following:

Theorem 5. *Let $T(G, c)$ be a satisfiable Tseitin formula where G is a connected graph with maximum degree at most Δ. Every smooth circuit in DNNF computing $T(G, c)$ has size at least $2^{\Omega(tw(G)/\Delta)}$.*

We compare Theorem 5 with existing work on the compilation of Tseitin formulas by Itsykson et al. [IRSS21, IRS22]. Before Theorem 5 was proved, Itsykson et al. had studied the compilation of Tseitin formulas to non-deterministic read-once branching programs (for short 1-NBP or nFBDD). They have introduced a new width measure for graphs called the *component width*, denoted by $compw(G)$, and proved that the nFBDD size for any satisfiable Tseitin formula $T(G, c)$ is between $2^{compw(G)}$ and $|E(G)| \times 2^{compw(G)} + 2$. They compared the component width of a graph to its treewidth and its

pathwidth and were able to show that $\frac{1}{2}(tw(G) - 1) \leq compw(G) \leq pw(G) + 1$ and that the two bounds are tight. On the one hand, DNNF $<$ nFBDD (see e.g. [ACMS20]) so our result is more general when the maximum degree of the graph $\Delta(G)$ is bounded by a constant. On the other hand, building upon the proof of Theorem 5, we can show the following variant:

Lemma 17. *Let $T(G, c)$ be a satisfiable Tseitin formula where G is a 3-connected graph with maximum degree at most Δ. Then every nFBDD computing $T(G, c)$ has size at least $2^{\Omega(pw(G)/\Delta)}$.*

One year after Theorem 5 was presented in [dCM21a], Itsykson, Riazanov and Smirnov have generalized it to the case of satisfiable Tseitin formulas for arbitrary graphs [IRS22], that is, they have successfully removed the dependence in the maximum degree of the graph:

Theorem 6. [IRS22, Theorem 4.1] *Let $T(G, c)$ be a satisfiable Tseitin formula. Every smooth circuit in DNNF computing $T(G, c)$ has size at least $2^{\Omega(tw(G))}$.*

The result of Istykson et al. is a very nice improvement of our. Their proof uses several elements of the proof of Theorem 5, including the adversarial multi-partition rectangle cover game that we introduce in the next section.

1.3.1 An Adversarial Rectangle Cover Game for Lower Bounds on DNNF Size

When trying to show parameterized lower bounds with Theorem 1, one often runs into the problem that it is somewhat inflexible: the partitions of the rectangles in covers have to be balanced, but in parameterized applications this is often undesirable. Instead, to show good lower bounds, one wants to be able to partition in places that allow to cut in complicated subparts of the problem. For instance this is the underlying technique in [Raz16]. To make this part of the lower bound proofs more explicit and the technique more reusable, we here introduce a refinement of Theorem 1.

We define the *adversarial multi-partition rectangle cover game* for a Boolean function f over variables X and a set $S \subseteq sat(f)$ to be played as follows: two players, the cover player Charlotte and her adversary Adam, construct in several rounds a set $\mathcal{R}$ of rectangles that cover the set S respecting f (that is, rectangles in $\mathcal{R}$ contain only models of f). The game starts with $\mathcal{R}$ as the empty set. Charlotte starts a round by choosing an assignment in S and vtree T of X. Now Adam chooses a partition of X induced by T, that is, a partition (X_1, X_2) such that $X_1 = var(T_v)$ for some node v in T. Charlotte ends the round by adding to $\mathcal{R}$ a combinatorial rectangle r with respect to this partition and such that $r \leq f$. The game is over when S is covered by $\mathcal{R}$.

Definition 29. *The adversarial multi-partition rectangle complexity of f and S, denoted by $aR(f, S)$ is the minimum number of rounds in which Charlotte can finish the adversarial multi-partition rectangle cover game, whatever the choices of Adam are. When $S = sat(f)$, we write $aR(f) = aR(f, sat(f))$.*

The linear adversarial multi-partition rectangle complexity $aR_\ell(f, S)$ of f and S is defined analogously with the difference that instead of a vtree Charlotte gives an order of X and Adam chooses (X_1, X_2) such that X_1 is a prefix of the order given by Charlotte. The following theorem gives the core technique for showing lower bounds later on.

Theorem 7. *Let D be a circuit in DNNF computing the function f and let $S \subseteq sat(f)$. Then $aR(f, S) \leq |D|$.*

Proof. Let $X = var(D) = var(f)$. We iteratively delete vertices from D and construct rectangles. The approach is as follows: Charlotte chooses an assignment $a \in S$ not yet in any rectangle she constructed before and a proof tree T of D satisfied by a. Now Adam chooses a partition (not necessarily balanced)

induced by T at a node v. Using Lemma 8, Charlotte chooses the rectangle $sat(D, v)$, deletes it from S and the game continues.

The vertex v in the above construction is different at every round of the game: by construction, Charlotte never chooses an assignment a that is in any set $sat(D, v)$ for a vertex v that has appeared before. Thus, no such v can appear in the proof tree of the chosen a. Consequently, a new vertex v is chosen for each assignment a that Charlotte chooses and thus the game will never last more than $|D|$ rounds. □

If the input is an FBDD instead of a circuit in DNNF, the proof trees chosen by Charlotte are linear, giving an order of X. Hence, we get the following corollary.

Corollary 1. *Let B be an FBDD computing a function f and let $S \subseteq sat(f)$. Then $aR_\ell(f, S) \leq |B|$.*

The adversarial multi-partition rectangle cover game can be modified so that the rectangle chosen by Charlotte at a given round is disjoint from the rectangles that she had already stored in $\mathcal{R}$. This variant of the game returns a set $\mathcal{R}$ containing pairwise disjoint rectangles.

Definition 30. *The adversarial multi-partition disjoint rectangle complexity of f and S, denoted by $aR_d(f, S)$ is the minimum number of rounds in which Charlotte can finish the adversarial multi-partition rectangle cover game by choosing rectangles that are pairwise disjoint, whatever the choices of Adam are. When $S = sat(f)$, we write $aR_d(f) = aR_d(f, sat(f))$.*

It is clear that $aR(f) \leq aR_d(f)$. Just like $aR(f)$ is a lower bound on the size of the smallest circuit in DNNF computing f, $aR_d(f)$ is a lower bound on the size of the smallest circuit in d-DNNF computing f. This latter bound is shown in the next theorem, using the techniques at work in [BCMS16] to prove Theorems 1 and 2.

Theorem 8. *Let D be a circuit in d-DNNF computing the function f and let $S \subseteq sat(f)$. Then $aR_d(f, S) \leq |D|$.*

Proof. Let $X = var(D) = var(f)$. Compared to the proof of Theorem 7, here we modify the circuit between two rounds. We denote by D^i the circuit in d-DNNF at round i for $i \geq 1$ with $D^1 = D$. Round 1 is played as before: Charlotte chooses an assignment $a \in S$ not yet in any rectangle she constructed before and a proof tree T^1 of D^1 satisfied by a. Then Adam chooses a partition induced by T^1 at a node v^1. Then Charlotte chooses the rectangle $sat_X(D^1, v^1)$ and deletes it from S. But now before the game continues, Charlotte removes v^1 from D^1 by disconnecting v^1 from its children, replacing v^1 by a constant 0, propagating that constant, and deleting all nodes that are not reachable from the root of D^1. The resulting circuit is called D^2 and the game continue.

Claim 1. *If D^i is a circuit in d-DNNF, then so is D^{i+1}.*

Proof. It is readily verified that for every node u in D^{i+1}, we have that $var(D^{i+1}_u) \subseteq var(D^i_u)$ so decomposability is preserved in D^{i+1}.

As for determinism, in a Boolean circuit C over X containing only literal inputs, constant inputs, $\vee$-nodes and $\wedge$-nodes, replacing a node by a 0-input yields a circuit C' such that $sat_X(C') \subseteq sat_X(C)$. Especially for every node u that is in both D^i and D^{i+1} we have $sat_X(D^{i+1}_u) \subseteq sat_X(D^i_u)$. If u is a $\vee$-node in D^{i+1} with children $u_1, \ldots, u_k$, then it is also a $\vee$-node in D^i and $u_1, \ldots, u_k$ are also children of u in D^i. By determinism in D^i we have $sat_X(D^i_{u_j}) \cap sat_X(D^i_{u_l}) = \emptyset$ for every $j \neq l$, so $sat_X(D^{i+1}_{u_j}) \cap sat_X(D^{i+1}_{u_l}) \subseteq sat_X(D^i_{u_j}) \cap sat_X(D^i_{u_l}) = \emptyset$. So determinism is preserved. □

Claim 2. $sat_X(D^{i+1}) = sat_X(D^i) \setminus sat_X(D^i, v^i)$.

Proof. Let C^i be the circuit obtained by disconnecting v_i from its children, replacing v^i by a fresh variable y, and deleting that are not reachable from the root. It is clear that C^i is a circuit in DNNF (but not in d-DNNF) and that $C^i|_{\overline{y}} \equiv D^{i+1}$. Now we look at what becomes of the proof trees of D^i during the transformation. Let $\mathcal{T}(D^i)$ be the set of proof trees of D^i and let $\mathcal{T}(C^i)$ be the set of proof trees of C^i. If $T \in \mathcal{T}(D^i)$ does not contain v^i, then T is in $\mathcal{T}(C^i)$. Otherwise if $T \in \mathcal{T}(D^i)$ contains v^i, then the proof tree T' obtained by replacing T_{v^i} by y is in $\mathcal{T}(C^i)$. It should be clear that they are not other proof trees in $\mathcal{T}(C^i)$. By Lemma 5 we have $C^i \equiv \bigvee_{T \in \mathcal{T}(C^i)} T$ and thus $D^{i+1} \equiv C^i|_{\overline{y}} \equiv \bigvee_{T \in \mathcal{T}(C^i) \cap \mathcal{T}(D^i)} T \equiv \bigvee_{T \in \mathcal{T}(D^i), T \not\ni v^i} T$. Hence $sat_X(D^{i+1}) = sat_X(D^i) \setminus sat_X(D^i, v^i)$. □

Claims 1 and 2 allow us to conclude that the game is sound and that Charlotte generates a set $\mathcal{R}$ of pairwise disjoint rectangles that cover S and respect f. □

Theorem 7 implies 1 and 8 implies 2.

1.3.2 Splitting Parity Constraints

In this section, we consider rectangles that respect a satisfiable Tseitin formula $T(G, c)$, that is, rectangles r over the same variables as $T(G, c)$ such that $r \leq T(G, c)$, or equivalently, such that $sat(r) \subseteq sat(T(G, c))$. Recall that $T(G, c)$ is a CNF formula that represents the conjunction of parity constraints $\bigwedge_{v \in V(G)} \chi_v$, where $\chi_v : \sum_{e \in E_G(v)} x_e = c(v) \mod 2$. We will see that such rectangles *split* the parity constraints of $T(G, c)$ in a certain sense and show how this is reflected in G. This will be crucial in proving the lower bound on the DNNF size in the next section with the adversarial multi-partition rectangle cover game.

Rectangles Induce Subconstraints for Tseitin formulas

Let r be a rectangle for the partition (E_1, E_2) of $E(G)$ such that $r \leq T(G, c)$. Assume that there is a vertex v of G incident to edges in E_1 and to edges in E_2, i.e., $E(v) = E_1(v) \cup E_2(v)$ where neither $E_1(v)$ nor $E_2(v)$ is empty. We show that r does not only respect χ_v, but it also respects a subconstraint of χ_v.

Definition 31. *Let χ be a parity constraint. A subconstraint of χ is a parity constraint χ' on a non-empty proper subset of the variables of χ.*

Example 16. Let $\chi : x + y + z = 0 \mod 2$. Then $\chi' : x + y = 1 \mod 2$ and $\chi'' : x = 0 \mod 2$ are subconstraints of χ. ◀

Lemma 18. *Let $T(G, c)$ be a satisfiable Tseitin formula and let r be a rectangle for the partition (E_1, E_2) of $E(G)$ such that $sat(r) \subseteq sat(T(G, c))$. If $v \in V(G)$ is incident to edges in E_1 and to edges in E_2, then there exists a subconstraint χ'_v of χ_v such that $sat(r) \subseteq sat(T(G, c) \wedge \chi'_v)$.*

Proof. Let $a_1 \cup a_2 \in sat(r)$ where a_1 is an assignment to E_1 and a_2 an assignment to E_2. Let $a_1(v)$ and $a_2(v)$ denote the restriction of a_1 and a_2 to $E_1(v)$ and $E_2(v)$, respectively. We claim that for all $a'_1 \cup a'_2 \in R$, we have that $a'_1(v)$ and $a_1(v)$ have the same parity, that is, $a_1(v)$ assigns an odd number of variables of $E_1(v)$ to 1 if and only if it is also the case for $a'_1(v)$. Indeed if $a_1(v)$ and $a'_1(v)$ have different parities, then so do $a_1(v) \cup a_2(v)$ and $a'_1(v) \cup a_2(v)$. So either $a_1 \cup a_2$ or $a'_1 \cup a_2$ falsifies χ_v, but both assignments are accepted by r, so $a_1(v)$ and $a'_1(v)$ cannot have different parities as this contradicts $sat(r) \subseteq sat(T(G, c))$. Let c_1 be the parity of $a_1(v)$, then we have that assignments accepted by r must satisfy $\chi'_v : \sum_{e \in E_1(v)} x_e = c_1 \mod 2$, so $sat(r) \subseteq sat(T(G, c) \wedge \chi'_v)$. □

Renaming χ'_v as χ^1_v and adopting notations from the proof, one sees that $\chi^1_v \wedge \chi_v \equiv \chi^1_v \wedge \chi^2_v$ where $\chi^2_v : \sum_{e \in E_2(v)} x_e = c(v) + c_1 \mod 2$. So r respects the formula $(T(G,c) - \chi_v) \wedge \chi^1_v \wedge \chi^2_v$ where $(T(G,c) - \chi_v)$ is the formula obtained by removing all clauses of χ_v from $T(G,c)$. In this sense, the rectangle is splitting the constraint χ_v into two subconstraints over disjoint variables. Since $\chi_v \equiv (\chi^1_v \wedge \chi^2_v) \vee (\overline{\chi}^1_v \wedge \overline{\chi}^2_v)$ it is plausible that potentially many models of χ_v are not accepted by r. We show that this is true in the next section.

Vertex Splitting and Subconstraints for Tseitin Formulas

Let $v \in V(G)$ and let (N_1, N_2) be a proper partition of $N(v)$, that is, neither N_1 nor N_2 is empty. The graph G' we get by *splitting* v along (N_1, N_2) is defined as the graph we get by deleting v, adding two vertices v^1 and v^2, and connecting v^1 to all vertices in N_1 and v^2 to all vertices in N_2. We now show that splitting a vertex v in a graph G has the same effect as adding a subconstraint of χ_v.

Lemma 19. *Let $T(G,c)$ be a Tseitin formula. Let $v \in V(G)$ and let (N_1, N_2) be a proper partition of $N(v)$. Let c_1 and c_2 be such that $c_1 + c_2 = c(v) \mod 2$ and let $\chi^i_v : \sum_{u \in N_i} x_{uv} = c_i \mod 2$ for $i \in \{1,2\}$ be subconstraints of χ_v. Call G' the result of splitting v along (N_1, N_2) and set*

$$c'(u) = \begin{cases} c(u), & \text{if } u \in V(G) \setminus \{v\} \\ c_i, & \text{if } u = v^i, i \in \{1,2\} \end{cases}$$

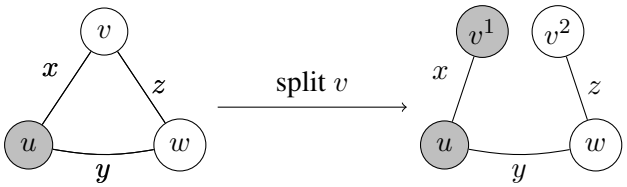
There is a bijection $\rho : var(T(G,c)) \to var(T(G',c'))$ acting as a renaming of the variables such that $T(G',c') \equiv (T(G,c) \wedge \chi^1_v) \circ \rho$.

Proof. Denote by $T(G,c) - \chi_v$ the formula equivalent to the conjunction of all χ_u for $u \in V(G) \setminus \{v\}$. Then $T(G,c) \wedge \chi^1_v \equiv (T(G,c) - \chi_v) \wedge \chi^1_v \wedge \chi^2_v$. The constraints χ_u for $u \in V(G) \setminus \{v\}$ appear in both $T(G',c')$ and in $T(G,c) - \chi_v$ and the subconstraints χ^1_v and χ^2_v are exactly the constraints for v^1 and v^2 in $T(G',c')$ modulo the variable renaming ρ defined by $\rho(x_{uv}) = x_{uv^1}$ when $u \in N_1$, $\rho(x_{uv}) = x_{uv^2}$ when $u \in N_2$, and $\rho(x_e) = x_e$ when v is not incident to e. □

Example 17. In Example 9, we introduced the Tseitin formula

$$T(G,c) = (\overline{x} \vee y) \wedge (x \vee \overline{y}) \wedge (y \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (x \vee z) \wedge (\overline{x} \vee \overline{z})$$

where G is the graph represented is shown below on the left. The splitting of v creates two nodes v^1 and v^2. The constraint corresponding to v in $T(G,c)$ is $\chi_v : x + z = 1 \mod 2$. Now we let $\chi^1_v : x = 1 \mod 2$ and $\chi^2_v : z = 0 \mod 2$ be two subconstraints of χ_v. Then the formulas $T(G,c) \wedge \chi^1_v$ and $T(G,c) \wedge \chi^2_v$ are both equivalent to the Tseitin formula $T(G',c')$, where G' results from splitting v. The following figure shows one of the outcome of that splitting.



Charges are indicated by a color code: gray vertices all have charge 0 and white vertices all have charge 1. Now we have:

$$T(G',c') = (\overline{x} \vee y) \wedge (x \vee \overline{y}) \wedge (y \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge x \wedge \overline{z}$$

Applying unit propagation on $T(G',c')$ and on $T(G,c) \wedge x = T(G,c) \wedge \chi^1_v$ shows that $T(G',c') \equiv x \wedge y \wedge \overline{z} \equiv T(G,c) \wedge \chi^1_v$. One can also verify that $T(G,c) \wedge \overline{z} = T(G,c) \wedge \chi^2_v \equiv T(G',c')$. ◀

Intuitively, Lemma 19 says that splitting a vertex in G and adding a subconstraint are essentially the same operation. This allows us to compute the number of models of a Tseitin formula to which a subconstraint was added.

Lemma 20. *Let $T(G, c)$ be a satisfiable Tseitin formula where G is connected. Define $T(G', c')$ as in Lemma 19. If G' is connected then $T(G', c')$ has $2^{|E(G)|-|V(G)|}$ models.*

Proof. Since $T(G, c)$ is satisfiable and since $\sum_{u \in V(G')} c'(u) = \sum_{u \in V(G)} c(u) = 0 \mod 2$, by Lemma 9 $T(G', c')$ is satisfiable. Using Lemma 10 yields that $T(G', c')$ has $2^{|E(G')|-|V(G')|+1} = 2^{|E(G)|-|V(G)|}$ models. □

Lemma 21. *Let $T(G, c)$ be a satisfiable Tseitin formula where G is connected. Let $\{v_1, \ldots, v_k\}$ be an independent set in G. For all $i \in [k]$ let (N_1^i, N_2^i) be a proper partition of $N(v_i)$ and let $\chi'_{v_i} : \sum_{u \in N_1^i} x_{uv_i} = c_i \mod 2$. If the graph obtained by splitting all v_i along (N_1^i, N_2^i) is connected, then the formula $T(G, c) \wedge \chi'_{v_1} \wedge \cdots \wedge \chi'_{v_k}$ has $2^{|E(G)|-|V(G)|-k+1}$ models.*

Proof. An easy induction based on Lemma 19 and Lemma 20. The induction works since, $\{v_1, \ldots, v_k\}$ being an independent set, the edges to modify by splitting v_i are still in the graph where $v_1, \ldots, v_{i-1}$ have been split. □

Vertex Splitting in 3-Connected Graphs

When we want to apply the results of the last sections to bound the size of rectangles, we require that the graph G remains connected after splitting vertices. This is obviously not true for all choices of vertex splits, but we will see that if G is sufficiently connected, then we can always chose a large subset of any set of potential splits such that, after applying the split for this subset, G remains connected.

Lemma 22. *Let G be a 3-connected graph and let $I = \{v_1, \ldots, v_k\}$ be an independent set in G. For every $i \in [k]$ let (N_1^i, N_2^i) be a proper partition of $N(v_i)$. Then there is a subset $S \subseteq I$ of size at least $k/3$ such that the graph resulting from splitting all $v_i \in S$ along the corresponding (N_1^i, N_2^i) is connected.*

Proof. Let $C_1, \ldots, C_r$ be the connected components of the graph G_1 that we get by splitting *all* v_i. If G_1 is connected, then we can set $S = I$ and we are done. So assume that $r > 1$ in the following. Now add for every $i \in [k]$ the edge (v_i^1, v_i^2). Call this edge set L (for *links*) and the resulting graph G_2. Note that G_2 is connected and for every edge set $E' \subseteq L$ we have that $G_2 \setminus E'$ is connected if and only if G is connected after splitting the vertices corresponding to the edges in E'. Denote by L_{in} the edges in L whose end points both lie in some component C_j and let $L_{out} = L \setminus L_{in}$. Note that $2k = 2|I| = |L_{out}| + |L_{in}|$.

Let $S^j = \{v \in I \mid |\{v^1, v^2\} \cap V(C_j)| = 1\}$ be the subset of I such that after splitting the vertices in I, exactly one side of the vertex ends up in C_j. We show that $|S^j| \geq 3$. Since G_2 is connected but the set C_j is a connected component of $G_2 \setminus L = G_1$, there must be at least one edge in L incident to a vertex in C_j. By construction this vertex is in I, say it is v_i. Since $N_1^i \neq \emptyset$ and $N_2^i \neq \emptyset$, we have that v_i has a neighbor w in C_j and $w \notin I$ (since I is an independent set). If we delete the vertices of S^j, then a subset of C_j becomes disconnected from the rest of G^2 (which is non-empty because there is at least one component different from C_j in G_2 which also contains a vertex not in I by the same reasoning as before). But then, because G is 3-connected, there must be at least three vertices in S^j.

Now we have that $2|L_{out}| = \sum_{i=1}^{r} |S^j| \geq 3r$, so

$$r \leq \frac{2}{3}|L_{out}|.$$

Contract all components C_j in G_2 and call the resulting multigraph G_3. Note that G_3 is connected, that $|V(G_3)| = r$, and that $E(G_3) = L_{out}$. Let E_T be the edges of a spanning tree of G_3 and let E^* be the remaining edges. Then $|L_{out}| = |E_T| + |E^*| = r - 1 + |E^*| < 2|L_{out}|/3 + |E^*|$ and thus

$$|E^*| > \frac{|L_{out}|}{3}.$$

Recall that each edge in $L_{out} = E(G_3)$ corresponds to a vertex in I that was split when going from G to G_1. Since G_3 remains connected when we delete the edge set E^*, splitting only the vertices corresponding to E^* in G yields a connected graph. Finally, the number of vertices of I that we can split in G while preserving connectivity is the same as the number of links in G_2 that can be deleted while preserving connectivity. If we choose not to delete the edges that, after contradiction of G_2 to G_3 constitute E_T, then the number of links of G_2 that we can delete while preserving connectivity is,

$$|L_{in} \cup E^*| = |L_{in}| + |E^*| = |L| - |L_{out}| + |E^*| > |L| - \frac{2}{3}|L_{out}| \geq \frac{1}{3}|L| = \frac{k}{3}.$$

□

1.3.3 Lower Bounds on the DNNF Size of Tseitin formulas

In this section, we use the results of the previous sections to show our lower bounds for circuits in DNNF computing Tseitin formulas. To this end, we will first show that we can restrict ourselves to the case of 3-connected graphs. But before that, we show that for any graph G, the size of the smallest circuit in DNNF computing a satisfiable $T(G, c)$ does not depends on c, especially we show that is is sufficient to show lower bounds when c assigns 0 to every vertex. Note that, by Lemma 9, $T(G, 0)$ is satisfiable for every G.

Lemma 23. *Let $T(G, c)$ be a satisfiable Tseitin formula with G connected. Let D be a circuit in DNNF computing $T(G, c)$. Then there is a circuit D_0 in DNNF computing $T(G, 0)$ obtained by switching the polarity of some literals in D, so $|D_0| = |D|$.*

Proof. Let v and w be two vertices of G sent to 1 by c. Denote $v_1 = v$ and $v_m = w$ and let $P = \{v_1v_2, \ldots, v_{m-1}v_m\}$ be a path from v to w in G, which exists by connectivity. Set $c'(v) = c'(w) = 0$ and $c'(u) = c(u)$ for $u \notin \{v, w\}$. $T(G, c')$ is satisfiable. Let $X = \{x_e \mid e \in E(G)\}$ and consider the bijection $\varphi : lit(X) \to lit(X)$ defined as $\varphi(\ell_{x_e}) = \overline{\ell_{x_e}}$ if $e \in P$ and $\varphi(\ell_{x_e}) = \ell_{x_e}$ otherwise. Seeing truth assignments as mappings from X to $lit(X)$, one can verify that, for all $u \in V(G)$, an assignment a satisfies $\sum_{e \in E(u)} x_e = c(u) \mod 2$ if and only if $\varphi \circ a$ satisfies $\sum_{e \in E(u)} x_e = c'(u) \mod 2$. Thus $sat(T(G, c')) = \{\varphi \circ a \mid a \in sat(T(G, c))\}$. Switching the polarity of literal inputs in a circuit does not impact decomposability, so the circuit D' obtained by switching the polarity of literals input according to φ in D is a circuit in DNNF computing $T(G, c')$. The number of vertices sent to 0 by c' is that of c minus 2. Repeating the procedure until reaching the 0-function yields D_0 computing $T(G, 0)$. □

Reduction from Connected to 3-Connected Graphs

We here show why it is sufficient to prove our bound for Tseitin formulas whose graphs are 3-connected. In [BK06], Bodlaender and Koster study how separators can be used in the context of treewidth. They call a separator S *safe for treewidth* if there exists a connected component of $G \setminus S$ whose vertex set V' is such that $tw(G[S \cup V'] + clique(S)) = tw(G)$, where $G[S \cup V'] + clique(S)$ is the graph induced on $S \cup V'$ with additional edges that pairwise connect all vertices in S.

Lemma 24. [BK06, Corollary 15] *Every separator of size 1 is safe for treewidth. When G has no separator of size 1, every separator of size 2 is safe for treewidth.*

Remember that a *topological minor* H of a G is a graph that can be constructed from G by iteratively applying the following operations [Die12]:

- edge deletion,
- deletion of isolated vertices, or
- subdivision elimination: if $\deg(v) = 2$ then delete v and, if its two neighbors are not already connected, then connect them.

Note that for subdivision elimination, if the two neighbors of v are already connected, then the subdivision elimination boils down to two edge deletions followed by the deletion of the now isolated vertex v.

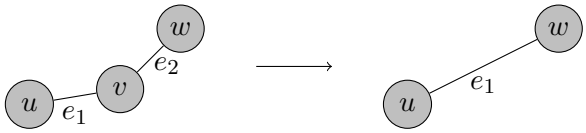
Lemma 25. *Let H be a topological minor of G. If the satisfiable Tseitin formula $T(G, 0)$ is computed by a circuit in DNNF of size s, then so does $T(H, 0)$.*

Proof. Let D be a circuit in DNNF computing $T(G, 0)$. We show how to obtain a circuit D' in DNNF representing $T(H, 0)$ with $|D'| = |D|$ when H is obtained by applying a single operation (edge deletion, isolated vertex deletion, subdivision elimination). The lemma will then follow by induction.

If H is obtained by deleting an isolated vertex v from G, then $T(H, 0) = T(G, 0)$ since isolated vertices give no constraints and thus no clauses in $T(G, 0)$. So in this case $D' = D$.

If H is obtained by deleting an edge $e = uv$ from G, then $T(H, 0) = T(G, 0)|_{\overline{x_e}}$. Conditioning a circuit in DNNF on a variable assignment does not increase its size: one can just replace in D every occurrence of x_e by 0 and every occurrence $\overline{x_e}$ by 1 to obtain D'. Clearly $|D'| = |D|$.

Finally assume that there is a vertex v of degree 2 in G incident to the edges $e_1 = uv$ and $e_2 = vw$ and say that H is constructed from G by subdivision elimination of v. There are two cases. If $uw \in E(G)$, then H is obtained by removing v, e_1 and e_2 from G. In other words, H is derived from G by two edge deletions (e_1 and e_2) followed by one isolated vertex deletion (v). We have already treated these operations so we know how to obtain D' from D in this case. If however $uw \notin E(G)$ then H is obtained by deleting v, e_1, e_2 and by connecting u to w. Since $c(v) = 0$, the constraint $\chi_v : x_{e_1} + x_{e_2} = 0 \mod 2$ implies that, in every satisfying assignment, x_{e_1} and x_{e_2} take the same value. Thus, abusing notation a little and calling e_1 the edge between u and w in H (as in the following figure) we obtain that $T(H, 0) \equiv T(G, 0)|_{x_{e_2}} \vee T(G, 0)|_{\overline{x_{e_2}}}$



We say that $T(H, 0)$ is obtained by *forgetting* x_{e_2} *from* $T(G, 0)$, denoted $T(H, 0) \equiv \exists x_{e_2}.T(G, 0)$. Forgetting is feasible on circuits in DNNF without size increase. Indeed forgetting x_{e_2} from D boils down to replacing *both* occurrences of x_{e_2} and occurrences of $\overline{x_{e_2}}$ by 1 [Dar01a]. Calling the resulting circuit D', we clearly have $|D'| = |D|$. □

We remark in passing that a result analogous to Lemma 25 is also true for nFBDD and FBDD instead of circuits in DNNF. For nFBDD this is directly clear since the forgetting operation that we use in the proof is also possible on nFBDD without any size increase and so the same proof works. However, for FBDD the proof is not immediate, since for them forgetting variables generally leads to an unavoidable

blow-up in size [DM02]. The proof can still be made to work nevertheless because we are in a special case in which the variable to forget is equivalent to a variable that remains and thus the operation is possible without any size increase. Since we do not use Lemma 25 for nFBDD or FBDD, we leave out the details.

Lemma 26. *Let G be a graph with treewidth at least 3. Then G has a 3-connected topological minor H with $tw(H) = tw(G)$.*

Proof. First we construct a topological minor of G with no separator of size 1 that preserves treewidth. Let $S = \{v\}$ be a separator of size 1 of G, then $G \setminus S$ has a connected component V' such that $G[S \cup V'] + clique(S) = G[S \cup V']$ has treewidth $tw(G)$. Let $G' = G[S \cup V']$, then $tw(G') = tw(G)$. Observe that G' is a topological minor (remove all edges not in $G[S \cup V']$ thus isolating all vertices not in $S \cup V'$, which are then deleted) where S is no longer a separator. Repeat the construction until G' has no separator of size 1.

Now assume $S = \{u, v\}$ is a separator of G'. If V' are the vertices of a connected component of $G' \setminus S$, then there is a path from u to v in $G[S \cup V']$ since otherwise either $\{u\}$ or $\{v\}$ is a separator of size 1 of G'. Lemma 24 ensures that there is a connected component H' in $G' \setminus S$ such that $H = (V(H') \cup S, E(G[V(H') \cup S]) \cup \{uv\})$ has treewidth $tw(H) = tw(G') = tw(G)$. Let us prove that H is topological minor of G'. Consider a connected component of $G' \setminus S$ distinct from H' with vertices V' and let P be a path connecting u to v in $G[S \cup V']$. Delete all edges of $G[S \cup V']$ not in P, then delete all isolated vertices in V' so that only P remains, finally use subdivision elimination to reduce P to a single edge uv. Repeat the procedure for all connected components of $G' \setminus S$ distinct from H', the resulting topological minor is $G[V(H') \cup S]$ with the (additional) edge uv, so it is$H$.

Repeat the construction until there are no separators of size 1 or size 2 left. Note that this process eventually terminates since the number of vertices decreases after every separator elimination. The resulting graph H is a topological minor of G of treewidth $tw(G)$ without separators of size 1 or 2. Since $tw(H) = tw(G) \geq 3$, we have that H has at least 4 vertices, so H is 3-connected. □

Lemma 26 does not hold when replacing treewidth by pathwidth. To see this, note that trees can have arbitrarily high pathwidth, see [Sch89], and trees with more than 3 vertices are clearly not 3-connected. Connected topological minors of trees are again trees, so a topological minor of a tree is 3-connected if and only if it has at most three vertices and therefore its pathwidth is at most 2.

Proof of the Lower Bound on the DNNF Size for Tseitin Formulas

We are now ready to prove the main result of this section.

Theorem 5. *Let $T(G, c)$ be a satisfiable Tseitin formula where G is a connected graph with maximum degree at most Δ. Every smooth circuit in DNNF computing $T(G, c)$ has size at least $2^{\Omega(tw(G)/\Delta)}$.*

Proof. By Lemma 23 we can set $c = 0$. By Lemmas 25 and 26 we can assume that G is 3-connected. We show that the adversarial multi-partition rectangle complexity is lower-bounded by 2^k for $k = \frac{2tw(G)}{9\Delta}$. To this end, we show that the rectangles that Charlotte can construct after Adam's answer are never bigger than $2^{|E(G)|-|V(G)|-k+1}$. Since $T(G, c)$ has $2^{|E(G)|-|V(G)|+1}$ models, the claim then follows.

So let Charlotte choose an assignment a and a vtree T. Note that since the variables of $T(G, 0)$ are the edges of G, the vtree T is also a branch decomposition of G. Now by the definition of branchwidth, Adam can choose a cut of T inducing a partition (E_1, E_2) of $E(G)$ for which there exists a set $V' \in V(G)$ of at least $bw(G) \geq \frac{2}{3}tw(G)$ vertices incident to edges in E_1 and to edges in E_2.

G has maximum degree Δ so there is an independent set $V'' \subset V'$ of size at least $\frac{|V'|}{\Delta}$. Since G is 3-connected, by Lemma 22 there is a subset $V^* \subseteq V''$ of size at least $\frac{|V''|}{3} \geq \frac{2tw(G)}{9\Delta} = k$ such that G

remains connected after splitting of the nodes in V^* along the partition of their neighbors induced by the edges partition (E_1, E_2). Using Lemma 18, we find that any rectangle r for the partition (E_1, E_2) respects a subconstraint χ'_v for each $v \in V^*$. So r respects $T(G,0) \wedge \bigwedge_{v \in V^*} \chi'_v$. Finally, Lemma 21 shows that $|sat(r)| \leq 2^{|E(G)|-|V(G)|-k+1}$, as required. □

Then we have shown a bound on the size of the smallest circuit in DNNF representing a satisfiable Tseitin formula, that can be summarized as follows. Let H be the 3-connected topological minor of G chosen by Lemma 26.

DNNF size for $T(G,c)$

$= $ DNNF size for $T(H,0)$ (Lemma 23)

$\geq $ DNNF size for $T(H,0)$ (Lemma 25)

$\geq aR(T(H,0))$ (Theorem 7)

$\geq 2^k$ where $k = 2\,tw(H)/(9(\Delta(H)+1))$ (proof of Theorem 5)

$= 2^k$ where $k = 2\,tw(G)/(9(\Delta(H)+1))$ (Lemma 26)

$\geq 2^k$ where $k = 2\,tw(G)/(9(\Delta(G)+1))$ ($\Delta(G) \geq \Delta(H)$)

We note that a similar lower bound using pathwidth instead of treewidth holds on the size of the smallest nFBDD computing a Tseitin formula. But then, as explained before, the reduction to Tseitin formulas structured by 3-connected graphs does not preserve the pathwidth. So we can only phrase our result with pathwidth for Tseitin formulas whose graphs are already 3-connected.

Lemma 17. *Let $T(G,c)$ be a satisfiable Tseitin formula where G is a 3-connected graph with maximum degree at most Δ. Then every nFBDD computing $T(G,c)$ has size at least $2^{\Omega(pw(G)/\Delta)}$.*

Proof. The proof is the similar to that of Theorem 5. Again by Lemma 23 we can set $c = 0$ and this time it is assumed in the lemma that G is 3-connected. The proof then follows the proof of Theorem 5 to show that the *linear* adversarial multi-partition rectangle complexity is lower-bounded by 2^k for $k = \frac{pw(G)}{3\Delta}$. So we use Corollary 1 instead of Theorem 7. One just needs to follow the second and third paragraphs of the proof of Theorem 5, replacing vtree by *linear* vtree, branch decomposition by *linear* branch decomposition, branchwidth with *linear* branchwidth, and using the inequality $bw_\ell(G) \geq pw(G)$ from Lemma 1. □

We recall that Lemma 17 can be compared with a result of Itsykson et al. who, as mentioned before, proved that the nFBDD size for *any* satisfiable Tseitin formula $T(G,c)$ is between $2^{compw(G)}$ and $|E(G)| \times 2^{compw(G)} + 2$ where $compw(G)$ is the component width of G, a width parameter whose definition we omit. The component width is related to the treewidth and the pathwidth by the following relation: $\frac{1}{2}(tw(G)-1) \leq compw(G) \leq pw(G)+1$. Lemma 17 also shows that the component width and the pathwidth are linearly related for 3-connected graphs whose maximum degree is bounded by a constant. However, since the component width of trees is 0 while the pathwidth is unbounded, Lemma 17 does not extend to cases where G is not 3-connected.

1.4 Conclusion and Perspectives

The multi-partition rectangles cover technique, initially a tool from communication complexity, has proved to be very practical for showing large lower bounds on the DNNF size of several functions ever since it has been introduced for knowledge compilation [BCMS16]. We have used this technique to prove exponential lower bounds on the DNNF size of two new classes of functions.

First we have shown that pseudo-Boolean constraints may, for specific thresholds and specific weights to the variables, have DNNF size exponential in the number of variables. Our result shows that methods that involve compiling every constraint of a problem in DNNF are ill-suited depending on the types of constraints (for instance techniques for finding propagation complete encodings of constraints [KS19]). But we have only analyze a worst case scenario. We leave open the more difficult problem of characterizing pseudo-Boolean constraints whose DNNF size is not polynomial in the number of variables.

We have also shown that the DNNF size of satisfiable Tseitin formulas, whose underlying graph has maximum degree bounded by a constant, is characterized by the treewidth of the graph: such formulas have DNNF size polynomial in the number of variables n if and only if the treewidth of the graph is at most $O(\log(n))$. To obtain this result, we have improved the multi-partition rectangles cover technique. The inspiration for the improved technique, called the *adversarial multi-partition rectangles cover game*, came from the work of Razgon in [Raz16]. It takes the form of a two-player game and its main advantage is that it relaxes the requirement of working with rectangles over balanced partitions.

One year after our exponential lower bound for Tseitin formulas was presented in [dCM21a], it was generalized by Itsykson, Riazanov and Smirnov in [IRS22] to the case of satisfiable Tseitin formulas over *any* graph (and not only graphs of bounded degree). Notably, they still use the adversarial multi-partition rectangles cover game to prove their result. There certainly are other functions that are hard to compile to DNNF and for which the classical multi-partition rectangle cover technique fails to give good lower bounds on the DNNF size. It would be interesting to see the extent to which our new technique is useful for these functions. With the result of Itsykson et al., there may not be much left to prove on the DNNF size of Tseitin formulas, but we are not done with Tseitin formulas in this thesis! Indeed the bound on their DNNF size is instrumental for proving several other results, that are presented in other chapters.

Chapter 2

Lower Bounds for Approximate Knowledge Compilation

When the compiled form of a function in a language L is too large, and when the context allows it, one can try to compile an approximation of the function with the hope that the its representation in L is smaller. The approximation error intuitively depends on the nature of the queries to be answered on the (approximate) function, but in any case this error must be controlled to avoid arguably useless approximations like constant 0 functions. In this chapter we first study a notion of approximation introduced for FBDDs and OBDDs that we call *weak approximation*, we show that there are functions whose weak approximations all have d-DNNF size exponential in the number of variables. Then we give arguments against weak approximation in the context of (approximate) model counting and introduce a second notion that we call *strong approximation* that avoid some of the problems of weak approximation. We show that there are functions that have arguably useless weak approximation that can be represented by small circuits in d-DNNF while their strong approximations have d-DNNF size exponential in the number of variables.

2.1 State of Approximate Knowledge Compilation

The viability of knowledge compilation for reasoning on a Boolean function depends on the size of the compiled form of that function. A large compiled form requires a long compilation time and, more importantly, often implies that answers to queries will take longer to get. As we have seen in the previous chapter, the compilation of Boolean functions into DNNF is often bound to generate very large circuits. Actually there is no language that supports polynomial time clausal entailment (like DNNF) where all Boolean functions have polynomial-size compiled form unless the polynomial hierarchy collapses [SK91, SK96]. To overcome this difficulty one may resort to approximation during compilation. Approximate knowledge compilation suggests compiling an approximation of the initial function when the exact compiled form is too large. Before using approximation, one must ensure that introducing errors is authorized by the context. Indeed, depending on the application of the function to be compiled, approximation is not necessarily an option, for instance a function that decides whether to administer a certain drug based on a patient symptoms should never generate erroneous results for obvious reasons. Even when approximation is an option, the approximation error must be controlled to avoid, for instance, that a function with hundreds of models out of thousands of truth assignments be approximated by the constant 0 function.

One canonical area where approximate knowledge compilation makes sense is probabilistic reasoning. In this setting, one wants to compile classifiers based on graphical models, for instance Bayesian

networks, to then reason about the classifiers by querying the compiled representation [CD03]. The most important language in this setting is that of circuits in d-DNNF, which allow for efficient (weighted) model counting and probability computation and are thus particularly well-suited for probabilistic reasoning [Dar01b]. Recall that circuits in d-DNNF are generalizations of other important models of computation like OBDD [Bry86] and SDD [Dar11], which have also found applications in probabilistic reasoning [CD03, CKD13, SCD19b]. Since graphical models like Bayesian networks are almost exclusively inferred by learning processes, they are inherently not exact representations of the world. In particular for Bayesian networks, the structure of the network can capture the exact dependencies between variables, but the conditional probability tables are approximate. Thus, when reasoning about graphical models, in most cases the results do no have to be exact but approximate reasoning is sufficient, assuming that the approximation error can be controlled and is small. It is thus natural in this context to consider approximate knowledge compilation. Recently, Chubarian and Turán [CT20] showed, building on [GKM+11], that this approach is feasible in some settings: it is possible to compile efficiently approximations of so-called Tree Augmented Naive Bayes classifiers (TAN) (or more generally bounded pathwidth Bayes classifiers) into OBDD. Note that efficient exact compilation is ruled out in this setting due to strong lower bounds for threshold pseudo-Boolean functions, as shown in Chapter 1 of this thesis, which imply lower bounds for TAN.

Horn Approximations. Approximation was considered already in some of the earliest work on knowledge compilation by Selman and Kautz [SK91, SK96]. Their approximate compilation scheme is to construct Horn formulas H_l and H_u that approximate an input CNF formula ϕ by $H_l \models \phi \models H_u$. The purpose of this method is to perform approximate clausal entailment on ϕ (clausal entailment being tractable on Horn formulas). However, the focus was different in this setting: Horn formulas are not fully expressive so the question becomes that of understanding the formulas that are the best out of all Horn formulas approximating a function, instead of requesting error guarantees for the approximation. Moreover the task of efficiently finding an optimal approximation in this scheme has some serious complexity impediments as pointed out in [KPS93].

DNNF and BDD Approximations. Regarding compilation to the DNNF language, the very first approximate compilation schemes were given by Darwiche along with the first exact compilation algorithms [Dar01a]. The exact algorithms for compiling CNF formulas proposed by Darwiche – and actually all subsequent top-down compilation algorithms – alternate between conditioning the formula on partial assignments to its variables and looking for disjoint components of a graph of the conditioned CNF (hypergraph, primal graph, dual graph, etc.) to create decomposable $\wedge$-nodes. The proposed approximation consisted in either ignoring variables or ignoring assignments when conditioning. In another direction, the approximation scheme introduced in [PD07] – initially as a first step for solving approximate MaxSAT – modifies the initial CNF formula by replacing some occurrences of variables by fresh literals before compilation. This replacement can only decrease the primal treewidth of the formula, a fortiori it decreases the value of known upper bounds on the smallest size of the compiled form in DNNF, d-DNNF or SDD (which we recall are exponential in the primal treewidth of the formula). After compiling the modified formula in DNNF, the additional variables can be forgotten to obtain a circuit in DNNF that approximately computes the initial formula and is entailed by it. These approximation schemes are mostly concerned with the size of the resulting circuit. No approximation error has been defined for these schemes, and the only guarantee on the quality of the approximate circuit in DNNF is that it either entails or is entailed by the initial formula. Since the sublanguages OBDD and FBDD of DNNF were known and studied before DNNF was introduced, the research on approximate compilation to these sublanguage has developed independently to that on compilation to DNNF and has been more concerned with the approximation error. A measure of the quality of an approximate OBDD or FBDD

has been defined in [BSW02] as the probability that this circuit disagrees with the initial function on a random truth assignment. Bollig et al. show that compiling good approximations to OBDD with respect to this measure is intractable for some classes of functions [BSW02].

Randomized BDDs. It is common that approximation methods rely on randomization. The randomized algorithms called Monte Carlo algorithms often have a step where some elements are sampled according to a probability distribution (often the uniform distribution) over a large set of objects. The algorithm gains time as it avoids exploring large sets of objects, but in exchange it returns is an estimate of the correct answer. A usual amplification technique to improve the confidence on the estimate is to perform several independent runs of the algorithm and to generate a global estimate (like the mean or the median of the results of every run). As an example, Karger algorithm is a randomized algorithm that generates a cut of a graph that is close to minimum by randomly sampling edges of the graph to contract [KS96]. OBDDs and FBDDs are, in a sense, programs to search the set of solutions of a function (hence the alternate name "branching programs") and one can introduce randomization in these circuits by giving them access to additional nodes called "guess nodes" and to an additional sink labelled by "?". The circuits are then called *randomized OBDDs* and *randomized FBDDs*. Guess nodes are unlabelled binary nodes, that essentially introduce non-determinism to the circuit, as do $\vee$-nodes in nOBDDs and nFBDDs. Given a truth assignment, a computation path is constructed in a randomized OBDD (resp. FBDD) circuit like in a normal OBDD (resp. FBDD), except that when a guess node is met, the child to add to the path is chosen with probability $1/2$ (this is a sampling step). So for a given assignment a randomized OBDD or FBDD has a certain probability to reach the sink 1. As such, these objects can be used to approximately represent Boolean functions. The approximation error is the probability that on a random assignment, the randomized OBDD (resp. FBDD) reaches the wrong sink (the sink 0 or "?" if the assignment is a model of the function, and the sink 1 or "?" otherwise). In [Weg00, Chapter 11], Wegener gives a collection of functions that are hard to exactly compile in OBDD and that have small approximations as randomized OBDDs whose error can be controlled, and also examples of functions that are only computed by large OBDDs and that cannot be approximated by small randomized OBDDs.

Contributions. Looking at what has been done for approximate compilation to DNNF or OBDD, one sees that the definition of a "good" approximation (regardless of the size of its compiled form) should intuitively depends on the reasoning performed on the compiled form. Typically, the expectations in terms of guarantees for an approximate function seems different when it is queried for clausal entailment than when it is used for counting. In the latter case, one may want the proportion of satisfying (resp. falsifying) assignments of the approximation to be close to that of the initial function. While in the former case one may focus on finding good upper and lower approximations with respect to entailment as in [SK91, SK96, Dar01a]. Then the answer of a clausal entailment query on the initial function is known when the upper approximation entails the given clause (then so does the initial function) or when the lower approximation does not entail the given clause (then neither does the initial function). In this chapter we study two notions of approximations. The first one, that we call *weak approximation*, is the one previously used for representing approximations by OBDDs [KSW99, BSW02]. We adapt a result of [BSW02] to find an infinite class of functions that are hard to compile in d-DNNF, due to their large d-DNNF size, and show that the same holds for any of their weak approximations. Then we will see that weak approximation has shortcomings that make it the wrong notion to use for approximate model counting when the number of models of the function to approximate is not large enough. We then remedy the situation by formalizing the new notion of *strong approximation*. While not formalized as such, it can be verified that the OBDDs of [CT20, GKM+11] are in fact strong approximations in our sense. We show the existence of an infinite class of functions for which the constant 0 function is a weak approximation (though a fairly useless one) that is clearly easily represented as a circuit in d-DNNF, but

such that the d-DNNF size of any strong approximation of the functions is exponential in the number of variables. The results of this chapter have been published in the article [dCM20] co-authored with Stefan Mengel.

2.2 Lower Bounds for Weak Approximation to d-DNNF

In this chapter, we measure the quality of an approximation by looking at how close its sets of models and counter-models are to that of the initial function. We start by considering the notion of approximation that has been studied for different forms of branching programs before, see e.g. [KSW99, BSW02]. To differentiate it from other notions, we give it the name *weak approximation*.

Definition 32 (Weak approximation)**.** *Let $\mathcal{D}$ be a distribution on the truth assignments to X and $\varepsilon > 0$. We say that $\tilde{f}$ is a* weak ε-approximation *of f (or weakly ε-approximates f) with respect to $\mathcal{D}$ if*

$$\Pr_{a\sim\mathcal{D}}\left[f(a) \neq \tilde{f}(a)\right] \leq \varepsilon.$$

When $\mathcal{D}$ is the uniform distribution $\mathcal{U}$, the condition of weak ε-approximability is equivalent to $|\{a \mid f(a) \neq \tilde{f}(a)\}| \leq \varepsilon 2^n$.

Note that weak ε-approximation is only useful when $\varepsilon < 1/2$. This is because every function has a trivial $(1/2)$-approximation: if $\Pr_{a\sim\mathcal{D}}[f(a) = 1] > 1/2$, then the constant 1-function is a $(1/2)$-approximation, otherwise this is the case for the constant 0-function. Note that it might be hard to decide which case is true – for instance if f is given as a formula then deciding whether $\Pr_{a\sim\mathcal{U}}[f(a) = 1] > 1/2$ is the PP-complete problem MAJSAT – but in any case we know that the approximation ratio of one of the constants is good.

When $\varepsilon < 1/2$, weak-approximation used in the context of approximate compilation suffers from some impediments. Indeed, Bollig *et al.* [BSW02] show that there are classes of functions such that all OBDDs computing any ε-approximation with respect to $\mathcal{U}$ have exponential size. We lift their techniques to circuits in d-DNNF showing that the same functions are also hard for these circuits.

Theorem 9. *For every $0 \leq \varepsilon < 1/2$, there is a class of Boolean functions $\mathcal{C}$ such that, for any $f \in \mathcal{C}$ on n variables, f can be represented by a formula of size $O(poly(n))$ but any circuit in d-DNNF computing a weak ε-approximation of f with respect to $\mathcal{U}$ has size $2^{\Omega(n)}$.*

The Boolean functions in $\mathcal{C}$ are actually bilinear forms on the vector space $\mathbb{F}_2^n$, whose elements are vectors of n elements in $\{0,1\}$, whose product operator is the element-wise conjunction and whose sum operator is the element-wise xor (exclusive disjunction). Bilinear forms on $\mathbb{F}_2^n$ are function from $\mathbb{F}_2^n \times \mathbb{F}_2$ to $\mathbb{F}_2^n$ of the form $f(x,y) = x^\top Ay$ for $x, y \in \{0,1\}^n$ and A an $n \times n$ matrix of 0s and 1s. These functions are seen as Boolean functions from $\{0,1\}^n \times \{0,1\}^n$ to $\{0,1\}$ and can be written as Boolean formulas over $2n$ variables using only the connectors $\wedge$ and $\oplus$ (exclusive disjunction), see Example 18.

It should be clear from the theorem statement that any exact compilation of functions of the class $\mathcal{C}$ to d-DNNF also results in circuits of size exponential in n (since f is a weak-approximation of f). So the theorem states that there are functions with polynomial-size representation as formulas (though not CNF formulas) but such that all exact and weak approximate representations in d-DNNF are too large. The proof of Theorem 9 follows exactly that of Bollig et al. for OBDDs, differing near the end only. A central element of the proof is the notion of *discrepancy*.

The discrepancy method. We want to use Theorem 2 to bound the size of circuits in d-DNNF computing a weak ε-approximation $\tilde{f}$ of f with respect to some distribution. To this end we study disjoint

balanced rectangle covers of $\tilde{f}$. Let r be a rectangle from such a cover. r can make *false positives* on f, i.e., have models that are not models of f. The set of false positives of r on f is $r^{-1}(1) \cap f^{-1}(0)$. Similarly, *true positives* are models shared by r and f, i.e., assignments in $r^{-1}(1) \cap f^{-1}(1)$. The *discrepancy* $\text{Disc}\,(f, r)$ of f on r is the difference between the number of false positives and true positives, normalized by the total number of assignments. Let $n = |var(f)|$, then

$$\text{Disc}\,(f, r) = \big||r^{-1}(1) \cap f^{-1}(1)| - |r^{-1}(1) \cap f^{-1}(0)|\big|/2^n.$$

A small discrepancy indicates that r has few models (i.e., both $r^{-1}(1) \cap f^{-1}(0)$ and $r^{-1}(1) \cap f^{-1}(1)$ are small) or that it makes roughly as many false positives as true positives on f (i.e., $|r^{-1}(1) \cap f^{-1}(0)| \approx |r^{-1}(1) \cap f^{-1}(1)|$). Proving large lower bounds by Theorem 2 is easier when the rectangles of the balanced cover are small. Intuitively, if all rectangles in a cover of $\tilde{f}$ have a small discrepancy then either they are small (as desired), or they make many false positives on f and therefore $\tilde{f}$ is not a good approximation of f. Discrepancy bounds based on this idea have been used before to prove results in distributional communication complexity [KN97, Chapter 3.5]. We show that when there is an upper bound on $\text{Disc}\,(f, r)$ for all rectangles r in every cover of $\tilde{f}$, one can obtain a lower bound on the size of the cover of $\tilde{f}$.

Lemma 27. *Let f be a Boolean function over n variables and let $\tilde{f}$ be a weak ε-approximation of f with respect to $\mathcal{U}$. Let $\tilde{f} = \bigvee_{k=1}^{K} r_k$ be a disjoint (balanced) rectangle cover of $\tilde{f}$ and assume that there is an integer $\Delta > 0$ such that $\text{Disc}\,(f, r_k) \leq \Delta/2^n$ for for all r_k. Then $K \geq (|f^{-1}(1)| - \varepsilon 2^n)/\Delta$.*

Proof. We have $|f \neq \tilde{f}| = |\{a \mid f(a) \neq \tilde{f}(a)\}|$

$$\begin{aligned}
&= |f^{-1}(1) \cap \tilde{f}^{-1}(0)| + |f^{-1}(0) \cap \tilde{f}^{-1}(1)| \\
&= \big|f^{-1}(1) \cap \textstyle\bigcap_{k=1}^{K} r_k^{-1}(0)\big| + \big|f^{-1}(0) \cap \bigcup_{k=1}^{K} r_k^{-1}(1)\big| \\
&= |f^{-1}(1)| - \sum_{k=1}^{K} (|r_k^{-1}(1) \cap f^{-1}(1)| - |r_k^{-1}(1) \cap f^{-1}(0)|) \\
&\geq |f^{-1}(1)| - 2^n \textstyle\sum_{k=1}^{K} \text{Disc}\,(f, r_k) \\
&\geq |f^{-1}(1)| - K\Delta
\end{aligned}$$

where the last equality is due to the rectangles being disjoint. The weak ε-approximation with respect to the uniform distribution $\mathcal{U}$ gives that $|\tilde{f} \neq f| \leq \varepsilon 2^n$, which we use to conclude. □

Combining Lemma 27 with Theorem 2, the proof of Theorem 9 boils down to showing that there are functions such that for every balanced rectangle r, the discrepancy $\text{Disc}\,(f, r)$ can be suitably bounded, as shown in [BSW02].

Proof sketch of Theorem 9. The hard functions are particular bilinear forms. Recall that a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is a *bilinear form* if it is linear in each of its two arguments. Every Boolean bilinear form is characterized by an $n \times n$ matrix A over $\{0,1\}$ by the relation $f(a, b) = a^\top Ab$. Such a function has $2^{2n-1}(1 - 2^{-\text{rk}(A)})$ models (there are $2^n - 2^{n-\text{rk}(A)}$ vectors b such that $Ab \neq 0$ and for each such Ab there are 2^{n-1} vectors a such that $a^\top Ab \neq 0$). Bilinear forms can be seen as Boolean functions from $\{0,1\}^{2n}$ to $\{0,1\}$, yet we find convenient to keep the notation $f(a, b)$ and refer to two sets of n variables X and Y.

A result of particular interest is the following lemma due to Ajtai [Ajt05]. It states that for n large enough, there exist matrices with a lower bound on the rank of any large enough submatrix.

Claim 3 (Ajtai Lemma)**.** [Ajt05] *Take* $0 < \delta \leq 1/2$ *such that* $\delta \log(1/\delta)^2 \leq 2^{-16}$*. There exist exponentially many matrices of size* $n \times n$ *for which each square submatrix of size at least* $\delta n \times \delta n$ *has rank at least* $\delta' n$*, where* $\delta' = \delta/(256 \log(1/\delta))^2$*.*

Let f be a bilinear form whose matrix H is that given by Ajtai Lemma and let $\tilde{f}$ be a weak ε-approximation with respect to $\mathcal{U}$, Bollig et al. use the discrepancy method to bound the discrepancy of f with respect to any rectangle r from a disjoint balanced rectangle cover of $\tilde{f}$. Their discrepancy bound comes from the following result shown by Babai et al.

Claim 4. [BHK01] *Let* X *and* Y *be sets of variables and* $|X| = |Y| = n$*. Let* $f' : \{0,1\}^X \times \{0,1\}^Y \to \{0,1\}$ *be a bilinear form characterised by the matrix* A*, and let* r' *be a rectangle over* $X \cup Y$ *with respect to the partition* (X, Y)*, then* $\text{Disc}\,(f', r') \leq \sqrt{\Pr_b\,[Ab = 0]} = 2^{-\text{rk}(A)/2}$*.*

Bollig et al. use several arguments to bound the discrepancy $\text{Disc}\,(f, r)$ from above by a fraction of $\text{Disc}\,(f', r')$ with f' a bilinear form characterised by some submatrix A of size $\delta n \times \delta n$ of the matrix H of f. Then using the properties of that matrix H they obtained the following:

Claim 5. [BSW02, Proof of Theorem 21] *Let* f *a bilinear form whose matrix is that given by Claim 3 and let* $\tilde{f}$ *a weak* ε*-approximation with respect to* $\mathcal{U}$*. We have* $\text{Disc}\,(f, r) \leq 2^{-\delta' n/2-2}$ *for every rectangle from a cover of* $\tilde{f}$ *with respect to a balanced partition, where* δ' *is defined as in Claim 3.*

Now let $\bigwedge_{k=1}^{R_d(\tilde{f})} r_k$ be a disjoint balanced rectangle cover of $\tilde{f}$. It follows from Lemma 27 and Claim 5 that

$$R_d(\tilde{f}) \geq \frac{|f^{-1}(1)|2^{-2n} - \varepsilon}{2^{-\delta' n/2-2}} \geq \frac{2^{-1} - 2^{-\delta' n-1} - \varepsilon}{2^{-\delta' n/2-2}} \geq 2^{\delta' n/2+2}\left(\frac{1}{2} - \varepsilon\right) - 2^{-\delta' n/2+2}$$

δ' is a constant defined independently of n (because δ' depends solely on δ which is defined independently of n) so this proves that the number of disjoint rectangles respecting balanced partition in the cover of $\tilde{f}$ is $2^{\Omega(n)}$. We use Theorem 2 to conclude. □

Theorem 9 is a straightforward generalization of the result on OBDDs in [BSW02] since the d-DNNF size of a Boolean function is never smaller than a constant factor time its OBDD size, and its actually exponentially smaller in some cases [DM02].

2.3 Strong ε-Approximations

That *some* functions have only weak approximations that cannot be computed by small circuits in d-DNNF is not the main argument against using weak approximation in approximate knowledge compilation. We discuss another shortcoming of weak approximation and propose a stronger notion of approximation that avoids it.

Definition 33. *Let* f *be a Boolean function.* f *is* trivially weakly ε-approximable *(with respect to some distribution) if the constant* 0 *function is a weak* ε*-approximation of* f*.*

Considering approximations with respect to the uniform distribution, it is easy to find classes of functions that are trivially weakly approximable.

Lemma 28. *Let* $\varepsilon > 0$ *and* $0 \leq \alpha < 1$*. Let* $\mathcal{C}$ *be a class of functions such that every function in* $\mathcal{C}$ *over* n *variables has at most* $2^{\alpha n}$ *models. Then there exists a constant* n_0*, such that any function from* $\mathcal{C}$ *over more than* n_0 *variables is trivially weakly* ε*-approximable with respect to the uniform distribution.*

Proof. Let $n_0 = \frac{1}{1-\alpha}\log(\frac{1}{\varepsilon})$ and choose f any function from $\mathcal{C}$ over $n > n_0$ variables. Then $|\{a \mid f(a) \neq f_0(a)\}| = |f^{-1}(1)| \leq 2^{\alpha n} < \varepsilon 2^n$. Therefore f_0 is a weak ε-approximation (with respect to the uniform distribution) of any function of $\mathcal{C}$ over sufficiently many variables. □

We remark that similar trivial approximation results can be shown for other distributions if the probability of a random assignment with respect to this distribution being a model is very small. As a consequence, weak approximation makes no sense for functions with "few" (or "improbable") models, especially in the context of model counting. However such functions are often encountered, for example, random k-CNF with sufficiently many clauses are expected to have few models, see [ACH+21] for the case $k = 2$. Furthermore, even for functions with "many" models, one often studies encodings over larger sets of variables. For instance, when using Tseitin encoding to transform Boolean circuits into CNF, one introduces auxiliary variables that compute the value of sub-circuits under a given assignment. Generally, auxiliary variables are often used in practice since they reduce the representation size of functions, see e.g. [Pre21]. The resulting encodings have more variables but most of the time the same number of models as the initial function (this is for instance the case of Tseitin encoding). Consequently, they are likely to be trivially weakly approximable from Lemma 28.

Example 18. The Tseitin encoding of $a \oplus b$ is $z \wedge (\overline{z} \vee a \vee b) \wedge (\overline{z} \vee \overline{a} \vee \overline{b}) \wedge (z \vee \overline{a} \vee b) \wedge (z \vee a \vee \overline{b})$ which we shorten as $z \wedge Ts(z, a \oplus b)$. For larger xor-sum we define the encoding inductively, for instance $a \oplus b \oplus c$ is encoded as $z' \wedge Ts(z', z \oplus c) \wedge Ts(z, a \oplus b)$. We also define the Tseitin encoding of $a \wedge b$ as $z \wedge Ts(z, a \wedge b) = z \wedge (\overline{z} \vee a) \wedge (\overline{z} \vee b) \wedge (z \vee \overline{a} \vee \overline{b})$. To illustrate that encoding can render a formula trivially weakly approximable, we give the Tseitin encoding of Boolean bilinear forms (which are the hard functions of Theorem 9). Let us encode

$$f(X,Y) = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

as the CNF formula

$$\begin{aligned} \phi(X,Y,Z) = {} & Ts(z_1, y_1 \oplus y_2) \wedge Ts(z_2, y_2 \oplus y_3) \wedge Ts(z_3, z_1 \oplus y_3) \wedge \\ & Ts(z_5, x_1 \wedge z_1) \wedge Ts(z_6, x_2 \wedge z_2) \wedge Ts(z_7, x_3 \wedge z_3) \wedge \\ & Ts(z_8, z_5 \oplus z_6) \wedge Ts(z_9, z_7 \oplus z_8) \wedge z_9 \end{aligned}$$

where $Z = \{z_1, \ldots, z_9\}$. We have that $f(X,Y) \equiv \exists Z.\phi(X,Y,Z)$ and $|sat(\phi)| = |f^{-1}(1)| = 28$. So ϕ is trivially ε-weakly approximated for any $\varepsilon \geq 28/2^{15} \approx 0,00085$. ◀

For these reasons we define a stronger notion of approximation.

Definition 34 (Strong approximation)**.** *Let $\mathcal{D}$ be a distribution of the truth assignments to X and $\varepsilon > 0$. Let f be a Boolean function over X. We say that the Boolean function $\tilde{f}$ over X is a* strong ε-approximation *of f (or strongly ε-approximates f) with respect to $\mathcal{D}$ if*

$$\Pr_{a \sim \mathcal{D}}\left[f(a) \neq \tilde{f}(a)\right] \leq \varepsilon \Pr_{a \sim \mathcal{D}}\left[f(a) = 1\right].$$

When $\mathcal{D}$ is the uniform distribution $\mathcal{U}$, the condition of strong approximability is equivalent to $|\{a \mid f(a) \neq \tilde{f}(a)\}| \leq \varepsilon |f^{-1}(1)|$. It is easy to see that strong approximation does not have the problem described in Lemma 28 for weak approximation. Strong approximation has been modelled to allow for efficient counting. In fact, a circuit in d-DNNF computing a strong ε-approximation of a function f allows approximate model counting for f with approximation factor ε.

Strong approximation has implicitly already been used in knowledge compilation. For instance it has been shown in [GKM+11] – although the authors use a different terminology – that for $\varepsilon > 0$, any knapsack function over n variables has a strong ε-approximation with respect to $\mathcal{U}$ that is computed by an OBDD of size polynomial in n and $1/\varepsilon$. The generalization of the result to Tree Augmented Naive Bayes Classifiers (TAN) [CT20] also uses strong approximations. These results are all the more significant since we know from [HTKY97, dC20] that there exist threshold functions that are only computed by OBDDs of size exponential in n.

Obviously, a strong approximation of f with respect to some distribution is also a weak approximation. Thus the statement of Theorem 9 can trivially be lifted to strong approximation. However the bilinear forms that serve as hard functions for Theorem 9 necessarily have sufficiently many models: if we are to consider only functions with few models, then they all are trivially weakly approximable. Yet we prove in the next section that there exist trivially weakly approximable functions such that the d-DNNF size of any of its strong ε-approximation is exponential in n. Our proof follows the discrepancy method but relies on the following variant of Lemma 27 for strong approximation.

Lemma 29. *Let f be a Boolean function over n variables and let $\tilde{f}$ be a strong ε-approximation of f with respect to $\mathcal{U}$. Let $\tilde{f} = \bigvee_{k=1}^{K} r_k$ be a disjoint (balanced) rectangle cover of $\tilde{f}$ and assume that there is an integer $\Delta > 0$ such that* $\mathrm{Disc}\,(f, r_k) \leq \Delta/2^n$ *for for all r_k. Then $K \geq (1-\varepsilon)|f^{-1}(1)|/\Delta$.*

Proof. The proof is essentially the same as for Lemma 27, differing only in the last lines where we use $|\tilde{f} \neq f| \leq \varepsilon |f^{-1}(1)|$ rather than $|\tilde{f} \neq f| \leq \varepsilon 2^n$. □

2.4 Large d-DNNF Size for Strong Approximations

In this section, we show a lower bound on the size of circuits in d-DNNF computing strong approximations of some functions that have weak approximations by small circuits in d-DNNF.

2.4.1 Large d-DNNF Size for Strong Approximations of Linear Codes

The functions we consider are characteristic functions of linear codes which we introduce now: a *linear code* of length n is a linear subspace of the vector space $\{0,1\}^n$. Vectors from this subspace are called *code words*. A linear code is characterized by an $m \times n$ *parity check matrix* H over $\{0,1\}$ as follows: a vector $a \in \{0,1\}^n$ is a code word if and only if $Ha = 0^m$. The *characteristic function* of a linear code is a Boolean function satisfied by exactly the code words and that can thus be represented compactly as a system of m parity constraints over n variables. Note that the characteristic function of a length n linear code whose check matrix is H has $2^{n-\mathrm{rk}(H)}$ models, where $\mathrm{rk}(H)$ denotes the rank of H. Following ideas developed in [DHJ+04], we focus on linear codes whose check matrices H have the following property: H is called *s-good* for some integer s if any sub-matrix obtained by taking at least $n/3$ columns[1] from H has rank at least s. The existence of s-good matrices for $s = m - 1$ is guaranteed by the next lemma.

Lemma 30. [DHJ+04] *Let $m = n/100$ and sample an $m \times n$ parity check matrix H uniformly at random over $\{0,1\}$. Then H is $(m-1)$-good with probability $1 - 2^{-\Omega(n)}$.*

Our interest in linear codes characterized by s-good matrices is motivated by another result from [DHJ+04] which states that the maximal size of any rectangle entailing the characteristic function of such a code decreases as s increases.

[1] Duris *et al.* [DHJ+04] limit to sub-matrices constructed from at least $n/2$ columns rather than $n/3$; however their result can easily be adapted.

Lemma 31. [DHJ+04] *Let f be the characteristic function of a linear code of length n characterized by the s-good matrix H. Let r be a rectangle with respect to a balanced variable partition and such that $r \leq f$. Then $|r^{-1}(1)| \leq 2^{n-2s}$.*

Combining Lemmas 30 and 31 with Theorem 2, one gets the following result that was already observed in [Men16]:

Theorem 10. *There exists a class of linear codes $\mathcal{C}$ such that, for any code from $\mathcal{C}$ of length n, any circuit in d-DNNF computing its characteristic function has size $2^{\Omega(n)}$.*

In the following, we will show that not only are characteristic functions hard to represent exactly as circuits in d-DNNF, they are even hard to strongly approximate.

Given the characteristic function f of a length n linear code of check matrix H, f has exactly $2^{n-\text{rk}(H)}$ models. When $\text{rk}(H) = \Omega(n)$, f satisfies the condition of Lemma 28, so for every $\varepsilon > 0$ and n large enough, f is trivially weakly ε-approximable with respect to the uniform distribution. However we will show that any strong ε-approximation $\tilde{f}$ of f with respect to the uniform distribution is computed only by circuits in d-DNNF of size exponential in n.

To show this result, we will use the discrepancy method: we are going to find a bound on the discrepancy of f on any rectangle from a disjoint balanced rectangle cover of $\tilde{f}$. Then we will use the bound in Lemma 29 and combine the result with Theorem 2 to finish the proof.

Note that it is possible that a rectangle from a disjoint rectangle cover of $\tilde{f}$ makes no false positives on f. In fact, if this is the case for all rectangles in the cover, then $\tilde{f} \leq f$ and in this case, lower bounds can be shown essentially as in the proof of Theorem 10. We assume that no rectangle makes more false positives on f than it accepts models of f, because if such a rectangle r exists in a disjoint cover of $\tilde{f}$, then deleting r leads to a better approximation of f than $\tilde{f}$. Thus it is sufficient to consider approximations and rectangle covers in which all rectangles verify $|r^{-1}(1) \cap f^{-1}(1)| \geq |r^{-1}(1) \cap f^{-1}(0)|$.

Definition 35. *Let r be a rectangle. A* core rectangle *(more succinctly a* core*) of r with respect to f is a rectangle r_{core} respecting the same partition as r and such that*

1. *$r_{core} \leq f$ and $r_{core} \leq r$,*
2. *r_{core} is maximal in the sense that there is no r' satisfying 1. such that $|r'^{-1}(1)| > |r_{core}^{-1}(1)|$.*

Note that if $r \leq f$, then the only core rectangle of r is r itself. Otherwise r may have several core rectangles. We next state a crucial lemma on the relation of discrepancy and cores whose proof we defer to later parts of this section.

Lemma 32. *Let f be the characteristic function of some length n linear code, let r be a rectangle with more true positives than false positives on f, and let r_{core} be a core rectangle of r with respect to f, then*

$$\text{Disc}\,(f, r) \leq \frac{1}{2^n} |r_{core}^{-1}(1)|.$$

Lemma 32 says the following: consider a rectangle $r_{\text{core}} \leq f$ which is a core of a rectangle r. If r accepts more models of f than r_{core}, then for each additional such model, r accepts at least one false positive. With Lemma 32, it is straightforward to show the main result of this section.

Theorem 11. *Let $0 \leq \varepsilon < 1$. There is a class of Boolean functions $\mathcal{C}$ such that for any $f \in \mathcal{C}$ on n variables*

- *f has a compact representation as a system of $O(n)$ parity constraints over $var(f)$,*

- *f is trivially weakly ε-approximable with respect to $\mathcal{U}$,*
- *any circuit in d-DNNF computing a strong ε-approximation of f with respect to $\mathcal{U}$ has size $2^{\Omega(n)}$.*

Proof. Choose $\mathcal{C}$ to be the class of characteristic functions for length n linear codes characterized by $(m-1)$-good check matrices with $m = n/100$. Existence of these functions as n increases is guaranteed by Lemma 30. Let $\tilde{f}$ be a strong ε-approximation of $f \in \mathcal{C}$ with respect to $\mathcal{U}$ and let $\bigvee_{k=1}^{K} r_k$ be a rectangle cover of $\tilde{f}$. Combining Lemma 32 with Lemma 31, we obtain $\text{Disc}\,(f, r_k) \leq 2^{-n}2^{n-2(m-1)}$. We then use Lemma 29 to get $K \geq (1-\varepsilon)2^{2m-n}|f^{-1}(1)|/4$. The rank of the check matrix of f is at most m so $|f^{-1}(1)| \geq 2^{n-m}$ and $K \geq (1-\varepsilon)2^m/4 = (1-\varepsilon)2^{\Omega(n)}$. We use Theorem 2 to conclude. □

Note that Theorem 11 is optimal with respect to ε since for $\varepsilon = 1$ there is always the trivial approximation by the constant 0 function.

It remains to show Lemma 32 in the remainder of this section to complete the proof of Theorem 11. To this end, we need another definition.

Definition 36. *Let (X_1, X_2) be a partition of $var(f)$. A* core extraction operator *with respect to f is a mapping $\mathcal{C}_f$ that maps every pair (S_1, S_2) of sets of assignments to X_1 and X_2, respectively, to a pair (S_1', S_2') such that*

a) $S_1' \subseteq S_1$ and $S_2' \subseteq S_2$,

b) assignments from $S_1' \times S_2'$ are models of f,

c) if f has no model in $S_1 \times S_2$, then $S_1' = S_2' = \emptyset$,

d) S_1' and S_2' are maximal in the sense that for every $S_1'' \subseteq S_1$ and every $S_2'' \subseteq S_2$ respecting the properties a), b) and c), we have $|S_1'||S_2'| \geq |S_1''||S_2''|$.

Intuitively S_1' and S_2' are the largest subsets one can extract from S_1 and S_2 such that assignments from $S_1' \times S_2'$ are models of f. Note that, similarly to rectangle cores, the sets S_1' and S_2' are not necessarily uniquely defined. In this case, we assume that $\mathcal{C}_f$ returns an arbitrary pair with the required properties. One can show that core extraction operators yield core rectangles, as their name suggests.

Claim 6. *Let $r = \rho_1 \wedge \rho_2$ be a rectangle respecting the partition (X_1, X_2) and denote $(A, B) = \mathcal{C}_f(\rho_1^{-1}(1), \rho_2^{-1}(1))$. Then the rectangle $\mathbb{1}_A \wedge \mathbb{1}_B$ is a core rectangle of r with respect to f.*

Proof. The rectangle $r_0 = \mathbb{1}_A \wedge \mathbb{1}_B$ respects the same variable partition as r. We now justify that it is a core rectangle for f, as defined in Definition 35:

1. $A \subseteq \rho_1^{-1}(1)$ and $B \subseteq \rho_2^{-1}(1)$ so $r_0 \leq r$ and all assignments in $A \times B$ are models of f so $r_0 \leq f$.
2. Assume r_0 is not maximal, that is, there exist $A' \subseteq \rho_1^{-1}(1)$ and $B' \subseteq \rho_2^{-1}(1)$ such that $r' = \mathbb{1}_{A'} \wedge \mathbb{1}_{B'} \leq f$ and $|r'^{-1}(1)| > |r_0^{-1}(1)|$. Then $|A'||B'| > |A||B|$, which contradicts the properties of $\mathcal{C}_f$.

□

At this point, recall that f is the characteristic function of a linear code for a $m \times n$ check matrix H.

Claim 7. *Let $r = \rho_1 \wedge \rho_2$ be a rectangle respecting the partition (X_1, X_2). Let $(A, B) = \mathcal{C}_f(\rho_1^{-1}(1), \rho_2^{-1}(1))$ and consider the core rectangle $r_{core} = \mathbb{1}_A \wedge \mathbb{1}_B$. Let $\overline{A} = \rho_1^{-1}(1) \setminus A$ and $\overline{B} = \rho_2^{-1}(1) \setminus B$. Then all assignments in $\overline{A} \times B$ and $A \times \overline{B}$ are false positives of r on f.*

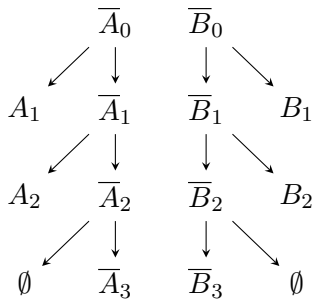


Figure 2.1: An iterative core extraction where $l = 2$.

Proof. Index the n columns of H with the variables in X (x_1 for column 1, x_2 for column 2, and so on). Let H_1 (resp. H_2) be the matrix obtained taking only the columns of H whose indices are in X_1 (resp. X_2). Clearly all assignments in $\overline{A} \times B$ and $A \times \overline{B}$ are models of r, but we will prove that they are not models of f. For every $a' \in \overline{A}$ there is $b \in B$ such that $H(a', b) = H_1 a' + H_2 b \neq 0^m$, otherwise the core rectangle would not be maximal. By definition of A and B, given $a \in A$, for all $b \in B$ we have $H(a, b) = H_1 a + H_2 b = 0^m$, so $H_2 b$ is constant over B. Therefore if $H_1 a' \neq H_2 b$ for *some* $b \in B$ then $H_1 a' \neq H_2 b$ for *all* $b \in B$. But then no assignment in $\{a'\} \times B$ can be a model of f and since a' has been chosen arbitrarily in $\overline{A}$, all assignments in $\overline{A} \times B$ are false positives. The case for $A \times \overline{B}$ follows analogously. □

For A and B defined as in Claim 7, we know that the assignments from $A \times B$ are models of f, and that those from $\overline{A} \times B$ and $A \times \overline{B}$ are not, but we have yet to discuss the case of $\overline{A} \times \overline{B}$. There may be additional models in this last set. The key to proving Lemma 32 is to iteratively extract core rectangles from $\mathbb{1}_{\overline{A}} \wedge \mathbb{1}_{\overline{B}}$ and control how many false positives are generated at each step of the iteration. To this end we define the collection $((A_i, B_i))_{i=0}^{l+1}$ as follows:

- $A_0 = \rho_1^{-1}(1)$ and $B_0 = \rho_2^{-1}(1)$,
- for $i \geq 1$, $(A_i, B_i) = \mathcal{C}_f(A_0 \setminus \bigcup_{j=1}^{i-1} A_j \ , \ B_0 \setminus \bigcup_{j=1}^{i-1} B_j)$,
- A_{l+1} and B_{l+1} are empty, but for any $i < l + 1$, neither A_i nor B_i is empty.

Denoting $\overline{A}_i = A_0 \setminus \bigcup_{j=1}^{i} A_j$ and $\overline{B}_i = B_0 \setminus \bigcup_{j=1}^{i} B_j$, we can write $(A_i, B_i) = \mathcal{C}_f(\overline{A}_{i-1}, \overline{B}_{i-1})$ (note that $\overline{A}_0 = A_0$ and $\overline{B}_0 = B_0$). Basically, we extract a core $(\mathbb{1}_{A_1} \wedge \mathbb{1}_{B_1})$ from r, then we extract a core $(\mathbb{1}_{A_2} \wedge \mathbb{1}_{B_2})$ from $(\mathbb{1}_{\overline{A}_1} \wedge \mathbb{1}_{\overline{B}_1})$, and so on until there is no model of f left in $\overline{A}_l \times \overline{B}_l$, in which case no core can be extracted from $(\mathbb{1}_{\overline{A}_l} \wedge \mathbb{1}_{\overline{B}_l})$ and $\mathcal{C}_f(\overline{A}_l, \overline{B}_l)$ returns $(\emptyset, \emptyset)$. The construction is illustrated in Figure 2.1.

Claim 8. *For any $i > 0$, all assignments in $F_i = (A_i \times \overline{B}_i) \cup (\overline{A}_i \times B_i)$ are false positives of r on f. Furthermore for every $i \neq j$ we have $F_i \cap F_j = \emptyset$.*

Proof. For the first part, it is clear from Claim 7 that assignments in $A_i \times \overline{B}_i$ and $\overline{A}_i \times B_i$ are false positives of $\mathbb{1}_{\overline{A}_{i-1}} \wedge \mathbb{1}_{\overline{B}_{i-1}}$ on f, and since $\mathbb{1}_{\overline{A}_{i-1}} \wedge \mathbb{1}_{\overline{B}_{i-1}} \leq r$, they are indeed false positives of r on f. For the second part, let $j > i > 0$, $F_i = (A_i \times \overline{B}_i) \cup (\overline{A}_i \times B_i)$ and $F_j = (A_j \times \overline{B}_j) \cup (\overline{A}_j \times B_j)$ are disjoint because both A_j and $\overline{A}_j$ are disjoint from A_i and both B_j and $\overline{B}_j$ are disjoint from B_i. □

Claim 9. *The function $\bigvee_{i=1}^{l}(\mathbb{1}_{A_i} \wedge \mathbb{1}_{B_i})$ is a disjoint rectangle cover of $r \wedge f$. Furthermore, if r respects a balanced partition, so do all rectangles in $\bigvee_{i=1}^{l}(\mathbb{1}_{A_i} \wedge \mathbb{1}_{B_i})$.*

Proof. By construction, the functions $(\mathbb{1}_{A_i} \wedge \mathbb{1}_{B_i})$ are rectangles with respect to the same partition as r. So if r is with respect to balanced partition, so do these rectangles.

For all i there is $(A_i \times B_i) \subseteq r^{-1}(1)$, so $\bigvee_{i=1}^{l}(\mathbb{1}_{A_i} \wedge \mathbb{1}_{B_i}) \leq r$. And by definition of $\mathcal{C}_f$, assignments from $A_i \times B_i$ are models of f, so $\bigvee_{i=1}^{l}(\mathbb{1}_{A_i} \wedge \mathbb{1}_{B_i}) \leq r \wedge f$.

To prove equality, assume that there exists a a model of r and f that is not a model of $\bigvee_{i=1}^{l}(\mathbb{1}_{A_i} \wedge \mathbb{1}_{B_i})$, that is, a does not belong to any $A_i \times B_i$ for $i > 0$. Then by Claim 3, a must be in $\overline{A}_l \times \overline{B}_l$ (figure 2.1 may help seeing this), but since $\overline{A}_l \times \overline{B}_l$ contains no models of f, this contradicts our assumption.

This proves that $\bigvee_{i=1}^{l}(\mathbb{1}_{A_i} \wedge \mathbb{1}_{B_i})$ is a rectangle cover of $r \wedge f$. The only thing left to prove is that the rectangles are disjoint. To see this, it is sufficient to observe that, for all $i > 1$, $A_i \subseteq \overline{A}_{i-1}$ which is disjoint from A_{i-1} and $B_i \subseteq \overline{B}_{i-1}$ which is disjoint from B_{i-1}. □

With Claim 8 and Claim 9, we can now prove Lemma 32.

Proof of Lemma 32. Claims 8 and 9 show that $\bigcup_{i=1}^{l}(A_i \times B_i) = r^{-1}(1) \cap f^{-1}(1)$ and that $\bigcup_{i=1}^{l}\left((A_i \times \overline{B}_i) \cup (\overline{A}_i \times B_i)\right) \subseteq r^{-1}(1) \cap f^{-1}(0)$, and that these unions are disjoint. First we focus on the models of f covered by r.

$$|r^{-1}(1) \cap f^{-1}(1)| = \sum_{i=1}^{l} |A_i||B_i| = |r_{\text{core}}^{-1}(1)| + \sum_{i=2}^{l} |A_i||B_i| \qquad (*)$$

where $r_{\text{core}} = \mathbb{1}_{A_1} \wedge \mathbb{1}_{B_1}$ is the first (therefore the largest) core rectangle extracted from r with respect to f. Now focus on the false positives of r on f

$$\begin{aligned} |r^{-1}(1) \cap f^{-1}(0)| &\geq \sum_{i=1}^{l} \left(|A_i||\overline{B}_i| + |\overline{A}_i||B_i|\right) \\ &\geq \sum_{i=1}^{l} \left(|A_i||B_{i+1}| + |A_{i+1}||B_i|\right) \end{aligned}$$

The maximality property of $\mathcal{C}_f$ implies $|A_i||B_i| \geq |A_{i+1}||B_{i+1}|$, but then

$$|A_{i+1}||B_{i+1}| \leq \max(|A_i||B_{i+1}|, |A_{i+1}||B_i|) \leq |A_i||B_{i+1}| + |A_{i+1}||B_i|$$

Thus using $(*)$ it follows that:

$$|r^{-1}(1) \cap f^{-1}(0)| \geq |r^{-1}(1) \cap f^{-1}(1)| - |r_{\text{core}}^{-1}(1)|.$$

By assumption, r accepts more models of f than false positives so $\text{Disc}\,(f, r) = (|r^{-1}(1) \cap f^{-1}(1)| - |r^{-1}(1) \cap f^{-1}(0)|)/2^n$ and the lemma follows directly. □

2.4.2 Large d-DNNF Size for Strong Approximations of Tseitin Formulas

The hard functions of Theorem 11 are of the form $\chi_1 \wedge \cdots \wedge \chi_m$ where each χ_i is an even parity constraint (so a constraint of the form $\sum_{x \in var(\chi_i)} x = 0 \mod 2$). Such functions are easily represented as Boolean formulas using $\oplus$ (xor) and $\wedge$ connectives. An interesting question is whether the result holds for functions given to us in another format. In this section we show a particular case of Theorem 11 where the hard functions are CNF formulas whose size are linear in the number of variables.

The idea is to use CNF formulas that represent systems of parity constraints that are more restricted than the characteristic functions of linear codes used in Theorem 11, in particular we will use specific satisfiable Tseitin formulas. Let us explain why we do not want to directly use CNF representations or CNF encodings of the hard functions used in Theorem 11. Let us first recall that the distinction between "CNF representations" and "CNF encodings" is that CNF encodings can use auxiliary variables

while CNF representations cannot: a CNF encoding of $f(X)$ is a CNF formula $\phi(X,Y)$ such that $\exists Y.\phi(X,Y) \equiv f(X)$, while a CNF representation of $f(X)$ is a CNF formula $\varphi(X)$ such that $\varphi(X) \equiv f(X)$. Our only candidate for CNF representations of $\chi_1 \wedge \cdots \wedge \chi_m$ requires representing each χ_i as a CNF of $2^{var(\chi_i)-1}$ distinct clauses where each clause contains all variables of χ_i and has an even number of literals that are non-negated variables (so an even number of literals of the form $\ell_x = x$). Let us call this the *canonical CNF representation*.

Example 19. The canonical CNF representation of $\chi : x + y + z = 0 \mod 2$ – or equivalently $\chi : x \oplus y \oplus z = 0$ – is the CNF formula $(x \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$. It contains $2^{|var(\chi)|-1} = 2^2 = 4$ clauses, each containing an even number of literals among $\{x, y, z\}$. ◀

Let n be the number of variables in $\chi_1 \wedge \cdots \wedge \chi_m$. If a single constraint contains more than $O(\log(n))$ variables, then the size of the canonical CNF representation is not polynomial in n. Proving large lower bounds on the d-DNNF size of large CNF formulas is not significant enough. Unfortunately, almost all systems of constraints used to prove Theorem 11 have only CNF representations whose size is exponential in the number of variables. Indeed these systems of constraints are characteristic functions of linear codes whose parity check matrices are $m \times n$ matrices that are $(m-1)$-good, where $m = n/100$. By Lemma 30, sampling every entry of the $m \times n$ matrix independently and uniformly at random in $\{0,1\}$ gives an $(m-1)$-good matrix with high probability. But the expected number of 1s in any row of such matrices is $n/2$ and by Chernoff bound, some row has $\Omega(n)$ 1s with high probability. In other words if the system of constraints is constructed using Lemma 30, then with high probability it contains a parity constraint over $\Omega(n)$ variables, thus the canonical representation of the system has size at least $2^{\Omega(n)}$.

Using CNF encodings instead, one can ensure that the size of the CNF remains polynomial in n, see for instance the CNF encoding in Example 18 for bilinear forms (that can easily be adapted for systems of parity constraints). But if strong approximations of $f(X)$ are impossible to represent by small circuits in d-DNNF, it is not obvious for us that the same applies to strong approximations of its CNF encodings. It could be that a CNF encoding $\phi(X,Y)$ has strong approximations $\widetilde{\phi}(X,Y)$ that have small circuits in d-DNNF, because it is not clear how to obtain in polynomial time a strong approximation of $f(X)$ from a circuit in d-DNNF computing $\widetilde{\phi}(X,Y)$. In particular it not clear that $\exists Y.\widetilde{\phi}(X,Y)$ should be a strong approximation of $f(X)$ (even though $\exists Y.\phi(X,Y) \equiv f(X)$). Even if $\exists Y.\widetilde{\phi}(X,Y)$ was a strong approximation $\tilde{f}(X)$ of $f(X)$, it is generally impossible to existentially forget variables from a circuit in d-DNNF in polynomial time while preserving determinism [Dar02a], thus knowing that $\tilde{f}(X)$ has exponential d-DNNF size would not be sufficient to derive the same for $\widetilde{\phi}(X,Y)$.

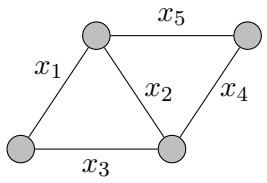
So using CNF encodings of systems of parity constraints to extend Theorem 11 seem difficult, and using their canonical CNF representations is inappropriate if these CNF representations get to large. In this section we show that we can extend Theorem 11 for specific systems of parity constraints whose canonical representations are provably small (that is, polynomial in the number of variables). In particular we use satisfiable Tseitin formulas.

Theorem 12. *Let $0 \leq \varepsilon < 1$. There is a class of satisfiable Tseitin formulas $\mathcal{T}$ such that for any $T(G) \in \mathcal{T}$*

- *$T(G)$ is CNF formula of size $O(|E(G)|) = O(|var(T(G))|)$*
- *for $|var(T(G))|$ large enough, $T(G)$ is trivially weakly ε-approximable with respect to $\mathcal{U}$,*
- *any circuit in d-DNNF computing a strong ε-approximation of $T(G)$ with respect to $\mathcal{U}$ has size $2^{\Omega(|var(T(G))|)}$.*

First, the Tseitin formulas studied in this section are all of the form $T(G,0)$, that is, the charge function assigns every vertex to 0. It is not too difficult to see such formulas as characteristic functions of linear codes.

Example 20. The Tseitin formula for the following graph



where all gray vertices all have charge 0, is

$$\begin{aligned} T(G,0) =&(\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_4} \vee x_5) \wedge (x_4 \vee \overline{x_5}) \wedge \\ &(x_1 \vee x_2 \vee \overline{x_5}) \wedge (\overline{x_1} \vee x_2 \vee x_5) \wedge (x_1 \vee \overline{x_2} \vee x_5) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_5}) \wedge \\ &(x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee x_3 \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \end{aligned}$$

It is the canonical CNF representation of the system of parity constraint $(x_1 + x_3 = 0 \mod 2) \wedge (x_4 + x_5 = 0 \mod 2) \wedge (x_1 + x_2 + x_5 = 0 \mod 2) \wedge (x_2 + x_3 + x_4 = 0 \mod 2)$, and thus of the equation

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

So this Tseitin formula represents the linear code whose parity check matrix is the 4×5 matrix above. ◀

Since the models of Tseitin formulas $T(G,0)$ are linear codes, Lemma 32 applies and, for a rectangle r with more true positives than false positives on $T(G,0)$, we have that $\mathrm{Disc}\,(T(G,0), r) \leq \frac{1}{2^{|E(G)|}} |r^{-1}_{\mathrm{core}}(1)|$ where r_{core} is a core rectangle of r with respect to $T(G,0)$.

To prove Theorem 12, we are going to make Adam and Charlotte play the adversarial rectangle cover game on a strong approximation $\tilde{f}$ of $T(G,0)$. The two players know that $\tilde{f}$ approximates $T(G,0)$ and Adam will use this knowledge when choosing partitions for the rectangles. Before that we introduce the class of graphs for the Tseitin formulas.

Lemma 33. *There is an infinite class $\mathcal{G}$ of graphs such that, for every $G \in \mathcal{G}$, all vertices of G have degree 3, G has maximum degree 3, and $tw(G) = \Omega(|V(G)|)$.*

Proof. It is known that there exists an infinite class $\mathcal{G}'$ of graphs that are 3-regular (that is, all vertices have degree 3) and whose treewidth is $\Omega(n)$, where n is the number of vertices, see for instance Theorem 5 and Proposition 1 in [GM09]. For each graph $G \in \mathcal{G}'$ whose treewidth is at least 3, we use Lemma 26 to construct a topological minor H of G that is 3-connected and that has the same treewidth has G, then we add it to $\mathcal{G}$. Since the three operations to construct topological minors (isolated vertex deletion, edge deletion and subdivision elimination) can only decrease the maximum degree of the graph, all vertices of H have degree at most 3. Furthermore, since H is 3-connected, no vertex in H has degree 0, 1 or 2, for the neighbours of such a vertex would be a separator of H of size at most 2. So $\mathcal{G}$ is indeed a class of graphs that are 3-connected, whose vertices all have degree 3, and that have linear treewidth.

Finally, we show that $\mathcal{G}$ is infinite. Since $\mathcal{G}'$ is infinite and since $tw(G) = \Omega(|V(G)|)$ for $G \in \mathcal{G}'$, we can choose an infinite sequence $G_1, G_2, \ldots$ of graphs in $\mathcal{G}'$ such that the sequence $tw(G_1), tw(G_2), \ldots$ is strictly increasing. Calling H_i the topological minor of G_i that is placed in $\mathcal{G}$, we obtain a sequence $H_1, H_2, \ldots$ of graphs in $\mathcal{G}$ such that $tw(H_i) = tw(G_i)$. So the sequence $tw(H_1), tw(H_2), \ldots$ is strictly increasing. Thus $\mathcal{G}'$ is infinite. □

We are now ready to prove Theorem 12.

Proof of Theorem 12. We choose the class $\mathcal{G}$ of graphs of Lemma 33. For $G \in \mathcal{G}$, the Tseitin formula $T(G,0)$ contains $O(|V(G)|)$ clauses of size at most 3 (because G has maximum degree 3). We have that $|E(G)| = \sum_{v \in V(G)} deg(v) = \frac{3|V(G)|}{2}$. So $|T(G,0)| = O(|E(G)|) = O(|var(T(G,0))|)$.

$T(G,0)$ is satisfiable and, by Lemma 10, since G is connected, $T(G,0)$ has $2^{|E(G)|-|V(G)|+1} = 2^{|E(G)|/3+1} = 2^{|var(T(G,0))|/3+1}$ models. By Lemma 28, for $|E(G)|$ large enough, $T(G,0)$ is trivially weakly ε-approximable (with respect to $\mathcal{U}$).

Now we prove that the d-DNNF size of a strong ε-approximation $\tilde{f}$ of $T(G,0)$ (with respect to $\mathcal{U}$) is exponential in $|var(T(G,0))|$. To this end we make Charlotte and Adam play the adversarial *disjoint* rectangle cover game to cover $\tilde{f}$. Let $k = 2tw(G)/3$.

First Charlotte chooses an assignment $a \in \tilde{f}^{-1}(1)$ and a vtree T over $var(T(G,0))$. Then Adam chooses a cut of T exactly as in the proof of Theorem 5, that is, a cut of T such that every rectangle of $T(G,0)$ respecting the resulting partition Π of $var(T(G,0))$ accepts at most $2^{|E(G)|-|V(G)|-k+1}$ models of $T(G,0)$. Next, Charlotte chooses a rectangle r with respect to Π such that $r \leq \tilde{f}$. By Lemma 32, we have that $\text{Disc}\,(T(G,0),r) \leq 2^{-|E(G)|}|r_{\text{core}}^{-1}(1)|$ where r_{core} is a core rectangle of r with respect to $T(G,0)$. Since a core rectangle respects the same partition as the initial rectangle (even if this partition is not balanced), we have that r_{core} is a rectangle with respect to Π such that $r_{\text{core}} \leq T(G,0)$. It follows that

$$\text{Disc}\,(T(G,0),r) \leq 2^{|E(G)|-|V(G)|-k+1}/2^{|E(G)|}.$$

At the end of the game, Charlotte and Adam have a set $r_1, \ldots, r_K$ of disjoint rectangles such that $r_i \leq \tilde{f}$, and r_i verifies the above inequality, and $\tilde{f} \equiv r_1 \vee \cdots \vee r_K$. So using Lemma 29 with $\Delta = 2^{|E(G)|-|V(G)|-k+1}$, then we obtain that

$$K \geq (1-\varepsilon)|sat(T(G,0))|/2^{|E(G)|-|V(G)|-k+1} = (1-\varepsilon)2^k = 2^{\Omega(tw(G))} = 2^{\Omega(|var(T(G,0))|)}$$

Finally, by Theorem 8, $\tilde{f}$ has d-DNNF size $2^{\Omega(|var(T(G,0))|)}$. □

2.5 Conclusion and Perspectives

We have formalized and studied two notions of approximations in knowledge compilation. We have called them weak and strong approximations and we have presented functions that are hard to approximate by circuits in d-DNNF with respect to these two notions. In particular, we have shown that strong approximations by circuits in d-DNNF generally require exponentially larger circuits in d-DNNF than weak approximations.

Let us sketch some directions for future research. One obvious question is to determine for which classes of functions there *are* efficient algorithms computing approximations by circuits in d-DNNF. In [CT20], it is shown that this is the case for certain Bayesian networks and for certain probability distributions over the variable assignments. It would be interesting to extend this to other settings to make approximation more applicable in knowledge compilation.

Another question is defining and analyzing more approximation notions beyond weak and strong approximations. In fact, the latter was designed to allow approximate (weighted) counting as needed in probabilistic reasoning. Are there ways of defining notions of approximation that are useful for other problems, say optimization or entailment queries? For entailment queries, notions of approximation measuring the probability that a clause (resp. a term) is entailed (resp. entails) by both the function and its approximating circuit come to mind. But one can also envision defining good approximating circuits as pairs of "sandwiching" estimators (one that entails the initial function, the other that is entailed by

it), like the approximating Horn formulas of Selman and Kautz [SK96]. Then the guarantees of the approximation could be the existence of a sequence of sandwiching estimators that converges towards the initial function (which is generally not the case of the estimators of Selman and Kautz) with a controlled growth in size between successive pairs of estimators.

A more technical question would be to determine lower bounds for circuits in DNNF, so non-deterministic circuits. In that setting, different rectangles may share the same false positives in which case our lower bound techniques based on discrepancy break down. Since we have a fairly good knowledge of techniques to prove lower bounds on the DNNF size of Tseitin formulas – which we have used to show Theorem 12 for circuits in d-DNNF – we leave the following open question as future work.

Open question 1. Is there a denumerable class of graphs $\mathcal{G}$ such that, for ε fixed, all strong ε-approximations of $T(G, 0)$ for $G \in \mathcal{G}$ have DNNF size $2^{\Omega(|var(T(G,0))|)}$?

Part II

Applications of Lower Bounds Inside and Outside Knowledge Compilation

Chapter 3

Lower Bounds on Intermediate Results in Bottom-Up Compilation

In this chapter we give a first application of our lower bound on the DNNF size of satisfiable Tseitin formulas proved in Chapter 1. We study a paradigm for compiling CNF formulas to structured languages such as str-DNNF, SDD or OBDD called *bottom-up compilation*. This paradigm has for specificity that it generates a sequence of intermediate circuits in the compilation language to construct the compiled form of the input formula. We describe a framework for bottom-up compilation that generalizes the behaviour of all practical bottom-up compilers that we are aware of. Then, relying on the lower bound of Theorem 5, we construct classes of CNF formulas that are guaranteed to generate intermediate circuits that are much larger than both the input CNF formula and the output circuit.

3.1 Bottom-Up Compilation

There are mainly two approaches for compiling a system of constraints into DNNF: *top-down compilation* and *bottom-up compilation*. The former roughly consists of remembering the trace of an exhaustive backtracking algorithm exploring the whole solution space [HD05], while the latter iteratively conjoins circuits in DNNF computing constraints of the system. Thus a bottom-up compiler needs an efficient procedure to perform (binary) conjunctions of circuits in DNNF. In practice the procedure is encompassed in the so-called *apply function* which, given two circuits in DNNF and a binary Boolean operation (from a given list), computes a circuit in DNNF that represents the function resulting from applying the operation on the two circuits. The most general fragment of DNNF known to have an efficient apply function for conjunctions is the language str-DNNF of circuits in DNNF structured by a vtree [PD08]. As a consequence, in practice, bottom-up compilers target sublanguage of str-DNNF such as SDD [Dar11, CD13] or OBDD [Bry86, Som09].

Bottom-up compilation is appealing from a theoretical perspective as we can establish a framework that describes the general behaviour of bottom-up compilers and abstract away implementation details. This framework gives us a nice environment for rigorous theoretical analysis of the efficiency of bottom-up compilers. We start by describing this framework. Let L be a language whose circuits are structured by vtrees and let L_T be the subclass of L where circuits are structured by a fixed vtree T. Let us make two assumptions on L:

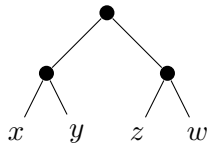
(H1) There is a polynomial-time procedure that compiles any clause C to L_T for any fixed vtree T such that $var(C) \subseteq var(T)$. We denote the circuit returned by the procedure by $\mathsf{Compile}(C, T)$.

(H2) There is a polynomial-time procedure that, for any vtree T, given two circuits $D, D' \in L_T$, returns

another circuit in L_T equivalent to $D \wedge D'$. We denote by $\mathsf{Apply}(D, D', \wedge)$ the circuit returned by the procedure.

Fact 1. *Hypotheses (H1) and (H2) are satisfied when* $L = \mathsf{SDD}$ *and when* $L = \mathsf{str\text{-}DNNF}$. *They are also satisfied when* $L = \mathsf{OBDD}$ *assuming* T *is a linear vtree.*

Note that hypotheses (H1) and (H2) are not satisfied by each sublanguage of $\mathsf{str\text{-}DNNF}$. For instance for $L = \mathsf{dec\text{-}DNNF} \cap \mathsf{str\text{-}DNNF}$, while it is known that L is complete, not every clause can be computed by a circuit in L_T when T is a non-linear vtree.

Example 21. Let $L = \mathsf{dec\text{-}DNNF} \cap \mathsf{str\text{-}DNNF}$ and consider the clause $C = x \vee y \vee z \vee w$. Let T be the vtree 

and suppose there exists a circuit D in L computing C. Let λ be the mapping from the nodes of D to the nodes of T. Every variable of C is essential so $var(C) = var(D)$. Let v be D's root node. It is already readily verified that v cannot be labelled by 0, by 1, or by a literal. If v is a $\wedge$-node, then $\lambda(v)$ must be the root node of T, for otherwise $var(D_v) = var(\lambda(v))$ would not be equal to $var(C)$. So there exist two functions $f : \{0,1\}^{\{x,y\}} \to \{0,1\}$ and $g : \{0,1\}^{\{w,z\}} \to \{0,1\}$ such that $D(x,y,w,z) \equiv f(x,y) \wedge g(w,z) \equiv C$. But then both $f(1,0) \wedge g(0,0)$ and $f(0,0) \wedge g(1,0)$ evaluate to 1, so $f(0,0) \wedge g(0,0)$ also evaluates to 1, which cannot be for otherwise we would have $C \wedge \overline{x} \wedge \overline{y} \wedge \overline{z} \wedge \overline{w} = 1$. So v is not a $\wedge$-node either. The only choice left is that v is a decision node. Say (without loss of generality) that v is a decision node labelled by x and call v_0 and v_1 its 0- and 1-child, respectively. A decision node is a compact representation of a circuit made of one $\vee$-node and two $\wedge$-nodes as shown in Figure 6a. Let u_0 and u_1 be the hidden $\wedge$-nodes with $D_{u_0} = \overline{x} \wedge D_{v_0}$ and $D_{u_1} = x \wedge D_{v_1}$. By structured decomposability, we have $\lambda(u_1) = \lambda(u_0)$. So either $\lambda(u_0)$ is the root of T, but then $var(D_{v_1}) \cup var(D_{v_0}) \subseteq \{w, z\}$ and y does not appear in D, or $\lambda(u_0)$ is the left child of the root of T, but then $var(D_{v_1}) \cup var(D_{v_0}) \subseteq \{y\}$ and w and z do not appear in D. In both cases we have that $var(D) \neq var(C)$, a contradiction. So D does not exist. ◀

Practical bottom-up compilers may implement a procedure that can modify the vtree of a circuit while preserving the function it computes, we call that procedure Restructure. Given $D \in L$ (and possibly a vtree T) Restructure returns another circuit $D' \in L$ equivalent to D, with $var(D') = var(D)$, but whose vtree may differ from that of D ($D' \in L_T$ if T is specified). We make no assumption on the running time of Restructure. One can imagine that Restructure performs an intractable task, for instance minimizing the input circuit in L (possibly in L_T if a vtree T is specified). In other words, Restructure could be used to minimize circuits, which is NP-hard even when $L = \mathsf{OBDD}$ [BW96, Sie02]. But one can also imagine that Restructure tries to find a circuit D' equivalent to D and smaller than D by performing several transformations on D that modify slightly its vtree, see for instance the SDD minimization in [CD13]. While it is important to keep in mind that bottom-up compilers have access to that procedure Restructure, the lack of running-time guarantees and the possible differences of implementation between compilers incite us to abstract that procedure. So we assume that a bottom-up compiler can arbitrarily manipulate any circuit in L in constant time as long as it returns another circuit in L that computes the same function. More formally, given any circuit $D \in L$, the compiler can generate *any* circuit $D' \in L$ equivalent to D in $O(1)$ time.

Definition 37 (Bottom-up compilation)**.** *Let* L *be a class of circuits structured by vtrees and let* $L_T \subset L$ *be the class of circuits of* L *structured by a fixed vtree* T. *Assume (H1) and (H2) hold for* L. *An* $L(\wedge, r)$-compilation *of a CNF* ϕ *to* L *is a finite sequence of circuits* $D_1, D_2, \ldots, D_N$ *in* L *culminating in* $D_N \equiv \phi$ *and such that, for each* $i \in [N]$,

$B_1 = \mathsf{Compile}(x_1 \vee \overline{x_2}, \pi)$

$B_2 = \mathsf{Compile}(\overline{x_1} \vee x_2, \pi)$

$B_3 = \mathsf{Apply}(B_1, B_2, \wedge)$

$B_4 = \mathsf{Compile}(x_1 \vee x_2 \vee x_3, \pi')$

$B_5 = \mathsf{Compile}(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}, \pi')$

$B_6 = \mathsf{Apply}(B_4, B_5, \wedge)$

$B_7 = \mathsf{Restructure}(B_3, \pi')$

$B_8 = \mathsf{Apply}(B_6, B_7, \wedge)$

with $\pi(x_1) < \pi(x_2) < \pi(x_3)$

and $\pi'(x_2) < \pi'(x_1) < \pi'(x_3)$

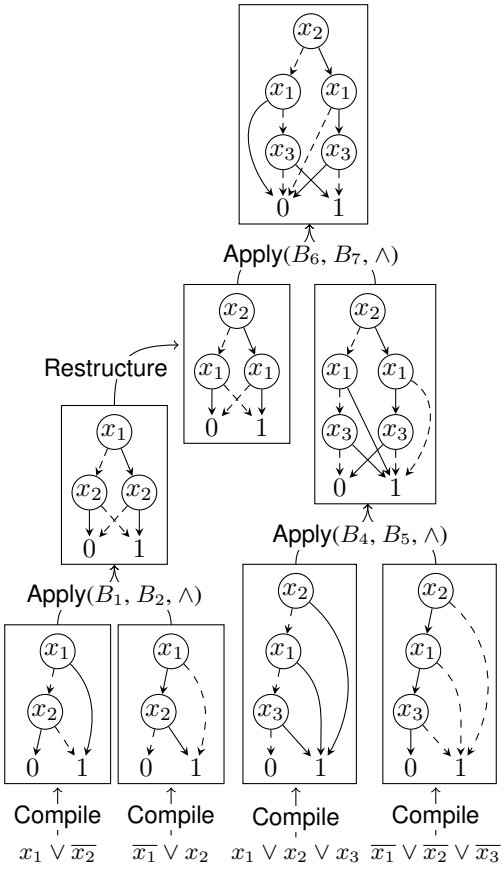


Figure 3.1: An $\mathsf{OBDD}(\wedge, r)$-compilation.

- $D_i = \mathsf{Compile}(C, T)$ *for some clause* C *of* ϕ *and some vtree* T*, or*
- $D_i = \mathsf{Apply}(D_j, D_k, \wedge)$ *with* $j, k < i$ *and* D_j *and* D_k *respect the same vtree, or*
- $D_i \equiv D_j$ *with* $j < i$ *and the circuits may respect different vtrees.*

An $L(\wedge)$-compilation *is an* $L(\wedge, r)$*-compilation where only the first two rules are available. When* ϕ *is unsatisfiable and when the satisfiability of a circuit in* L *can be tested in polynomial time, we assume that* D_N *is a single node labelled by* 0*. In this case we talk of* $L(\wedge, r)$-refutations *and* $L(\wedge)$-refutations.

Example 22. Figure 3.1 shows an bottom-up compilation to OBDD of the CNF formula $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$ where restructuring is allowed thanks to a $\mathsf{Restructure}$ procedure. Vtrees are replaced by variable orderings in the example. This is done without loss of generality since the vtrees respected by OBDDs are linear and correspond to orderings, or permutations, of X. A sequence of instructions that leads to an OBDD representing the input CNF is written on the left. ◀

As discussed before, regarding the third rule of Definition 37, the time needed to obtain D_i from D_j is not taken into account since the lower bounds that we seek are on the size of intermediate circuits. We do not even assume that $D_i \equiv D_j$ is easily verifiable in our framework. The "r" in "$L(\wedge, r)$-compilation" indicates that this third rule (the *r*estructuring rule) is allowed. In contrast, an $L(\wedge)$-compilation is a bottom-up compilation where all intermediate circuits respect a common vtree. We will focus on $\mathsf{str\text{-}DNNF}(\wedge, r)$ compilations and refutations.

We are interested in the amount of memory used when compiling CNF formulas bottom-up. Regardless of implementation details, a bottom-up compiler that fits our framework must keep every D_i in memory at some point. Note that the whole sequence $D_1, \ldots, D_N$ never has to be kept in memory entirely since earlier D_i can be deleted from memory when they are not needed any more. Thus, the size of the largest intermediate result

$$\max_{1 \leq i \leq N} |D_i|$$

is a lower bound on the space needed, a fortiori on the time taken, by the compilation process. Our framework is general enough to encapsulate practical bottom-up compilers to SDD [CD13] and to OBDD [Som09] so all lower bounds that we can prove on the size of intermediate results apply to these compilers.

One can envision a bottom-up compilation process whose input formula and whose final circuit are much smaller than the biggest intermediate circuit, i.e., $\max(|F|, |D_N|) \ll \max_{1 \leq i \leq N} |D_i|$. Then, using a bottom-up compiler appears intuitively wasteful. This is most visible when compiling unsatisfiable CNF formulas as the smallest compiled form is a single node labelled by 0 and yet, its bottom-up compilation may require a large memory space to store intermediate circuits.

The size of the D_i can differ dramatically depending on the sequence of apply operations, i.e., the order with which the clauses are conjoined. However, our main result for this chapter is to show that there are formulas that have constant-size representations as circuits in str-DNNF but for which every possible str-DNNF$(\wedge, r)$-compilation must produce big intermediate results.

3.2 State of Bottom-Up Compilation and Contribution

We review what is known on the bottom-up compilation paradigm.

Successive Clause Aggregation. In a compiler like the one included in the SDD package, the CNF formula is compiled into the target language by aggregating the clauses one by one to a main circuit that will be the compiled form in the end. This approach can be represented in our framework by assuming that all Apply operations are of the form Apply$(D, \text{Compile}(C), \wedge)$ for some clause C of the formula. For example the bottom-up compilation represented Figure 3.1 does not follow this approach since the last Apply combines two OBDDs, none of which computes a clause. This approach raises the question of the order with which the clauses are fed to the Apply. Indeed it has been noticed experimentally that choosing a "bad" order to aggregate clauses can significantly increase the size of intermediate circuits [NW07, HD04]. In particular, Narodytska and Walsh ([NW07]) introduce heuristics for choosing an order in which to conjoin the clauses to try to decrease the size of intermediate OBDDs and show experimentally that it works well when compiling certain configuration problems bottom-up.

Practical Bottom-Up Compilers. As mentioned earlier, there exist practical bottom-up compilers or packages for manipulating circuits from which bottom-up compilers can be "easily" crafted, in particular the SDD package[2] to compile into the language SDD and the CUDD package[3] to compile into the language OBDD [Som09]. Both packages provide an implementation of the Apply function for the conjunction and for other operations (disjunction, exclusive OR, etc.), and even ready-to-use compilers for the SDD package. They also enable restructuring the SDDs or OBDDs. For both packages, restructuring consists in navigating the space of possible structures to find one that decreases the size of

[2] http://reasoning.cs.ucla.edu/sdd/
[3] https://github.com/ivmai/cudd

the circuit: the CUDD package, gives access to a wide range of algorithms that modify the variable-ordering of an OBDD – for instance algorithms described in [Rud93, PSP94, PS95, BLW95] – and the SDD package gives access to a set of functions that allow to navigate the space of all possible vtrees over the variables of an SDD [CD13]. In both packages, restructuring is used for minimization purposes, and can be triggered dynamically (usually when the size of the circuit exceeds a certain threshold). As a side note, bottom-up compilers have also been developed for functions beyond Boolean functions that are out of the scope of this thesis. For instance, real-valued functions whose variables take value in $\{0, 1\}$ can be compiled into *arithmetic decision diagrams* (ADDs) using compilers like SALADD[4][Sch15] or the CUDD package as it also allows to manipulate ADDs.

Existing Lower Bounds. Lower bounds on the size of intermediate results in bottom-up compilations have been studied through OBDD-based refutation (or proof) systems, see for instance [Kra08, Seg08, TSZ10, FX13], so in the case when the formula to be compiled is unsatisfiable. We are not aware of refutation systems using circuits in str-DNNF that are not OBDD or branching programs. This is largely explained by the fact that refutation systems have the requirement that every step of a refutation must be verifiable in polynomial time. This is not the case of str-DNNF$(\wedge, r)$-refutations since testing the equivalence of two circuits in str-DNNF is generally intractable. However, testing the equivalence of two OBDDs is feasible in polynomial-time [GM94b], so OBDD$(\wedge, r)$-refutations are verifiable and fit in the definition of a proof system. Since OBDDs are generally exponentially larger than circuits in str-DNNF, lower bounds on str-DNNF$(\wedge, r)$-refutations are even more significant. Moreover, restructuring is not always allowed in the OBDD-based refutation system while it is in our framework for compilation. Indeed, the bounds in [Kra08, Seg08, TSZ10, FX13] are stated for OBDD-based refutations in which the variable order can be arbitrarily chosen at the beginning of the refutation but cannot be changed on-the-fly. Also we do not require any specific order in which the clauses are conjoined, which is a restriction used for some bounds in, e.g., [FX13].

The research that is the closest to our work has been conducted by Itsykson, Knop, Romashchenko and Sokolov in [IKRS20]. They show lower bounds for OBDD$(\wedge, r)$-refutations of specific unsatisfiable CNFs that are exponential in the number of variables. They work on the class of unsatisfiable Tseitin formulas whose underlying graphs are regular algebraic expanders, and on a class of formulas encoding the pigeon hole principle (PHP). For the latter class, they show the following

Theorem 13. [IKRS20, Theorem 3.17] *Let* $PHP_n^m = \bigwedge_{i=1}^{m}(p_{i,1} \vee \cdots \vee p_{i,n}) \wedge \bigwedge_{j=1}^{n} \bigwedge_{i \neq i'} (\overline{p_{i,j}} \vee \overline{p_{i',j}})$ *be an encoding of the pigeon hole principle for* n *holes and* m *pigeons, then every* OBDD$(\wedge, r)$ *refutation of* PHP_n^{n+1} *produces an intermediate circuit of size at least* $2^{\Omega(n)}$.

Their lower bounds for Tseitin formulas are very closely related to the results proved in this chapter as they deal with unsatisfiable Tseitin formulas $T(G, c)$ for particular graphs G. In [IKRS20], the graphs are supposed to be (d, n, α)-expander graphs, that is, graphs on n vertices, whose vertices all have degree d, and such that the absolute value of the second largest eigenvalue of the adjacency matrix, or spectral expansion[5], is not greater than αd. When the underlying graph of an unsatisfiable Tseitin formula is an (d, n, α)-algebraic expander, they show the following:

Theorem 14. [IKRS20, Theorem 3.14] *For* d *a constant large enough and* α *another constant small enough, every* OBDD$(\wedge, r)$ *refutation of an unsatisfiable Tseitin formula whose underlying graph is an* (d, n, α)*-algebraic expander produces an intermediate circuit of size at least* $2^{\Omega(n)}$.

[4] https://www.irit.fr/ Helene.Fargier/BR4CP/CompilateurSALADD.html

[5] Often the spectral expansion will be 1 minus the absolute value of the second largest eigenvalue of the adjacency matrix

A close inspection at the proofs of the two theorems above reveals a common pattern that can roughly be summed up as: look at the last `Apply` of the OBDD$(\wedge, r)$-refutation, let it be `Apply`$(B, B', \wedge)$, then show that given B and B' one can construct in polynomial time an OBDD computing a satisfiable function whose smallest equivalent circuit in OBDD has exponential size. Then B or B' must have exponential size. For Theorem 13, this function is PHP_k^k, which encodes all $k!$ ways to place k pigeons in k holes, with $k = \Omega(n)$. For Theorem 14, it is a satisfiable Tseitin formula $T(G', c)$ with G' an algebraic expander graph that have $\Omega(n)$ vertices. Then the proof of both theorems requires an exponential lower bound on the size of OBDDs computing PHP_k^k or $T(G', c)$.

Contributions. Our contributions are the following results:

Theorem 15. *Let G be a graph on n vertices whose maximum degree is at most Δ. Every* str-DNNF$(\wedge, r)$ *refutation of an unsatisfiable Tseitin formula $T(G, c)$ produces an intermediate circuit of size at least $2^{\Omega(tw(G)/\Delta)}poly(n)$.*

Theorem 16. *There is a class $\mathcal{F}$ of satisfiable CNF formulas such that every $\phi \in \mathcal{F}$ is computed by a circuit in str-DNNF of constant size, but every* str-DNNF$(\wedge, r)$ *compilation of ϕ produces an intermediate circuit of size at least $2^{\Omega(tw(\phi))}poly(|var(\phi)|)$, where $tw(\phi)$ is the primal treewidth of ϕ.*

Theorem 15 improves over Theorem 14 in two directions. First, we look at str-DNNF$(\wedge, r)$ refutations, that encompass OBDD$(\wedge, r)$ refutations and can be expected to be more space efficient since the str-DNNF size of a function is fewer, and sometimes exponentially fewer, than its OBDD size (for instance every DNF formula can be turned into a circuit in str-DNNF for any vtree in polynomial-time, but some DNF formulas have exponential OBDD size). Second, our theorem reaches the same conclusion as Theorem 14 when restricted to (d, n, α)-algebraic expander graphs, but encompass a larger class of graphs and is parameterized on the treewidth of G, rather than on the spectral expansion of G. In all fairness, given Theorem 5, the proof of Theorem 14 would also hold for str-DNNF$(\wedge, r)$ refutations with only a few modifications. So the main contribution from our side is to have shown the exponential dependence on the treewidth. Spectral expansion and treewidth are more or less related. On the one hand, the spectral expansion of a graph that has more than one connected component is 0 while the treewidth of such a graph can be linear in the number of vertices. On the other hand, in the case of constant-degree graphs the ratio $tw(G)/n$, where n is the number of vertices, is tightly linked to the spectral expansion when this one is constant. Indeed it is known that a constant spectral expansion for a constant-degree graph is equivalent to a constant vertex expansion (whose definition we omit) [HLW06], that a graph has constant vertex expansion only if its treewidth is linear in the number n of vertices, and that a graph has treewidth $\Omega(n)$ only if it has a subgraph whose vertex expansion is constant [GM09]. When restricted to (d, n, α)-algebraic expander graphs, our theorem reaches the same conclusion as Theorem 14, but our theorem generalizes Theorem 14 as it does not assume constant spectral expansion in the premises. In addition we think that it is nice that our theorem shows a dependence on a monotone parameter like treewidth, that is, if H is a subgraph of G then $tw(H) \leq tw(G)$ (spectral expansion is not a monotone property of graphs, see for instance the case of disconnected graphs).

The proof of Theorem 15 follows the pattern behind Theorem 14: use the circuits from the last `Apply` to construct in polynomial time a circuit in DNNF representing a satisfiable Tseitin formula, then use Theorem 5 to conclude that one of the two input circuits must be large. Yet, our proof has differences due to the dependence on the treewidth of the underlying graph rather than its spectral expansion. This forces us to use heavy machinery from graph theory to show intermediate results on graph bipartitions that maintain a large treewidth on both side of the partition.

The results of this section have been published in the article [dCM22b] co-authored with Stefan Mengel.

Succeeding Lower Bounds. A few months after the our contribution has been presented in [dCM22b], Itsykson, Riazanov and Smirnov proved that the $\mathsf{OBDD}(\wedge, r)$-refutations of unsatisfiable Tseitin formulas for every graph G (and not only graphs whose maximum degree is bounded) generate intermediate circuits of size at least $2^{\Omega(tw(G))}$ [IRS22, Theorem 3.1]. Itsykson et al. refer to our work in [dCM22b] and remark that their result can be extended to $\mathsf{str\text{-}DNNF}(\wedge, r)$-refutations [IRS22, Remark 3.7]. These are very neat results that generalized ours. Moreover the proof avoid complicated (but in itself interesting) machinery from graph theory that we use.

3.3 Refuting Tseitin Formulas in $\mathsf{str\text{-}DNNF}(\wedge, r)$

In this section we show that there are classes of CNF formulas of size polynomial in the number of variables and that have constant size representations in $\mathsf{str\text{-}DNNF}$, but such that $\mathsf{str\text{-}DNNF}(\wedge, r)$-compilations of these formulas create intermediate circuits of exponential size (in the number of variables). To this end we study the space complexity of $\mathsf{str\text{-}DNNF}(\wedge, r)$-compilations of unsatisfiable Tseitin formulas whose underlying graph is connected. We parameterize our bounds by the treewidth of the graph. Recall that for exponential lower bounds to be relevant, we need an input CNF formula whose length is polynomial in the number of variables and that we achieve this by restricting our study to graphs of maximum degree bounded by some constant Δ. This is a common restriction that leads to an upper bound of $|V| \times 2^{\Delta-1}$ on the number of clauses in the Tseitin formulas.

Let us first recall two properties of circuits in str-DNNF that we will use several time.

Proposition 1. *There is an algorithm that, given a circuit D in str-DNNF respecting the vtree T and an assignment a to a set of variables, returns a circuit in str-DNNF respecting T and computing $D|a$ in time $O(|D| \times |var(a)|)$. We often denote this circuit directly by $D|a$.*

Proposition 2. *There is an algorithm that, given two circuits D and D' in str-DNNF respecting the same vtree T, returns a circuit in str-DNNF respecting T and computing $D \wedge D'$ in time $O(|D| \times |D'|)$.*

We start with a simple observation that essentially says that, given a bottom-up compilation of a function f, one can easily infer a bottom-up compilation of $f|a$, for any partial assignment a. This will be useful in several upcoming proofs.

Lemma 34. *Let ϕ be a CNF formula and $D_1, \ldots, D_N$ be a $\mathsf{str\text{-}DNNF}(\wedge, r)$-compilation of ϕ. Let a be a partial assignment to $var(\phi)$, then $D_1|a, \ldots, D_N|a$ is a $\mathsf{str\text{-}DNNF}(\wedge, r)$ compilation of $\phi|a$.*

Proof. For every i between 1 and N let D'_i be $D_i|a$. By Proposition 1, $|D'_i| \leq |D_i|$ and D'_i and D_i respect a common vtree. We have $D_N \equiv \phi$, so $D'_N \equiv \phi|a$ follows. We will prove that, for every i, either D'_i is a circuit in str-DNNF computing a clause of $F|a$, or there are $j, k < i$ such that $D'_i = \mathsf{Apply}(D'_j, D'_k, \wedge)$ where all three circuits respect a common vtree, or there is $j < i$ such that $D'_i \equiv D'_j$ and the vtree of D'_i and D'_j may differ.

Take an arbitrary i between 1 and N. If D_i is a circuit in str-DNNF computing a clause C of ϕ, that is, $D_i \equiv C$, then $D'_i = D_i|a \equiv C|a$ and $C|a$ is indeed a clause of $\phi|a$. Otherwise if D_i is the circuit in str-DNNF returned by $\mathsf{Apply}(D_j, D_k, \wedge)$, then $D_i \equiv D_j \wedge D_k$ and all three circuits share a common vtree. Then $D'_i = D_i|a \equiv (D_j \wedge D_k)|a \equiv D_j|a \wedge D_k|a = D'_j \wedge D'_k$. Since the vtree is not modified by conditioning we can feed D'_j and D'_k to an Apply to obtain $D'_i = \mathsf{Apply}(D'_j, D'_k, \wedge)$. Finally in the case when D_i is equivalent to D_j with potentially a vtree modification, it is clear that $D'_i = D_i|a \equiv D_j|a = D'_j$. □

We will prove our main result, Theorem 15, later in this section after some discussion and preparations. First, note that there are graphs of bounded degree with treewidth linear in the number of vertices, see e.g. [GM09]. It follows that there are formulas where the intermediate results have exponential size.

Corollary 2. *There is a denumerable family of unsatisfiable CNF formulas such that every formula on n variables has $O(n)$ clauses and all its* str-DNNF$(\wedge, r)$*-refutations produce intermediate results of size $2^{\Omega(n)}$.*

This can also be inferred from the findings of Itsykson et al., summarized in Theorem 13 and Theorem 14, modulo modifications of their proofs to make them work for str-DNNF$(\wedge, r)$-refutations instead of OBDD$(\wedge, r)$-refutations.

Lower bounds on large intermediate results for str-DNNF$(\wedge, r)$-refutations might look somewhat unconvincing since they only talk about the compilation of unsatisfiable formulas, a setting in which costly compilation can be substituted by a usually much less expensive single call to a SAT solver. In fact, some knowledge compilers, e.g. the top-down knowledge compiler D4 [LM17], make a call to a SAT solver before trying to compile the input to avoid wasting time when compiling unsatisfiable instances. However, equipped with Lemma 34, we can lift them to satisfiable formulas that are computed by constant size circuits in str-DNNF with a simple trick.

Theorem 16. *There is a class $\mathcal{F}$ of satisfiable CNF formulas such that every $\phi \in \mathcal{F}$ is computed by a circuit in str-DNNF of constant size, but every* str-DNNF$(\wedge, r)$ *compilation of ϕ produces an intermediate circuit of size at least $2^{\Omega(tw(\phi))}poly(|var(\phi)|)$, where $tw(\phi)$ is the primal treewidth of ϕ.*

Proof. Consider a class of unsatisfiable Tseitin formulas $\mathcal{T} = \{T(G) \mid G \in \mathcal{G}\}$ for any denumerable class of graphs $\mathcal{G}$ whose degree is bounded by a constant, and let x be a fresh variable not used in any of these formulas. For each $T(G)$ let $F(G)$ be the formula $T(G)$ with the additional literal x added to all clauses. Clearly, $F(G) \equiv x \vee T(G) \equiv x$, so the smallest circuit in str-DNNF computing $F(G)$ consists of a single input node labelled by x. By Lemma 34, given a str-DNNF$(\wedge, r)$ compilation of $F(G)$, we can condition all intermediate circuits in str-DNNF on $x = 0$ to obtain a str-DNNF($\wedge$,r) refutation of $T(G)$. Since conditioning does not increase the size of circuits in str-DNNF, it follows from Theorem 15 that str-DNNF$(\wedge, r)$ compilations of $F(G)$ produce intermediate circuits of size $2^{\Omega(tw(G))}poly(|var(T(G))|) = 2^{\Omega(tw(G))}poly(|var(F(G))|)$.

Now the primal graph of $F(G)$ is the same as the primal graph of $T(G)$ with an additional vertex v_x for the variable x which is connected to all other vertices. Thus, for any tree decomposition of the primal graph of $T(G)$, one can add v_x to every bag of that decomposition to obtain a tree decomposition of the primal graph of $F(G)$. It follows that the primal treewidth $F(G)$ is at most the primal treewidth of $T(G)$ plus one. The result then follows from Lemma 12 which states that, since the degree of G is bounded by a constant, the primal treewidth of $T(G)$ is within a constant of $tw(G)$. □

As a first step towards Theorem 15, let $T(G)$ be unsatisfiable where $G = (V, E)$ is connected. We look at the very last Apply in the str-DNNF$(\wedge, r)$-refutation of $T(G)$:

$$D_N = \mathsf{Apply}(D^\ell, D^r, \wedge)$$

where $D_N \equiv 0$ and D^ℓ and D^r are two satisfiable circuits in str-DNNF structured by the same vtree. Roughly put, we proceed as follows:

1. We prove that there is a partition (A, B) of V such that *both* $G[A]$ *and* $G[B]$ have treewidth $\Omega(tw(G))$.

2. For that partition we show how to construct from D^ℓ and D^r in polynomial time a circuit D^* in str-DNNF computing a *satisfiable* Tseitin formula $T(G[A])$ or $T(G[B])$.

3. From Theorem 5 we derive that $|D^*| = 2^{\Omega(tw(G))}$ and use $|D^*| = O(|D^\ell| \times |D^r|)$ to conclude.

For convenience we denote $G_A = G[A]$ and $G_B = G[B]$. In the second step, we can not really control which of $T(G_A)$ or $T(G_B)$ is satisfiable. But the first step frees us from worrying about this: since both G_A and G_B have large treewidth, D^* has size exponential in the treewidth of G regardless of whether it represents $T(G_A)$ or $T(G_B)$.

We can already build toward the proof of Theorem 15 by assuming that the following two lemmas hold.

Lemma 35. *goodPartition Let $G = (V, E)$ be a 2-connected graph with maximum degree Δ. There is a partition (A, B) of V such that $G[A]$ is connected, $G[B]$ is 2-connected, and $\min(tw(G[A])), tw(G[B])) \geq \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$ where $\alpha > 0$ is a fixed universal constant.*

Lemma 36. *technicalLemma Let $\text{Apply}(D^\ell, D^r, \wedge)$ be the last step of a str-DNNF$(\wedge, r)$ refutation of $T(G, c)$ where G is 2-connected. Assume that there is a partition (A, B) of V such that $G[A]$ is connected, $G[B]$ is 2-connected, and both $G[A]$ and $G[B]$ have treewidth at least 2. Then there is a circuit in str-DNNF of size $O(|D^\ell| \times |D^r|)$ computing a satisfiable Tseitin formula whose underlying graph is $G[A]$ or $G[B]$ (potentially minus one vertex).*

We will also use the following result from [BK06] which we reformulate to simplify notations.

Theorem 17. [BK06] *Let G be a graph with a 1-separator u. Then $G - u$ contains a connected component $G' = (V', E')$ such that $tw(G) = tw(G[V' \cup \{u\}])$.*

Proof of Theorem 15. First, using Lemmas 35 and 36 and Theorem 5, we prove the result when G is 2-connected. Let Δ be an upper bound on the maximum degree of all our graphs. Fix a graph $G = (V, E)$ and consider the partition (A, B) of V given by Lemma 35. Let $k = tw(G)$ and $n = |E(G)|$. We can choose the constant hidden in $2^{\Omega(k)}$ of the statement so that the theorem becomes trivial whenever $\lfloor \alpha k/\Delta^2 \rfloor < 2$, hence we assume $\lfloor \alpha k/\Delta^2 \rfloor \geq 2$ in the remainder.

The conditions on (A, B) described in Lemma 36 are met so we obtain a circuit D^* in str-DNNF computing a *satisfiable* Tseitin formula $T(G^*)$ where G^* is G_A, or G_B, or G_A minus one vertex, or G_B minus one vertex. In any case we have that $tw(G^*) \geq \min(tw(G_A), tw(G_B)) - 1$. By Proposition 2 we have $|D^*| \leq \gamma \times |D^\ell| \times |D^r|$ for some $\gamma > 0$. Now Theorem 5 says that there is a constant $\beta > 0$ such that $|D^*| \geq 2^{\beta k/\Delta^3}/n$. So we have $\min(|D^\ell|, |D^r|) \geq 2^{\beta k/2\Delta^3}/(\gamma n)$. This completes the proof in the case when G is 2-connected.

Now we show how to go from the general case to the case when G is 2-connected. Assume G has a 1-separator $\{u\}$ and let $U_1, \ldots, U_s$ be the vertex sets of the connected components of G after removal of u. We know from Theorem 17 that there is some $i \in [s]$ such that $tw(G[U_i \cup \{u\}]) = tw(G)$, say $i = 1$. Now there is a proper subset $E' \subset E(u)$ such that removing E' from G yields two connected components G_A and G_B, with $U_i \cup \{u\} \subseteq A$. So $E' = E(A, B)$ and, by Lemma 11, we can choose an assignment a to E' such that $T(G)|a = T(G_A) \wedge T(G_B)$ where $T(G_B)$ is satisfiable and $T(G_A)$ is unsatisfiable.

Let a_B be a satisfying assignment of $T(G_B)$. Using Lemma 34 we can condition any str-DNNF $(\wedge,r)$-refutation of $T(G)$ on the assignment $a \cup a_B$ to obtain a str-DNNF $(\wedge,r)$ refutation of $T(G_A)$ without size increase. G_A has fewer 1-separators than G and $tw(G_A) = tw(G)$. We repeat the procedure until obtaining a str-DNNF $(\wedge,r)$ refutation of $T(G')$, where G' is a subgraph of G that has the same treewidth of G and has no 1-separator. So G' is 2-connected, and the refutation of $T(G')$ obtained is at most as large as that of $T(G)$ we have started from. □

3.4 From Unsatisfiable to Satisfiable Tseitin Formulas (Lemma 36)

Given a str-DNNF$(\wedge, r)$ compilation of a CNF formula ϕ producing a sequence $D_1, \ldots, D_N$ of circuits in str-DNNF, we call $clause(D_i)$ the set of clauses of ϕ that were used to construct D_i.

Example 23. In the OBDD$(\wedge, r)$ compilation represented in Figure 3.1, the set of clauses for the OBDD B_3 is $clause(B_3) = \{x_1 \vee \overline{x_2}, \overline{x_1} \vee x_2\}$. ◀

Recall that for a parity constraint χ, $clause(\chi)$ is the set of clauses of its equivalent canonical CNF. That is $C \in clause(\chi)$ if and only if $var(C) = var(\chi)$ and the number of non-negated literals of C (literals of the form $\ell_x = x$) is odd if χ is an odd parity constraint, and even otherwise. One can observe that if an assignment a to $var(\chi)$ falsifies a clause $C \in clause(\chi)$, then it satisfies every clause in $clause(\chi) \setminus \{C\}$.

Example 24. Let $\chi : x + y + z = 1$, this is an even parity constraint where $var(\chi) = \{x, y, z\}$ so $clause(\chi) = \{(x \vee \overline{y} \vee \overline{z}), (\overline{x} \vee y \vee \overline{z}), (\overline{x} \vee \overline{y} \vee z), (x \vee y \vee z)\}$. The assignment a to $\{x, y, z\}$ defined by $a(x) = a(y) = 1$ and $a(z) = 0$ falsifies the clause $(\overline{x} \vee \overline{y} \vee z)$ but it satisfies the remaining clauses $(x \vee \overline{y} \vee \overline{z})$, $(\overline{x} \vee y \vee \overline{z})$ and $(x \vee y \vee z)$. ◀

Definition 38. *A parity constraint χ is called* incomplete *in a CNF formula ϕ when $clause(\chi) \cap clause(\phi) \neq clause(\chi)$. Otherwise it is called* complete *in ϕ.*

Given a circuit D in str-DNNF from a str-DNNF$(\wedge, r)$ *compilation of a CNF formula ϕ, χ is called* incomplete *in D when $clause(\chi) \cap clause(D) \neq clause(\chi)$. Otherwise it is called* complete *in D.*

Incomplete constraints are defined in such a way that χ is called incomplete in ϕ even when all clauses of $clause(\chi) \setminus clause(\phi)$ are entailed by ϕ. It is readily verified that there is a constraint $\chi_v(G, c)$ that is incomplete in a subformula ϕ of $T(G, c)$ if and only if ϕ is a proper subformula of $T(G, c)$. It is also clear that for every circuit D in str-DNNF form a str-DNNF$(\wedge, r)$ compilation of $T(G, c)$, except the last one, there is a constraint $\chi_v(G, c)$ that is incomplete in D for some $v \in V(G)$.

Lemma 37. *Let $T(G, c)$ be a satisfiable Tseitin formula where G is connected, let $E' \subseteq E(G)$ and let G' be the graph G where the edges E' have been removed. If G' is connected, then for every assignment a to $X_{E'}$, $T(G, c)|a$ is satisfiable.*

Proof. For every $v \in V(G)$, let $E'(v) = E(v) \cap E'$. Let $c' : V(G') \to \{0, 1\}$ be such that $c'(v) = c(v) + \sum_{u \in E'(v)} a(x_{uv}) \mod 2$. Then $T(G, c)|a = T(G', c')$. Since G' is connected, by Lemma 9 we simply have to show that $\sum_{v \in V(G')} c'(v) = 0 \mod 2$. Note that $V(G') = V(G)$.

$$\begin{aligned}\sum_{v \in V(G)} c'(v) &= \sum_{v \in V(G)} \left(c(v) + \sum_{u \in E'(v)} a(x_{uv})\right) \mod 2 \\ &= \sum_{v \in V(G)} c(v) + 2 \times \sum_{e \in E'} a(x_e) \mod 2 \\ &= \sum_{v \in V(G)} c(v) = 0 \mod 2\end{aligned}$$

Where the last equality comes from Lemma 9 applied to $T(G, c)$. □

Lemma 38. *Let $T(G, c)$ be an unsatisfiable Tseitin formula. If G is 2-connected, then every proper subformula of $T(G, c)$ is satisfiable.*

Proof. Let F be a proper subformula of $T(G, c)$ and let $C \in clause(T(G, c)) \setminus clause(F)$. Let a be the unique assignment to $var(C)$ that falsifies C. To prove the lemma, we show that $F|a$ is a subformula of a *satisfiable* Tseitin formula. Let $v \in V(G)$ be the unique vertex such that $C \in clause(\chi_v(G, c))$. Recall that the charge function $c + \mathbb{1}_v \mod 2$ is equal to c except on v. Since G is connected, by Lemma 9, $T(G, c + \mathbb{1}_v \mod 2)$ is satisfiable.

Now let F' be defined by $clause(F') = clause(F) \setminus clause(\chi_v(G,c))$. Since a satisfies all clauses of $\chi_v(G,c)$ but C, we have that $F|a = F'|a$. Observe that F' is a subformula of $T(G, c + \mathbb{1}_v \mod 2)$ so $F'|a$ is a subformula of $T(G, c + \mathbb{1}_v \mod 2)|a$. We know that $T(G, c + \mathbb{1}_v \mod 2)|a$ is a Tseitin formula, so it remains to justify that it is satisfiable.

Let G' be the graph G where v has been removed and let $c' : V(G') \to \{0,1\}$ be the charge function defined by $c'(u) = c(u) + 1 \mod 2$ if $u \in N(v)$ and $a(x_{vu}) = 1$, and $c'(u) = c(u)$ otherwise, then $T(G, c+\mathbb{1}_v \mod 2)|a = T(G', c')$. Since G is 2-connected, we have that G' is connected, so Lemma 37 gives us that $T(G', c')$ is satisfiable. □

Observe that for $\chi_v(G,c)$ a parity constraint of the Tseitin formula $T(G,c)$ and a a partial variable assignment, $\chi_v(G,c)|a$ is a parity constraint of the Tseitin formula $T(G,c)|a$.

Lemma 39. *Let F be a subformula of $T(G,c)$ and assume that there is a partition (A,B) of V such that both G_A and G_B are connected and have treewidth at least 2. Suppose the constraint $\chi_v(G,c)$ is incomplete in F. Let $C \in clause(\chi_v(G,c)) \setminus clause(F)$ and write $C = C' \vee C''$ where C'' is the restriction of C to $X_{E(A,B)}$. Let a be an assignment to $X_{E(A,B)}$ that falsifies C''. Then $\chi_v(G,c)|a$ is incomplete in $F|a$.*

Proof. $T(G,c)|a$ is a Tseitin formula $T(G',c')$ where $V(G') = V(G)$ and $e \in E(G')$ if and only if $e \in E(G)$ and the corresponding variable is not in $var(a)$. We prove that $\chi_v(G',c')$ is incomplete in $F|a$. In particular we show that C' is a clause of $\chi_v(G',c')$ that is not in $F|a$.

First we explain why C' is in $clause(\chi_v(G',c'))$. Since $C' = C|a$, and since $C \in T(G,c)$, C' must be a clause of $T(G,c)|a = T(G',c')$. Moreover $clause(T(G',c')) = \bigcup_{v \in V} clause(\chi_v(G',c'))$ and the clauses of $\chi_v(G',c')$ are exactly the clauses of $T(G',c')$ whose set of variables is $var(\chi_v(G',c'))$. Since $var(C') = var(\chi_v(G',c'))$ it follows that $C' \in clause(\chi_v(G',c'))$.

Now suppose that $C' \in clause(F|a)$. So there is a clause $\Gamma \in clause(F)$ such that $\Gamma|a = C'$. We show that Γ is not a in $clause(\chi_v(G,c))$. Suppose we have $\Gamma \in clause(\chi_v(G,c))$. So $var(\Gamma) = var(C)$ and we write $\Gamma = \Gamma' \vee \Gamma''$ where Γ'' is the restriction of Γ to $X_{E(A,B)}$. Now $X_{E(A,B)} \cap var(\Gamma) = X_{E(A,B)} \cap var(C)$, so $\Gamma|a = \Gamma' \vee (\Gamma''|a)$. Note that a falsifies Γ'' otherwise $\Gamma|a$ would be 1, so $\Gamma|a = \Gamma' = C'$. But since $var(\Gamma'') = var(C'')$ and since a falsifies both C'' and Γ'', it follows that $C'' = \Gamma''$. Thus $\Gamma = C$, a contradiction.

Now that we know that $\Gamma \notin clause(\chi_v(G,c))$ we consider two cases. If $|var(C')| > 1$ then let $u, w \in V(G)$ be distinct vertices such that $\{x_{vu}, x_{vw}\} \subseteq var(C')$. Then $\{x_{vu}, x_{vw}\} \subseteq \Gamma$ and since $\chi_v(G,c)$ is the only constraint of $T(G,c)$ that contains both x_{vu} and x_{vw}, it follows that $\Gamma \in clause(\chi_v(G,c))$, a contradiction.

Otherwise $|var(C')| = 1$, say $var(C') = \{x_{vu}\}$. Then since in $T(G,c)$ the variable x_{vu} appears only in the clauses of $\chi_u(G,c)$ and in the clauses of $\chi_v(G,c)$, it follows that $\Gamma \in clause(\chi_u(G,c))$. But then a assigns a value to all variables of $X_{E(v)}$ except x_{uv} and to all variables of $X_{E(u)}$ except x_{uv}. Which means that uv is a connected component of G', so uv is either $G[A]$ or $G[B]$, which contradicts the fact that $tw(G[A])$ and $tw(G[B])$ are at least 2. □

The proof of Lemma 36 intuitively works by considering the following two cases:

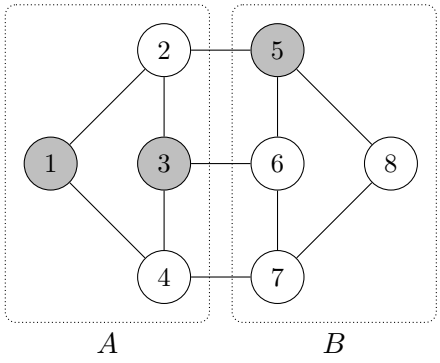
1. For *some* $D \in \{D^\ell, D^r\}$, at most two constraints for vertices in B are incomplete in D.
2. For *every* $D \in \{D^\ell, D^r\}$, at least three constraints for vertices in B are incomplete in D.

By Lemma 11 we have $T(G,c)|a = T(G_A, c_a^A) \wedge T(G_B, c_a^B)$ for every assignment a to $X_{E(A,B)}$, which we recall is the set of variables corresponding to the edges that have one endpoint in A and the other in B. In the first case, we assume that almost all constraints $\chi_v(G,c)$ for $v \in B$ are complete in D. Then

we choose an assignment a to $X_{E(A,B)}$ such that $T(G_B, c_B^a)$ is satisfiable. Each clause in $clause(D)$ either belongs to $\bigcup_{v\in A} clause(\chi_v(G,c))$ or belongs to $\bigcup_{v\in B} clause(\chi_v(G,c))$, the idea is then to "extract" in polynomial time from D a circuit in str-DNNF computing the CNF formula whose clauses are $\bigcup_{v\in B} clause(\chi_v(G,c))$. This formula is almost $T(G_B, c_B^a)$, since there are only two constraints for vertices in B that are incomplete in D. Then we conjoin to the extracted circuit the few missing constraints without increasing its size too much, so that it computes $T(G_B, c_B^a)$. In the second case, many constraints $\chi_v(G,c)$ for $v \in B$ are incomplete in both D^ℓ and D^r. In that case, we can choose an assignment a to $X_{E(A,B)}$ such that $T(G_A, c_A^a)$ is satisfiable. Then we can work on D^r and D^ℓ using transformations that can only decrease the size of the circuits and finally we conjoin the resulting circuits in polynomial time to get a circuit in str-DNNF computing $T(G_A, c_A^a)$.

We will illustrate each case using the following example.

Example 25. Consider the Tseitin formula for the following graph G. Let $V = \{1,2,3,4,5,6,7,8\}$ be its set of vertices. For every two vertices $i, j \in V$ connected by an edge in G, we will denote by x_{ij} (or x_{ji}) the variable for the edge between i and j. The charge of each vertex is indicated by a color code: gray vertices have charge 0 and white vertices have charge 1. Since the graph is connected and since there is an odd number of white vertices, by Lemma 9 the corresponding Tseitin formula is unsatisfiable.



We will use $A = \{1,2,3,4\}$ and $B = \{5,6,7,8\}$. So $X_{E(A,B)} = \{x_{25}, x_{36}, x_{47}\}$. The constraint for $i \in \{1,2,3,4,5,6,7,8\}$ is denoted by χ_i. ◀

Lemma 40 (Lemma 36, case 1)**.** *Use the notation of Lemma 36. If for some $D \in \{D^\ell, D^r\}$ at most two constraints of $T(G,c)$ for vertices of B are incomplete in D, then there is a circuit in str-DNNF of size $O(|D|)$ computing a satisfiable Tseitin formula whose underlying graph is G_B or G_B minus one vertex.*

Proof. Let F be the CNF formula such that $clause(F) = clause(D)$. D is satisfiable, so there is an assignment a to $X_{E(A,B)}$ such that $F|a$ is satisfiable. Clearly $F|a$ is a subformula of $T(G,c)|a$.

By Lemma 11, we have $T(G,c)|a = T(G_A, c_A) \wedge T(G_B, c_B)$ for some charge functions c_A and c_B over A and B, respectively. Thus we write $F|a = F_A \wedge F_B$ where F_A is a subformula of $T(G_A, c_A)$ and F_B is a subformula of $T(G_B, c_B)$. Let a_A be an assignment to $var(F_A)$ satisfying F_A (a_A must exist otherwise $F|a$ is unsatisfiable) and observe that, since $var(F_A) \cap var(F_B) = \emptyset$, we have $D|a \cup a_A \equiv (F_A \wedge F_B)|a_A \equiv F_A|a_A \wedge F_B = F_B$. Now we consider two cases:

- In the first case, $T(G_B, c_B)$ is satisfiable. Since at most two constraints $\chi_v(G,c)$ for $v \in B$ are incomplete in F, we have that at most two constraints $\chi_v(G_B, c_B)$ for $v \in B$ are incomplete in F_B. If no constraint of $T(G_B, c_B)$ is incomplete in F_B, then $F_B = T(G_B, c_B)$, then $D|a \cup a_A \equiv T(G_B, c_B)$ and by Proposition 1 we have $|D|a \cup a_A| = O(|D|)$. Now suppose there are two constraints $\chi_u(G_B, c_B)$ and $\chi_v(G_B, c_B)$ incomplete in F_B (choose $u = v$ for the case of a single incomplete

constraint). One can construct two circuits D_u and D_v in str-DNNF that compute $\chi_u(G_B, c_B)$ and $\chi_u(G_B, c_B)$, that both respect the same vtree as D, and whose size is $O(\Delta) = O(1)$. Finally, using Propositions 1 and 2 we obtain a circuit in str-DNNF computing $D|(a \cup a_A) \wedge D_u \wedge D_v \equiv T(G_B, c_B)$ whose size is $O(|D|)$.

- In the second case, $T(G_B, c_B)$ is unsatisfiable. Let $C \in clause(T(G_B, c_B)) \setminus clause(F_B)$ (C must exist, otherwise F_B and $F|a$ would be unsatisfiable), let v be the vertex of B such that $C \in \chi_v(G_B, c_B)$, and let a_v be the unique assignment to $var(C)$ that falsifies C. Let F'_B be the subformula of F_B whose clauses are $clause(F_B) \setminus clause(\chi_v(G_B, c_B))$. We have that a_v satisfies all clauses of $\chi_v(G_B, c_B)$ except C, so $F_B|a_v = F'_B|a_v$. Now observe that F'_B is a subformula of the satisfiable Tseitin formula $T(G_B, c_B + \mathbb{1}_v \mod 2)$. Thus $F'_B|a_v$ is a subformula of $T(G_B, c_B + \mathbb{1}_v \mod 2)|a_v = T(G'_B, c'_B)$ where G'_B is G_B without the vertex v, and c'_B is defined by $c'_B(u) = c_B(u) + a_v(x_{vu}) \mod 2$ if $u \in N_{G_B}(v)$ and by $c'_B(u) = c_B(u)$ otherwise. Since G_B is 2-connected, we have that G'_B is connected, so Lemma 37 gives us that $T(G'_B, c'_B)$ is satisfiable.

  Finally, $D|a \cup a_A \cup a_v$ is equivalent to $F'_B|a_v$ and at most two constraints of $T(G'_B, c'_B)$ are incomplete in $F'_B|a_v$, so using the same argument as in the first case, we find a circuit in str-DNNF of size $O(|D|)$ that computes $T(G'_B, c'_B)$.

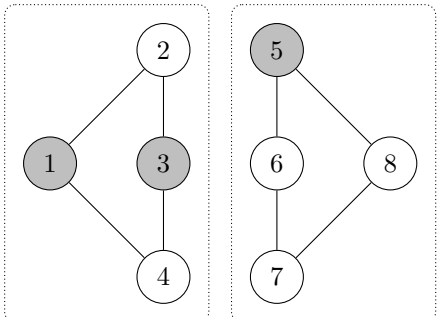
□

We illustrate the construction described in the above proof using the following example.

Example 26. For the Tseitin formula $T(G, c)$ shown in Example 25, say that we have a circuit D in str-DNNF where

$$
\begin{aligned}
clause(D) = \{&(x_{25} \vee \overline{x_{56}} \vee x_{58}), && \text{[clauses for } \chi_5\text{]}\\
&(x_{36} \vee \overline{x_{56}} \vee \overline{x_{67}}), (\overline{x_{36}} \vee x_{56} \vee \overline{x_{67}}), (\overline{x_{36}} \vee x_{56} \vee \overline{x_{67}}), (x_{36} \vee x_{56} \vee x_{67}), && \text{[clauses for } \chi_6\text{]}\\
&(x_{47} \vee \overline{x_{67}} \vee \overline{x_{78}}), (\overline{x_{47}} \vee x_{67} \vee \overline{x_{78}}), && \text{[clauses for } \chi_7\text{]}\\
&(x_{58} \vee \overline{x_{78}}), (\overline{x_{58}} \vee x_{78}), ...\} && \text{[clauses for } \chi_8\text{]}
\end{aligned}
$$

and all clauses not shown are clauses of χ_1, χ_2, χ_3 or χ_4. For the vertices of B (that is vertices 5, 6, 7 and 8) we have that χ_6 and χ_8 are complete in D and that χ_5 and χ_7 are complete in D. We are in the situation where at most two constraints for B are incomplete in D. Now consider the assignment a to $X_{E(A,B)}$ defined by $a(x_{25}) = a(x_{36}) = a(x_{47}) = 0$. Then $T(G, c)|a$ is the Tseitin formula represented by the following figure:



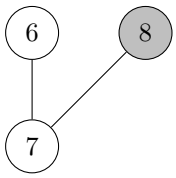
We have $T(G)|a = T(G_A) \wedge T(G_B)$ where $T(G_A)$ is satisfiable and $T(G_B)$ is unsatisfiable. Let a_A be any model of $T(G_A, c_A)$ then we have

$$clause(D|a \cup a_A) = \{(\overline{x_{56}} \vee x_{58}), (\overline{x_{56}} \vee \overline{x_{67}}), (x_{56} \vee x_{67}), (\overline{x_{67}} \vee \overline{x_{78}}), (x_{58} \vee \overline{x_{78}}), (\overline{x_{58}} \vee x_{78})\}$$

Again we have that the constraints of $T(G_B)$ for 5 and 7 are incomplete in $D|a \cup a_A$ and the constraints of $T(G_B)$ for 6 and 8 are complete in $D|a \cup a_A$. The circuit $D|a \cup a_A$ is satisfiable but $T(G_B)$ is not, so we are in the second subcase of Lemma 36, case 1. We consider the clause $(x_{56} \vee \overline{x_{58}})$ of $T(G_B)$ that is not in $clause(D|a \cup a_A)$. We denote by a_5 the unique assignment that falsifies a_5, so $a_5(x_{56}) = 0$ and $a(x_{58}) = 1$. Then we have

$$clause(D|a \cup a_A \cup a') = \{(x_{67}), (\overline{x_{67}} \vee \overline{x_{78}}), (x_{78})\}$$

Now $(x_{67}) \wedge (\overline{x_{67}} \vee \overline{x_{78}}) \wedge (x_{78})$ is a subformula of the following *satisfiable* Tseitin formula:



and we just have to conjoin $D|a \cup a_A \cup a'$ with the circuit in str-DNNF that computes the few constraints for this Tseitin formula that is incomplete in $D|a \cup a_A \cup a'$. Here we have to conjoin $D|a \cup a_A \cup a'$ with a circuit in str-DNNF that computes $x_{67} + x_{78} = 1 \mod 2$. ◀

Now we move on to the second case of Lemma 36.

Lemma 41 (Lemma 36, case 2)**.** *Use the notation of Lemma 36. If for every $D \in \{D^\ell, D^r\}$ at least three constraints of $T(G)$ for vertices of B are incomplete in D, then there is a circuit in str-DNNF of size $O(|D^\ell| \times |D^r|)$ computing a satisfiable Tseitin formula whose underlying graph is G_A.*

Proof. Let F^ℓ and F^r be the CNF formulas such that $clause(F^\ell) = clause(D^\ell)$ and $clause(F^r) = clause(D^r)$. $\mathsf{Apply}(D^\ell, D^r, \wedge)$ is the last apply of the refutation so we must have $F^\ell \wedge F^r = T(G)$. Let a be an assignment to $X_{E(A,B)}$. By Lemma 11, $T(G, c)|a$ is of the form $T(G_A, c_A^a) \wedge T(G_B, c_B^a)$. For convenience we drop some superscripts a. Since F^ℓ and F^r are subformulas of $T(G, c)$, it follows that

$$F^\ell|a = F_A^\ell \wedge F_B^\ell \quad \text{and} \quad F^r|a = F_A^r \wedge F_B^r$$

where F_A^ℓ and F_A^r are subformulas of $T(G_A, c_A)$ and F_B^ℓ and F_B^r are subformulas of $T(G_B, c_B)$.

Claim 10. *Assume there is an assignment a to $X_{E(A,B)}$ such that $T(G_A, c_A)$ and $F^\ell|a$ and $F^r|a$ are satisfiable. Then there is a circuit in str-DNNF of size $O(|D^r||D^\ell|)$ that computes $T(G_A, c_A)$.*

Proof. Using Proposition 1 we obtain a circuit $D^\ell|a \equiv F^\ell|a$ in str-DNNF of size $O(|D|^\ell)$ and another circuit $D^r|a \equiv F^r|a$ in str-DNNF of size $O(|D|^r)$. Since D^ℓ and D^r are structured by the same vtree, so are $D^\ell|a$ and $D^r|a$. By assumption $F^\ell|a$ and $F^r|a$ are satisfiable so let a_B^ℓ be a model of F_B^ℓ and let a_B^r be a model of F_B^r. Using Proposition 1 and the fact that $var(F_A^\ell) \cap var(F_B^\ell) = var(F_A^r) \cap var(F_B^r) = \emptyset$, we have that

$$D^\ell|(a \cup a_B^\ell) = F_A^\ell \quad \text{and} \quad D^r|(a \cup a_B^r) = F_A^r$$

are two circuits in str-DNNF structured by the same vtree and of respective size $O(|D^\ell|)$ and $O(|D^r|)$. It then suffices to use Proposition 2 to obtain a circuit in str-DNNF computing

$$D^\ell|(a \cup a^\ell_B) \wedge D^r|(a \cup a^r_B) \equiv F^\ell_A \wedge F^r_A = T(G_A, c_A)$$

and whose size is $O(|D^r||D^\ell|)$. □

In the rest of the proof we explain how to construct the assignment a of Claim 10. By assumption a constraint of $T(G, c)$ for a vertex $u^r \in B$ is incomplete in D^r and three constraints of $T(G, c)$ for $u^\ell, v^\ell, w^\ell \in B$ are incomplete in D^ℓ. The latter three vertices are distinct, so at least two of them are different from u^r. Suppose, without loss of generality, that $u^r \neq v^\ell$ and $u^r \neq w^\ell$. For convenience, rename $u = u^r$, $v = v^\ell$ and $w = w^\ell$. Let C_u be a clause of $\chi_u(G, c)$ that is not in $clause(D^r)$ and let C_v and C_w be clauses of $\chi_v(G, c)$ and $\chi_w(G, c)$ that are not in $clause(D^\ell)$. We write $C_u = C'_u \vee C''_u$, $C_v = C'_v \vee C''_v$ and $C_w = C'_w \vee C''_w$ where C''_u, C''_v, C''_w are the restrictions of C_u, C_v, C_w to $X_{E(A,B)}$, respectively. Note that C''_u or C''_v or C''_w may be empty. Let $E''(u)$, $E''(v)$ and $E''(w)$ be the set of edges corresponding to $var(C''_u)$, $var(C''_v)$ and $var(C''_w)$, respectively. By definition, all three sets are subsets of $E(A, B)$.

Claim 11. *We have $E(A, B) \neq E''(u) \cup E''(v)$ or $E(A, B) \neq E''(u) \cup E''(w)$.*

Proof. If $E''(u) = \emptyset$ or $E''(v) = \emptyset$ or $E''(w) = \emptyset$, then the claim holds because otherwise $E(A, B)$ would be a subset or $E(u)$, or a subset of $E(v)$, or a subset of $E(w)$, which is not possible since G is 2-connected.

Otherwise, if neither $E''(u)$ nor $E''(v)$ nor $E''(w)$ is empty, then the three sets are pairwise disjoint since $u, v, w \in B$. So if $E(A, B) = E''(u) \cup E''(v)$ were to hold, then we would have $E(A, B) \neq E''(u) \cup E''(w)$ because otherwise $E''(v) = E''(w) \neq \emptyset$ would hold, which is impossible. □

Suppose, without loss of generality, that $E(A, B) \neq E''(u) \cup E''(v)$. Let a''_u and a''_v be the assignments to $var(C''_u)$ and $var(C''_v)$ that falsify C''_u and C''_v, respectively (if C''_u is empty then so is a''_u, and if C''_v is empty then so is a''_v). Conditioning $T(G, c)$ on $a''_u \cup a''_v$ gives an unsatisfiable Tseitin formula on the graph G' obtained by removing $E''(u) \cup E''(v)$ from G. Since G, G_A, and G_B are connected, and since $E''(u) \cup E''(v)$ is a *proper* subset of $E(A, B)$, we have that G' is connected. Using Lemma 11, again since $E''(u) \cup E''(v)$ is a *proper* subset of $E(A, B)$, we have an assignment a to $X_{E(A,B)}$ that extends $a''_u \cup a''_v$ and such that $T(G, c)|a = T(G_A, c_A) \wedge T(G_B, c_B)$ where $T(G_A, c_A)$ is satisfiable and $T(G_B, c_B)$ is unsatisfiable.

Now we have $F^\ell|a \equiv F^\ell_A \wedge F^\ell_B$ and $F^r|a \equiv F^r_A \wedge F^r_B$ where F^ℓ_A and F^r_A are satisfiable (because $T(G_A, c_A)$ is satisfiable). Recall that $C_u \notin clause(D^r)$ and $C_v \notin clause(D^\ell)$. By construction, a falsifies C''_u and C''_v, moreover by assumption, both G_A and G_B are connected and have treewidth at least 2, thus by Lemma 39 the constraints $\chi_u(G, c)|a$ and $\chi_v(G, c)|a$ are incomplete in $D^r|a$ and $D^\ell|a$, respectively. Since u and v belong to B, it follows that F^ℓ_B and F^r_B are proper subformulas of $T(G_B, c_B)$. Then since G_B is 2-connected, by Lemma 38 we have that both F^ℓ_B and F^r_B are satisfiable. Finally since $var(F^\ell_A) \cap var(F^\ell_B) = var(F^r_A) \cap var(F^r_B) = \emptyset$, we have that $F^\ell|a$ and $F^r|a$ are satisfiable. At this point we invoke Claim 10 to finish the proof. □

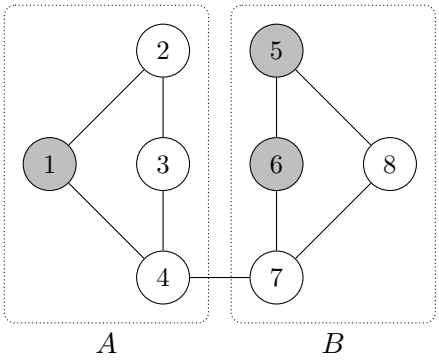
Again we use running example to describe the construction of the circuit in str-DNNF in the second case.

Example 27. For the Tseitin formula $T(G, c)$ shown in Example 25, suppose that D^ℓ and D^r have the following clauses

$$\begin{aligned} clause(D^\ell) = \{&(\overline{x_{25}} \vee x_{56} \vee x_{58}), (x_{25} \vee \overline{x_{56}} \vee x_{58}), && \text{[clauses for } \chi_5\text{]} \\ &(x_{36} \vee \overline{x_{56}} \vee \overline{x_{67}}), (\overline{x_{36}} \vee x_{56} \vee \overline{x_{67}}), && \text{[clauses for } \chi_6\text{]} \\ &(x_{47} \vee \overline{x_{67}} \vee \overline{x_{78}}), (\overline{x_{47}} \vee x_{67} \vee \overline{x_{78}}), && \text{[clauses for } \chi_7\text{]} \\ &(x_{58} \vee \overline{x_{78}}), (\overline{x_{58}} \vee x_{78}) && \text{[clauses for } \chi_8\text{]} \\ &\ldots\} && \text{[clauses for } \chi_1, \chi_2, \chi_3, \chi_4\text{]} \end{aligned}$$

$$\begin{aligned} clause(D^r) = \{&(x_{25} \vee x_{56} \vee \overline{x_{58}}), (\overline{x_{25}} \vee \overline{x_{56}} \vee \overline{x_{58}}), && \text{[clauses for } \chi_5\text{]} \\ &(\overline{x_{36}} \vee \overline{x_{56}} \vee x_{67}), (x_{36} \vee x_{56} \vee x_{67}), && \text{[clauses for } \chi_6\text{]} \\ &(\overline{x_{47}} \vee \overline{x_{67}} \vee x_{78}), (x_{47} \vee x_{67} \vee x_{78}), && \text{[clauses for } \chi_7\text{]} \\ &(x_{58} \vee \overline{x_{78}}), (\overline{x_{58}} \vee x_{78}), && \text{[clauses for } \chi_8\text{]} \\ &\ldots\} && \text{[clauses for } \chi_1, \chi_2, \chi_3, \chi_4\text{]} \end{aligned}$$

The constraint χ_8 is complete in both D^ℓ and D^r. The constraints χ_5, χ_6 and χ_7 are incomplete in both D^ℓ and D^r. We are in the second case of Lemma 36. We take a clause missing from D^ℓ and a clause missing from D^r. We choose clauses for distinct parity constraints: $(x_{25} \vee x_{56} \vee \overline{x_{58}}) \in clause(\chi_5) \setminus clause(D^\ell)$ and $(\overline{x_{36}} \vee x_{56} \vee \overline{x_{67}}) \in clause(\chi_6) \setminus clause(D^r)$. The restriction of the two clauses to $X_{E(A,B)}$ is (x_{25}) and $(\overline{x_{36}})$, respectively. We consider the assignment a'' to $\{x_{25}, x_{36}\}$ that falsifies the two restricted clauses, so $a''(x_{25}) = 0$ and $a''(x_{36}) = 1$. Then $T(G, c)|a''$ is the Tseitin formula described by the following figure:



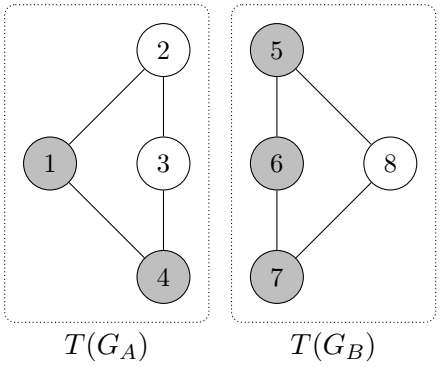
We want to assign x_{47} a value so that the Tseitin formula over A is satisfiable. So we extend a'' to the assignment a defined by $a(x_{25}) = 0$ and $a(x_{36}) = a(x_{47}) = 1$. Then we have $T(G, c)|a$ corresponding to the following figure:

$T(G_B)$ is unsatisfiable, but using a' we have ensured that some clauses of $T(G_B)$ are not in $D^\ell|a$ and that some clauses of $T(G_B)$ are not in $D^r|a$:

$$clause(D^\ell|a) = \{(\overline{x_{56}} \vee x_{58}), (x_{56} \vee \overline{x_{67}}), (x_{67} \vee \overline{x_{78}}), (x_{58} \vee \overline{x_{78}}), (\overline{x_{58}} \vee x_{78}), \text{[clauses for } A\text{]}\}$$

$$clause(D^r|a) = \{(x_{56} \vee \overline{x_{58}}), (\overline{x_{56}} \vee x_{67}), (\overline{x_{67}} \vee x_{78}), (x_{58} \vee \overline{x_{78}}), (\overline{x_{58}} \vee x_{78}), \text{[clauses for } A\text{]}\}$$

It follows that both $(\overline{x_{56}} \vee x_{58}) \wedge (x_{56} \vee \overline{x_{67}}), (x_{67} \vee \overline{x_{78}}) \wedge (x_{58} \vee \overline{x_{78}}) \wedge (\overline{x_{58}} \vee x_{78})$ and $(x_{56} \vee \overline{x_{58}}) \wedge (\overline{x_{56}} \vee x_{67}) \wedge (\overline{x_{67}} \vee x_{78}) \wedge (x_{58} \vee \overline{x_{78}}) \wedge (\overline{x_{58}} \vee x_{78})$ are satisfiable. For instance the assignment a^ℓ_B



defined by $a^\ell_B(x_{56}) = a^\ell_B(x_{67}) = a^\ell_B(x_{78}) = 0$ and $a^r_B(x_{58}) = 1$ satisfies the first CNF formula. And the assignment a^r_B defined by $a^r_B(x_{56}) = a^r_B(x_{67}) = a^r_B(x_{78}) = a^r_B(x_{58}) = 1$ satisfies the second CNF formula. Then we have that

$$clause(D^\ell|a \cup a^\ell_B) \cup clause(D^r|a \cup a^r_B) = clause(T(G_A))$$

and it only remains to conjoin $D^\ell|a \cup a^\ell_B$ and $D^r|a \cup a^r_B$ to obtain a circuit in str-DNNF computing the satisfiable Tseitin formula $T(G_A)$, which is feasible in polynomial time using Proposition 2 since D^ℓ and D^r (a fortiori $D^\ell|a \cup a^\ell_B$ and $D^r|a \cup a^r_B$) have the same vtree. ◀

3.5 Graph Bi-Partition with Large Treewidth on Both Sides

Lemma 35 is shown with the help of Theorem 18 below combined with Theorem 17. Before diving into the proof, which is quite intricate, we discuss some of the underlying graph theory, in particular the following result.

Theorem 18. *There exists a constant $0 < \alpha \leq 1$ such that, for all graphs $G = (V, E)$ with maximum degree at most Δ, there is a partition (A, B) of V such that $tw(G[A]) \geq \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$ and $tw(G[B]) \geq \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$.*

To illustrate Theorem 18, we look at the particular case of grid graphs. The $n \times n$ grid has treewidth $n-1$ and maximum degree 4. It is straightforward to partition its vertices to obtain an $n \times \lfloor n/2 \rfloor$ grid on one side, and an $n \times \lceil n/2 \rceil$ on the other. Using this partition for (A, B) we see that $G[A]$ and $G[B]$ both have an $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ induced grid and therefore both have treewidth at least $\lfloor n/2 \rfloor - 1 \geq (n-1)/4$. Of course, the constant α in the theorem is way smaller than $1/4$.

We provide some arguments to justify the veracity of Theorem 18. This theorem is an adaptation of the following result by Chekuri and Chuzhoy [CC13].

Theorem 19. *Let h and r be integers and let $G = (V, E)$. There are positive constants β and c such that, if $h^3 r \leq \beta \frac{tw(G)}{\log^c(tw(G))}$, then there is an* efficient *algorithm to partition V into $(V_1, \ldots, V_h)$, with $tw(G[V_i]) \geq r$ true for all $i \in [h]$.*

Theorem 18 is almost a subcase of Theorem 19 with $h = 2$. The only problem is that in Theorem 18, r would be roughly $\alpha tw(G)/\Delta^2$, and thus be too big for Theorem 18 where it must be fewer that $O(tw(G)/\log^c(tw(G)))$. A careful examination of Chekuri and Chuzhoy's proof shows that the log-divisor has two reasons: (1) a preprocessing of G to decrease its degree and (2) the use of an approximation algorithm to make the partition efficiently computable. Since we work with graphs of bounded

degree and only care about the existence of a partition and not its computation, we can adapt the proof for $h = 2$ and make some other adjustments to get rid of the $\log^c(tw(G))$ to obtain Theorem 18.

The proof of Theorem 18 is split into several subsections. The first two subsections introduce preliminary notions, then in the next subsection we give two lemmas that help proving Theorem 18, and the remaining subsections contain the proofs of these two lemmas.

We define the following numerical constants: $\gamma = 1/2000$, $\beta = 6/\gamma = 12000$ and $\alpha = 1/(200\beta) = 1/2400000$ the constant from Theorem 18. When the set of vertices, the set of edges, or the maximum degree of a graph G is not specified, we denote it by $V(G)$, $E(G)$, and $\Delta(G)$, respectively. We assume $\frac{\alpha tw(G)}{\Delta^2} \geq 1$, otherwise the theorem is trivial.

3.5.1 Well-linkedness

In a graph G, a set $S \subseteq V(G)$ is called *well-linked* when, for every pair $X, Y \subseteq S$ such that $|X| = |Y|$, there exist $|X|$ vertex-disjoint paths from X to Y. Note that X and Y are not necessarily distinct and that paths of size zero are allowed. The well-linked number of G, denoted $wl(G)$, is the size of the largest well-linked set in G. It is known that $tw(G) \leq wl(G) + 1 \leq 3 \times tw(G)$ [HW17].

Lemma 42. *Let G be a simple graph. There is a set $S^* \subseteq V(G)$ of size $tw(G) \leq |S^*| + 1 \leq 3 \times tw(G)$ such that, for every partition (A, B) of $V(G)$, it holds that*

$$|E(A, B)| \geq \min(|A \cap S^*|, |B \cap S^*|).$$

Proof. Let S^* be the largest well-linked set in G and consider a partition (A, B) of $V(G)$. The bounds on $|S^*|$ stem from the relation between $wl(G)$ and $tw(G)$. Let $S^*_A = S^* \cap A$ and $S^*_B = S^* \cap B$. Assume, without loss of generality, that $|S^*_A| \leq |S^*_B|$. Take an arbitrary subset $Z \subseteq S^*_B$ of size $|S^*_B| - |S^*_A|$ and let $X = S^*_A \cup Z$ and $Y = S^*_B$. Then we have $X, Y \subseteq S^*$ and $|X| = |Y|$.

By definition of S^* there are at least $|X| = |S^*_A| + |Z|$ vertex-disjoint paths from X to Y, $|S^*_A|$ among them start from $S^* \cap A$ and end in $S^* \cap B$. Since $A \cap B = \emptyset$, each of these paths has size at least 1, and since they are vertex-disjoint, there are at least $|S^*_A| = |S^* \cap A| = \min(|S^* \cap A|, |S^* \cap B|)$ edges going from A to B. Thus $|E(A, B)| \geq |S^*_A| = |S^* \cap A| = \min(|S^* \cap A|, |S^* \cap B|)$. □

In the remainder of this section, S^* is the subset of V described by Lemma 42, $k = |S^*|$ and $r = \frac{2\alpha k}{\Delta^2}$. Observe that $r \geq \frac{\alpha tw(G)}{\Delta^2} \geq 1$.

Lemma 43. [CC13] *Let G be a simple graph whose maximum degree is Δ, assume that there is $S \subseteq V(G)$ and $\gamma \in [0, 1]$ such that, for every partition (A, B) of $V(G)$, it holds that $|E(A, B)| \geq \gamma \min(|S \cap A|, |S \cap B|)$. Then $tw(G) \geq \frac{\gamma |S|}{3\Delta} - 1$.*

3.5.2 Acceptable partitions

In a graph G, given a subset $S \subseteq V(G)$, we denote by $out_G(S)$ the set of edges of G that have one endpoint in S and the other in $V(G) \setminus S$. We drop the G subscript when it is clear from context which graph we are working with. Let $\mathcal{C} = (V_1, \ldots, V_s)$ be a partition of $V(G)$ (s is arbitrary). We denote by $G_{\mathcal{C}}$ the multigraph with s vertices $\{\nu_1, \ldots, \nu_s\}$ such that for each $i \neq j$ (no self-loop) there are $|E(V_i, V_j)|$ edges between ν_i and ν_j. One can construct $G_{\mathcal{C}}$ from G by contracting every subset of vertices V_i into a single vertex, which is ν_i. We then call $G_{\mathcal{C}}$ a *contracted multigraph*. We abuse the notations and consider the vertices of $G_{\mathcal{C}}$ to be $V_1, \ldots, V_s$ (so $\mathcal{C}$ can be seen as $V(G_{\mathcal{C}})$). It will be important that all multigraphs $G_{\mathcal{C}}$ considered for different partitions $\mathcal{C}$ have enough edges and have maximum degree bounded by $\Delta' = \beta r \Delta^2$ where $\beta = 12000$. We will show that these properties are guaranteed when the partition $\mathcal{C}$ is *acceptable*.

Definition 39. *A partition $\mathcal{C} = (V_1, \ldots, V_s)$ of $V(G)$ is called* acceptable *when $|out(V_i)| \leq \Delta'$ and $|V_i \cap S^*| \leq \frac{k}{2}$ hold for all $i \in [s]$.*

Acceptable partitions of $V(G)$ exist, the simplest one is $\mathcal{C} = \{\{v\} \mid v \in V(G)\}$. Indeed we have $|\{v\} \cap S^*| \leq 1 \leq \frac{k}{2}$ since $k \geq 2$ and $|out(\{v\})| = \deg(v) \leq \Delta \leq \beta r \Delta^2 = \Delta'$ since $\beta, r \geq 1$.

Claim 12. *Let $\mathcal{C}$ be an acceptable partition of $V(G)$. Then $\Delta(G_\mathcal{C}) \leq \Delta'$.*

Proof. Let $\mathcal{C} = \{V_1, \ldots, V_s\}$ and let ν_i be the vertex of $G_\mathcal{C}$ corresponding to V_i. Then $\deg(\nu_i) = |out(V_i)| \leq \Delta'$. □

Claim 13. *Let $\mathcal{C}$ be an acceptable partition of $V(G)$. Then $|E(G_\mathcal{C})| \geq \frac{k}{4}$.*

Proof. Let $\mathcal{C} = \{V_1, \ldots, V_s\}$. Assume that $|V_1 \cap S^*| \geq |V_2 \cap S^*| \geq \cdots \geq |V_s \cap S^*|$ holds. We have $\sum_{i=1}^{s} |V_i \cap S^*| = |S^*| = k$. Since $|V_1 \cap S^*| \leq \frac{k}{2}$ we can find $l \in [s-1]$ the largest integer such that $\sum_{i=1}^{l} |V_i \cap S^*| \leq \frac{3k}{4}$. We clearly have $\sum_{i=l+1}^{s} |V_i \cap S^*| \geq \frac{k}{4}$. We also have $\sum_{i=1}^{l} |V_i \cap S^*| \geq \frac{k}{4}$ for otherwise we would have $|V_{l+1} \cap S^*| = \sum_{i=1}^{l+1} |V_i \cap S^*| - \sum_{i=1}^{l} |V_i \cap S^*| > \frac{3k}{4} - \frac{k}{4} = \frac{k}{2}$, which contradicts the choice of l.

Let $A = V_1 \cup \cdots \cup V_l$ and $B = V_{l+1} \cup \cdots \cup V_s$. By construction we have $|A \cap S^*| \geq \frac{k}{4}$ and $|B \cap S^*| \geq \frac{k}{4}$. And by definition of S^* we have $|E(A,B)| \geq \min(|A \cap S^*|, |B \cap S^*|) \geq \frac{k}{4}$. When contracting the V_i to obtain $G_\mathcal{C}$, the edges $E(A,B)$ survive, so $|E(G_\mathcal{C})| \geq |E(A,B)| \geq \frac{k}{4}$. □

3.5.3 Proof of Theorem 18

The proof of Theorem 18 boils down to the following two lemmas, which we will show in later sections.

Lemma 44. *Let $\mathcal{C}$ be an acceptable partition of $V(G)$. Then there is a partition $(\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$ of $\mathcal{C}$ such that, for all $i \in \{1,2,3\}$*

$$|E(G_\mathcal{C}[\mathcal{U}_i])| \geq \frac{|E(G_\mathcal{C})|}{180}.$$

Recall that $\mathcal{C} = (V_1, \ldots, V_s)$, where $V_1 \cup \cdots \cup V_s = V(G)$. From a partition $(\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$ of $\mathcal{C}$ we define a corresponding partition (U_1, U_2, U_3) of $V(G)$ where U_j is obtained by uncontracting all nodes in $\mathcal{U}_j$. More formally, $U_j = \bigcup_{V_i \in \mathcal{U}_j} V_i$. Note that $\mathcal{U}_j$ is partition of U_j.

Lemma 45. *Let $\mathcal{C} = (V_1, \ldots, V_s)$ be an acceptable partition of $V(G)$. Let $\mathcal{U} \subseteq \mathcal{C}$ such that $|E(G_\mathcal{C}[\mathcal{U}])| \geq \frac{|E(G_\mathcal{C})|}{180}$ and let $U = \bigcup_{V_i \in \mathcal{U}} V_i$. If $tw(G[U]) < \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$ and $|U \cap S^*| \leq \frac{k}{2}$, then there is a partition $\mathcal{U}'$ of U such that $\mathcal{C}' = (\mathcal{C} \setminus \mathcal{U}) \cup \mathcal{U}'$ is an acceptable partition of $V(G)$ and such that $|E(G_{\mathcal{C}'})| < |E(G_\mathcal{C})|$.*

With Lemmas 44 and 45, we can show Theorem 18 relatively easily.

Proof of Theorem 18. We know that acceptable partitions of $V(G)$ exist. Let $\mathcal{C}$ be the acceptable partition of $V(G)$ such that $|E(G_\mathcal{C})|$ is minimal, that is, for all other acceptable partitions $\mathcal{C}'$ we have $|E(G_{\mathcal{C}'})| \geq |E(G_\mathcal{C})|$. Let $(\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$ be the partition of $\mathcal{C}$ given by Lemma 44 and (U_1, U_2, U_3) be the corresponding partition of $V(G)$.

Assume, without loss of generality, that $tw(G[U_1]) \geq tw(G[U_2]) \geq tw(G[U_3])$. If $tw(G[U_2]) \geq \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$ then we take $A = U_1$ and $B = U_2 \cup U_3$ and we are done. Suppose otherwise that $tw(G[U_3]) \leq tw(G[U_2]) < \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$. There must be $|U_j \cap S^*| \leq \frac{|S^*|}{2} = \frac{k}{2}$ for some $j \in \{2,3\}$, say for $j = 3$. But then Lemma 45 gives a partition $\mathcal{U}_3'$ of U_3 such that $\mathcal{C}' = \mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{U}_3'$ is an acceptable partition of $V(G)$ such that $|E(G_{\mathcal{C}'})| < |E(G_\mathcal{C})|$, a contradiction. □

3.5.4 Proof of Lemma 44

Lemma 44 is a consequence of the more general Lemma 46 below. Lemma 46 essentially says that if the number of edges of a multigraph is greater than a factor of its degree, then there is a partition of its vertices into three parts such that many edges remain in every part.

Lemma 46. *Let H be a multigraph with no self-loops and with $|E(H)| \geq 25\Delta(H)$. Then there is a partition $V(H) = (V_r, V_b, V_g)$ (red,blue,green) such that $|E(H[V_c])| \geq |E(H)|/180$ holds for all $c \in \{r, b, g\}$.*

Proof of Lemma 44. $\mathcal{C}$ is an acceptable partition, so $|E(G_{\mathcal{C}})| \geq k/4$ and $\Delta(G_{\mathcal{C}}) \leq \Delta' = \beta r \Delta^2 = 2\beta\alpha k \leq k/100 \leq |E(G_{\mathcal{C}})|/25$. Now Lemma 44 is a direct application of Lemma 46 with $H = G_{\mathcal{C}}$. □

The proof of Lemma 46 is probabilistic. It uses the **Paley-Zygmund inequality** which, given a non-negative random variable Z with finite variance and $\theta \in (0, 1]$, is

$$\Pr[Z \geq \theta \mathrm{E}[Z]] \geq (1-\theta)^2 \frac{\mathrm{E}[Z]^2}{\mathrm{E}[Z^2]}$$

Proof of Lemma 46. Let $m = |E(H)|$, $D = \Delta(H)$ and $\eta = \frac{1}{25}$. By assumption $D \leq \eta m$. The vertices of H are assigned a color in $\{r(ed), b(lue), g(reen)\}$ uniformly at random. An edge $e = uv$ is red when both its endpoints are red, it is blue when both its endpoints are blue, it is green when both its endpoints are green, and otherwise it has no color. Let X_e^c be the event that the edge e has color $c \in \{r, b, g\}$ and let $E_c = \sum_{e \in E(H)} X_e^c$ be the number of edges colored with c after random coloring of the vertices. It is clear that $\Pr[X_e^c] = \frac{1}{9}$ and that $\mathrm{E}[E_c] = \frac{m}{9}$. The statement of the lemma follows from proving that

$$\Pr\left[E_r < \frac{m}{180} \text{ or } E_b < \frac{m}{180} \text{ or } E_g < \frac{m}{180}\right] < 1.$$

By the union bound, it is sufficient to show that $\Pr\left[E_c < \frac{m}{180}\right] < \frac{1}{3}$ holds, for c fixed in $\{r, b, g\}$. We will use Paley-Zygmund inequality to prove $\Pr\left[E_c \geq \frac{m}{180}\right] > \frac{2}{3}$, so we need to compute $\mathrm{E}\left[E_c^2\right]$.

$$\mathrm{E}\left[E_c^2\right] = \mathrm{E}\left[\left(\sum\nolimits_e X_e^c\right)^2\right] = \sum_{e \in E(G)} \sum_{e' \in E(G)} \mathrm{E}[X_e^c X_{e'}^c] = \sum_{e \in E(G)} \sum_{e' \in E(G)} \Pr[X_e^c \text{ and } X_{e'}^c]$$

Let us look at $\Pr\left[X_e^c \text{ and } X_{e'}^c\right]$. Let u_e and v_e be the endpoints of e.

- If e' has the same endpoints as e, and $e' \in E(u_e) \cap E(v_e)$, then the probability is $\Pr\left[X_e^c \text{ and } X_{e'}^c\right] = \Pr[X_e^c] = \frac{1}{9}$.
- If e' shares exactly one endpoint with e, and $e' \in (E(u_e) \cup E(v_e)) \setminus (E(u_e) \cap E(v_e))$, then the probability is $\Pr\left[X_e^c \text{ and } X_{e'}^c\right] = \frac{1}{27}$.
- If e' has no endpoint in common with e, and $e' \in E(H) \setminus (E(u_e) \cup E(v_e))$, then the probability is $\Pr\left[X_e^c \text{ and } X_{e'}^c\right] = \frac{1}{81}$.

So we obtain

$$\begin{aligned}\mathrm{E}\left[E_c^2\right] &= \sum_{e \in E(G)} \left(\frac{|E(u_e) \cap E(v_e)|}{9} + \frac{|E(u_e)| + |E(v_e)| - 2|E(u_e) \cap E(v_e)|}{27} \right. \\ &\qquad \left. + \frac{m - (|E(u_e)| + |E(v_e)| - |E(u_e) \cap E(v_e)|)}{81} \right) \\ &= \frac{m^2}{81} + \frac{2}{81} \sum_{e \in E(G)} |E(u_e) \cap E(v_e)| + \frac{2}{81} \sum_{e \in E(G)} (|E(u_e)| + |E(v_e)|)\end{aligned}$$

Recall that we are dealing with multigraphs, so $\sum_{e\in E(G)} |E(u_e) \cap E(v_e)|$ is not necessarily m. But we do have that $|E(u_e) \cap E(v_e)|$, $|E(u_e)|$ and $|E(v_e)|$ are all at most $D \leq \eta m$ so

$$\mathrm{E}\left[E_c^2\right] \leq \frac{m^2}{81} + \frac{6mD}{81} \leq \frac{m^2}{81} + \frac{6\eta m^2}{81}.$$

Finally we apply Paley-Zygmund inequality:

$$\Pr\left[E_c \geq m/180\right] \geq \left(1 - \frac{1}{20}\right)^2 \frac{\mathrm{E}\left[E_c\right]^2}{\mathrm{E}\left[E_c^2\right]} \geq \left(1 - \frac{1}{20}\right)^2 \frac{1}{1+6\eta} > \frac{2}{3}$$

□

3.5.5 Proof of Lemma 45

Lemma 45. *Let $\mathcal{C} = (V_1, \ldots, V_s)$ be an acceptable partition of $V(G)$. Let $\mathcal{U} \subseteq \mathcal{C}$ such that $|E(G_{\mathcal{C}}[\mathcal{U}])| \geq \frac{|E(G_{\mathcal{C}})|}{180}$ and let $U = \bigcup_{V_i \in \mathcal{U}} V_i$. If $tw(G[U]) < \lfloor \frac{\alpha tw(G)}{\Delta^2} \rfloor$ and $|U \cap S^*| \leq \frac{k}{2}$, then there is a partition $\mathcal{U}'$ of U such that $\mathcal{C}' = (\mathcal{C} \setminus \mathcal{U}) \cup \mathcal{U}'$ is an acceptable partition of $V(G)$ and such that $|E(G_{\mathcal{C}'})| < |E(G_{\mathcal{C}})|$.*

We are going to construct the new partition $\mathcal{U}'$ of U. Recall that $\Delta' = \beta r \Delta^2$ is the maximum degree of the contracted multigraph and that $\frac{6\Delta^2 r}{\Delta'} = \frac{6}{\beta} = \gamma < 1$. From Lemma 43 we deduce that for all $S \subseteq U$ with $|out(S)| \geq \Delta'$, there is a partition (A_U, B_U) of U, such that $|E(A_U, B_U)| < \gamma \min(|A_U \cap S|, |B_U \cap S|)$, for otherwise we would get

$$tw(G[U]) \geq \frac{\gamma|S|}{3\Delta} - 1 \geq \frac{\gamma|out(S)|}{3\Delta^2} - 1 \geq \frac{\gamma\Delta'}{3\Delta^2} - 1 = 2r - 1 \geq r \geq \frac{\alpha tw(G)}{\Delta^2}$$

where we have used that $|S| \geq |out(S)|/\Delta$ and $r = 2\alpha k/\Delta^2 \geq \alpha tw(G)/\Delta^2 \geq 1$.

The Split Function. We define a routine $Split(Y)$ whose input is a subset $Y \subseteq U$ with $|out(Y)| \geq \Delta'$. $Split(Y)$ first chooses the smallest subset $S \subseteq Y$ whose vertices are endpoints of edges in $out(Y)$, and such that $|out(S) \cap out(Y)| \geq \Delta'$. Since $|S| \geq |out(S)|/\Delta$, it follows that $|S| \geq |out(S)|/\Delta \geq \Delta'/\Delta > 1$. Then $Split(Y)$ returns a partition (A_Y, B_Y) of Y such that $|E(A_Y, B_Y)| < \gamma \min(|S \cap A_Y|, |S \cap B_Y|)$. We know such a partition exists because otherwise $tw(G[U]) \geq tw(G[Y]) \geq r$ would hold. We always assume that $|out(A_Y)| \leq |out(B_Y)|$. Observe that neither $S \cap A_Y$ nor $S \cap B_Y$ is empty.

Lemma 47. *Let $(A_Y, B_Y) = Split(Y)$, then*

$$|E(A_Y, B_Y)| < \gamma\Delta' \tag{3.1}$$

and

$$|E(A_Y, B_Y)| < \gamma \min(|out(Y) \cap out(A_Y)|, |out(Y) \cap out(B_Y)|) \tag{3.2}$$

Proof. Consider the subset $S \subseteq Y$ chosen by $Split(Y)$. Every $v \in S$ is the endpoint of an edge in $out(Y)$ by definition, so $|out(S \cap A_Y) \cap out(Y)| \geq |S \cap A_Y|$ and $|out(S \cap B_Y) \cap out(Y)| \geq |S \cap B_Y|$. Thus $|out(A_Y) \cap out(Y)| \geq |S \cap A_Y|$ and $|out(B_Y) \cap out(Y)| \geq |S \cap B_Y|$ hold. $Split(Y)$ returns the partition (A_Y, B_Y) such that $|E(A_Y, B_Y)| < \gamma \min(|S \cap A_Y|, |S \cap B_Y|)$. Combining this inequality with the ones we have just obtained gives (3.2).

We have $|out(S \cap A_Y) \cap out(Y)| < \Delta'$ and $|out(S \cap B_Y) \cap out(Y)| < \Delta'$, for otherwise $|S|$ would not be minimal. Thus $|E(A_Y, B_Y)| < \gamma \min(|S \cap A_Y|, |S \cap B_Y|) < \gamma \min(|out(S \cap A_Y) \cap out(Y)|, |out(S \cap B_Y) \cap out(Y)|) < \gamma\Delta'$. □

Algorithm 1: BetterPartition(U)

Let $P = (U)$
while there exists $Y \in P$ such that $|out(Y)| \geq \Delta'$ **do**
  Let $(A_Y, B_Y) = Split(Y)$
  Remove Y from P and add A_Y and B_Y to P
end
$\mathcal{U}' = \emptyset$
for $Y \in P$ **do**
  Add all connected components of $G[Y]$ to $\mathcal{U}'$
end
Return $\mathcal{U}'$

Lemma 48. *Let $(A_Y, B_Y) = Split(Y)$. Then $|out(A_Y)| \leq \frac{|out(Y)|}{2(1-\gamma)}$ and $|out(B_Y)| \leq |out(Y)|$.*

Proof. By definition of $Split$ we have $|out(A_Y)| \leq |out(B_Y)|$ and $|E(A_Y, B_Y)| < \gamma|out(A_Y) \cap out(Y)|$. For the first part, observe that $|out(A_Y)| + |out(B_Y)| - 2|E(A_Y, B_Y)| = |out(Y)|$ so

$$\begin{aligned} |out(Y)| &\geq 2|out(A_Y)| - 2|E(A_Y, B_Y)| \\ &\geq 2|out(A_Y)| - 2\gamma|out(A_Y) \cap out(Y)| \\ &\geq 2(1-\gamma)|out(A_Y)| \end{aligned}$$

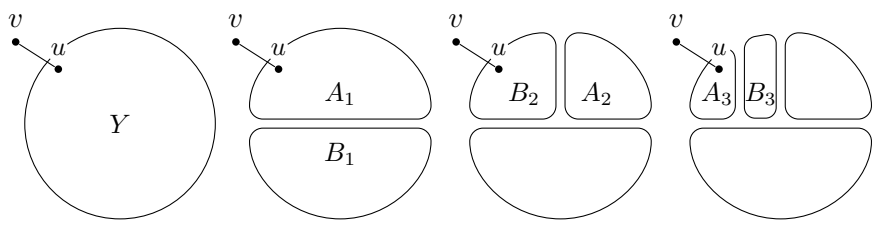
For the second part, observe that $|out(B_Y)| = |out(B_Y) \cap out(Y)| + |E(A_Y, B_Y)|$ so

$$\begin{aligned} |out(B_Y)| &\leq |out(B_Y) \cap out(Y)| + \gamma|out(A_Y) \cap out(Y)| \\ &\leq \gamma|out(Y)| + (1-\gamma)|out(B_Y) \cap out(Y)| \\ &\leq \gamma|out(Y)| + (1-\gamma)|out(B_Y)| \end{aligned}$$

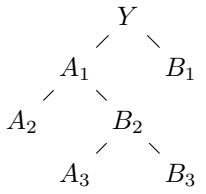
and therefore $\gamma|out(B_Y)| \leq \gamma|out(Y)|$ holds. □

A Partition Algorithm. Recall that we have a partition $\mathcal{U}$ of $U \subseteq V(G)$ and that we want to replace it with an new partition $\mathcal{U}'$ such that $\mathcal{C}' = (\mathcal{C} \setminus \mathcal{U}) \cup \mathcal{U}'$ is an acceptable partition of $V(G)$ with $|E(G_{\mathcal{C}'})| < |E(G_{\mathcal{C}})|$. The new partition $\mathcal{U}'$ is given by the algorithm BetterPartition(U). The algorithm starts from the partition $P = (U)$ of size 1. Then, as long as P has a component Y with a border too big (i.e., $|out(Y)| \geq \Delta'$), the algorithm calls $Split$ to divide Y in two parts and replaces Y by the two parts. The algorithm ends because $Split(Y)$ returns a partition (A_Y, B_Y) where neither A_Y nor B_Y is empty, so Y is replaced by two smaller sets. In the worst case we would reach the point where every set in P contains a single vertex, i.e., is of the form $\{v\}$, and the *while* loop ends since $|out(\{v\})| = \deg(v) \leq \Delta < \Delta'$. The trace of all splits occurring in BetterPartition(U) forms a rooted binary tree T where each internal $t \in T$ corresponds to a subset of U. We encode this with a mapping λ from nodes of T to subset of U: if $\lambda(t) = Y$ and $(A_Y, B_Y) = Split(Y)$, then $\lambda(t_l) = A_Y$ and $\lambda(t_r) = B_Y$ where t_l and t_r are the children of t. See for instance Figure 3.2 (a), it represents a sequence of splits whose trace is the tree shown Figure 3.2 (b).

Lemma 49. *Let $\mathcal{U}'$ be the output of BetterPartition(U). Then $\mathcal{C}' = (\mathcal{C} \setminus \mathcal{U}) \cup \mathcal{U}'$ is an acceptable partition.*



(a) $(A_1, B_1) = Split(Y)$, $(A_2, B_2) = Split(A_1)$, $(A_3, B_3) = Split(B_2)$.



(b) The trace of the splits.

Figure 3.2: A sequence of splits and its trace.

Proof. Consider P at the end of the *while* loop in BetterPartition(U). Let $Y \in P$ and let $Y^* \subseteq Y$ be such that $G[Y^*]$ is a connected component of $G[Y]$. It is readily verified that $|out(Y^*)| \leq |out(Y)| < \Delta'$. By assumption there is $|U \cap S^*| \leq \frac{k}{2}$ so $|Y^* \cap S^*| \leq \frac{k}{2}$ holds as well. Now $\mathcal{U}'$ is a partition of U whose elements are sets like Y^*. By what we have shown, since $\mathcal{C}$ is an acceptable partition, so is $\mathcal{C}' = (\mathcal{C} \setminus \mathcal{U}) \cup \mathcal{U}'$. □

Lemma 49 is a first step towards proving Lemma 45. It remains to show that $|E(G_{\mathcal{C}'})| < |E(G_{\mathcal{C}})|$. First observe that

$$\begin{aligned} |E(G_{\mathcal{C}'})| &= |E(G_{\mathcal{C}})| - |E(G_{\mathcal{C}}[\mathcal{U}])| - |out_{G_{\mathcal{C}}}(\mathcal{U})| + \Big|\bigcup_{Y \in \mathcal{U}'} out_G(Y)\Big| \\ &= |E(G_{\mathcal{C}})| - |E(G_{\mathcal{C}}[\mathcal{U}])| - |out_G(U)| + \Big|\bigcup_{Y \in P} out_G(Y)\Big| \end{aligned}$$

$\bigcup_{Y \in P} out(Y)$ contains all the edges of $out(U)$ plus the edges $E(A_Y, B_Y)$ for every split $(A_Y, B_Y) = Split(Y)$ done by the algorithm. So $|\bigcup_{Y \in P} out(Y)|$ equals $|out(U)|$ plus some value M

$$|E(G_{\mathcal{C}'})| = |E(G_{\mathcal{C}})| - |E(G_{\mathcal{C}}[\mathcal{U}])| + M \tag{3.3}$$

M is unknown yet, it will be bounded later.

Charging Scheme.

We bound M by replaying BetterPartition(U) with a charging scheme that puts non-negative real numbers, called *charges*[6], on the edges of $E(G[U]) \cup out(U)$. Initially all charges are 0. When a split $(A_Y, B_Y) = Split(Y)$ occurs, each edge in $|E(A_Y, B_Y)|$ adds a charge $|out(A_Y) \cap out(Y)|^{-1}$ to every edge in $out(A_Y) \cap out(Y)$ (which we recall is not empty). So for the split $(A_Y, B_Y) = Split(Y)$, a total charge of $|E(A_Y, B_Y)|$ is created in the graph. This is the only way to add charges in the graph therefore, when the algorithm ends, the total charge equals M.

Existing charges are also moved in the graph in such a way that when the algorithm ends, only edges of $out(U)$ have non-zero charges. This will allow us to bound M as a fraction of $|out(U)|$. Movements of charges occur during splits along with charges creation. When the split $(A_Y, B_Y) = Split(Y)$ occurs, instead of having every edge in $E(A_Y, B_Y)$ give charges $|out(A_Y) \cap out(Y)|^{-1}$, we decide that every edge $e' \in E(A_Y, B_Y)$ that already has a charge $c_{e'}$ adds a charge $(1 + c_{e'})/|out(A_Y) \cap out(Y)|$ to every edge in $out(A_Y) \cap out(Y)$. So e' contributes a total charge $1 + c_{e'}$ to $out(A_Y) \cap out(Y)$. The charge of e' is then reset to 0 since its old charge $c_{e'}$ has been stored in $out(A_Y) \cap out(Y)$. Algorithm Charging(T, λ) shows an implementation of the charging scheme. Its input (T, λ) encodes the splits done during the course of BetterPartition(U).

[6]These charges are *not* related to the charge functions of Tseitin formulas.

Algorithm 2: Charging(T, λ)

1 All c_e are set to 0
2 Mark all leaves of T as visited
3 **while** there is $t \in T$ not marked such that t_r and t_l are marked **do**
4 Let $Y = \lambda(t)$, $A_Y = \lambda(t_l)$ and $B_Y = \lambda(t_r)$
5 **for** $e \in out(A_Y) \cap out(Y)$ **do**
6 Set $c_e = c_e + \sum_{e' \in E(A_Y, B_Y)} \frac{1+c_{e'}}{|out(A_Y) \cap out(Y)|}$
7 **end**
8 **for** $e' \in E(A_Y, B_Y)$ **do**
9 Set $c_{e'} = 0$
10 **end**
11 Mark t as visited
12 **end**

Fix an edge $e = uv$. Suppose that some set Y is split into (A_Y, B_Y) and that e is in $out(Y)$. Either $u \in Y$ and $v \notin Y$ in which case charges are added to e if and only if $u \in A_Y$, or $v \in Y$ and $u \notin Y$ in which case charges are added to e if and only if $v \in A_Y$. In the first case we say that e is charged via u, in the other case e is charged via v. See for instance Figure 3.2 (a). In the leftmost figure, Y is represented by a circle and we have an edge $e = uv \in out(Y)$ with $u \in Y$. Next Y is split into $(A_1, B_1) = Split(Y)$. Since $u \in A_1$, a charge is added to e via u. Next A_1 is split into $(A_2, B_2) = Split(A_1)$, but no charge is added to e because u is in B_2, not in A_2.

Lemma 50. *Let $\mathcal{U}'$ be the output partition of BetterPartition(U) and let (T, λ) be the trace of the splits done by BetterPartition(U). Let $\mathcal{C}' = (\mathcal{C} \setminus \mathcal{U}) \cup \mathcal{U}'$. After running Charging(T, λ), the total charge in G equals $M = |E(G_{\mathcal{C}'})| - |E(G_{\mathcal{C}})| + |E(G_{\mathcal{C}}[\mathcal{U}])|$.*

Proof. M equals the sum of $|E(A_Y, B_Y)|$ over all splits $(A_Y, B_Y) = Split(Y)$ done during the course of BetterPartition(U).

Fix an iteration of the *while* loop of Charging(T, λ). Let $(A_Y, B_Y) = Split(Y)$ be the split for that iteration. In the first *for* loop, a charge of $|E(A_Y, B_Y)| + \sum_{e' \in E(A_Y, B_Y)} c_{e'}$ is created and added to the graph. Then the second *for* loop sets the charges for all $e' \in E(A_Y, B_Y)$ to 0, so a charge $\sum_{e' \in E(A_Y, B_Y)} c_{e'}$ is lost. Thus the total charge created during that iteration of the *while* loop is $|E(A_Y, B_Y)|$. There is one iteration per split done by BetterPartition(U) so when Charging(T, λ) finishes the total charge in the graph equals M. □

Lemma 51. *Let (T, λ) be the trace of the splits done by BetterPartition(U). Let $e = uv$ be an edge with at least one endpoint in U. After running Charging(T, λ) the total charge on e is $c_e \leq 9\gamma$. Moreover if $e \notin out(U)$ then $c_e = 0$.*

Proof. If $e \notin out(U)$ and there is no Y in the trace of BetterPartition(U) such that $e \in E(A_Y, B_Y)$, then c_e never moves from its initial value, that is 0. If $e \notin out(U)$ and $e \in E(A_Y, B_Y)$ for some $(A_Y, B_Y) = Split(Y)$, then c_e is set to 0 at line 9 of Charging(T,λ). Now for all Y' that are processed after Y (so they are closer to the root of the trace than Y) either Y is disjoint from Y', or $Y \subseteq A_{Y'}$ or $Y \subseteq B_{Y'}$, and therefore e can never be in $out(A_{Y'}) \cap out(Y')$. This leaves the case $e \in out(U)$. We show that $c_e \leq 9\gamma$ by proving the following more general result.

Claim 14. *At every moment during the course of Charging(T, λ), there exists an $r \in \mathbb{N}$ such that for each e there is $c_e \leq \sum_{j=1}^{r} (8\gamma)^j$.*

Proof. The proof is by induction. The claim is clearly true at the beginning of the algorithm when all c_e are 0. For the general case let $e = uv$. We define $c_e(u)$ and $c_e(v)$ as the charges given to e via u and via v. Clearly $c_e = c_e(u) + c_e(v)$. We focus on $c_e(u)$. Consider the sequence $Y_{z+1} \subset Y_z \subset \cdots \subset Y_1$ where $|out(Y_{z+1})| < \Delta'$, $u \in Y_{z+1}$, $v \notin Y_1$ (so $e \in out(Y_i)$ for all i) and such that, for all i, e is charged via u when Y_i is processed in the *while* loop. Note that Y_{i+1} may not be A_{Y_i}, see for instance Figure 3.2, it shows that $u \in A_3 \subset B_2 \subset A_1$ so there is an i such that $Y_i = A_1$ and $Y_{i+1} = A_3$ is different from $A_{Y_i} = A_2$. Despite this observation, Lemma 48 still gives us that $|out(Y_{i+1})| \leq \frac{|out(Y_i)|}{2(1-\gamma)}$ for all $i \in [z]$. Since we also have $|out(Y_i)| \geq \Delta'$ for all $i \in [z]$ (otherwise Y_i would not be split) we obtain

$$|out(Y_i)| \geq (2(1-\gamma))^{z-i}|out(Y_z)| \geq (2(1-\gamma))^{z-i}\Delta'$$

We can Z_i the set such that $(Y_i, Z_i) = Split(Y_i, Z_i)$.

Claim 15. *For all $i \leq z$, it holds that*

$$\frac{|E(Y_i, Z_i)|}{|out(Y_i) \cap out(Y_i \cup Z_i)|} \leq \frac{\gamma}{1-\gamma}\left(\frac{1}{2(1-\gamma)}\right)^{z-i}$$

Proof. $|out(Y_i) \cap out(Y_i \cup Z_i)| = |out(Y_i)| - |E(Y_i, Z_i)| \geq |out(Y_i)| - \gamma|out(Y_i) \cap out(Y_i \cup Z_i)| \geq (1-\gamma)|out(Y_i)| \geq (1-\gamma)\Delta'(2(1-\gamma))^{z-i}$. We then use Lemma 47 (3.1) to conclude. □

When processing the split of Y_i the charge added to e via u is

$$\frac{\sum_{e' \in E(Y_i, Z_i)}(1 + c_{e'})}{|out(Y_i) \cap out(Y_i \cup Z_i)|} \leq \frac{(1 + \max_{e'} c_{e'})|E(Y_i, Z_i)|}{|out(Y_i) \cap out(Y_i \cup Z_i)|}$$

By induction hypothesis, $1+\max_{e'} c_{e'} \leq \sum_{j=0}^{r}(8\gamma)^j$ and we know by Claim 15 that $|E(Y_i, Z_i)|/|out(Y_i) \cap out(Y_i \cup Z_i)| \leq \frac{\gamma}{1-\gamma}\left(\frac{1}{2(1-\gamma)}\right)^{z-i}$, so when creating Y_i the charge added to e via u is at most

$$\frac{\gamma}{1-\gamma}\left(\sum_{j=0}^{r}(8\gamma)^j\right)\left(\frac{1}{2(1-\gamma)}\right)^{z-i}$$

We miss a bound on the charge added to e when creating Y_{z+1}. We just use that it is less than γ by Lemma 48 (3.2). So we obtain that the charge added to $e = uv$ via u when creating $Y_1, Y_2, \ldots Y_{z+1}$ is at most

$$\begin{aligned} c_e(u) &\leq \gamma + \frac{\gamma}{1-\gamma}\left(\sum_{j=0}^{r}(8\gamma)^j\right)\sum_{i=1}^{z}\left(\frac{1}{2(1-\gamma)}\right)^{z-i} \\ &\leq \gamma + \frac{2\gamma}{1-2\gamma}\sum_{j=0}^{r}(8\gamma)^j \leq \gamma + 3\gamma\sum_{j=0}^{r}(8\gamma)^j \end{aligned}$$

Where $2\gamma/(1-2\gamma) \leq 3\gamma$ follows from $\gamma = 1/2000$. The bound holds for $c_e(v)$ as well so $c_e = c_e(u) + c_e(v)$ is at most

$$\begin{aligned} 2\gamma + 6\gamma\sum_{j=0}^{r}(8\gamma)^j &= 8\gamma + 6\gamma\sum_{j=1}^{r}(8\gamma)^j \\ &\leq 8\gamma + 8\gamma\sum_{j=1}^{r}(8\gamma)^j = \sum_{j=1}^{r+1}(8\gamma)^j \end{aligned}$$

This finishes the proof of Claim 14. □

So for $e \in out(U)$ we have that $c_e \leq \sum_{j=1}^{\infty}(8\gamma)^j = \frac{1}{1-8\gamma} - 1$, which is less than 9γ when $\gamma = 1/2000$. □

Now we can finish the proof of Lemma 45.

Proof of Lemma 45. Run BetterPartition(U) and let $\mathcal{U}'$ be the output partition of U. Let $\mathcal{C}' = (\mathcal{C} \setminus \mathcal{U}) \cup \mathcal{U}'$. By Lemma 49, $\mathcal{C}'$ is an acceptable partition. It remains to show $|E(G_{\mathcal{C}'})| < |E(G_{\mathcal{C}})|$. Recall Equation (3.3):

$$|E(G_{\mathcal{C}'})| = |E(G_{\mathcal{C}})| - |E(G_{\mathcal{C}}[\mathcal{U}])| + M$$

Let (T, λ) be the trace of the splits done by BetterPartition(U). By Lemma 50, the total charge in the graph after running Charging(T, λ) is M which, by Lemma 51 is at most $9\gamma|out(U)|$. Using this bound in Equation (3.3) yields

$$\begin{aligned}
|E(G_{\mathcal{C}'})| &\leq |E(G_{\mathcal{C}})| - |E(G_{\mathcal{C}}[\mathcal{U}])| + 9\gamma|out(U)| \\
&\leq |E(G_{\mathcal{C}})| - |E(G_{\mathcal{C}}[\mathcal{U}])| + 9\gamma|E(G_{\mathcal{C}})| \\
&\leq |E(G_{\mathcal{C}})| + \left(9\gamma - \frac{1}{180}\right)|E(G_{\mathcal{C}})| \\
&< |E(G_{\mathcal{C}})|
\end{aligned}$$

□

3.6 Conclusion and Perspectives

The contributions of this chapter rely on two key results. The first one is the exponential lower bound on the DNNF size of satisfiable Tseitin formulas whose underlying graph has maximum degree bounded by a constant. The second is the existence of a constant $\alpha > 0$ such that, for every graph G whose maximum degree is bounded by a constant, there is a bipartition of the vertices of G such that the induced graphs on both sides of the partition have treewidth at least $\alpha \times tw(G)$. The two results have the annoying requirement that the graphs must have bounded maximum degree. Thanks to Itsykson, Riazanov and Smirnov [IRS22], this requirement has been lifted for the DNNF size of satisfiable Tseitin formulas. We wonder whether the requirement of the degree being bounded by a constant can also be lifted in the second result.

Open question 2. Is there a constant $\alpha > 0$ such that, for every graph G, there is a bipartition (A, B) of $V(G)$ such that both $tw(G[A]) \geq \alpha \times tw(G)$ and $tw(G[B]) \geq \alpha \times tw(G)$ hold?

The question is not concerned with efficient algorithms to find the aforementioned bipartitions but only with the existence of these bipartitions for every graph. Answering the question positively is not necessary to get rid of the maximum degree $\Delta(G)$ in our main results. Indeed Itsykson et al. have already generalized our contributions to the case when the graph of the Tseitin formulas have unbounded maximum degree [IRS22, Theorem 3.1, Remark 3.7]. Yet we think that the graph property suggested by our question is interesting in its own right and could be useful to prove lower bounds in other situations.

Going back to bottom-up compilation, the paradigm has this advantage over top-down compilation in that we have a simple framework that describes the underlying behavior of most bottom-up compilers, and that allows us to analyze the space efficiency of these compilers. Interestingly, our framework is not restricted to the strategy where a *single* intermediate circuit is kept in memory by the compiler, and where constraints (or clauses) are aggregated one by one to this circuit. Yet this technique, that we call of successive constraint aggregation, is the only one that we aware of in practice. So we ask the following question:

Open question 3. Is there a denumerable class of CNF formulas for which all str-DNNF$(\wedge, r)$-compilations using successive clause aggregation generate intermediate circuits of exponential size, while there are general str-DNNF$(\wedge, r)$-compilations that avoid these exponentially large intermediate circuits?

Chapter 4

The Length of Regular Resolution Refutations of Unsatisfiable Tseitin Formulas

In this chapter we give a second application of our lower bound on the size of circuits in DNNF representing satisfiable Tseitin formulas from Chapter 1. Lower bounds are negative results and, unsurprisingly, so are their applications. We show how Theorem 5 can be used to derive a lower bound on the length of regular resolution refutations of unsatisfiable Tseitin formulas. This lower bound is characterized by a single exponential dependence in the treewidth of the underlying graph of the formula (so a dependence in $2^{\Omega(tw(G))}$). The exponential dependence also appears in known upper bounds and can therefore not be improved. We present (regular) resolution before giving the state of the art and our lower bound.

4.1 Regular Resolution Refutation

The *resolution rule* says that if a function entails the clauses $C \vee x$ and $C' \vee \overline{x}$ for some variable x, then it also entails the clause $C \vee C'$, formally:

$$\frac{C \vee x \qquad C' \vee \overline{x}}{C \vee C'}$$

The variable x is called the *resolved variable* and the clause $C \vee C'$ is called the *resolvent*. We will use the notation $C \vee C' = \texttt{Resolution}(C \vee x, C' \vee \overline{x}, x)$.

Let ϕ be an unsatisfiable CNF formula. A *resolution refutation* of ϕ is a sequence of, not necessarily pairwise distinct clauses, $C_1, \ldots, C_N$, such that C_N is the empty clause and such that for all $1 \leq i \leq N$

- either C_i is a clause of ϕ,
- or C_i is the resolvent of the resolution rule applied on clauses C_j and C_k for some $j, k < i$.

The *length* of a resolution refutation $C_1, \ldots, C_N$ is the number of clauses in the sequence. We assume, without loss of generality, that for every $i < N$, the clause C_i is used as a premise in a resolution rule to derive C_j for some $j > i$, in other words, with the exception of the empty clause, a clause is contained in the sequence only if it is used for inferring other clauses by resolution.

Under this assumption, the resolution refutation $C_1, \ldots, C_N$ of ϕ may be represented graphically as a directed acyclic graph whose N vertices are the clauses C_i and such that, if C_i is the resolvent of the resolution rule applied on C_j and C_k, then there is an edge directed from C_j to C_i and another directed

$C_1 = x \vee \overline{y}$
$C_2 = \overline{x} \vee \overline{z}$
$C_3 = \overline{y} \vee z$
$C_4 = y \vee \overline{z}$
$C_5 = \overline{x} \vee y$
$C_6 = x \vee z$
$C_7 = \texttt{Resolution}(C_1, C_2, x) = \overline{y} \vee \overline{z}$
$C_8 = \texttt{Resolution}(C_5, C_6, x) = y \vee z$
$C_9 = \texttt{Resolution}(C_3, C_7, z) = \overline{y}$
$C_{10} = \texttt{Resolution}(C_4, C_8, z) = y$
$C_{11} = \texttt{Resolution}(C_9, C_{10}, y) = \emptyset$

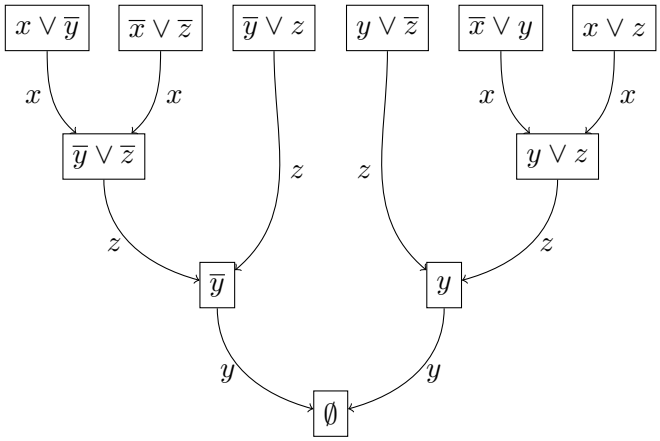


Figure 4.1: A resolution refutation and its DAG representation.

from C_k to C_j. The edges are labelled by the resolved variables. The DAG as a single sink which is the empty clause, and as several sources that are clauses from the CNF formulas. A resolution refutation is called *regular* when no variable labels more than one edge on any path of the DAG.

Example 28. The CNF formula $\phi = (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (y \vee \overline{z}) \wedge (\overline{y} \vee z) \wedge (x \vee z) \wedge (\overline{x} \vee \overline{z})$ is unsatisfiable. A possible resolution refutation of ϕ is shown Figure 4.1. This refutation is regular since no variable appears twice on any path of the DAG connecting a source to the sink. The refutation is of length 11. ◀

It is known that the (regular) resolution refutation system is sound, meaning that all unsatisfiable CNF have a (regular) resolution refutation [DP60]. So we denote by $Res(\phi)$ the length of the shortest resolution refutation of an unsatisfiable CNF formula ϕ and we denote by $RRes(\phi)$ the length of the shortest regular resolution refutation of an unsatisfiable CNF formula ϕ. It is also known that for certain unsatisfiable CNF formulas, the shortest regular resolution refutation is unavoidably much longer than the shortest non-regular resolution refutation. More formally, Alekhnovich et al. proved in [AJPU07] that there exists an infinite class $\mathcal{F}$ of unsatisfiable CNF formulas, a constant $\delta > 0$, and a polynomial p such that for every $\phi \in \mathcal{F}$, $Res(\phi) \leq p(|\phi|)$ and $RRes(\phi) \geq 2^{|\phi|^\delta}$. Yet, regular resolution remains interesting as the underlying procedure behind DPLL-style algorithms [DP60, DLL62]. Studying the length of the shortest regular resolution refutation for some formulas is one way to assess the efficiency of DPLL-style algorithms on unsatisfiable instances, hence the research conducted on this refutation system (see [Urq87, BBI12, ABdR+18, BI13] for a small sample).

4.2 State of Resolution Refutation of Tseitin Formulas and Contribution

Tseitin formulas have been introduced by G. S. Tseitin in the sixties as examples of hard CNF formulas for the resolution proof (or refutation) system [Tse68] (see [Tse83] for an English version). We have already introduced Tseitin formulas in the preliminaries. We just recall here that they are CNF formulas that encode systems of parity constraints structured by a graph, that there is a simple criterion for deciding whether a Tseitin formula is satisfiable given in Lemma 9, and that this criterion allows us to construct unsatisfiable Tseitin formulas for any graph. The first exponential lower bounds on the length of resolution refutation of Tseitin formulas have been established by Urquhart for the case when

the underlying graphs of the formulas are expander graphs [Urq87], which are graphs with particular connectivity properties. We refer the reader to [HLW06] for details on expander graphs. Similar lower bounds on the length of the refutations of Tseitin formulas in other refutations systems have been proved under similar assumptions on the underlying graphs [Ben02, IO13].

It is known that different properties of the underlying graph characterize different parameters of the resolution refutations for Tseitin formulas [AR11, IO13, GTT20]. Extending this line of work, we show that treewidth determines the length of regular resolution refutations for Tseitin formulas. An upper bound was already proved by Alekhnovich and Razborov:

Theorem 20. [AR11] *There is a constant c such that for every graph G and for every unsatisfiable Tseitin formula $T(G, c)$, the length of the shortest regular resolution refutation of $T(G, c)$ is at most $2^{O(tw(G))}|V(G)|^c$.*

Our contribution in this chapter is to provide a matching lower bound:

Let G be a connected graph with maximum degree at most Δ and let $T(G, c)$ be an unsatisfiable Tseitin formula. The length of the shortest regular resolution refutation of $T(G, c)$ is at least $2^{\Omega(tw(G)/\Delta)}|V(G)|^{-1}$.

The two theorems have the following corollary that gives a characterization of unsatisfiable Tseitin formulas that have regular resolution refutations of length polynomial in the number of variables:

Corollary 3. *Let $\Delta \geq 0$ be any integer. Unsatisfiable Tseitin formulas for graphs whose maximum degree is bounded by Δ have regular resolution refutations of length polynomial in the number of variables if and only if the treewidth of the graphs is bounded logarithmically in their size.*

Theorem 4.5 is not the first lower bounds for the length of resolution refutations of Tseitin formulas based on treewidth. For general resolution, a $2^{\Omega(tw(G)^2)/|V(G)|}$ lower bound can be inferred combining the width-length relation of [BW01] with lower bounds shown in [GTT20]. This gives a tight $2^{\Omega(tw(G))}$ bound when the treewidth of G is linear in its number of vertices. For smaller treewidth, there are also bounds from [GIRS19] for the stronger proof system of constant depth Frege proofs which, for resolution, imply a $2^{\Omega(tw(G)^\delta)}$ lower bound. But since the top exponent δ is significantly smaller than 1, these results are incomparable to ours. For the length of regular resolution resolution, the best lower bound in the general case was $2^{\Omega(tw(G)/\Delta(G))/\log(|V(G)|)}$ shown by Itsykson et al. in [IRSS21]. We build on the work of Itsykson et al. and improve their bounds. To do so we use the lower bound on the DNNF size of satisfiable Tseitin formulas – whereas Itsykson et al. use a looser bound on the nFBDD size of Tseitin formulas – to eliminate the division by $\log(|V(G)|)$ in the exponent.

4.3 Branching Programs for Search Relations

We avoid dealing directly with regular resolution refutations thanks to a well-known connection between these refutations and branching programs representations of the *search relation*. Let ϕ be an unsatisfiable CNF formula. We define $Search(\phi)$ as the relation consisting of all pairs (a, C) where a is an assignment to $var(\phi)$ and C is a clause of ϕ falsified by a. A read-once branching program, or FBDD, B over variables $var(\phi)$ and with domain $clauses(\phi)$ computes $Search(\phi)$ if and only if for every assignment a to $var(\phi)$ we have $(a, B(a)) \in Search(\phi)$, where $B(a)$ is the clause of ϕ reached following the computation path corresponding to a in B. Note that there may be pairs $(a, C) \in Search(\phi)$ such that $C \neq B(a)$, this occurs when a falsifies more than one clause of ϕ. Now we present the connection between FBDDs computing $Search(\phi)$ and regular resolution refutations of ϕ.

Theorem 21. [Kra95] *For every unsatisfiable CNF formula ϕ, $RRes(\phi)$ equals the size of the smallest FBDD over $var(\phi)$ and with domain $clauses(\phi)$ that computes $Search(\phi)$.*

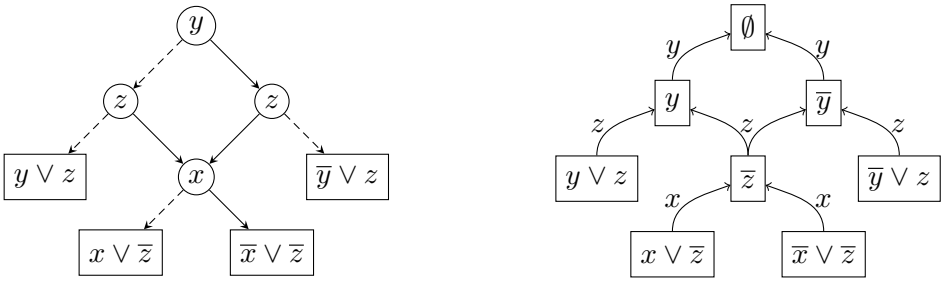


Figure 4.2: From regular resolution refutation to FBDD computing a search relation

The reader curious of the proof is referred to Theorem 4.2.3. in [Kra95]. Here we just give some insight as to why the theorem holds by describing a transformation from regular resolution refutations of ϕ to FBDD computing $Search(\phi)$. Let $C_1, \ldots, C_N$ be a regular resolution refutation of ϕ and let f be a mapping from $\{C_1, \ldots, C_N\}$ to nodes of an FBDD defined as follows: if C_i is a clause of ϕ, then $f(C_i)$ is a sink labelled by C_i, but if $C_i = \texttt{Resolution}(C_j, C_k, x)$ with $C_j = C'_j \vee x$ and $C_k = C'_k \vee \overline{x}$ then $f(C_i)$ is a decision node labelled by x whose 1-child is $f(C_k)$ and whose 0-child is $f(C_j)$. One can show that $f(C_N)$ is the source (or the root) of an FBDD computing $Search(\phi)$ (see [Kra95] for details). Moreover there is a reverse transformation to obtain a regular resolution refutation of ϕ from an FBDD computing $Search(\phi)$. Let $u_1, \ldots, u_N$ be the nodes of such an FBDD B, given in depth-first order (so u_1 is the root of B). The function g maps the nodes of B to clauses as follows: if u_i is a sink labelled by C then $g(u_i) = C$, and if u_i is a decision node for x with 0-child u_j and 1-child u_k, then $g(u) = \texttt{Resolution}(g(u_k), g(u_j), x)$. One can show that $g(u_N), \ldots, g(u_1)$ is a regular resolution refutation of ϕ.

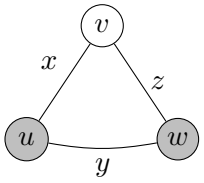
Example 29. Consider the following unsatisfiable CNF formula: $\phi = (x \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (y \vee z) \wedge (\overline{y} \vee z)$. Figure 4.2 shows an FBDD computing $Search(\phi)$ on the right and the corresponding regular resolution refutation of ϕ on the left. The refutation is represented by a DAG whose sources are at the bottom and whose sink is at the top. A single clause is created in the refutation for each decision node and for each sink of the FBDD, so the FBDD and the refutation have the same size: the number of nodes of the FBDD (including sinks) equals the number of clauses in the corresponding refutation. ◀

For a Tseitin formula, from an unsatisfied clause we can directly infer an unsatisfied parity constraint. Recall that the parity constraints of $T(G, c)$ are in bijection with $|V(G)|$. When $T(G, c)$ is unsatisfiable, Itsykson et al. [IRSS21] define the search relation $SearchVertex(G, c)$ to be the set of pairs (a, v) with a an assignment to $var(T(G, c))$ and v a vertex of G such that $\chi_v(G, c)$ is falsified by a. We say that an FBDD B over $var(T(G, c))$ and with domain $V(G)$ computes $SearchVertex(G, c)$ if, for every assignment a to $var(T(G, c))$ we have $(a, B(a)) \in SearchVertex(G, c)$. Note that if a falsifies more than one parity constraint of $T(G, c)$ then there are pairs $(a, v) \in SearchVertex(G, c)$ such that $v \neq B(a)$.

Corollary 4. *For every unsatisfiable Tseitin formula $T(G, c)$, $RRes(T(G, c))$ is at least the size of the smallest FBDD computing $SearchVertex(G, c)$.*

Proof. Let B be an FBDD computing $Search(T(G, c))$. For each sink of B labelled by a clause C, there is a vertex $v \in V(G)$ such that C is a clause of the CNF representation of $\chi_v(G, c)$, then change the label of the sink from C to v. The resulting FBDD computes $SearchVertex(G, c)$ and has the same size as B. Using this transformation and Theorem 21, the result is immediate. □

Example 30. The CNF formula $(\overline{x} \vee y) \wedge (x \vee \overline{y}) \wedge (x \vee z) \wedge (\overline{x} \vee \overline{z}) \wedge (y \vee \overline{z}) \wedge (\overline{y} \vee z)$ for which a regular resolution refutation is shown Figure 4.1 is a Tseitin formula for the following graph



where charges are indicated by a color code: every gray node has charge 0 and every white node has charge 1. Figure 4.3 shows the transformation of that refutation to an FBDD computing $Search(T(G,c))$ and the transformation of that FBDD to another FBDD computing $SearchVertex(G,c)$. ◀

4.4 Well-Structured Branching Programs Computing SearchVertex

The proof of Theorem 4.5 relies heavily on techniques developed in [IRSS21]. There it is shown how to use an FBDD B computing $SearchVertex(G,c)$, where $T(G,c)$ is an *unsatisfiable* Tseitin formula, to construct an FBDD B^* computing any *satisfiable* Tseitin formula $T(G,c^*)$ with the same underlying graph. Note that B is an FBDD over $var(T(G,c))$ with domain $V(G)$ while B^* is an FBDD over $var(T(G,c^*)) = var(T(G,c))$ with domain $\{0,1\}$. The construction of [IRSS21] results in $|B^*|$ quasi-polynomial in $|B|$. Thus good lower bounds on the size of B^* yield lower bounds for regular refutation by Corollary 4. To give tighter results, we give a version of the reduction from unsatisfiable to satisfiable Tseitin formulas where the target representation for $T(G,c')$ is not the FBDD language but the more succinct DNNF language. This lets us decrease the size of the representation from quasi-polynomial to polynomial which, thanks to Theorem 5, will yield Theorem 4.5. So we are about to prove the following:

Theorem 22. *Let $T(G,c)$ be an unsatisfiable Tseitin formula where G is connected. There exists, for every satisfiable Tseitin formula $T(G,c^*)$, a circuit in DNNF of size $O(RRes(T(G,c)) \times |V(G)|)$ computing it.*

To prove Theorem 22, we use the notion of *well-structured* FBDDs introduced in [IRSS21]. We present that notion in this section, but before that, we recall how Tseitin formulas behave under conditioning of variables whose corresponding edges are bridges of the underlying graph.

Lemma 52. *Let G be a connected graph and let $T(G,c)$ be an unsatisfiable Tseitin formula. Let $e \in E(G)$ be a bridge of G, that is, calling a and b the vertices connected by e, the graph G' obtained by removing e has two connected components. Let G^a be the component containing a and let G^b be the*

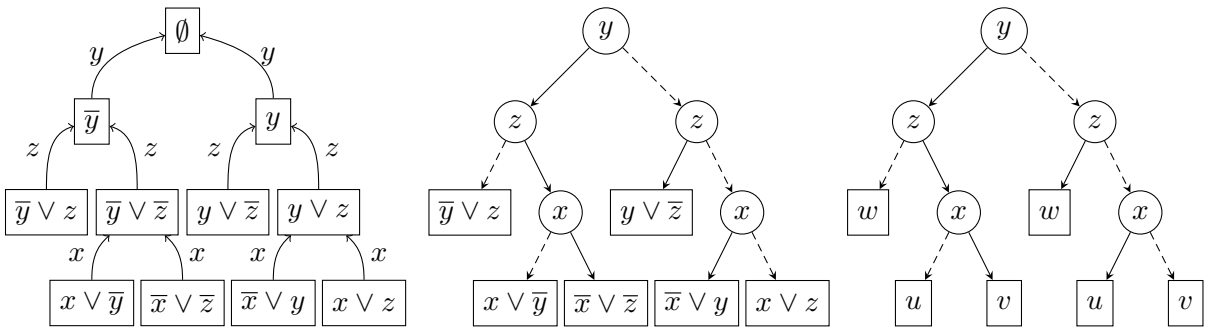


Figure 4.3: From a regular resolution refutation of unsatisfiable Tseitin formulas to FBDD computing $SearchVertex$.

component containing b. Define $\gamma(\overline{x_e}) = c$ and $\gamma(x_e) = c + \mathbb{1}_a + \mathbb{1}_b \mod 2$. Finally for $\ell_e \in \{x_e, \overline{x_e}\}$, let $\gamma^a(\ell_e)$ be the restriction of $\gamma(\ell_e)$ to $V(G^a)$, and $\gamma^b(\ell_e)$ be the restriction of $\gamma(\ell_e)$ to $V(G^b)$. Then for every ℓ_e,

$$T(G,c)|\ell_e = T(G^a, \gamma^a(\ell_e)) \wedge T(G^b, \gamma^b(\ell_e))$$

and either $T(G^a, \gamma^a(\ell_e))$ is unsatisfiable or $T(G^b, \gamma^b(\ell_e))$ is unsatisfiable, but not both.

Example 31. Consider the graph G shown on the left in Figure 4.4. The vertices of G are charged according to a function c. We have

$$\begin{aligned} T(G,c) = {} & (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \\ & \wedge (\overline{x_2} \vee x_3 \vee x_e) \wedge (x_2 \vee \overline{x_3} \vee x_e) \wedge (x_2 \vee x_3 \vee \overline{x_e}) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_e}) \\ & \wedge (\overline{x_4} \vee x_5) \wedge (x_4 \vee \overline{x_5}) \wedge (\overline{x_5} \vee x_6) \wedge (x_5 \vee \overline{x_6}) \\ & \wedge (x_4 \vee \overline{x_6} \vee \overline{x_e}) \wedge (\overline{x_4} \vee x_6 \vee \overline{x_e}) \wedge (\overline{x_4} \vee \overline{x_6} \vee x_e) \wedge (x_4 \vee x_6 \vee \overline{x_e}) \end{aligned}$$

By Lemma 9, $T(G,c)$ is unsatisfiable since G is connected and since the number of vertices that have charge 1 is odd. Let $e = ab$. Removing e from G yields two connected components G^a and G^b. Assume x_e is set to 1, then $T(G,c)|x_e$ is the Tseitin formula for the graph and the charges represented on the right of Figure 4.4. We have

$$\begin{aligned} T(G^a, \gamma^a(x_e)) = {} & (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \\ & \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3}) \\ T(G^b, \gamma^b(x_e)) = {} & (\overline{x_4} \vee x_5) \wedge (x_4 \vee \overline{x_5}) \wedge (\overline{x_5} \vee x_6) \wedge (x_6 \vee \overline{x_6}) \\ & \wedge (x_4 \vee \overline{x_6}) \wedge (\overline{x_4} \vee x_6) \end{aligned}$$

and one can verify that $T(G,c)|x_e = T(G^a, \gamma^a(x_e)) \wedge T(G^b, \gamma^b(x_e))$. By Lemma 9, $T(G^a, \gamma^a(x_e))$ is unsatisfiable since G^a is connected and since it has three vertices with charge 1. However $T(G^b, \gamma^b(x_e))$ is satisfiable since G^b is connected and since it has no vertex with charge 1. The case when x_e is assigned the value 0 corresponds to the graph shown in the middle of Figure 4.4. This time $T(G,c)|\overline{x_e} = T(G^a, \gamma^a(\overline{x_e})) \wedge T(G^b, \gamma^b(\overline{x_e}))$ and $T(G^b, \gamma^b(\overline{x_e}))$ is unsatisfiable while $T(G^a, \gamma^a(\overline{x_e}))$ is satisfiable. ◀

Definition 40. *Let $T(G,c)$ be an unsatisfiable Tseitin formula where G is a connected graph. A branching program B computing $SearchVertex(G,c)$ is* well-structured *when, for all nodes u_k of B, there exists a connected subgraph G_k of G and a charge function c_k such that $T(G_k, c_k)$ is unsatisfiable, u_k computes $SearchVertex(G_k, c_k)$, and*

1. *if u_k is the source, then $G_k = G$ and $c_k = c$,*
2. *if u_k is a sink corresponding to $v \in V(G)$, then $G_k = (\{v\}, \emptyset)$ and $c_k = \mathbb{1}_v$,*
3. *if u_k is a decision node for x_{ab} with 0- and 1-child u_{k_0} and u_{k_1}, set $\ell_0 = \overline{x_{ab}}$ and $\ell_1 = x_{ab}$, then for all $i \in \{0,1\}$, $(G_{k_i}, c_{k_i}) = (G^a_k, \gamma^a_k(\ell_i))$ if $T(G^a_k, \gamma^a_k(\ell_i))$ is unsatisfiable, otherwise $(G_{k_i}, c_{k_i}) = (G^b_k, \gamma^b_k(\ell_i))$.*

We remark that our definition is a slight simplification of that given by Itsykson et al. [IRSS21]. It can easily be seen that ours is implied by theirs (see Definition 3.2 and Proposition 3.4 in [IRSS21]).

Essentially, in a well-structured FBDD computing $SearchVertex(G,c)$, each decision node u_k computes a SearchVertex relation for some unsatisfiable formula $T(G_k, c_k)$. And when moving to a child of u_k by assigning its variable x_{ab} the value 0 or 1, we look at the formula $T(G_k, c_k)$ conditioned

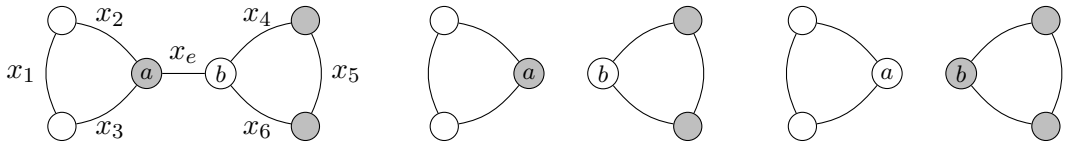


Figure 4.4: Evolution of the charges of a graph after removal of a bridge.

on this variable assignment. This is again a Tseitin formula but on the graph $G'_k = G_k - ab$. The well-structured property states that the children of u_k represents the $SearchVertex$ relation for that formula restricted to the connected component of G'_k where it is unsatisfiable.

Let us give more details on the third point of the definition. Let $e = ab$. If e is not a bridge in G_k, then $G^a_k = G^b_k = G'_k$ and $\gamma^a_k = \gamma^b_k = \gamma_k$. Since $T(G_k, c_k)$ is unsatisfiable, so are $T(G'_k, \gamma_k(x_e)) = T(G_k, c_k)|x_e$ and $T(G'_k, \gamma_k(\overline{x_e})) = T(G_k, c_k)|\overline{x_e}$. So both u_{k_0} and u_{k_1} computes $SearchVertex$ over the graph G'_k. But if ab is a bridge in G_k, then G^a_k and G^b_k are two distinct component of G'_k. Then u_{k_0} computes $SearchVertex$ restricted to the component that is responsible for $T(G_k, c_k)|\overline{x_e}$ being unsatisfiable. Similarly, u_{k_1} computes $SearchVertex$ restricted to the component that is responsible for $T(G_k, c_k)|x_e$ being unsatisfiable. Note that, by Lemma 52, if the component for u_{k_0} is G^a_k, then the component for u_{k_1} must be G^b_k, and vice versa.

Example 32. Let G_k be the graph shown on the left in Figure 4.4 with the charges c_k indicated by a color code: all gray vertices have charge 0 and all with vertices have charge 1. The edge $e = ab$ is a bridge of G_k. Let G^a_k (resp. G^b_k) be the connected component of $G_k - e$ containing a (resp. b). If a node u_k in a well-structured FBDD computes the relation $SearchVertex(G_k, c_k)$, then its 0-child u_{k_0} computes the relation $SearchVertex$ for the graph G^b_k with the charge function restricted to $V(G^b_k)$, because as the middle picture shows, the charge distribution on component G^b_k is the reason for $T(G_k, c_k)|\overline{x_e}$ being unsatisfiable. Analogously, the 1-child u_{k_1} computes the relation $SearchVertex$ for the graph G^a_k with the charge function restricted to $V(G^a_k)$, because as the rightmost picture shows, the charge distribution on component G^a_k is the reason for $T(G_k, c_k)|x_e$ being unsatisfiable. ◀

We will use the following result from [IRSS21].

Lemma 53. [IRSS21, Lemma 1.4] *Let $T(G, c)$ be an unsatisfiable Tseitin formula where G is connected and let B be an FBDD of minimal size computing the relation $SearchVertex(G, c)$. Then B is well-structured.*

4.5 From Unsatisfiable to Satisfiable Tseitin Formulas

In this section we prove Theorem 22. Similarly to Theorem 14 in [IRSS21], the proof is based on a reduction from a well-structured FBDD for $SearchVertex(G, c)$ to a circuit in DNNF computing the *satisfiable* formula $T(G, 0)$.

Lemma 54. *Let G be a connected graph. Let $T(G, 0)$ and $T(G, c)$ be Tseitin formulas where $T(G, c)$ is unsatisfiable. For every well-structured FBDD B computing $SearchVertex(G, c)$ there exists a circuit in DNNF of size $O(|B| \times |V(G)|)$ computing $T(G, 0)$.*

Proof. Let $S = |B|$ and denote by $u_1, \ldots, u_S$ the nodes of B such that if u_j is a successor of u_i, then $j < i$ (thus u_S is the source of B). For every $1 \leq i \leq S$, the node u_i computes $SearchVertex(G_i, c_i)$. We will show how to iteratively construct circuits $D^1, \ldots, D^S$ in DNNF such that for every i, D^i is a subcircuit of D^{i+1} and

for all $v \in V(G_i)$, there is a node s in D^i such that D^i_s computes $T(G_i, c_i + \mathbb{1}_v \mod 2)$. $(*)$

For convenience, we remove the mod2 notation in this proof and assume that all arithmetic is over $\mathbb{F}_2$. The graphs $G_1, \cdots, G_S$ are each connected so by Lemma 9, if $T(G_i, c_i)$ is unsatisfiable, then $T(G_i, c_i + \mathbb{1}_v)$ is satisfiable for every $v \in V(G_i)$. We show by induction on i how to construct D^i from D^{i-1} while respecting $(*)$.

Base case. u_1 is a sink of B, so it computes $SearchVertex(G_v, \mathbb{1}_v)$ where $G_v = (\{v\}, \emptyset)$ for some vertex $v \in V(G)$. Thus we define D_1 as a single node labelled by the constant 1, which indeed computes $T(G_v, \mathbb{1}_v + \mathbb{1}_v) = T(G_v, 0) \equiv 1$. So D_1 is a circuit in DNNF respecting $(*)$.

Now suppose that we have the circuit D^{k-1} in DNNF satisfying $(*)$ and consider the node u_k of B. There are two cases:

Inductive case 1: u_k is a sink of B. We argue as for D_1, but since we already have a node labelled by the constant 1 in D^{k-1}, we just define $D^k = D^{k-1}$ and we are done.

Inductive case 2: u_k is a decision node for the variable x_e with 0-child u_{k_0} and 1-child u_{k_1}. Recall that B_{u_k} computes $SearchVertex(G_k, c_k)$ and let $e = ab$. There are two cases. If e is not a bridge in G_k then $G^a_k = G^b_k = G_k - e$ and, by the well-structured property, $B_{u_{k_0}}$ computes $SearchVertex(G_k - e, c_k)$ and $B_{u_{k_1}}$ computes $SearchVertex(G_k - e, c_k + \mathbb{1}_a + \mathbb{1}_b)$. For every $v \in V(G_k)$, since $k_0, k_1 < k$, by induction there is a node s_v in D^{k_0} such that $D^{k_0}_{s_v}$ computes $T(G_k - e, c_k + \mathbb{1}_v)$ and a node σ_v in D_{k_1} such that $D^{k_1}_{\sigma_v}$ computes $T(G_k - e, c_k + \mathbb{1}_a + \mathbb{1}_b + \mathbb{1}_v)$. So for every $v \in V(G_k)$ we add to D^{k-1} an $\vee$-node g_v whose left input is $\overline{x_e} \wedge s_v$ and whose right input is $x_e \wedge \sigma_v$. By construction, g_v computes $T(G_k, c_k + \mathbb{1}_v)$ and the new $\wedge$-nodes are decomposable since e is not an edge of $G_k - e$ and therefore x_e and $\overline{x_e}$ do not appear in D^{k_0} or D^{k_1}.

If otherwise $e = ab$ is a bridge in G_k, then by the well-structured property, there exist $i \in \{0, 1\}$ and a literal $\ell_e \in \{\overline{x_e}, x_e\}$ such that $B_{u_{k_i}}$ computes $SearchVertex(G^a_k, \gamma^a_k(\ell_e))$ and $B_{u_{k_{1-i}}}$ computes $SearchVertex(G^b_k, \gamma^b_k(\overline{\ell_e}))$. We construct a gate g_v computing $T(G_k, c_k + \mathbb{1}_v)$ for each $v \in V(G_k)$. Assume, without loss of generality, that $v \in V(G^a_k)$, then

- $T(G_k, c_k + \mathbb{1}_v)|\overline{\ell_e} \equiv T(G^a_k, \gamma^a_k(\overline{\ell_e}) + \mathbb{1}_v) \wedge T(G^b_k, \gamma^b_k(\overline{\ell_e})) \equiv 0$ (because $T(G^b_k, \gamma^b_k(\overline{\ell_e})) \equiv 0$), and
- $T(G_k, c_k + \mathbb{1}_v)|\ell_e \equiv T(G^a_k, \gamma^a_k(\ell_e) + \mathbb{1}_v) \wedge T(G^b_k, \gamma^b_k(\ell_e))$

For the second item, since $k_0, k_1 < k$, by induction there is a gate s_v in D^{k_i} such that $D^{k_i}_{s_v}$ computes $T(G^a_k, \gamma^a_k(\ell_e) + \mathbb{1}_v)$ and there is a gate s_b in $D^{k_{1-i}}$ such that $D^{k_{1-i}}_{s_b}$ computes $T(G^b_k, \gamma^b_k(\overline{\ell_e}) + \mathbb{1}_b)$. But $\gamma_k(\ell_e) = \gamma_k(\overline{\ell_e}) + \mathbb{1}_a + \mathbb{1}_b$, so $\gamma^b_k(\ell_e) = \gamma^b_k(\overline{\ell_e}) + \mathbb{1}_b$, therefore $D^{k_{1-i}}_{s_b}$ computes the formula $T(G^b_k, \gamma^b_k(\ell_e))$. So we add an $\wedge$-node g_v whose left input is ℓ_e and whose right input is $s_v \wedge s_b$ and add it to D^{k-1}. Note that $\wedge$-gates are decomposable since G^a_k and G^b_k share no edge and therefore D^{k_0} and D^{k_1} are on disjoint sets of variables.

Let D^k be the circuit after all g_v have been added to D^{k-1}. Then D^k is a circuit in DNNF that contains D^{k-1} as a subcircuit and that satisfies $(*)$.

It only remains to bound $|D^S|$. To this end, observe that when constructing D^k from D^{k-1} we add at most $3 \times |V_k|$ gates, so $|D^S|$ is at most $3(|V_1| + \cdots + |V_S|) = O(S \times |V(G)|)$. Finally, take any root of D^S and delete all gates not reached from it, the resulting circuit is a circuit D in DNNF computing a satisfiable Tseitin formula $T(G, c')$. We get a circuit in DNNF computing $T(G, 0)$ using Lemma 23. $\square$

Combining Theorem 21, Corollary 4, Lemma 53 and Lemma 54 as follows yields Theorem 22:

Regular Refutation Length for an unsatisfiable $T(G, c)$

$\geq$ FBDD size for $SearchClause(T(G, c))$ (Theorem 21)

$\geq$ FBDD size for $SearchVertex(T(G, c))$ (Corollary 4)

$=$ well-structured FBDD size for $SearchVertex(T(G, c))$ (Lemma 53)

$\geq$ DNNF size for $T(G, 0)/O(|V(G)|)$ (Lemma 54, Lemma 23)

It suffices to combine Theorem 22 with our lower bound on the DNNF size of satisfiable Tseitin formula proven in Chapter 1 to obtain the main result of this chapter, which is that the length of any regular resolution refutation of any unsatisfiable Tseitin formula over a graph G is exponential in $tw(G)/\Delta(G)$: Let G be a connected graph with maximum degree at most Δ and let $T(G, c)$ be an unsatisfiable Tseitin formula. The length of the shortest regular resolution refutation of $T(G, c)$ is at least $2^{\Omega(tw(G)/\Delta)}|V(G)|^{-1}$.

4.6 Conclusion and Perspectives

The main result of this chapter is a characterization result: we have fully characterized unsatisfiable Tseitin formulas over graphs, whose maximum degree is bounded by a constant, that have regular resolution refutations of polynomial size. The two downsides of this characterization are, first the restriction to graphs of bounded degree, and, second, the restriction to *regular* resolution refutations. The first restriction can actually be avoided thanks to the recent result of Itsykson, Riazanov and Smirnov who proved that the DNNF size of satisfiable Tseitin formulas over *any* graph (and not only those of bounded degree) is exponential in the treewidth of the graph [IRS22]. Concerning the assumption of regularity, one could perhaps lift it by proving that the shortest *general* resolution refutation of every unsatisfiable Tseitin formula is essentially regular. We ask whether this conjecture is true.

Open question 4. For unsatisfiable Tseitin formulas, are the length of the shortest regular resolution refutation and the length of the shortest resolution refutation polynomially related?

One could give a positive answer to the question by proving that the smallest branching programs for the $SearchClause$ predicate for unsatisfiable Tseitin formulas are read-once branching programs. It can be shown that this result would not extend to the $SearchVertex$ predicate: the smallest branching programs for this predicate are generally not read-once. But there is hope that the result holds for the $SearchClause$ predicate. We leave this open for future work.

Part III

Extending the Knowledge Compilation Map to New Fields

Chapter 5

Enumeration Queries on Compilation Languages

The compilation of knowledge into a language is only half of knowledge compilation. The second part, the online part, is reasoning on the circuit resulting from the compilation process. One of the goals of a knowledge compilation map is to help a user choosing a compilation language for its problem by linking each language to queries and transformations that are tractable for the language. It is then up to the user to decide which queries and transformations are useful to solve the problem at hand, and to choose the compilation language accordingly. In Darwiche and Marquis' compilation map [DM02], eight queries that occur naturally in logical reasoning (satisfiability, model counting, clausal entailment, etc.) and eight transformations corresponding to basic logical operations (conjunction, disjunction, negation, etc.) are studied. However each new application of knowledge compilation brings its bucket of new queries to be analyzed for each compilation language. Examples of new queries include forgetting queries in QBF solving [CBLM05, FM06], minimization queries with respect to a cost function in optimization problems [KBLM16], and verification and explanation queries in the context of eXplainable AI (XAI) [AKM20]. In this chapter, we focus on the hardness of particular queries that are *enumeration queries*.

Enumeration problems consist in listing the elements of a set. This type of problems has appeared more than 50 years ago in graph theory with the enumeration of cycles [Tie70] and have since found applications in many other domains of computer science including logic, database theory, data mining, etc. (see [Was16] for an ongoing compendium of enumeration algorithms). Enumeration queries differ from decision and function queries in that the expected answer is not a single element but potentially a large sequence of solutions. Thus one has to worry about how the solutions are listed: are they given together in one big set or are they listed incrementally and, in the latter case, what about guarantees on their incremental generation? As an example, one of the eight queries of Darwiche and Marquis' map is *model enumeration* that asks to generate the set of all satisfying assignments of a circuit in output-polynomial-time. The existence of such output-polynomial time algorithm is a desirable but one is often more interested in more efficient enumeration algorithms, see e.g. [Str19] for a survey on the different enumeration complexity classes. Given the reasoning power of the different compilation languages, one can expect efficient enumeration algorithms in these languages. This is certainly true for model enumeration [DM02, CS21, ABJM17] but in this chapter we show more mitigated results for the enumeration of specific implicants, namely prime implicants, sufficient reasons and subset-minimal abductive explanations.

After a preliminary section where we introduce the specific implicants to be considered and some enumeration complexity classes, we study the enumeration of prime implicants from circuits in dec-

DNNF and prove that it is in IncP, the class of polynomial incremental time enumeration problems. We then focus on two closely related, but seemingly harder, enumeration problems where further restrictions are put on the implicants to be generated. In the first problem, one is only interested in prime implicants representing subset-minimal abductive explanations, a notion much investigated in AI for more than three decades. In the second problem, the target are prime implicants representing sufficient reasons, a recent yet important notion in the emerging field of eXplainable AI, since they aim to explain predictions achieved by Machine Learning classifiers. We provide evidence showing that enumerating prime implicants corresponding to subset-minimal explanations or to sufficient reasons is not even feasible in output-polynomial time. The results of this chapter have been presented in the article [dCM22a] co-authored with Pierre Marquis.

For the sake of readability, in this chapter, several proofs are deferred to Section 5.5..

5.1 Enumeration Queries and Enumeration Complexity

5.1.1 Enumeration Complexity

We introduce some enumeration complexity classes as described in [Str19]. Let Σ be an alphabet and let A be a binary predicate in $\Sigma^* \times \Sigma^*$. Given an instance $x \in \Sigma^*$ (the input), $A(x)$ (the set of solutions) denotes the set of all $y \in \Sigma^*$ such that $A(x, y)$. The enumeration problem Enum·A is the function mapping x to $A(x)$. We say that Enum·A is in the class EnumP if for every $y \in A(x)$, $|y|$ is polynomial in $|x|$, and if deciding whether y is in $A(x)$ is in P. Contrary to other enumeration complexity classes, EnumP does not capture the complexity of *computing* the set of solutions $A(x)$, it serves more as a counterpart of NP for enumeration problems.

The computation model used for the enumeration of solutions is the random access machine (RAM) model, see e.g. [AHU74] for a description of this model and [Str19] for details on why RAM have been chosen rather than Turing machines. We do not feel compelled to give details on the RAM model as we do not deal with fine-grained complexity results in this chapter (we are satisfied by just proving that such or such algorithm runs or does not run in time polynomial in the size of its input). A RAM solves Enum·A if, for all x, it returns a sequence $y_1, \ldots, y_m$ of pairwise distinct elements such that $\{y_1, \ldots, y_m\} = A(x)$. We say that Enum·$A$ is in OutputP if there is a RAM solving Enum·A in time $O(poly(|x| + |A(x)|))$ on every input x. OutputP is a relevant enumeration class when the whole set of solutions is explicitly asked for. For instance, the dualization of a monotone CNF formula ϕ is the task of generating a DNF formula equivalent to ϕ. Because of the monotony condition on ϕ, the terms used in any smallest DNF formula equivalent to ϕ are precisely its prime implicants. Thus, the dualization problem boils down to enumerating *all* the prime implicants of ϕ. In this context an output-polynomial time algorithm is relevant.

For other applications, computing only a fixed number of solutions may be enough. A RAM solves Enum·A in incremental time $f(t)g(n)$ if on every x, it runs in time $O(f(t)g(|x|))$ and returns a sequence $y_1, \ldots, y_t$ of t pairwise distinct elements of $A(x)$ when $t \leq |A(x)|$, and the whole set $A(x)$ when $t > |A(x)|$. We say that Enum·A is in IncP_a if there is a RAM that solves A in incremental time $O(t^a n^b)$ for some constant b. We write $\mathsf{IncP} = \bigcup_{a \geq 0} \mathsf{IncP}_a$. IncP has a simple characterization that uses the function problem AnotherSol·A which, given x and $S \subseteq A(x)$, returns some $y \in A(x) \setminus S$ when $S \neq A(x)$, and *false* otherwise. Recall that FP is the functional version of P, that is, FP is the class of function problems (and not only decision problems) that can be solved by a deterministic Turing machine in polynomial time. It is known that:

Proposition 3 ([Str19])**.** *A problem Enum·A in* EnumP *is in* IncP *if and only if AnotherSol·A is in* FP*. Moreover, if there is a algorithm answering AnotherSol·$A(x, S)$ in $O(|S|^a |x|^b)$ for some constants a and*

b, then Enum·A is in IncP_{a+1}.

It is readily verified that $\mathsf{IncP}_a \subseteq \mathsf{OutputP}$ for every $a \geq 0$, thus $\mathsf{IncP} \subseteq \mathsf{OutputP}$. Moreover the inclusion is believed to be strict [Str19].

Our investigation of the tractability of enumeration problems in compilation remains mostly theoretical. When practical applications are envisioned, enumeration algorithms are usually split into two phases: a preprocessing phase at the end of which the input (x in our case) is enriched with indexes, and the enumeration phase itself. Efficient enumeration algorithms are then subject to strict requirements which are that the preprocessing phase must be feasible in linear time, and that the delay between successive solutions in the enumeration phase be constant or at worst linear in the size of the solution. In particular, the delay between successive solution must be independent of the size of the indexed input (see [Seg14] for a survey on constant-delay enumeration). Thus both constant-delay enumeration and linear-delay enumeration are more restrictive than incremental polynomial-time enumeration and the interest of the class IncP is more theoretical than practical.

5.1.2 Enumeration Queries in the Compilation Map

In the settings that interest us, the instance previously denoted by x is restricted to circuits in a given compilation language L. We use the term "Enum·A from L" to refer to the enumeration problem Enum·A whose input is taken in L. Let us clearly set the boundaries of our investigation on enumeration queries in compilation languages: for a given enumeration problem Enum·A over a compilation language L, our goal is first to determine whether Enum·A from L is in OutputP, and if the answer is positive, then we want to determine whether Enum·A from L is in IncP. In particular, when Enum·A from L is *not* (or thought not to be) in OutputP, we provide evidence supporting this claim but we do not further investigate the complexity of Enum·A from L.

We illustrate the enumeration classes defined in the previous section using enumeration queries that are already (more or less directly) studied in the compilation map. In the following definitions, D is a circuit in L. The predicates *MOD* and *CMOD* describe the models and the counter-models of a circuit D.

$$MOD(D, a) \Leftrightarrow a \text{ is an assignment to } var(D) \text{ that satisfies } D$$
$$CMOD(D, a) \Leftrightarrow a \text{ is an assignment to } var(D) \text{ that falsifies } D$$

Thus we have $MOD(D) = sat(D)$ and $CMOD(D) = \{0,1\}^{var(D)} \setminus sat(D)$. Enum·*MOD* from DNNF, d-DNNF, etc. is one of the eight initial query from the knowledge compilation map of Darwiche and Marquis [DM02]. Indeed "Enum·*MOD* from L" corresponds to the query **ME** (Model Enumeration). A language L is said to support **ME** if and only if Enum·*MOD* from L is in OutputP. Since it is shown in [DM02] that all sublanguages of DNNF, and more generally all languages where conditioning and satisfiability check are tractable, supports **ME** we already have the following:

Lemma 55. [DM02, Lemma A.3] *Let L be a language supporting polynomial-time conditioning and polynomial-time satisfiability check, then Enum·MOD from L is in* OutputP.

It is straightforward to adapt the algorithm given in [DM02, Lemma A.3] so that it incrementally generates of models of a circuit in DNNF. The algorithm is for instance details in the proof of [FM14, Proposition 3].

Lemma 56. [FM14, Proposition 3] *Let L be a language supporting polynomial-time conditioning and polynomial-time satisfiability check, then Enum·MOD from L is in* IncP_1.

The dual problem of enumerating counter-models can be studied analogously. The idea is simply to replace satisfiability queries by validity queries, i.e., to ask whether a circuit computes a function that has

no counter-models. So for the problem "Enum·*CMOD* from L" one just has to know whether validity testing – a query known as **VA** in [DM02] – (and conditioning) is feasible in polynomial-time in L.

Lemma 57. *Let L be a compilation language that does not support polynomial-time validity check unless a hypothesis H from complexity theory fails, then Enum·CMOD from L is not in* OutputP *unless H fails.*

Proof. Assume there exists an algorithm that returns the set M of counter-models of a circuit $D \in L$ in less than $p(|D|)q(|M|)$ elementary steps (so in $O(p(|D|)q(|M|))$ time) with p and q two polynomials. Then one can decide the validity of D by running that algorithm for at most $p(|D|)q(1)$ steps. If the algorithm has not stopped by then, then D has at least one counter-model and thus is not valid. □

Replacing satisfiability queries by validity queries in the algorithm for Enum·*MOD* described in [FM14, Proposition 3], it follows that:

Lemma 58. *Let L be a compilation language that supports polynomial-time validity check and conditioning then Enum·CMOD from L is in* IncP_1.

Consequences of these lemmas are that Enum·*CMOD* from DNNF is not in OutputP unless P = NP while Enum·*CMOD* from d-DNNF is in IncP_1. We should note that by this latter result, enumerating the models of the negation of a circuit D in d-DNNF is easy, while no definitive answer (or even conditional answer) to the question "does there exist a polynomial-time algorithm to generate a circuit in d-DNNF computing $\neg D$" is known.

5.1.3 New Enumeration Queries

We now turn to the enumeration of objects more complexe than models and counter-models. The objects that we focus on are derivatives of the dual concepts of *prime implicants* and *prime implicates*. Recall that an implicant t of a function f, thus a term t such that $t \models f$, is a prime implicant of f if and only if no implicant t' of f distinct from t is such that $t \models t'$. Similarly a prime implicate of f is a clause c such that $f \models c$ and no implicate c' of f distinct from c is such that $c' \models c$. We define the predicates

$$IP(f, t) \Leftrightarrow t \text{ is a prime implicant of } f$$
$$PI(f, c) \Leftrightarrow c \text{ is a prime implicate of } f$$

So $IP(f)$ and $PI(f)$ are the set of prime implicants of f and the set of prime implicates of f, respectively. Within AI, prime implicants and prime implicates have been considered for modelling and solving a number of problems, including compiling knowledge [RdK87] and generating explanations of various kinds. This is the case in logic-based abductive reasoning (see e.g., [SL90, EG95]), a form of inference required in many applications when the available knowledge base is incomplete (e.g., in medicine) and because of such an incompleteness, it cannot alone explain the observations that are made about the state of the world. Formally, the explanations one looks for are terms over a preset alphabet (composed of the so-called *abducible variables*, e.g., representing diseases) such that the manifestations that are reported (e.g., some symptoms) are logical consequences of the background knowledge when completed by such a term. In order to avoid trivial explanations, one also asks those terms to be consistent with the knowledge base. Explanations that are the less demanding ones from a logical standpoint (i.e., subset-minimal ones) can be characterized as specific prime implicants. Thus we introduce the predicate *SE*, for *subset minimal explanations*, that capture implicants of a function over a variable set X given as part of the input:

$$SE((f, X), t) \Leftrightarrow t \text{ is a prime implicant of } f \text{ such that } var(t) \subseteq X$$

In model-based diagnosis, one is interested in computing kernel diagnoses [dKMR92], which are also specific prime implicants of the Boolean function representing the available knowledge about the system to be diagnosed (i.e., the description of the system, together with the observations made, e.g., the inputs and outputs of the system). Only those prime implicants built up from variables used to represent the states of the components of the system (normal or not) are considered. More recently, deriving explanations justifying why certain predictions have been made has appeared as essential for ensuring trustworthy Machine Learning (ML) technologies [Mil19, Mol19]. In the research area of eXplainable AI (XAI), recent work has shown how ML classifiers of various types (including black boxes) can be associated with Boolean circuits (alias transparent or "white" boxes), exhibiting the same input-output behaviours [NKR+18, SCD18a, SCD19a]. Thanks to such mappings, XAI queries about classifiers can be delegated to the corresponding circuits. The notion of sufficient reasons of an instance given a Boolean function f modelling a binary classifier has been introduced in [DH20]. Given an assignment a to the problem variables such that $f(a) = 1$ (resp. $f(a) = 0$), a sufficient reason for a is a subset-minimal partial assignment a' which is compatible with a (i.e., a and a' give the same values to the variables that are assigned in a') and which satisfies the property that for every extension a'' of a' we have $f(a'') = 1$ (resp. $f(a'') = 0$). The features assigned in a' (and the way they are assigned) can be viewed as explaining why a has been classified by f as a positive (or as a negative) instance. Thus we introduce another predicate for sufficient reasons:

$$\mathit{SR}((f, a), t) \Leftrightarrow t \text{ is a prime implicant of } f \text{ satisfied by the assignment } a$$

For simplicity we use the ternary notations $\mathit{SR}(f, a, t)$ and $\mathit{SE}(f, a, t)$ and we let $\mathit{SR}(f, a)$ denote the set of sufficient reason for a given f and $\mathit{SE}(f, X)$ denote the set of subset-minimal abductive explanations over X given f.

Whatever the way prime implicants are used, generating them is in general a computationally demanding task, for at least two reasons. On the one hand, deriving a single prime implicant of a Boolean function represented by a propositional formula (or circuit) is NP-hard since such a formula is satisfiable when it has a prime implicant, and it is valid precisely when this prime implicant is the empty term. On the other hand, a source of complexity is the number of prime implicants that may prevent from computing them all. Indeed, it is well-known that the number of prime implicants of a Boolean function can be exponential in the number of variables of the function, and, for many representations of the function, also exponential in the size of the representation (just consider the parity function as an example). In more detail, the number of prime implicants of a Boolean function can be larger than the number of models of the function [DF59]; there also exist families of Boolean functions over n variables having $\Omega(\frac{3^n}{n})$ prime implicants [CM78].

In the next sections we study Enum·*IP*, Enum·*PI*, Enum·*SE* and Enum·*SR* from sublanguages of DNNF, in particular from dec-DNNF. We start with the general problems of enumerating prime implicants and prime implicates before addressing the cases of specific implicants (sufficient reasons and subset minimal explanations).

5.2 Enum·*IP* and Enum·*PI* from dec-DNNF are in OutputP

We start with a couple of easy results. First of all, since there is a linear-time procedure to verify that a term is an implicant of a circuit in d-DNNF, there is a polynomial-time algorithm to decide whether a given term is a prime implicant of a circuit in d-DNNF. Similarly, there is a linear-time algorithm for checking whether a given clause is an implicate of a circuit in DNNF, so there also is a polynomial-time algorithm to decide whether it is a prime implicate of the circuit. On the other hand, there is no

polynomial-time algorithm answering validity queries on arbitrary circuits in DNNF unless P = NP. Since deciding the validity of a function f boils down to deciding whether $IP(f) = \{t_\emptyset\}$ we get:

Lemma 59.

- *Enum·IP from* d-DNNF *is in* EnumP.
- *Enum·IP from* DNNF *is not in* EnumP *unless* P = NP.
- *Enum·PI from* DNNF *is in* EnumP.

Deciding the validity of a function f is also equivalent to deciding whether $PI(f) = \emptyset$ thus

Lemma 60. *Enum·PI from* DNNF *is not in* OutputP *unless* P = NP.

The question of the membership of Enum·*IP* from d-DNNF and of Enum·*PI* from d-DNNF in OutputP eludes us, but we are able to give an answer for the sublanguage dec-DNNF. Before that we impose a restriction on the circuits in dec-DNNF studied in this chapter. We say that a a circuit D in dec-DNNF is *reduced* when, for every node v of D, we have $D_v \equiv 0$ if and only if v is a leaf labelled by 0, and $D_v \equiv 1$ if and only if v is a leaf labelled by 1. Since satisfiability queries and validity queries are feasible in linear time in dec-DNNF, reducing a circuit in dec-DNNF is a linear-time preprocessing.

It is known that Enum·*IP* from OBDDs is in OutputP [MC91], and it is not hard to extend this result to circuits in dec-DNNF. We briefly describe the output-polynomial-time construction of $IP(D)$ for a circuit D in dec-DNNF as it is important for subsequent results. The construction is based on the two following folklore propositions. Given a set of Boolean functions $\mathcal{F}$, we denote by $\max(\mathcal{F}, \models)$ (resp. $\min(\mathcal{F}, \models)$) the subset of functions $f \in \mathcal{F}$ such that no $f' \in \mathcal{F} \setminus \{f\}$ verifies $f \models f'$ (resp. $f' \models f$).

Proposition 4. [Mar93] *Let f and g be Boolean functions. Then $IP(f \wedge g) = \max(\{t \wedge t' \mid t \in IP(f), t' \in IP(g)\}, \models)$. Furthermore, if $var(f) \cap var(g) = \emptyset$, then $IP(f \wedge g) = \{t \wedge t' \mid t \in IP(f), t' \in IP(g)\}$.*

Proposition 5. *Let f be a Boolean function and let x be a variable. Then*

$$IP(f) = \{t \wedge \overline{x} \mid t \in IP(f|\overline{x}),\ t \not\models f|x\} \cup \{t \wedge x \mid t \in IP(f|x),\ t \not\models f|\overline{x}\} \cup IP(f|\overline{x} \wedge f|x)$$

Note that $t \in IP(f|\overline{x})$ (resp. $IP(f|x)$) entails $f|x$ (resp. $f|\overline{x}$) if and only if t is subsumed by some term in $IP(f|\overline{x} \wedge f|x)$. As a consequence, from $IP(f|\overline{x})$ and $IP(f|x)$, one construct $IP(f|\overline{x} \wedge f|x)$ in polynomial time thanks to Proposition 4 and we use it to derive $IP(f)$ thanks to Proposition 5.

We also have that:

Proposition 6. *Let f be a Boolean function and let x be a variable. Then*

$$|IP(f)| \geq \max(|IP(f|\overline{x})|, |IP(f|x)|).$$

For the proof of Proposition 6, we point to the algorithm for conditioning a prime implicant representation provided in [DM02]. The algorithm constructs the prime implicants of $f|\ell$ as the logically weakest terms of the DNF formula $\bigvee_{t \in IP(f)} t|\ell$, or more formally $IP(f|\ell) = \max(\{(t|\ell) \mid t \in IP(f)\}, \models)$. This is enough to derive $|IP(f|\ell)| \leq |IP(f)|$.

Consider now a circuit D in dec-DNNF and an internal node v with two children u and w. If the sets $IP(D_u)$ and $IP(D_w)$ are provided, then $IP(D_v)$ is obtained in polynomial-time using Proposition 4 if v is a decomposable $\wedge$-gate, and using Proposition 5 if v is a decision node. Furthermore, in both cases, we have $|IP(D_v)| \geq \max(|IP(D_u)|, |IP(D_w)|)$. These observations lead to a simple algorithm that generates $IP(D)$ by computing the sets $IP(D_v)$ for every node v of D considered in a depth-first traversal. Since constructing the set of prime implicants for any node given that of its children is tractable, since this set is smaller than $|IP(D)|$, and since it is computed only once, the algorithm runs in time $O(poly(|D| + |IP(D)|))$. Thus, we get:

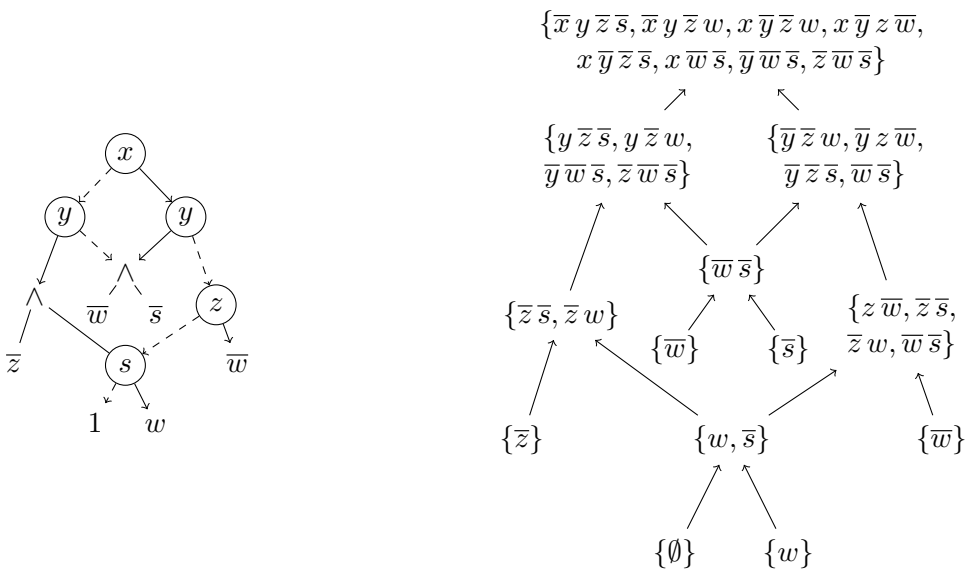


Figure 5.1: An output-polynomial time construction of the set of prime implicants of a dec-DNNF.

Lemma 61. *Enum·IP from* dec-DNNF *is in* OutputP.

Example 33. Figure 5.1 shows a circuit in dec-DNNF and the construction of the sets of prime implicants of each of its sub circuit. Following Proposition 5, we detail the construction of the set $\{z\,\overline{w}, \overline{z}\,\overline{s}, \overline{z}\,w, \overline{w}\,\overline{s}\}$ of prime implicants of the circuit rooted under the right-most decision node labelled by y. Call that circuit D. For the construction we assume that we have already computed the set of prime implicants of the children of the root of D, in that case $\{\overline{w}\,\overline{s}\}$ and $\{z\,\overline{w}, \overline{z}\,\overline{s}, \overline{z}\,w, \overline{w}\,\overline{s}\}$. To construct $IP(D)$ from Proposition 5 we first look at the terms obtained conjoining $\overline{y}$ to prime implicants of $D|\overline{y}$, we obtain $\overline{y}\,z\,\overline{w}$, $\overline{y}\,\overline{z}\,\overline{s}$, $\overline{y}\,\overline{z}\,w$ and $\overline{y}\,\overline{w}\,\overline{s}$. Since only the fourth term is an implicant of $D|y$ we have $\{\overline{y}\,z\,\overline{w}, \overline{y}\,\overline{z}\,\overline{s}, \overline{y}\,\overline{z}\,w\} \subseteq IP(D)$. Then we look at the terms obtained conjoining y to prime implicants of $D|\overline{y}$, we have only one: $y\,\overline{w}\,\overline{s}$. Since that term is an implicant of $D|\overline{y}$, it is not in $IP(D)$. Finally the remaining implicants of D are exactly the terms in $IP(D|y \wedge D|\overline{y}) = \max(\{z\,\overline{w}\,\overline{s}, \overline{z}\,\overline{w}\,\overline{s}, \overline{z}\,w\,\overline{w}\,s, \overline{w}\,\overline{s}\}, \models) = \{\overline{w}\,\overline{s}\}$. ◀

Recall that circuits in dec-DNNF are not structured (by a vtree) generally. In particular, for the sublanguage of read-once binary decision diagrams, it is not required that all computation paths respect the same variable ordering for the result of Lemma 61 to hold (the result applies to FBDDs as much as to OBDDs).

It is worth explaining in a few lines why the output-polynomial-time construction of $IP(D)$ for D a circuit in dec-DNNF is not easily extended to circuits in d-DNNF. A key component of the construction is the fact that for a node v of the circuit D in dec-DNNF that has two children u and w, it holds that $|IP(D_v)| \geq \max(|IP(D_u)|, |IP(D_w)|)$. This property can be interpreted by saying that it is not a waste of time or resource to compute $IP(D_u)$ and $IP(D_w)$: we might as well compute $IP(D_u)$ and $IP(D_w)$ to construct $IP(D_v)$ since the latter set is necessarily larger and can be computed easily from the former two. In a circuit in d-DNNF, this property can be lost at deterministic $\vee$-nodes. Indeed, consider the simple case of a deterministic circuit D' over $X \cup \{y\}$ whose root v is a $\vee$-node that has two children u and w. D'_u computes the function $y \wedge even(X)$ where $even(X)$ accepts exactly the assignments to X that set an even number of variables to 1, and D'_w computes the function $y \wedge odd(X)$ where $odd(X)$ accepts exactly the assignments to X that set an odd number of variables to 1. So $D' = D'_v \equiv D'_u \vee D'_w \equiv y$. We have that D' is reduced and that $IP(D') = \{y\}$. However the prime implicants of $even(X)$ and $odd(X)$

are exactly their respective models so both $IP(y \wedge even(X))$ and $IP(y \wedge odd(X))$ have size $2^{|X|-1}$. This simple example shows that we can have $|IP(D_v)| \ll \min(|IP(D_u)|, |IP(D_w)|)$ in a reduced circuit in d-DNNF. This discourages us from applying a procedure to compute the prime implicants of a circuit in d-DNNF similar to that used for circuits in dec-DNNF. However this does *not* mean that there is no output-polynomial-time algorithm for Enum·IP from d-DNNF. The existence of such an algorithm still eludes us.

We rapidly discuss the case of Enum·PI from dec-DNNF. The concept of prime implicates is dual to that of prime implicants and the above propositions for prime implicants have a dual version from prime implicates.

Proposition 7. *Let f and g be Boolean functions. Then $PI(f \vee g) = \min(\{c \vee c' \mid c \in PI(f), c' \in PI(g)\}, \models)$.*

Proposition 8. *Let f and g be Boolean functions. If $var(f) \cap var(g) = \emptyset$ and if f and g are satisfiable then $PI(f \wedge g) = PI(f) \cup PI(g)$ and $PI(f) \cap PI(g) = \emptyset$.*

Proposition 9. *Let f be a Boolean function and let x be a variable.*

$$\begin{aligned} PI(f) = &\{c \vee \overline{x} \mid c \in PI(f|\overline{x}),\ f|x \not\models c\} \\ &\cup \{c \vee x \mid c \in PI(f|x),\ f|\overline{x} \not\models c\} \\ &\cup IP(f|\overline{x} \vee f|x) \end{aligned}$$

Proposition 10. *Let f be a Boolean function and let x be a variable. Then*

$$|PI(f)| \geq \max(|PI(f|\overline{x})|, |PI(f|x)|).$$

Propositions 8 and 9 ensure that for u a node in a circuit D in dec-DNNF that has two children v and w, $PI(D_u)$ can be computed from $PI(D_v)$ and $PI(D_w)$ in polynomial-time. And Propositions 8 and 10 ensure that $|PI(D_u)| \geq \max(|PI(D_v)|, |PI(D_w)|)$. Thus the procedure that we use to prove that Enum·*IP* from dec-DNNF is in OutputP can be modified to compute prime implicates and therefore

Lemma 62. *Enum·PI from* dec-DNNF *is in* OutputP.

Note that extending the procedure to circuits in d-DNNF is again not an option, even if Proposition 7 gives a polynomial time construction of $PI(D_v)$ from $PI(D_u)$ and $PI(D_w)$ when v is an $\vee$-node. Indeed we have the problem that $|PI(D_v)|$ may be much smaller than $\min(|PI(D_u)|, |PI(D_w)|)$. For instance when D_u computes $even(X)$ and D_w computes $odd(X)$ we have that $|PI(D_v)| = 0$ while $|PI(D_u)| = |PI(D_w)| = 2^{|X|-1}$. Again, the impossibility to extend our output-polynomial-time construction from circuits in dec-DNNF to circuits in d-DNNF does *not* mean that Enum·*PI* from d-DNNF is in not OutputP. This is still an open question.

5.3 Enum·*IP* and Enum·*PI* from dec-DNNF are in IncP

We investigate Enum·*IP* and Enum·*PI* from dec-DNNF from the incremental enumeration perspective. Based on Proposition 3, we design a tractable algorithm `AnotherIP` for solving the problem AnotherSol·*IP*, thus showing that Enum·*IP* from dec-DNNF is in IncP. The case of Enum·*PI* is again dual to that of Enum·*IP*.

5.3.1 Solving the decision variant of AnotherSol·IP

We first consider the decision variant of AnotherSol·*IP* from dec-DNNF: given a circuit D in dec-DNNF and a set $S \subseteq IP(D)$, return *false* if and only if $S \neq IP(D)$. The problem is easily answered by a simple hack of the output-polynomial-time algorithm described in the previous section. The idea is simply to run the output-polynomial-time algorithm but every time a set $IP(D_v)$ has been constructed for some node v in D, we compare $|IP(D_v)|$ to $|S|$. If $|IP(D_v)| > |S|$ then since $|IP(D)| \geq |IP(D_v)|$ we have that $S \neq IP(D)$.

Lemma 63. *There is an algorithm which, given a circuit D in dec-DNNF and a set $S \subseteq IP(D)$, returns* false *if $S \neq IP(D)$ and* true *otherwise, and runs in time $O(|S|^2 poly(|D|))$.*

Proof. The output-polynomial-time algorithm to generate $IP(D)$ described in the previous section works by constructing the sets $IP(D_v)$ for every node v of D taken in depth-first order. As explained before, to decide whether $S = IP(D)$, we can modify the algorithm so that it stops as soon as it has constructed a set $IP(D_v)$ for some node v such that $|IP(D_v)| > |S|$. If no such node is found then $IP(D) = S$ and the algorithm has taken $O(|IP(D)|poly(|D|)) = O(|S|poly(|D|))$ time to terminate. Otherwise assume the algorithm ends after visiting k nodes $v_1, \ldots, v_k = v$ (in that order) and finding that $|IP(D_v)| > |S|$. Then we have that $|IP(D_{v_i})| \leq S$ for all $i < k$ (otherwise the algorithm would have terminated before). The running time of the algorithm in this case is $O(\sum_{i=1}^{k} |IP(D_{v_i})|poly(|D|)) = O((|IP(D_v)| + |S|(k-1))poly(|D|)) = O((|IP(D_v)| + |S||D|)poly(|D|))$. We just have to show that $|IP(D_v)| = O(|S|^2)$ to conclude. $IP(D_v)$ is constructed from two sets $IP(D_{v_i})$ and $IP(D_{v_j})$ with $i, j < k$. Either v is a decomposable $\wedge$-node and by Proposition 4 we have $|IP(D_v)| = |IP(D_{v_i})||IP(D_{v_j})| \leq |S|^2$, or v is a decision node for a variable x and by Proposition 5 we have $|IP(D_v)| \leq |IP(D_{v_i})|+|IP(D_{v_j})|+|IP(D_{v_i} \wedge D_{v_j})| \leq 2|S|+|IP(D_{v_i} \wedge D_{v_j})|$. Since by Proposition 4 we have $|IP(D_{v_i} \wedge D_{v_j})| \leq |IP(D_{v_i})||IP(D_{v_j})|$, we have $|IP(D_v)| \leq 2|S| + |S^2| = O(|S^2|)$. □

5.3.2 Propagating a prime implicant in D

Consider the algorithm of the previous subsection that decides whether $S = IP(D)$ and assume it has returned *false* because it has constructed a set $IP(D_v)$ for some node v in D such that $|IP(D_v)| > |S|$. We do not know how to modify the algorithm so that, instead of returning *false*, it returns a prime implicant of D that is not in S. More precisely, there is only one "good" situation where we know how to adapt the algorithm. In this section, we describe this "good" situation as it allows us to introduce the notion of *propagation* of a prime implicant that will be useful later.

The "good" situation is when there is a term $t \in IP(D_v)$ such that t is not entailed by any $t' \in S$. Then we can construct a prime implicant of D that entails t, and therefore is not in S. To do so, we choose a path from the root of D to v and we *propagate t along that path* using the following proposition and Proposition 4.

Proposition 11. *Let f be a Boolean function, let x be a variable, and let $\ell \in \{x, \overline{x}\}$. Consider $t \in IP(f|\ell)$. If $t \models f|\overline{\ell}$, then $t \in IP(f)$, otherwise $t \wedge \ell \in IP(f)$.*

Proof. Suppose $t \models f|\overline{\ell}$. Then t is an implicant of f since $t \models (f|\overline{x} \wedge f|x) \models ((\overline{x} \wedge f|\overline{x}) \vee (x \wedge f|x)) \equiv f$. To prove that it is prime let t' be a strict subterm of t and assume $t' \models f$. We have $x \notin var(t)$ since $x \notin var(f|\ell)$, so $t'|\ell = t'$. If $t' \models f$ then $t' = t'|\ell \models f|\ell$ and t is not a prime implicant of $f|\ell$, a contradiction.

Suppose $t \not\models f|\overline{\ell}$. Then $t \wedge \ell$ is an implicant of f since $t \wedge \ell \models (\ell \wedge f|\ell) \models ((\overline{x} \wedge f|\overline{x}) \vee (x \wedge f|x)) \equiv f$. To prove that it is prime, let t' be a strict subterm of $t \wedge \ell$ and assume $t' \models f$. If $\ell \notin t'$ then $t' = t'|\ell \models f|\ell$ and t is not a prime implicant of $f|\ell$, a contradiction. If however $\ell \in t'$, then write $t' = t'' \wedge \ell$ and observe that $t'' = t'|\ell \models f|\ell$, so t is not a prime implicant of $f|\ell$, another contradiction. □

The idea behind the propagation of t is simple. Consider the parent node p of v. If p is a decomposable $\wedge$-node whose children are v and v', then by Proposition 4 we find a prime implicant $IP(D_p)$ by conjoining any prime implicant of $D_{v'}$ to t. One can find a prime implicant of $D_{v'}$ in $O(poly(|D|))$ time using a procedure called `GenerateIP` (that will be given later). Otherwise, if p is a decision node for a variable x, then by Proposition 11 either t or $t \vee \ell_x$ is a prime implicant of D_p, where ℓ_x is some literal in x, and deciding which of the two is a prime implicant of D_p only requires deciding whether $t \models D_p|\overline{\ell_x} = D_{v'}$, which takes $O(|D|)$ time. So in both cases we can construct a prime implicant of D_p that entails t in $O(poly(|D|))$ time. We repeat the construction along the path until reaching the root of D, and end up with a prime implicant of D that entails t and thus is not in S.

Example 34. Figure 5.2b shows the propagation of the implicant $\overline{s} \in IP(D_{v_3})$ along the path (v_0, v_1, v_2, v_3). We construct a prime implicant of D_{v_2} from $\overline{s}$, here since v_3 is the 0-child of v_2 and since $\overline{s}$ does not entail the 1-child of v_2 we obtain $\overline{z}\,\overline{s} \in IP(D_{v_2})$. Then we construct a prime implicant of D_{v_1} from $\overline{z}\,\overline{s}$, here since v_2 is the 0-child of v_1 and since $\overline{z}\,\overline{s}$ does not entail the 1-child of v_1 we obtain $\overline{y}\,\overline{b}\,\overline{s} \in IP(D_{v_1})$. Repeating the step one more time leads to $x\,\overline{y}\,\overline{z}\,\overline{w} \in IP(D_{v_0}) = IP(D)$. ◀

Now the problem is that it is not clear that the "good" situation always occurs when $|IP(D_v)| > |S|$. In the next subsection we follow another direction and present another algorithm to solve Another·*IP*. The purpose of the current subsection was mainly to introduce the prime implicant propagation, which is useful in the next subsection.

5.3.3 Augmenting an incomplete subset of IP(D)

Recall the construction of $IP(D)$ discussed before Lemma 61. To address AnotherSol·*IP* on inputs D and S, a reverse procedure is performed under the assumption that $S = IP(D)$. Thus the idea is to traverse the circuit top-down while updating S, until we find some kind of "contradiction" – which we will describe later – at a node of D. Then we will create a particular prime implicant of the circuit rooted at that node, and we will propagate it along the path that has been followed to find the contradiction. Figures 5.2a and 5.2b – that will be explained later – give an idea of the two-step procedure: "parse the circuit top-down until finding contradiction" into "propagation of a new prime implicant".

Before defining what a contradiction means in this setting, recalling some notations and propositions is useful. For t a term and X a set of variables, t_X denotes the restriction of t to variables in X. If X and $var(t)$ are disjoint, then t_X is the empty term. Note that the notation $t_\emptyset$ is consistent with the notation t_X.

It is convenient to introduce propositions that "reverse" Proposition 4 and 5. The proofs are written in Section 5.5.

Proposition 12. *Let D be a circuit in dec-DNNF and let $S \subseteq IP(D)$. If the root of D is an $\wedge$-node, let u and w be its children and let $S_u = \{t_{var(D_u)} \mid t \in S\}$ and $S_w = \{t_{var(D_w)} \mid t \in S\}$. Then $S_u \subseteq IP(D_u)$ and $S_w \subseteq IP(D_w)$ hold, and*

$$S = IP(D) \text{ iff } S_u = IP(D_u) \text{ and } S_w = IP(D_w) \text{ and } S = \{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}.$$

Proposition 13. *Let D be a circuit in dec-DNNF whose root is a decision node labelled by x. Let u be its 0-child and w be its 1-child. Given $S \subseteq IP(D)$, let $S_u = \{t \mid t \wedge \overline{x} \in S\} \cup (S \cap IP(D_u))$, $S_w = \{t \mid t \wedge x \in S\} \cup (S \cap IP(D_w))$ and $S' = \{t \mid t \in S, x \notin var(t)\}$. Then $S_u \subseteq IP(D_u)$ and $S_w \subseteq IP(D_w)$ hold, and*

$$\begin{aligned} S = IP(D) \text{ iff } & S_u = IP(D_u) \text{ and } S_w = IP(D_w) \\ & \text{and } S' = \max(\{t_u \wedge t_w \mid t_u \in S_u,\, t_w \in S_w\}, \models). \end{aligned}$$

We point out that, since verifying that a term is a prime implicant of a circuit in dec-DNNF is tractable, the sets S_u and S_w in Proposition 13 can be determined in polynomial time given S and D. Let v be a node in D and let $S_v \subseteq \mathit{IP}(D_v)$. When asked to determine whether $S_v = \mathit{IP}(D_v)$, we say that we have a *contradiction* at node v when

(c1) $S_v = \emptyset$ while D_v is satisfiable, or

(c2) v is a decision node with children u and w, and we have $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$ but $S' \neq \max(\{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}, \models)$, where S_u, S_w and S' are defined as in Proposition 13, or

(c3) v is a decomposable $\wedge$-node and $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$ but $S \neq \{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}$.

A contradiction at node v guarantees that $S_v \neq \mathit{IP}(D_v)$. We will construct the set S_v from S such that, if $S_v \neq \mathit{IP}(D_v)$ then $S \neq \mathit{IP}(D)$. Checking that a contradiction (c1) occurs at node v is easy. Contradictions (c2) and (c3) on the other hand require checking that $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$. While this can be done using Lemma 63, we choose to do it in another way that ensures that if

$$S_u \neq \mathit{IP}(D_u)$$

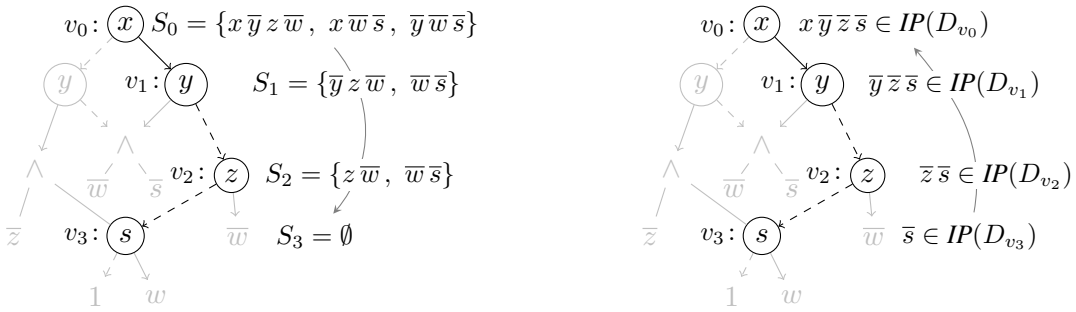
(resp. $S_w \neq \mathit{IP}(D_w)$), then a contradiction is found at a node below u (resp. w).

Now let us describe in more details how we determine whether $S = \mathit{IP}(D)$ (without using Lemma 63). Let r be the root of D. We assume that r is an internal node (a decision node or a decomposable $\wedge$-node) whose children are u and w. If there is no contradiction (c1) at r then we use Propositions 12 or 13 to construct the sets S_u and S_w, and we recursively ask whether $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$. Either the recursion ends because a contradiction has been found at a node under u or w, in which case we have that $S \neq \mathit{IP}(D)$, or it stops by itself (i.e., when reaching the leaves of the circuit without ever finding a contradiction), which shows that $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$. In the latter case, if r is a decomposable $\wedge$-node then we have to construct $\{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}$ to see whether it equals S. If it does then by Proposition 12 we have $S = \mathit{IP}(D)$, otherwise we have a contradiction (c3) at node r. If r is a decision node then we have to construct S' as in Propositions 12 and check whether S' equals $\max(\{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}, \models)$. If the two sets differ then we have a contradiction (c2) at node r and by Proposition 13 we derive that $S \neq \mathit{IP}(D)$, otherwise $S = \mathit{IP}(D)$.

If we find a contradiction at a node v in D when asked whether $\mathit{IP}(D_v) = S_v$, then we construct a prime implicant in $\mathit{IP}(D) \setminus S$ by first generating a prime implicant in $\mathit{IP}(D_v) \setminus S_v$ and by propagating it in D along the path that lead to v.

The procedure is given by Algorithm `MissingIP`. The inputs are a circuit D in dec-DNNF, a set $S \subseteq \mathit{IP}(D)$ and a path P in D. A function λ mapping the nodes of D to integers is used for memoization purposes. Initially $\lambda(v) = -1$ for every node v, but $\lambda(v)$ may be assigned a non-negative value at some point. More precisely, the first time a call $\texttt{MissingIP}(D_v, S, P)$ returns *false*, we learn that $S = \mathit{IP}(D_v)$ and set $\lambda(v)$ to $|S|$. Then for each later call $\texttt{MissingIP}(D_v, S', P')$ with $S' \subseteq \mathit{IP}(D_v)$, we check whether $S' = \mathit{IP}(D_v)$ by verifying that $\lambda(v) = |S'|$. P is used to remember the ancestor nodes that were visited before reaching a contradiction and, once a contradiction has been found, P helps us finding a prime implicant of $\mathit{IP}(D) \setminus S$.

Example 35. Consider calling $\texttt{MissingIP}(D, S_0, \emptyset)$ with D the circuit of Figure 5.1 and $S_0 = \{x\,\overline{y}\,z\,\overline{w}, x\,\overline{w}\,\overline{s}, \overline{y}\,\overline{w}\,\overline{s}\}$ a set of prime implicants of D. Figure 5.2a shows a scenario when $\texttt{MissingIP}(D, S_0, \emptyset)$ calls $\texttt{MissingIP}(D_{v_1}, S_1, (v_0))$, which in turn calls $\texttt{MissingIP}(D_{v_2}, S_2, (v_0, v_1))$, which finally calls $\texttt{MissingIP}(D_{v_3}, S_3, (v_0, v_1, v_2))$. Since $S_3 = \emptyset$ and D_{v_3} is reduced and different from 0,



(a) A path followed in `MissingIP`.

(b) Propagation of an implicant.

Figure 5.2: Generation of a new prime implicant from a circuit in dec-DNNF.

the algorithm has reached a contradiction (c1) at node v_3 and has not returned *false*, thus indicating that $S_0 \neq IP(D)$. The path followed to reach the contradiction is $P = (v_0, v_1, v_2, v_3)$. A prime implicant in $IP(D) \setminus S_0$ is generated by first finding a prime implicant in of D_{v_3}, say it is $\overline{s}$, and then by propagating it along P as described in Example 34. ◀

The proof of the next propositions on the soundess and running time of algorithm are deferred to Section 5.5.

Proposition 14. *Given a reduced circuit D in dec-DNNF and $S \subseteq IP(D)$, `MissingIP`$(D, S, \emptyset)$ runs in time $O(poly(|S| + |D|))$, and it returns false if and only if $S = IP(D)$.*

The algorithm `AnotherIP` that generates a new prime implicant breaks into two steps. First `MissingIP`$(D, S, \emptyset)$ searches for a contradiction. It returns $false$ if none is found, otherwise it returns a pair (t, P) with P the path followed to reach a node v where the first contradiction has been found, and t a prime implicant of D_v that could not be derived from S. The procedure `GenerateIP` is used to generate t. `GenerateIP` runs in polynomial time thanks to tractable model enumeration and linear-time implicant check on circuits in dec-DNNF. Finally `PropagateIP` is called to propagate t along the path P. The next proposition shows the correctness of `AnotherIP`:

Proposition 15. *Let D be a reduced circuit in dec-DNNF and let $S \subseteq IP(D)$. `AnotherIP`(D, S) runs in time $O(poly(|S| + |D|))$. It returns false if $S = IP(D)$, otherwise it returns a prime implicant of D that does not belong to S.*

On this basis, the existence of a polynomial incremental time enumeration of prime implicants for circuits in dec-DNNF can be easily established:

Lemma 64. *Enum·IP from* dec-DNNF *is in* IncP.

Proof. Using Proposition 15, k prime implicants of D can be generated in time $O(poly(k + |D|))$ by simply calling `AnotherIP`(D, S) k times, each time adding to S the new prime implicant that has been computed. This shows that Enum·*IP* from dec-DNNF is in IncP. □

5.3.4 The Dual Case of Prime Implicates

We briefly justify that the previous algorithm can be adapted for enumerating prime implicates. First let us consider the simple subcase where D is an FBDD instead of a circuit in dec-DNNF. It is easy to see that Enum·*PI* from FBDD is in IncP. Indeed, it is well-known that $PI(f) = \{\neg t \mid t \in IP(\neg f)\}$

Algorithm 3: `MissingIP`(D, S, P)

Promises: D is reduced, $S \subseteq IP(D)$

Let v be the root of D and let $P' \leftarrow P \cup (v)$
if $\lambda(v) = |S|$ **then** return *false*
if $S = \emptyset$ **then**
  if v is labelled by 0 **then** set $\lambda(v)$ to 0, return *false*
  else return (`GenerateIP`(D), P')
end
if v *is a* $\wedge$-node with children u and w **then**
  Construct S_u and S_w as in Proposition 12
  $r \leftarrow$ `MissingIP`(D_u, S_u, P')
  if $r \neq$ *false* **then** return r
  $r \leftarrow$ `MissingIP`(D_w, S_w, P')
  if $r \neq$ *false* **then** return r
  $S^* \leftarrow \{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}$
  if $S \neq S^*$ **then** return (t, P') for any $t \in S^* \setminus S$
else if v is a decision node with children u and w **then**
  Construct S_u, S_w, S' as in Proposition 13
  $r \leftarrow$ `MissingIP`(D_u, S_u, P')
  if $r \neq$ *false* **then** return r
  $r \leftarrow$ `MissingIP`(D_w, S_w, P')
  if $r \neq$ *false* **then** return r
  $S^* \leftarrow \max(\{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}, \models)$
  if $S^* \neq S'$ **then** for any $t \in S^* \setminus S'$ return (t, P')
end
Set $\lambda(v)$ to $|S|$ and return *false*

for every Boolean function, where $\neg t$ here denotes a clause. So given an FBDD B, one can negate B in constant time by switching the 0-sink with the 1-sink, and then enumerate the prime implicants of $\neg B$. Each prime implicant generated is then negated using De Morgan law to obtain a prime implicate of B. Unfortunately it is still unknown whether negation of a circuit in dec-DNNF (into another circuit in dec-DNNF) is feasible in polynomial-time. But we remark that the propositions used to design and analyse `MissingIP` have the following dual versions:

Proposition 16. *Let D be a satisfiable circuit in dec-DNNF and let $S \subseteq PI(D)$. If the root of D is an $\wedge$-node, let u and w be its children and let $S_u = \{c \mid c \in S, var(c) \cap var(D_u) \neq \emptyset\}$ and $S_w = \{c \mid c \in S, var(c) \cap var(D_w) \neq \emptyset\}$ (note that in this situation, $var(c) \cap var(D_u) \neq \emptyset$ if and only if $var(c) \subseteq var(D_u)$, and similarly for D_w). Then $S_u \subseteq PI(D_u)$ and $S_w \subseteq PI(D_w)$ hold, and*

$$S = PI(D) \text{ iff } S_u = PI(D_u) \text{ and } S_w = PI(D_w)$$

Proposition 17. *Let D be a circuit in dec-DNNF whose root is a decision node labelled by x. Let u be its 0-child and w be its 1-child. Given $S \subseteq PI(D)$, let $S_u = \{c \mid c \vee \overline{x} \in S\} \cup (S \cap PI(D_u))$, $S_w = \{c \mid c \vee x \in S\} \cup (S \cap PI(D_w))$ and $S' = \{c \mid c \in S, x \notin var(c)\}$. Then $S_u \subseteq PI(D_u)$ and $S_w \subseteq PI(D_w)$ hold, and*

$$\begin{aligned} S = PI(D) \text{ iff } & S_u = PI(D_u) \text{ and } S_w = PI(D_w) \\ & \text{and } S' = \min(\{c_u \vee c_w \mid c_u \in S_u,\ c_w \in S_w\}, \models). \end{aligned}$$

Algorithm 4: `GenerateIP`(D)

Promise: D is satisfiable

Find a satisfying assignment a of D
Let t be the corresponding term: $t = \bigwedge_{a(x)=1} x \wedge \bigwedge_{a(x)=0} \overline{x}$
while there is $\ell \in t$ such that $t - \ell \models D$ **do**
    Remove ℓ from t
end
return t

Algorithm 5: `PropagateIP`$(D, t, P = (v_0, \ldots, v_i))$

Promise: D is reduced, its root is v_0, P is a path in D

if $|P| = 1$ **then** return t
if v_{i-1} is a $\wedge$-node with children u and w **then**
    if $v_i = u$ **then** $t' \leftarrow$ `GenerateIP`(D_w)
    if $v_i = w$ **then** $t' \leftarrow$ `GenerateIP`(D_u)
else if v_{i-1} is a decision node for variable x with 0-child u and 1-child w **then**
    if $v_i = u$ **then**
        if $t \models D_w$ **then** $t' \leftarrow t_\emptyset$ **else** $t' \leftarrow \overline{x}$
    else
        if $t \models D_u$ **then** $t' \leftarrow t_\emptyset$ **else** $t' \leftarrow x$
end
`PropagateIP`$(D, t \wedge t', (v_0, \ldots, v_{i-1}))$

So one can design a polynomial-time algorithm `MissingPI` which, given a circuit D in dec-DNNF and a set $S \subseteq PI(D)$, returns *false* if and only if $S \neq PI(D)$. One just has to replace Propositions 12 and 13 by Propositions 16 and 17 and to replace the contradictions (c1), (c2) and (c3) by "dual versions":

(c4) $S_v = \emptyset$ while D_v is unsatisfiable, or

(c5) v is a decision node, $S_u = PI(D_u)$ and $S_w = PI(D_w)$, but $S' \neq \min(\{c_u \vee c_w \mid c_u \in S_u, c_w \in S_w\}, \models)$.

We remark that there are two kinds of contradictions and not three (like for `MissingIP`). The reason is that when v is a decomposable node, we have that $S_v = PI(D_v) = PI(D_u) \cup PI(D_w)$ if and only if $S_u = PI(D_u)$ and $S_w = PI(D_w)$, and if one the two equality fails then there is a contradiction (c4) of (c5) at a node before u or w. Note that the set S^* computed line 21 is now $\min(\{c_u \vee c_w \mid c_u \in S_u, c_w \in S_w\}, \models)$, which is computable in time $O(|S_u| \times |S_w|)$ given S_u and S_w.

`GenerateIP` is easily modified into an algorithm `GeneratePI` using that circuits in dec-DNNF support linear-time clausal entailment, and `PropagateIP` is turned into the algorithm `PropagatePI` to propagate a prime implicate along a path of the circuit using Propositions 16 and the following, dual proposition to Proposition 11:

Proposition 18. *Let f be a Boolean function, let x be a variable, and let $\ell \in \{x, \overline{x}\}$. Consider $c \in PI(f|\ell)$. If $f|\overline{\ell} \models c$, then $c \in PI(f)$, otherwise $c \vee \ell \in PI(f)$.*

Notice that there is a difference between `PropagatePI` and `PropagateIP` in the way they handle decomposable $\wedge$-node.

Algorithm 6: `AnotherIP`(D, S)

Promise: D is reduced, $S \subseteq IP(D)$

```
r ← MissingIP(D, S, ∅)
if r = false then return false
else if r = (t, P) then return PropagateIP(D, t, P)
```

Algorithm 7: `PropagatePI`$(D, c, P = (v_0, \ldots, v_i))$

Promise: D is reduced, its root is v_0, P is a path in D

```
if |P| = 1 then return c
if v_{i-1} is a ∧-node then PropagateIP(D, c, (v_0, ..., v_{i-1}))
else if v_{i-1} is a decision node for variable x with 0-child u and 1-child w then
    if v_i = u then
        if D_w ⊨ c then c' ← c_∅ else c' ← x̄
    else
        if D_u ⊨ c then c' ← t_∅ else c' ← x
end
PropagatePI(D, c ∧ c', (v_0, ..., v_{i-1}))
```

So we have a polynomial-time algorithm `AnotherPI` which, given a circuit D in dec-DNNF and a set $S \subseteq PI(D)$, returns *false* if $S = PI(D)$ and a clause in $PI(D) \setminus S$ otherwise. Thus:

Lemma 65. *Enum·PI from* dec-DNNF *is in* IncP.

5.4 Enumerating Specific Prime Implicants

For some applications, enumerating all prime implicants of f makes sense, even though there can be exponentially many. We have already mentioned the dualization of monotone CNF formulas as an example. In this section, we turn our attention to the problems Enum·*SR* and Enum·*SE* from dec-DNNF and its sublanguages. Recall that sufficient reasons for an assignment a and subset-minimal explanations are specific prime implicants.

To illustrate the two notions, we use the function f computed by the circuit of Figure 5.1 as a toy example. We introduce some context to make the example more accessible. f encodes a very incomplete characterization of human-like creatures in Tolkien's Middle Earth based on four physical attributes: presence of beard and facial hair (z), small size (s), human-like skin (x), pointy ears (w), plus the indication of whether the creature is enrolled in the armies of evil (y). We imagine that there are only seven possible creatures: hobbits ($x\,\overline{y}\,\overline{z}\,w\,s$), elves ($x\,\overline{y}\,\overline{z}\,w\,\overline{s}$), dwarfs ($x\,\overline{y}\,z\,\overline{w}\,s$), men and women ($x\,{*}\,{*}\overline{w}\,\overline{s}$),[7] ents ($\overline{x}\,\overline{y}\,{*}\,\overline{w}\,\overline{s}$), orcs ($\overline{x}\,y\,\overline{z}\,w{*}$) and trolls ($\overline{x}\,y\,\overline{z}\,\overline{w}\,\overline{s}$). The satisfying assignments of f describe these creatures. Its prime implicants are the smallest combinations of attributes which guarantee the existence of a creature in our Middle Earth.

5.4.1 Subset-Minimal Explanations and Abductive Explanations

Subset-minimal explanations are linked to abductive explanations, see e.g. [SL90, EG95]), that can be defined as follows:

[7] $*$ denotes that both choices are possible for the variable, for example here humans may fight for evil, humans and ents may or may not have beards, and orcs have a wide range of size.

Definition 41 (Abductive explanation). *Given a Boolean function f over variables X, a subset $H \subseteq X$, and a term m on $X \setminus H$, an* abductive explanation *is a term t on H such that $f \wedge t$ is satisfiable and $f \wedge t \models m$.*

The abduction problem asks whether an abductive explanation t exists for the input (f, H, m).

Example 36. Consider our toy example. We look for combinations of physical attributes that guarantee that the creature is evil. This is an abduction problem with $H = \{x, z, w, s\}$ and $m = y$. For instance the term $\overline{x} \wedge w$ is an abductive explanation because there exist creatures with pointy ears and a skin that is not human-like, and all of them are evil (in this case only the orcs fit this description). ◀

An abductive explanation t is in fact an implicant of $\neg f \vee m$ with the conditions that $f \wedge t$ is satisfiable and that t is restricted to variables in H (the abducibles). In other words, the abductive explanation t is an implicant of $\neg f \vee m$ whose variables are restricted H and such that $t \not\vdash f$. Since abduction is not a truth-preserving form of inference, one is often interested in generating abductive explanations that are also subset-minimal explanations over H. In other words, one wants *prime* implicants of $\neg f \vee m$ such that $f \wedge t$ is satisfiable and t is restricted to variables in H. We call *subset-minimal abductive explanations* these prime implicants.

Obviously enough, the abduction problem we focus on (the existence of an abductive explanation) is the same, would we consider subset-minimal abductive explanations or not. Indeed, deciding whether an abductive explanation exists is equivalent to deciding whether a subset-minimal abductive explanation exists. Unfortunately, the condition that only variables in H are allowed in abductive explanations is already too demanding from an enumeration perspective. Indeed, even in the case when an OBDD or a decision tree (DT) (recall that OBDD and DT are subsets of dec-DNNF) computing $\neg f \vee m$ is given, there is no much hope for a polynomial-time procedure to extract from it a single implicant of this circuit, that is built over H.

Proposition 19. *Unless* P $=$ NP*, there is no polynomial-time algorithm which, given an OBDD or a decision tree computing a function f over X and a set $Y \subseteq X$, decides whether f has an implicant t with $var(t) \subseteq Y$.*

Proof. Let ϕ be a CNF formula with m clauses $c_1, \ldots, c_m$. Create m fresh variables $z_1, \ldots, z_m$. Let $B_1, \ldots, B_m$ be OBDDs respecting the same variable ordering and computing $c_1, \ldots, c_m$, respectively. These OBDDs can be computed in polynomial time (and can even be chosen in DT). Define now the OBDDs $B^{(i)} = (\overline{z_i} \wedge B_i) \vee (z_i \wedge B^{(i+1)})$ for $1 \leq i \leq m$, with $B^{(m+1)} = 1$. $B^{(1)}$ is an OBDD on $\{z_1, \ldots, z_m\} \cup var(\phi)$ built in polynomial time from ϕ and whose size is in $O(|\phi|)$.

Claim 16. *An implicant t of $B^{(1)}$ such that $var(t) \subseteq var(\phi)$ exists if and only if ϕ is satisfiable.*

Proof. For the "only if" direction, assume the implicant exists. t is an implicant of $B^{(1)} = (\overline{z_1} \wedge B_1) \vee (z_1 \wedge B^{(2)})$. Since $z_1 \notin var(t)$, we have $t \models B_1 \equiv c_1$ and $t \models B^{(2)}$. Following the same line of reasoning with $B^{(2)}$ instead of $B^{(1)}$ we also have that $t \models B_2 \equiv c_2$ and $t \models B^{(3)}$. And we repeat the argument until reaching, $t \models c_1$, $t \models c_2$, $\ldots$, $t \models c_m$, $t \models B^{(m+1)} = 1$. So indeed $t \models \phi$ and thus ϕ is satisfiable.

For the "if" direction, assume that ϕ is satisfiable. Then there exists an implicant t of ϕ with $var(t) \subseteq var(\phi)$. Let a be a truth assignment to $var(\phi) \cup \{z_1, \ldots, z_m\}$ that satisfies t. If $a(z_i) = 1$ for all $i \in \{1, \ldots, m\}$, then $B^{(1)}|a \equiv B^{(2)}|a \equiv \cdots \equiv B^{(m+1)}|a \equiv 1$. Otherwise let j be the least integer such that $a(z_j) = 0$. Then $B^{(1)}|a \equiv B^{(2)}|a \equiv \cdots \equiv B^{(j)}|a \equiv B_j|a \equiv c_j|a$. Since t is an implicant of ϕ, we have that $t \models c_j$, so $c_j|a \equiv 1$. Thus every assignment a that satisfies t also satisfies $B^{(1)}$, in other words t is an implicant of $B^{(1)}$. □

So if the algorithm from the proposition statement exists, we can run it on inputs $B^{(1)}$ and $Y = var(\phi)$ to decide in polynomial time whether ϕ is satisfiable.

Finally note that if one had chosen to represent $B_1, \ldots, B_m$ as decision trees from DT (which is also feasible in polynomial time), then $B^{(1)}$ would have been a decision tree. So the statement also holds for DT. □

5.4.2 Sufficient Reasons

The notion of sufficient reason [DH20] is sometimes also referred to as "prime implicant explanation" [SCD18b] or as "abductive explanations" [INM19, INAM20]. We stick to "sufficient reason" to avoid any confusion with the distinct concept of abductive explanations as discussed in the previous section.

Example 37. Consider again our toy example. There is no creature which is small, has human-like skin, pointy ears, no facial hair, and is evil. Finding the reasons of why such a creature cannot exist means finding sufficient reasons for the assignment a defined by $a(x) = a(y) = a(w) = a(s) = 1$ and $a(z) = 0$ given $\neg f$. In this case $x\,y\,w \in SR(\neg f, a)$ explains why such a creature cannot exist: there are no creatures that are evil and have both human-like skin and pointy ears. It is a minimal explanation in that there are such creatures that are non-evil (hobbits and elves), and there are evil creatures that have pointy ears (orcs) or human-like skin (men). There are other sufficient reasons for a given $\neg f$, for instance $x\,y\,s \in SR(\neg f, a)$. ◀

Some results about the complexity of computing sufficient reasons have been pointed out for the past few years. Obviously enough, when no assumption is made on the representation of f, computing a single sufficient reason for an assignment a is already NP-hard (for pretty much the same reasons as for the prime implicant case, i.e., f is valid if and only if for any a, the unique sufficient reason for a given f is the empty term). Furthermore, the number of sufficient reasons for an assignment a given f can be exponential in the number of variables even when f is represented in DT [ABB+21a]. Contrary to subset-explanations, it is computationally easy to generate a single sufficient reason from $SR(D, a)$ when D is an OBDD or a decision tree representing f. A greedy algorithm can be used to this end: if a satisfies D (resp. $\neg D$), then start with the canonical term having a as its unique satisfying assignment and remove literals from this term while ensuring that it still is an implicant of D (resp. $\neg D$), until no more literal can be removed. In addition, when D is in DT, we can generate in polynomial time a monotone CNF formula Ψ such that $IP(\Psi) = SR(D, a)$ (see [DM21] for details), and then take advantage of a quasi-polynomial time algorithm for enumerating the elements of $IP(\Psi)$ [GK99]. In contrast, deciding whether a preset number of sufficient reasons for a given a exists is intractable (NP-hard), even when the Boolean function f is monotone (see Theorem 3 in [MGC+21]). We complete those results by providing evidence that Enum·SR from any language among dec-DNNF, OBDD, or DT is a difficult problem, despite the fact that those languages are quite convenient for many reasoning tasks [DM02, KLMT13a].

Let us first give an inductive computation of $SR(D, a)$ similar to that of $IP(D)$.

Proposition 20. *Let f and g be Boolean functions with $var(f) \cap var(g) = \emptyset$ and let a be a truth assignment to a superset of $var(f) \cup var(g)$. Then $SR(f \wedge g, a) = \{t \wedge t' \mid t \in SR(f, a), t' \in SR(g, a)\}$.*

Proposition 21. *Let f be a Boolean function, let a be a truth assignment to a superset of $var(f)$ and let $x \in var(f)$. If a satisfies the literal ℓ on variable x then*

$$SR(f, a) = \{t \wedge \ell \mid t \in SR(f|\ell, a),\ t \not\models f|\overline{\ell}\} \cup SR(f|\overline{x} \wedge f|x, a).$$

Recall that a key property used by the procedure of Lemma 61 is that $|IP(f)| \geq \max(|IP(f|\overline{x})|, |IP(f|x)|)$. Beyond this size relation between $IP(f)$ and $IP(f|\ell_x)$, every implicant of $IP(f|\ell_x)$ is kept in some form through $IP(f)$, thus computing $IP(f|x)$ and $IP(f|\overline{x})$ is not wasteful in the computation of $IP(f)$. This led to the output-polynomial procedure to generate $IP(f)$ for OBDDs and more generally for circuits in dec-DNNF. On the other hand, it is not guaranteed that $SR(f, a)$ is larger than $SR(f|x, a)$ and $SR(f|\overline{x}, a)$ so there is no straightforward adaptation of this procedure from Enum·IP to Enum·SR, as shown by the following example.

Example 38. Let D be the circuit of Figure 5.1 and consider the subcircuit D_{v_1} of D rooted at node v_1 indicated in Figure 5.2a. The assignment a to $\{y, z, w, s\}$ defined by $a(y) = a(z) = 1$ and $a(w) = a(s) = 0$ satisfies D_{v_1}. Recall that the set $IP(D_{v_1})$ has been constructed in Example 33 and observe that $SR(D_{v_1}, a) = \{\overline{w}\,\overline{s}\}$. Now the 0-child of v_1 is v_2 and looking at the set $IP(D_{v_2})$ constructed in Example 33, we see that $SR(D_{v_2}, a) = \{\overline{w}\,\overline{s}, z\,\overline{w}\}$. Since $D_{v_2} = D_{v_1}|\overline{y}$, we have that $|SR(D_{v_1}, a)| < \max(|SR(D_{v_1}|\overline{y}, a)|, |SR(D_{v_1}|y, a)|)$. ◀

We give evidence that enumerating sufficient reasons from dec-DNNF, or even OBDD or DT, is not in OutputP by reducing to it the problem of enumerating the minimal transversals of a hypergraph, a well-known problem whose membership to OutputP is a long-standing question. As a reminder, a hypergraph is composed of a set of vertices and of a set of subsets of vertices called hyperedges. A transversal of a hypergraph is a subset S of its vertices such that every hyperedge contains at least one vertex in S. A minimal transversal is a transversal that ceases to be a transversal when deprived from any of its vertices [Bre13].

Proposition 22. *If Enum·SR from* OBDD *is in* OutputP *or Enum·SR from* DT *is in* OutputP*, then enumerating the minimal transversals of a hypergraph is in* OutputP.

Proof. The proof adapts on the proof of Theorem 2 in [KPS93]. Let $\mathcal{H}$ be a hypergraph. Vertices are integers $1, \ldots, n$ and associated to variables $x_1, \ldots, x_n$, thus the hyperedges $H \in \mathcal{H}$ are sets of integers. Let $tr(\mathcal{H})$ be the set of transversals of $\mathcal{H}$ and let $tr_{\min}(\mathcal{H})$ be the set of minimal transversals of $\mathcal{H}$. For each $S \subseteq \{1, \ldots, n\}$ of vertices let a_S be the assignment such that $a_S(x_i) = 0$ if and only if $i \in S$, and let $\gamma_S = \bigvee_{i \in S} \overline{x_i}$. Observe that a_S satisfies $\gamma_{S'}$ if and only if $S \cap S' \neq \emptyset$. Let f be the function whose satisfying assignments are exactly the a_H for $H \in \mathcal{H}$. Denote by $sat(f)$ the set of satisfying assignments of f.

Now we look at prime implicates of f which are, by de Morgan laws, the negation of the prime implicants of $\neg f$. We have the following:

$$\begin{aligned} f \models \gamma_S &\Leftrightarrow \forall H \in \mathcal{H}, a_H \text{ satisfies } \gamma_S \\ &\Leftrightarrow \forall H \in \mathcal{H}, H \cap S \neq \emptyset \\ &\Leftrightarrow S \text{ is a transversal of } \mathcal{H} \end{aligned}$$

It follows that the set of implicates of f containing only negative literals is $\{\gamma_T \mid T \in tr(\mathcal{H})\}$, and that the set of *prime* implicates of f containing only negative literals is $\{\gamma_T \mid T \in tr_{\min}(\mathcal{H})\}$. Since the prime *implicants* of $\neg f$ are exactly the negations of the prime *implicates* of f, we get that the set of prime implicants of $\neg f$ containing only positive literals is $\{\bigwedge_{i \in T} x_i \mid T \in tr_{\min}(\mathcal{H})\}$. Observe that $a_\emptyset$ is the assignment that sets all x_i to 1 and that

$$SR(\neg f, a_\emptyset) = \left\{ \bigwedge_{i \in T} x_i \mid T \in tr_{\min}(\mathcal{H}) \right\}.$$

From $\mathcal{H}$ we construct $sat(f)$ as a list of assignments in polynomial time. Then from $sat(f)$ we construct in polynomial time an OBDD B equivalent to f. Then we obtain an OBDD B' equivalent to $\neg f$ by

switching the 0-sink and the 1-sink of B. Given the bijection between $SR(B', a_\emptyset)$ and $tr_{\min}(\mathcal{H})$, any algorithm for enumerating sufficient reasons from OBDD can be run with inputs B' and $a_\emptyset$ to enumerate the minimal transversals of $\mathcal{H}$. So if Enum·*SR* from OBDD is in OutputP then enumerating the minimal transversals of a hypergraph is in OutputP.

Finally, note that from $sat(f)$ one can construct a decision tree representing f in polynomial time (instead of an OBDD), and that negating such a decision tree boils down to turning 0-leaves into 1-leaves and vice-versa. So the statement also holds for Enum·*SR* from DT. □

5.5 Missing Proofs

In this section we present the proofs of the lemmas and propositions that we have omitted to avoid breaking the flow of the chapter.

Proposition 8. *Let f and g be Boolean functions. If $var(f) \cap var(g) = \emptyset$ and if f and g are satisfiable then PI$(f \wedge g)$ = PI$(f) \cup$ PI(g) and PI$(f) \cap$ PI$(g) = \emptyset$.*

Proof. Since prime implicates of a function are defined on the variable of that function, it is readily verified that PI$(f) \cap$ PI$(g) = \emptyset$. We now prove that PI$(f \wedge g)$ = PI$(f) \cup$ PI(g).

For the $\subseteq$ direction, let $c \in PI(f \wedge g)$ and assume neither $c_{var(f)}$ nor $c_{var(g)}$ is the empty clause. If $f \models c_{var(f)}$ then since $f \wedge g \models f$, c would not be a prime implicant of $f \wedge g$. So $f \not\models c_{var(f)}$ and similarly $g \not\models c_{var(g)}$. Then let $a_f \in sat(f)$ be a model of f not accepted by $c_{var(f)}$, and let $a_g \in sat(g)$ be a model of g not accepted by $c_{var(g)}$. By decomposability, $a_f \cup a_g$ is a model of $f \wedge g$, yet it is not a model of $c_{var(f)} \vee c_{var(g)} = c$, a contradiction. This proves that c contains only variables in $var(f)$ or only variables in $var(g)$. Assume, without loss of generality, that it is the first case. Then we have that $f \wedge g \models c_{var(f)}$ and after conditioning both sides on any model of g, by decomposability we obtain that $f \models c_{var(f)}$. Moreover $c_{var(f)}$ must be a prime implicate of f for $f \models c' \models c_{var(f)}$ yields that $f \wedge g \models c'$ (because $f \wedge g \models f$).

For the $\supseteq$ direction, let $c_f \in$ PI(f). Since $f \wedge g \models f$, c_f is an implicate of $f \wedge g$. To prove that c_f is a prime implicate assume $f \wedge g \models c' \models c_f$ for some clause c'. Then $var(c') \subseteq var(c_f)$, but then after conditioning both side on any model of g by decomposability we obtain that $f \models c'$ thus $c_f = c'$ for otherwise c_f would not be a prime implicate of f. The case for $c_g \in$ PI(f) is analogous. □

Proposition 12. *Let D be a circuit in dec-DNNF and let $S \subseteq$ IP(D). If the root of D is an $\wedge$-node, let u and w be its children and let $S_u = \{t_{var(D_u)} \mid t \in S\}$ and $S_w = \{t_{var(D_w)} \mid t \in S\}$. Then $S_u \subseteq$ IP(D_u) and $S_w \subseteq$ IP(D_w) hold, and*

$$S = \text{IP}(D) \text{ iff } S_u = \text{IP}(D_u) \text{ and } S_w = \text{IP}(D_w) \text{ and } S = \{t_u \wedge t_w \mid t_u \in S_u, t_w \in S_w\}.$$

Proof. If $S =$ IP(D) then by Proposition 4 $S = \{t_u \wedge t_v \mid t_u \in$ IP$(D_u), t_v \in$ IP$(D_v)\}$ so IP$(D_u) = \{t_{var(D_u)} \mid t \in S\} = S_u$ and IP$(D_u) = \{t_{var(D_v)} \mid t \in S\} = S_v$ and thus $S = \{t_u \wedge t_v \mid t_u \in S_u, t_v \in S_v\}$.

If $S \neq$ IP(D), let $t^* \in$ IP$(D) \setminus S$ and let $t^*_u = t^*_{var(D_u)}$ and $t^*_v = t^*_{var(D_v)}$. By Proposition 4, t^*_u (resp. t^*_v) is in IP(D_u) (resp. IP(D_v)), so either $t^*_u \notin S_u$ or $t^*_v \notin S_v$and we are done, or $t^*_u \in S_u$ and $t^*_v \in S_v$ in which case $\{t_u \wedge t_v \mid t_u \in S_u, t_v \in S_v\} \neq S$ since $t^*_u \wedge t^*_v$ is in $\{t_u \wedge t_v \mid t_u \in S_u, t_v \in S_v\}$ but not in S. □

Proposition 13. *Let D be a circuit in dec-DNNF whose root is a decision node labelled by x. Let u be its 0-child and w be its 1-child. Given $S \subseteq$ IP(D), let $S_u = \{t \mid t \wedge \overline{x} \in S\} \cup (S \cap$ IP$(D_u))$,*

$S_w = \{t \mid t \wedge x \in S\} \cup (S \cap \mathit{IP}(D_w))$ *and* $S' = \{t \mid t \in S, x \notin var(t)\}$. *Then* $S_u \subseteq \mathit{IP}(D_u)$ *and* $S_w \subseteq \mathit{IP}(D_w)$ *hold, and*

$$S = \mathit{IP}(D) \text{ iff } S_u = \mathit{IP}(D_u) \text{ and } S_w = \mathit{IP}(D_w) \\ \text{and } S' = \max(\{t_u \wedge t_w \mid t_u \in S_u,\, t_w \in S_w\}, \models).$$

Proof. For convenience we denote $S^* = \max(\{t_u \wedge t_w \mid t_u \in S_u,\, t_w \in S_w\}, \models)$. First we prove that $S_w \subseteq \mathit{IP}(D_w)$ (the proof that $S_u \subseteq \mathit{IP}(D_u)$ is analogous). Clearly $S \cap \mathit{IP}(D_w) \subseteq \mathit{IP}(D_w)$ so we just need to show that $\{t \mid t \wedge x \in S\} \subseteq \mathit{IP}(D_w)$. Let $t \wedge x$ be in S, then $t \wedge x \models D$. t is an implicant of D_w since $t \equiv (t \wedge x)|x \models D|x \equiv D_w$. Now if there exists $t' \neq t$ such that $t \models t' \models D_w$ then $t \wedge x \models t' \wedge x \models D$ holds, and therefore $t \wedge x$ is not a prime implicant of D, a contradiction. So $\{t \mid t \wedge x \in S\} \subseteq \mathit{IP}(D_w)$.

Second we prove that $S_w \neq \mathit{IP}(D_w)$ implies $S \neq \mathit{IP}(D)$ (the proof is similar for $S_u \neq \mathit{IP}(D_u)$). Assume there exists $t \in \mathit{IP}(D_w) \setminus S_w$. If $t \models D_u$ then t is in $\mathit{IP}(D)$ by Proposition 11. But t cannot be in S for otherwise it would be in $S \cap \mathit{IP}(D_w) \subseteq S_w$. This shows that $S \neq \mathit{IP}(D)$ in this case. If however $t \not\models D_u$ then $t \wedge x$ is in $\mathit{IP}(D)$ by Proposition 11. But $t \wedge x$ cannot be in S for otherwise t would be in $\{\tau \mid \tau \wedge x \in S\} \subseteq S_w$. So again $S \neq \mathit{IP}(D)$.

Now we prove that $(S' \neq S^*) \Rightarrow (S \neq \mathit{IP}(D))$. We may assume that $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$, otherwise $S \neq \mathit{IP}(D)$ holds regardless of $S' = S^*$. Since $D_u \equiv D|\overline{x}$ and $D_w = D|x$ we have that $S^* = \max(\{t_u \wedge t_w \mid t_u \in \mathit{IP}(D|\overline{x}),\, t_w \in \mathit{IP}(D|x)\}, \models) = \mathit{IP}(D|\overline{x} \wedge D|x)$ by Proposition 4. Now $S = S' \cup \{t \mid t \in S, \overline{x} \in t\} \cup \{t \mid t \in S, x \in t\}$ so, by Proposition 5, if $S = \mathit{IP}(D)$ then S' corresponds to the set $\mathit{IP}(D|\overline{x} \wedge D|x)$. So

$$(S' \neq S^*) \Rightarrow (S' \neq \mathit{IP}(D|\overline{x} \wedge D|x)) \Rightarrow (S \neq \mathit{IP}(D))$$

Now for the other direction, assume there exists $t \in \mathit{IP}(D) \setminus S$. First suppose that $t = t' \wedge x$. On the one hand, t' is in $\mathit{IP}(D|x) = \mathit{IP}(D_w)$. On the other hand t' is clearly not in $\{\tau \mid \tau \wedge x \in S\}$, and since it is not an implicant of D, it is not in $S \cap \mathit{IP}(D_w)$ either. This means that $t' \in \mathit{IP}(D_w) \setminus S_w$ and therefore $S_w \neq \mathit{IP}(D_w)$. In the case when $t = t' \wedge \overline{x}$, a similar proof gives that $S_u \neq \mathit{IP}(D_u)$. It remains to consider the situation where neither x nor $\overline{x}$ is in t. By Proposition 5, t is contained in $\mathit{IP}(D|\overline{x} \wedge D|x)$. As before, we can assume that $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$. We have already explained that this assumption yields $S^* = \mathit{IP}(D|\overline{x} \wedge D|x)$. Since t is not in S and $\overline{x} \notin t$ and $x \notin t$, we have that $t \notin S'$. So $t \in S^* \setminus S'$, and therefore $S \neq S^*$. □

Proposition 14. *Given a reduced circuit* D *in dec-DNNF and* $S \subseteq \mathit{IP}(D)$, `MissingIP`$(D, S, \emptyset)$ *runs in time* $O(poly(|S| + |D|))$, *and it returns false if and only if* $S = \mathit{IP}(D)$.

Proof. For the proof we can ignore the value of the input P since `MissingIP`(D, S, P) returns *false* if an only if `MissingIP`(D, S, P') returns *false* for every $P \neq P'$. So we write a $*$ for the third argument of `MissingIP`.

Soundness: We prove soundness by induction on the depth of D. If D has depth 1 then it is a single node v labelled by 0, 1 or a literal ℓ. The promise states that $S \subseteq \mathit{IP}(D)$. If v is labelled by 0, then S must be $\emptyset$ and the algorithm returns *false* at line 2. If v is is labelled by 1 then either $S = \{t_\emptyset\} = \mathit{IP}(1)$ and the algorithm returns *false* at line 24, or $S = \emptyset$ and the algorithm returns $(t_\emptyset, P')$ at line 5. Finally if v is labelled by ℓ, either $S = \{\ell\} = \mathit{IP}(\ell)$ and the algorithm returns *false* at line 24, or $S = \emptyset$ and the algorithm returns (ℓ, P') at line 5. In all cases the algorithm returns *false* if and only if $S = \mathit{IP}(D)$, and it sets $\lambda(v)$ to $|\mathit{IP}(D_v)|$ before returning *false*.

Now if D has depth more than 1, its root node v is either a decomposable $\wedge$-node or a decision node. Since D is reduced, it cannot be unsatisfiable, so if $S = \emptyset$ the algorithm returns something different

from *false* at line 5. From now on, we suppose that $S \neq \emptyset$. If v is a decomposable $\wedge$-node with children u and w. By Proposition 12, since we are promised that $S \subseteq IP(D)$, we have that $IP(D) = S$ if and only if $IP(D_u) = S_u$ and $IP(D_w) = S_w$ and $S = S^*$, with S_u and S_w defined as in Proposition 12 and S^* defined line 13. By induction, $IP(D_u) \neq S_u$ or $IP(D_w) \neq S_w$ if and only if the output of $\texttt{MissingIP}(D_u, S_u, *)$ or $\texttt{MissingIP}(D_w, S_w, *)$ is distinct from *false*. So if $S \neq IP(D)$, then line 10, 12 or 14 returns something different from *false*. And if $S = IP(D)$, then no return call is triggered lines 10, 12 or 14 and the algorithm returns *false* at line 24 after setting $\lambda(v)$ to $|S| = |IP(D)| = |IP(D_v)|$.

If v is a decision node for variable x with 0-child u and 1-child w. By Proposition 13, since we are promised that $S \subseteq IP(D)$, we have that $IP(D) = S$ if and only if $IP(D_u) = S_u$ and $IP(D_w) = S_w$ and $S' = S^*$ with S_u, S_w and S' defined as in Proposition 13 and S^* defined line 21. By induction $IP(D_u) = S_u$ and $IP(D_w) = S_w$ if and only if the output of $\texttt{MissingIP}(D_u, S_u, *)$ or $\texttt{MissingIP}(D_w, S_w, *)$ is not *false*. So if $S \neq IP(D)$, then lines 18, 20 or 22 returns something different from *false*. And if $S = IP(D)$, then no return call is triggered lines 18, 20 or 22 the algorithm returns *false* at line 24 after setting $\lambda(v)$ to $|S| = |IP(D)| = |IP(D_v)|$.

Running time: Consider the time spent in $\texttt{MissingIP}(D, S, *)$ before a return statement or a recursive call is triggered. The procedure may end at line 2, 4 or 5 in $O(1)$ time. Now if the algorithm does not end at the first lines, most of the running time is spent building sets of terms from S lines 8, 13, 16, 21. Constructing S_u and S_w line 8 only requires projecting the terms in S onto $var(D_u)$ and $var(D_w)$, which takes time $O(|S|)$. Constructing S^* line 13 takes $O(|S_u| \times |S_v|) = O(|S|^2)$. At line 8, S' is obtained in time $O(|S|)$ and S_u and S_w are obtained in time $O(poly(|S| + |D|))$ thanks to polynomial-time prime implicant check on circuits in dec-DNNF. Finally building S^* at line 21 and comparing it to S' can be done in time $O(poly(|S|_u| + |S_w|)) = O(poly(|S|)$. So before a return statement or a recursive call is triggered, the algorithm spends $O(poly(|S| + |D|)$ time. One can observe that $|S_u|$, $|S_w|$ are fewer than $|S|$, so for every node v in D, a call $\texttt{MissingIP}(D_v, S', *)$ takes $O(poly(|S| + |D|))$ time before triggering a return or a recursive call. Thanks to memoization – implemented via λ – the $O(poly(|S| + |D|))$ time procedure is done only once per node. So the total running time of the algorithm is also in $O(poly(|S| + |D|))$. □

Proposition 15. *Let D be a reduced circuit in dec-DNNF and let $S \subseteq IP(D)$. $\texttt{AnotherIP}(D, S)$ runs in time $O(poly(|S| + |D|))$. It returns false if $S = IP(D)$, otherwise it returns a prime implicant of D that does not belong to S.*

Proof. **Soundness.** First $\texttt{AnotherIP}(D, S)$ calls $\texttt{MissingIP}(D, S, \emptyset)$. The soundness of algorithm $\texttt{MissingIP}$ has been established in Proposition 14 so if $S = IP(D)$ then $\texttt{MissingIP}(D, S, \emptyset)$ returns *false* and so does $\texttt{AnotherIP}(D, S)$.

Now let us assume that $\texttt{MissingIP}(D, S, \emptyset)$ has not returned *false* but the pair (t, P) with $P = (v_0, \ldots, v_m)$ a path from v_0 (the root of D) to v_m and t a term. Use the notation $P_i = (v_0, \ldots, v_{i-1})$ for all $1 \leq i \leq m$. Then calling $\texttt{MissingIP}(D, S, \emptyset)$ has triggered a sequence of recursive calls $\texttt{MissingIP}(D_{v_1}, S_1, P_1)$, $\texttt{MissingIP}(D_{v_2}, S_2, P_2)$,…,$\texttt{MissingIP}(D_{v_m}, S_m, P_m)$ and a contradiction has been found at the last step. Thus $\texttt{MissingIP}(D_{v_m}, S_m, P_m)$ ends line 5 for a contradiction of type (c1), or line 14for a contradiction of type (c3), or line 22 and returns (t, P) with t some term that we claim is in $IP(D_{v_m}) \setminus S_m$.

Claim 17. $t \in IP(D_{v_m}) \setminus S_m$.

Proof. This is clear if $\texttt{MissingIP}(D_{v_m}, S_m, P_m)$ ends line 5.

If $\texttt{MissingIP}(D_{v_m}, S_m, P_m)$ ends line 14, then v_m is a decomposable $\wedge$-node whose children are u and w. The sets S_u and S_w have been generated and that it has been shown that $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$ (otherwise a return statement line 10 or 12 would have been triggered). So $S^* = \{t_u \wedge t_w \mid t_u \in \mathit{IP}(D_u), t_w \in \mathit{IP}(D_w)\} = \mathit{IP}(D_{v_m})$ by Proposition 4 and we have indeed $t \in S^* \setminus S_m = \mathit{IP}(D_{v_m}) \setminus S_m$.

If $\texttt{MissingIP}(D_{v_m}, S_m, P_m)$ ends line 22, then v_m is a decision node for x with 0-child u and 1-child w. The sets S_u, S_w, S' and S^* have been generated and that it has been shown that $S_u = \mathit{IP}(D_u)$ and $S_w = \mathit{IP}(D_w)$ (otherwise a return statement line 18 or 20 would have been triggered). So $S^* = \mathit{IP}(D_u \wedge D_w) = \mathit{IP}(D_{v_m}|\overline{x} \wedge D_{v_m}|x)$ by Proposition 4. We have $t \in S^* \setminus S'$ so it is clear that $x \notin var(t)$. Furthermore S' contains all terms from S_m in which neither x nor $\overline{x}$ appears, so $t \in S^* \setminus S'$ translates into $t \in S^* \setminus S_m = \mathit{IP}(D_{v_m}|\overline{x} \wedge D_{v_m}|x) \setminus S_m \subseteq \mathit{IP}(D_{v_m}) \setminus S_m$. □

Now $\texttt{AnotherIP}(D, S)$ returns the output of $\texttt{PropagateIP}(D, t, P)$. To prove that the output is a term in $\mathit{IP}(D) \setminus S$, it is sufficient to show that, for every $1 \leq i \leq m$, if $t_i \in \mathit{IP}(D_{v_i}) \setminus S_i$ then $\texttt{PropagateIP}(D, t_i, P_i)$ calls $\texttt{PropagateIP}(D, t_{i-1}, P_{i-1})$ with $t_{i-1} \in \mathit{IP}(D_{v_{i-1}}) \setminus S_{i-1}$. The rest is an easy induction (where $S_0 = S$ and $D_{v_0} = D$).

Claim 18. *Let $t_i \in \mathit{IP}(D_{v_i}) \setminus S_i$ with $i \geq 1$ then $\texttt{PropagateIP}(D, t_i, P_i)$ calls $\texttt{PropagateIP}(D, t_{i-1}, P_{i-1})$ where $t_{i-1} \in \mathit{IP}(D_{v_{i-1}}) \setminus S_{i-1}$.*

Proof. $\texttt{PropagateIP}(D, t_i, P_i)$ calls $\texttt{PropagateIP}(D, t_i \wedge t', P_{i-1})$. Let $t_{i-1} = t_i \wedge t'$. We need to show that it is in $\mathit{IP}(D_{v_{i-1}}) \setminus S_{i-1}$. First assume that v_{i-1} is a decomposable $\wedge$-node with children v_i and w, then t' is obtained line 4 and clearly $t' \in \mathit{IP}(D_w)$. By Proposition 4, $t_i \wedge t' \in \mathit{IP}(D_{v_i} \wedge D_w) = \mathit{IP}(D_{v_{i-1}})$. By construction $S_i = \{t_{var(D_{v_i})} \mid t \in S_{i-1}\}$. If $t_i \wedge t'$ was in S_{i-1} then its restriction t_i to $var(D_{v_i})$ would be S_i, a contradiction. So $t_i \wedge t' \notin S_{i-1}$.

Now suppose v_{i-1} is a decision node for x with 0-child u and 1-child w. Let v_i be u (the case $v_i = w$ is analogous). By construction $S_i = S_u$. t' is obtained line 7 and, by Proposition 11, $t_i \wedge t' \in \mathit{IP}(D_{v_{i-1}})$. To prove that $t_i \wedge t' \notin S_{i-1}$, first assume that $t_i \models D_w$. Then t' is the empty term $t_\emptyset$. So $t_i \wedge t' = t_i$ and $t_i \in \mathit{IP}(D_{v_{i-1}})$. If t_i was in S_{i-1} then we would have $t_i \in S_{i-1} \cap \mathit{IP}(D_{v_i}) \subseteq S_i$, a contradiction. So when $t_i \models D_w$, we have $t_i \wedge t' \in \mathit{IP}(D_{v_{i-1}}) \setminus S_{i-1}$. Now if $t_i \not\models D_w$, then $t_i \wedge t' = t_i \wedge \overline{x}$ and $t_i \wedge \overline{x}$ is not in S_{i-1} for otherwise we would have $t_i \in \{\tau \mid \tau \wedge \overline{x} \in S_{i-1}\} \subseteq S_i$. So again we have $t_i \wedge t' \in \mathit{IP}(D_{v_{i-1}}) \setminus S_{i-1}$. □

Running time. The running time of $\texttt{MissingIP}(D, S, P)$ is $O(poly(|S|+|D|))$. As for $\texttt{PropagateIP}(D, t, P)$, $|P|$ recursive calls are made and the cost between two consecutive recursive calls is either one call to `GenerateIP` line 3 or 4, or one implicant check line 7 or 9. An implicant test on a circuit in dec-DNNF takes linear-time and `GenerateIP` makes at most $|var(D)|$ such tests, so it runs in time $O(poly(|D|))$. Thus $\texttt{PropagateIP}(D, t, P)$ runs in time $O(|P| \times poly(|D|)) = O(poly(|D|))$. □

Proposition 16. *Let D be a satisfiable circuit in dec-DNNF and let $S \subseteq \mathit{PI}(D)$. If the root of D is an $\wedge$-node, let u and w be its children and let $S_u = \{c \mid c \in S, var(c) \cap var(D_u) \neq \emptyset\}$ and $S_w = \{c \mid c \in S, var(c) \cap var(D_w) \neq \emptyset\}$ (note that in this situation, $var(c) \cap var(D_u) \neq \emptyset$ if and only if $var(c) \subseteq var(D_u)$, and similarly for D_w). Then $S_u \subseteq \mathit{PI}(D_u)$ and $S_w \subseteq \mathit{PI}(D_w)$ hold, and*

$$S = \mathit{PI}(D) \text{ iff } S_u = \mathit{PI}(D_u) \text{ and } S_w = \mathit{PI}(D_w)$$

Proof. Immediate from Proposition 8. □

Proposition 17. *Let D be a circuit in dec-DNNF whose root is a decision node labelled by x. Let u be its 0-child and w be its 1-child. Given $S \subseteq \mathit{PI}(D)$, let $S_u = \{c \mid c \vee \overline{x} \in S\} \cup (S \cap \mathit{PI}(D_u))$,*

$S_w = \{c \mid c \vee x \in S\} \cup (S \cap \mathit{PI}(D_w))$ *and* $S' = \{c \mid c \in S, x \notin var(c)\}$. *Then* $S_u \subseteq \mathit{PI}(D_u)$ *and* $S_w \subseteq \mathit{PI}(D_w)$ *hold, and*

$$S = \mathit{PI}(D) \text{ iff } S_u = \mathit{PI}(D_u) \text{ and } S_w = \mathit{PI}(D_w) \text{ and } S' = \min(\{c_u \vee c_w \mid c_u \in S_u,\, c_w \in S_w\}, \models).$$

Proof. This is the dual of Proposition 13 using that $c \in \mathit{PI}(f)$ if and only if $\neg c \in \mathit{IP}(\neg f)$. □

Proposition 18. *Let f be a Boolean function, let x be a variable, and let $\ell \in \{x, \overline{x}\}$. Consider $c \in \mathit{PI}(f|\ell)$. If $f|\overline{\ell} \models c$, then $c \in \mathit{PI}(f)$, otherwise $c \vee \ell \in \mathit{PI}(f)$.*

Proof. This is the dual of Proposition 11 using that $c \in \mathit{PI}(f)$ if and only if $\neg c \in \mathit{IP}(\neg f)$. □

Proposition 20. *Let f and g be Boolean functions with $var(f) \cap var(g) = \emptyset$ and let a be a truth assignment to a superset of $var(f) \cup var(g)$. Then $\mathit{SR}(f \wedge g, a) = \{t \wedge t' \mid t \in \mathit{SR}(f, a),\, t' \in \mathit{SR}(g, a)\}$.*

Proof. Comes from Proposition 4:

$$\begin{aligned}
&\mathit{SR}(f \wedge g, a) = \{\tau \in \mathit{IP}(f \wedge g) \mid a \text{ satisfies } \tau\} \\
&= \{t \wedge t' \mid t \in \mathit{IP}(f), t' \in \mathit{IP}(g), a \text{ satisfies } t \wedge t'\} \\
&= \{t \wedge t' \mid t \in \mathit{IP}(f), t' \in \mathit{IP}(g), a \text{ satisfies both } t \text{ and } t'\} \\
&= \{t \wedge t' \mid t \in \mathit{SR}(f, a),\, t' \in \mathit{SR}(g, a)\}
\end{aligned}$$

□

Proposition 21. *Let f be a Boolean function, let a be a truth assignment to a superset of $var(f)$ and let $x \in var(f)$. If a satisfies the literal ℓ on variable x then*

$$\begin{aligned}
\mathit{SR}(f, a) = &\{t \wedge \ell \mid t \in \mathit{SR}(f|\ell, a),\, t \not\models f|\overline{\ell}\} \\
&\cup \mathit{SR}(f|\overline{x} \wedge f|x, a).
\end{aligned}$$

Proof. Comes from Proposition 5:

$$\begin{aligned}
\mathit{SR}(f, a) =& \{t \in \mathit{IP}(f) \mid a \text{ satisfies } t\} \\
=& \{t \wedge \overline{\ell} \mid t \in \mathit{IP}(f|\overline{\ell}),\, t \not\models f|\ell, a \text{ satisfies } t\} \\
&\cup \{t \wedge \ell \mid t \in \mathit{IP}(f|\ell),\, t \not\models f|\overline{\ell}, a \text{ satisfies } t\} \\
&\cup \{t \in \mathit{IP}(f|\overline{x} \wedge f|x) \mid, a \text{ satisfies } t\} \\
=& \{t \wedge \ell \mid t \in \mathit{SR}(f|\ell, a),\, t \not\models f|\overline{\ell}\} \\
&\cup \mathit{SR}(f|\overline{x} \wedge f|x, a).
\end{aligned}$$

□

5.6 Conclusion and Perspectives

We have introduced new enumeration queries for compilation languages. Enumeration queries cannot be analyzed like decision or function queries due to the format and the size of the output. We have considered them through the lens of enumeration complexity, in other words, for us enumeration queries for compilation languages are enumeration problems where the input is a circuit in a compilation language. We have considered the enumeration of prime implicants (and prime implicates) of circuits in dec-DNNF.

On the positive side, we have described an incremental polynomial-time algorithm for enumerating all prime implicants of a given circuit in dec-DNNF. On the negative side, we have shown that, when restricting the output to implicants over a given subset of the circuit variables, or to sufficient reasons for a given solution of the circuit, it is unlikely that there exists an output-polynomial time algorithm for generating these implicants from OBDDs or DTs. In particular for the problem of enumerating sufficient reasons of OBDDs or DTs, finding one sufficient reason for a given variable assignment is easy, but we have shown a reduction from this problem to the problem of enumerating minimal transversals of a hypergraph. For this latter problem the best known algorithm is output-pseudo-polynomial, that is, the set of solutions is returned in $O((n+m)^{\log(n+m)})$ where n is the size of the input and m is the size of the output [FK96]. We wonder if a similar algorithm exists for enumerating sufficient reasons of OBDDs and DTs.

Open question 5. Is there an algorithm that, given an OBDD (resp. a DT) B, a model a of B, and an integer k, returns k distinct sufficient reasons for a with respect to B in time $O((|B|+k)^{\log(|B|+k))})$ (or returns $\mathit{SR}(B,a)$ if $k > |\mathit{SR}(B,a)|$)?

We think that the answer is positive for DTs, but we have yet to write the details of the proof. The idea is to encode the DT into a CNF formula in polynomial time and then universally forget the literals that are not in a from that formula (in polynomial time as well). The resulting CNF formula is then monotone and its prime implicants are exactly the sufficient reasons for a with respect to the initial DT. Thus one can use the algorithm of Fredman and Kachiyan for the dualization of monotone formulas to enumerate the prime implicants of the monotone CNF formula [FK96].

The enumeration complexity classes that we have studied in this chapter, that is, OutputP and IncP are major classes in enumeration complexity, but are not necessarily relevant for practical applications. It seems that the "really tractable problems" are in IncP_1, that is, that there exists an incremental *linear* time algorithm for the problem. We have shown that enumerating prime implicants from circuits in dec-DNNF is in IncP but we have not given its membership to a class IncP_i for a specific i. We think it is unlikely that our algorithm works in better than increment *quadractic* time.

Open question 6. Find a small i such that Enum·*IP* from dec-DNNF is in IncP_i.

Note that finding the *least* i such that Enum·*IP* from dec-DNNF is in IncP_i would be much more complicated. In particular i could be a non-integer number but must be greater than 1 (the classes IncP_i for $i < 1$ are empty). We are not aware of any problem that are not in IncP_1 and for which the *least* i such that it belongs to IncP_i is known.

Chapter 6

Unconditional Succinctness Maps for Arithmetic Circuits

In this chapter we make an opening to the compilation of non-Boolean functions. We consider classes of arithmetic circuits (AC) that compute pseudo-Boolean functions, that is, functions over a set of Boolean variables and whose output are real numbers (not to confuse with PB-constraints from Chapter 1). Today ACs play an important role in artificial intelligence because they encompass several classes of circuits with practical applications in probabilistic reasoning. Every property defined for circuits in NNF in the preliminaries (decomposability, determinism, etc.) has an equivalent formulation on ACs. Much like circuits in NNF, it has been noticed that several operations intractable on general ACs become tractable for ACs that respect the right combination of properties. This explains the recent surge of interest for AC implementing specific properties. In the next sections, we recall or define several classes of AC and study the succinctness relations between these classes. We first show that there is a tight connection between subclasses NNF and classes of particular AC computing non-negative pseudo-Boolean functions called monotone AC. We use this connection to extend the succinctness map for circuits in NNF to monotone ACs. Then we show how techniques for finding lower bounds on the size of specific circuits in NNF (in particular circuits in str-DNNF) can be adapted to some classes of ACs that are non-monotone but still computes non-negative functions. We stress that all separations between classes and all lower bounds that we prove in this chapter are *unconditional* (so no "unless P = NP" or similar assumptions are needed).

6.1 Arithmetic Circuits for Pseudo-Boolean Functions

An *arithmetic circuit* (short AC) is a computational circuit over $\mathbb{R}$ whose nodes are labelled by sum operators ($+$-nodes) and by product operators ($\times$-nodes). ACs are not only very natural representations for real-valued polynomials, but also give programs for computing them; this can e.g. be traced back to [Val80] who called them $(+,\times)$-programs. We consider ACs that compute pseudo-Boolean functions, that is, functions from $\{0,1\}^X$ to $\mathbb{R}$ where X is a set of Boolean variables. So the leaves of our ACs are labelled either by a Boolean literal or by a constant in $\mathbb{R}$. We assume that every constant labelling a leaf is stored in memory using a fixed number of bits. Thus we assume that the *size* of an AC C, denoted by $|C|$, is only the number of nodes in C.

Several classes of circuits with practical applications in probabilistic reasoning can be seen as specific ACs, for instance probabilistic sentential decision diagrams (PSDD) [KdBCD14] or sum product networks (SPN) with indicator variable [PD11]. ACs are also strongly related to concepts such as AND/OR-circuits [DM07] and Cutset Networks [RKG14]. When used in probabilistic reasoning, ACs always represent non-negative functions and are therefore called (somewhat misleadingly perhaps) *pos-*

itive ACs. Formally, for all assignments a to its variables, a positive AC must return a value greater or equal to 0. We denote the class of all positive ACs by AC_p. Positive ACs constitute a subclass of what in the probabilistic graphical models community is called *probabilistic circuits* [VCL+21]. In the literature, positivity is is often syntactically enforced by assuming that all constants in the computation are non-negative, see e.g. [Dar03, PD11], in which case the ACs are called *monotone*. Formally, the class of *monotone* ACs, denoted AC_m, is the class of ACs whose leaves are labelled by variables or by *non-negative* constants. Essentially, compared to their monotone counterparts, positive ACs encode programs which allow subtraction as an additional operation. This has no impact on the tractability of most operations performed on the ACs [Den16] and it is known already since [Val80] that it can decrease the size of ACs exponentially.

While research on arithmetic circuits in complexity theory focuses almost exclusively on trying to show lower bounds on the size of ACs representing notoriously challenging polynomials, see e.g. [JS82, Raz09, SY10], the goals pursued in artificial intelligence are often different: on the one hand, algorithms for generating ACs from other models like Bayesian networks [CD08, CKD13, KdBCD14], or by learning from data [LD08, RL16], are a major focus. On the other hand, it is studied how imposing properties on the structure of ACs can render tractable operations like computation of marginals or of maximum a posteriori hypotheses (MAP) or more complex queries [HCD06, VCL+21, KCL+19]. The classes of ACs considered in this chapter correspond to circuits whose nodes enforce one or more of the properties defined in the preliminaries: smoothness, determinism, decomposability, and the less well-known notion of weak decomposability. For C an AC, the set of assignments to $var(C)$ for which C computes a non-zero value is called the *support* of C denoted by $supp(C)$.

Example 39. For the arithmetic circuit represented Figure 6.1, the only assignment a to $\{x, y, z\}$ over which the circuit evaluates to 0 is the assignment $a(x) = 0$, $a(y) = 1$ and $a(z) = 1$. So the support of this circuit is every assignment to $\{x, y, z\}$ but a. ◀

Note that when a node v of C is labelled with a literal ℓ_x, then $supp(C_v) = \{\ell_x\}$ and that when v is labelled by a constant α in $\mathbb{R}$ there is $supp(C_v) = \emptyset$ when $\alpha = 0$ and $supp(\alpha) = \{a_\emptyset\}$ when $\alpha \neq 0$, where $a_\emptyset$ is the empty assignment whose representation as a set of literals is the empty set. Smoothness is defined for ACs in the same way as for circuits in NNF but applies to $+$-nodes instead of $\vee$-nodes. Similarly, decomposable $\times$-nodes for ACs are defined like decomposable $\wedge$-nodes for circuits in NNF. As for determinism, an $+$-node v of an AC C is called *deterministic* when, calling its children $v_1, \ldots, v_m$, for every assignment a to $var(v_1) \cup \cdots \cup var(v_m)$ we have $C_{v_i}(a) \times C_{v_j}(a) = 0$ for every $i \neq j$.

Definition 42 (Weak Decomposability)**.** *An internal node v of a circuit C over Boolean variables is called* weakly decomposable *when, denoting the children of v by $v_1, \ldots, v_m$, for every $i \neq j$ and for every $x \in var(C_{v_i}) \cap var(Cv_j)$, there is a literal ℓ_x in x such that no leaves under C_{v_i} or C_{v_j} are labelled by ℓ_x (so $\overline{x}$ appears in C_{v_i} and C_{v_j}, or x appears in C_{v_i} and C_{v_j}, but not both).*

An AC (resp. a circuit in NNF) is weakly decomposable *when all its $\times$-nodes (resp. $\wedge$-nodes) are weakly decomposable.*

Weak decomposability is sometimes referred to as *consistency*, but we avoid using this term here since for Boolean circuits it is often used to mean satisfiability. Note that a decomposable AC or circuit in NNF is by definition weakly decomposable.

Example 40. The circuits represented in Figure 6.1 are weakly decomposable but not decomposable. Indeed the $\wedge$-node (resp. $\times$-nodes) that is not circled is decomposable but the $\wedge$-node (resp. $\times$-nodes) that is circled is not decomposable but is weakly-decomposable. Indeed, the two subcircuits of this circled $\wedge$-node (resp. $\times$-node) share one variable – the variable x – so the node is not decomposable, but $\overline{x}$ does not label any leaf under that node, so it is weakly-decomposable. ◀

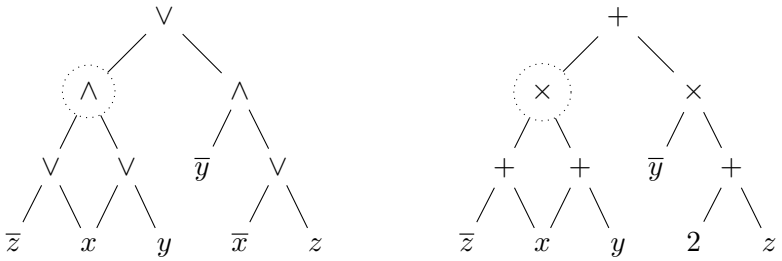


Figure 6.1: A weak decomposable circuit in NNF and a weak decomposable AC.

Generally, more restrictive properties render new operations tractable. For instance *structured decomposability* is a restricted version of decomposability that makes computing the product of two ACs with the same structure feasible in polynomial time [KdBCD14, VCL+21]. But one may also seeks less restrictive properties that are sufficient to ensure tractability of important operations. For instance *weak* decomposability relaxes decomposability but, combined with determinism, it is sufficient to ensure tractable MAP. It turns out that, for all problems studied so far, interesting combinations all include either decomposability or weak decomposability. So the classes studied in this chapter are those whose name is given by the following grammar L:

$$\begin{aligned} L &:= L' \mid \mathsf{s}L' \\ L' &:= L'' \mid \mathsf{d}L'' \\ L'' &:= \mathsf{D}L''' \mid \mathsf{wD}L''' \\ L''' &:= \mathsf{NNF} \mid \text{-}\mathsf{AC}_m \mid \text{-}\mathsf{AC}_p \end{aligned}$$

The grammar L accept the names of all subclasses of positive ACs, monotone ACs and circuits in NNF (L''') that implement decomposability or weak decomposability (L''), and possibly determinism (L'), and possibly smoothness (L). So eight subclasses of NNF and sixteen classes of ACs.

Among subclasses of NNF described by L, the subclasses of DNNF have already been described in the preliminaries but some superclasses based on wDNNF, the class of weakly decomposable circuits in NNF, are new. Since DNNF $\subset$ wDNNF and since DNNF is complete, the eight subclasses of NNF whose name is accepted by L are complete. Similarly, the sixteen classes of ACs whose names are accepted by L are complete for pseudo-Boolean functions. Indeed let X be a finite set of Boolean variables, every function $f : \{0,1\}^X \to \mathbb{R}_+$ has a representation in all these classes of AC_m and AC_p. To see this, one can just write f as $f(X) = \sum_{a \in supp(f)} f(a)\mathbb{1}_a(X)$, where $\mathbb{1}_a : \{0,1\}^X \to \{0,1\}$ is the function that maps a to 1 and every other assignments to 0. The terms $f(a)\mathbb{1}_a(X)$ are easily encoded as positive ACs with only decomposable $\times$-nodes, then constructing a positive AC computing f and implementing smoothness, determinism and decomposability from those "term" AC is straightforward. We make the assumption that the ACs in the rest of this chapter are such that every $+$-node and every $\times$-node has exactly two children. Any AC can be transformed in linear time into an equivalent AC respecting this property by a similar transformation as that described for circuits in NNF in the preliminaries.

The trade-off between usefulness and succinctness has been observed for Boolean circuits in NNF and attracted a lot of attention there [DM02, PD08, BCMS16, ACMS20]. Indeed, all classes of circuits in NNF respecting some combination of properties mentioned above (decomposability, determinism, etc.) have been studied almost exhaustively. In particular, for circuits in NNF, succinctness maps have been drawn that intuitively describe the relative succinctness for the classes of circuits one gets by applying different combinations of properties. When it comes to ACs, research on lower bounds in complexity theory focused on classes with properties such as bounded-depth, tree-like structure, or multilinearity [GK98, Raz09, Raz10, SY10] that have deep implications in theory but are not particularly desirable

in practice – with the exception of *syntactic* multilinearity which is in fact decomposability. In comparison to Boolean circuits, the succinctness analysis for classes of arithmetic circuits of practical interest is fairly young and far from complete.

In this chapter, we initiate a systematic succinctness map for ACs modeled after that proposed in [DM02] for circuits in NNF. Most of our results deal with classes of *monotone* ACs and are obtained by lifting results from the existing succinctness map for circuits in NNF. Indeed we will observe that understanding the succinctness relationships between different classes of monotone ACs reduces to understanding that between classes of circuits in NNF with analogous properties. However, several subclasses of NNF obtained with the reduction, namely those respecting weak decomposability, have only recently been introduced [AAC+19] and thus their position in the maps has not been studied. To analyze monotone ACs, we thus prove the missing succinctness relations for these classes. From the map for circuits in NNF and the lifting technique, we obtain the complete map linking the eight classes of *monotone* ACs one gets combining the different properties. In a modest contribution to the understanding of *positive* ACs, we show that under particular properties, all including determinism, the expressive power of classes of positive ACs coincide with that of their monotone counterparts. Thus some succinctness relations in the monotone map easily extend to the positive map. However, for positive ACs, several relations between classes remain open. Finally, in an effort to motivate further research on the succinctness relations left to prove, we describe a technique to show lower bounds on the size of positive ACs. We apply it to prove lower bounds for positive ACs with *structured* decomposability, which is the case for e.g. PSDD [KdBCD14]. The results of this chapter have been published in the article [dCM21b] co-authored with Stefan Mengel.

6.2 Preliminaries: Smoothing circuits in wDNNF

For a wD-AC (resp. circuit in wDNNF) C on variables X, we introduce *term subcircuits* of C as generalizations of proof trees for circuits in DNNF. A term subcircuit is constructed very much like a proof tree: starting from the source, whenever a $\times$-node (resp. $\wedge$-node) is encountered, its two successors are added to the subcircuit, and whenever a $+$-node (resp. $\vee$-node) is encountered, exactly one arbitrary successor is added to the subcircuit. These objects have already been studied under the name *complete subcircuits* in [CD06]. Each term subcircuit of a wD-AC (resp. a circuit in wDNNF) computes a weighted product of literals (resp. a term).

Lemma 66. *Let C be a wD-AC (resp. a circuit in wDNNF) and let T be term subcircuit of C. Call $lit(T)$ the set of literals labelling leaves of T. Then $T \equiv \alpha \prod_{\ell \in lit(T)} \ell$ (resp. $T \equiv \alpha \wedge \bigwedge_{\ell \in lit(T)} \ell$) for some constant $\alpha \in \mathbb{R}$ (resp. $\alpha \in \{0,1\}$). Furthermore, if C is smooth then $var(T) = var(C)$.*

By distributivity, the sum (resp. the disjunction) of all term subcircuits of C is equivalent to C.

Lemma 67. *Let C be a wD-AC (resp. a circuit in wDNNF) and let $\mathcal{T}$ be the set of all term subcircuits of C. Then $C = \sum_{T \in \mathcal{T}} T$ (resp. $C \equiv \bigvee_{T \in \mathcal{T}} T$).*

For circuit in DNNF, term subcircuits correspond exactly to the proof trees, but term subcircuits of circuits in wDNNF and of wD-ACs are generally not shaped like trees.

It was shown by Peharz et al. [PTPD15] that transforming general wD-AC$_m$ into swD-AC$_m$ leads to an unavoidable exponential blow-up. By Proposition 24, the same is true for wDNNF and s-wDNNF. Here we show that this is not the case when all term subcircuits have the same variables.

Lemma 68. *Let D be a circuit in wDNNF over n variables such that for any two term subcircuits T and T', $var(T) = var(T')$ holds. Then there is a smooth circuit D^* in wDNNF equivalent to D of size $|D^*| = O(n|D|)$. Furthermore, if D is deterministic, then so is D^*.*

Lemma 68 will be used in the next section to prove the succinctness map for NNF. Before we prove it, we give some more definitions.

Definition 43. *Let $\ell_x \in \{x, \overline{x}\}$. An $(\wedge \ell_x)$-link is a $\wedge$-node whose successors include a leaf labelled by ℓ_x. Let g be a node and let p be a predecessor of g, then inserting an $(\wedge \ell_x)$-link between g and p means replacing the connection*  *by*  .

We call the intermediate $\wedge$-node in the construction above the *link node*. A succession of link nodes is a *chain of links*. We remark that links have already been used by Peharz et al. [PTPD15] to analyse the impact of smoothness on weakly decomposable monotone AC, but we use them here in a different way.

For a term subcircuit T containing a node v, recall that T_v denotes the sub-circuit of T under v, and let $T_{\overline{v}}$ be the sub-circuit of T corresponding to all nodes accessible from the source without passing through v. Observe that because of *weak* decomposability, some nodes that are descendants of v may be reached by paths in T not passing through v, so T_v and $T_{\overline{v}}$ are not necessarily disjoint.

Proof (of Lemma 68). Let v be an $\vee$-node whose two children v_l and v_r are such that $var(D_{v_l}) \neq var(D_{v_r})$. Say, without loss of generality, that there is $x \in var(D_{v_l})$ such that $x \notin var(D_{v_r})$. There exists an $\wedge$-node that is an ancestor of v in D, otherwise not all term subcircuits of D would have the same variables. So, for all term subcircuits T of D containing v, $T_{\overline{v}}$ is not empty. Moreover x must be contained in $var(T_{\overline{v}})$ for otherwise we can construct a term subcircuit that does not contain x by extending $T_{\overline{v}}$ to a term subcircuit choosing v_r as the child of v.

We claim that x appears as a unique literal under v_l. To see this, assume first that x appears positively in $T_{\overline{v}}$. Now if $\overline{x}$ labels a node v^* below v_l, then we could extend $T_{\overline{v}}$ to a term subcircuit T^* containing v^* and thus the variable $\overline{x}$. But then T^* would contain both x and $\overline{x}$ which is impossible because C is weakly decomposable. If x appears negatively in $T_{\overline{v}}$, we reason analogously, so that in any case, x appears as a unique literal under v_l. We assume in the remainder that it appears positively, the other case is similar.

Analogously to above, one sees that for all term subcircuits T' containing v, we have that $T'_{\overline{v}}$ contains x. So, for all T' passing through v, we have $T \equiv T \wedge x$. Now insert an $(\wedge x)$-link between v and v_r and let D' be the resulting circuit in wDNNF. We write $vv_r \in T$ when the wire from v to v_r is in the term subcircuit T of D. There is a bijection λ between the term subcircuits of D and those of D': for a term subcircuit T of D, set $\lambda(T) = T$ if $vv_r \notin T$, and let $\lambda(T)$ be the term subcircuit of D' we get from T by inserting the $(\wedge x)$-link between v and v_r otherwise. Clearly, when $vv_r \in T$, then $\lambda(T) \equiv T \wedge x$, and we have already seen that $T \wedge x \equiv T$ in that case. So

$$D' \equiv \bigvee_{T\,:\,vv_r \in T} \lambda(T) \vee \bigvee_{T\,:\,vv_r \notin T} \lambda(T) \equiv \bigvee_{T\,:\,vv_r \in T} T \vee \bigvee_{T\,:\,vv_r \notin T} T \equiv D$$

Observe that $var(D_v) = var(D'_v)$ since x was already in $var(D_{v_l}) \subseteq var(D_v)$. Observe also that the $\wedge$-link node is decomposable. So D' is a circuit in wDNNF and, in D', the variable x appears under both children of v. We repeat that process until the children of v have the same set of variables, so until v is smooth. Doing this for all non-smooth $\vee$-nodes yields a circuit D^* in wDNNF that is smooth. The construction only adds chains of links between nodes that were originally in D, and since there are n variables, at most n links are inserted between any two connected nodes of D, hence $|D^*| = O(n|D|)$.

Finally we argue that if D is deterministic, then so is D^*. We just need to prove this for D', i.e., for a single addition of an $(\wedge x)$-link. Assume that v is deterministic in D. Let v'_r be the $\wedge$-node inserted between v and v_r in D'. The children of v'_r are x and v_r. Assume there is an assignment a' to $var(D'_v)$ whose restrictions a'_l and a'_r to $var(D'_{v_l})$ and $var(D'_{v'_r})$ are in $sat(D'_{v_l})$ and $sat(D'_{v'_r})$ respectively. Then a'_r satisfies D_{v_r} and v is not deterministic in D. This is a contradiction, so v remains deterministic in D'. □

6.3 From Monotone ACs to circuits in NNF

One attractive approach towards understanding the succinctness relations between classes of AC is lifting the corresponding map for classes of circuits in NNF to classes of AC. This is because the map for circuits in NNF is quite substantial and well understood by now, so building the map for ACs upon it would save us the trouble of many proofs. Here we show that we can apply this approach for classes of *monotone* ACs. The idea is that separating the classes of Boolean functions corresponding to the support of monotone ACs is enough to separate these classes of ACs.

Given a monotone AC C, we define a Boolean circuit $\varphi(C)$ that has the same underlying graph as C and is obtained by just modifying the labels on the nodes of C. Nodes labelled by x or $\overline{x}$ or the constant 0 are unchanged, but nodes labelled by constants different from zero are now labelled by the constant 1. For internal nodes, all $\times$-nodes become $\wedge$-nodes and all $+$-nodes become $\vee$-nodes. Clearly $var(C) = var(\varphi(C))$ and, since C and $\varphi(C)$ have the same graph, we have $|C| = |\varphi(C)|$.

We also define the reverse transformation: given a circuit D in NNF we denote by $\psi(D)$ the monotone AC that has the same underlying graph D and is obtained by turning all $\wedge$-nodes into $\times$-nodes and all $\vee$-nodes into $+$-nodes. Note that we have that $\varphi(\psi(D)) = D$ but that, given a monotone AC C we generally have $\psi(\varphi(C)) \neq C$ because φ modifies the labels of the leaves while ψ does not.

Lemma 69. *When C is a monotone AC, $\varphi(C)$ is a circuit in NNF whose models are $supp(C)$. Moreover if C is (weakly) decomposable, deterministic, or smooth, then $\varphi(C)$ is as well.*

Proof. The graph of $\varphi(C)$ is that of C and φ contains only $\wedge$- and $\vee$-nodes, thus $\varphi(C)$ is in NNF. It is easy to see that for each node v in C we have $var(C_v) = var(\varphi(C)_v)$, so smoothness and (weak) decomposability are preserved.

We prove that by induction on the depth of C that (1) $sat(\varphi(C)) = supp(C)$ and (2) if C is deterministic, then so is $\varphi(C)$. If C has depth 1, then it is a single node labelled either by a constant or by a literal. In case the node is labelled by a constant α, if $\alpha > 0$ then $supp(C) = \{a_\emptyset\} = sat(1) = sat(\varphi(C))$. If $\alpha = 0$ then $supp(C) = \emptyset = sat(0) = sat(\varphi(C))$. In case the node is labelled by a literal ℓ_x, then $\varphi(C) = C$ so we are done. Now assume (1) and (2) hold for all ACs of depth at most k and suppose C has depth $k+1$. Let v be its source node and let v_l and v_r be its children.

If v is a $\times$-node, then $C(a) = 0$ if and only if $C_{v_l}(a_l) = 0$ or $C_{v_r}(a_r) = 0$, where a_l and a_r denote the restrictions of a to $var(C_{v_l})$ and $var(C_{v_r})$ respectively. So $a \notin supp(C)$ if and only if $a_l \notin supp(C_{v_l})$ or $a_r \notin supp(C_{v_r})$. By induction $supp(C_{v_l}) = sat(\varphi(C)_{v_l})$ and $supp(C_{v_r}) = sat(\varphi(C)_{v_r})$, so $a \notin supp(C)$ if and only if $a \notin sat(\varphi(C)_{v_l} \wedge \varphi(C)_{v_r}) = sat(\varphi(C))$. So (1) holds.

If v is a $+$-node, then $C(a) = 0$ if and only if $C_{v_l}(a_l) = 0$ and $C_{v_r}(a_r) = 0$. So $a \notin supp(C)$ if and only if $a_l \notin supp(C_{v_l})$ and $a_r \notin supp(C_{v_r})$. Again by induction it follows that $a \notin supp(C)$ if and only if $a \notin sat(\varphi(C)_{v_l} \vee \varphi(C)_{v_r}) = sat(\varphi(C))$. So (1) holds. For (2), if $a_r \in supp(C_{v_r})$ implies $a_l \notin supp(C_{v_l})$ and vice-versa, then $supp(C_{v_l}) = sat(\varphi(C)_{v_l})$ and $supp(C_{v_r}) = sat(\varphi(C)_{v_r})$ yield that the root $\vee$-node of $\varphi(C)$ is deterministic. □

Lemma 70. *For every circuit D in NNF, $\psi(D)$ is an AC of size $|D|$ whose support is the set of models of D. Moreover if D is (weakly) decomposable, deterministic, or smooth, then so is $\psi(D)$.*

Proof. It is readily verified that $|D| = |\psi(D)|$ and that, for each node v in D, $var(D_v) = var(\psi(D)_v)$, so smoothness and (weak) decomposability are preserved in $\psi(D)$. Moreover $\varphi(\psi(D)) = D$, so $sat(D) = supp(\psi(D))$. Determinism is preserved since $sat(D_v) = supp(\psi(D)_v)$ holds for every node v. □

For a class $\mathcal{C}$ of AC, we define the class of circuits in NNF $\varphi(\mathcal{C}) = \{\varphi(C) \mid C \in \mathcal{C}\}$. Lemma 69 and Lemma 70 directly yield the following:

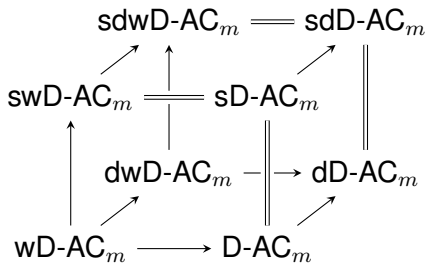


Figure 6.2: Succinctness map for monotone ACs.

Proposition 23. *Let γ be any combination of properties from* {s,d,D,wD}*, then* $\varphi(\gamma\text{-AC}_m) = \gamma\text{-NNF}$.

For instance $\varphi(\text{AC}_m) = \text{NNF}$, $\varphi(\text{D-AC}_m) = \text{DNNF}$, $\varphi(\text{dD-AC}_m) = \text{d-DNNF}$, etc. Moreover, since the circuit size is preserved by φ, the following holds:

Proposition 24. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be classes of monotone AC. Then $\mathcal{C}_1 \leq \mathcal{C}_2$ only if $\varphi(\mathcal{C}_1) \leq \varphi(\mathcal{C}_2)$.*

Proof. Assume $\varphi(\mathcal{C}_1) \not\leq \varphi(\mathcal{C}_2)$. There is a denumerable family $\mathcal{F}$ of Boolean functions for which there is no polynomial p such that ($\varphi(\mathcal{C}_1)$-size of f) $\leq p$($\varphi(\mathcal{C}_2)$-size of f).

Let $D_{2,f}$ be the smallest circuit in $\varphi(\mathcal{C}_2)$ that computes $f \in \mathcal{F}$ and consider $\psi(D_{2,f}) \in \mathcal{C}_2$ for every $f \in \mathcal{F}$. No $C_2 \in \mathcal{C}_2$ computes the same function as $\psi(D_{2,f})$ and is smaller than $\psi(D_{2,f})$, for otherwise we would have that $\varphi(C_2)$ is a circuit in $\varphi(\mathcal{C}_2)$ that computes f and is smaller than $D_{2,f}$.

Now if $\mathcal{C}_1 \leq \mathcal{C}_2$ then there exists a polynomial p such that ($\mathcal{C}_1$-size of f) $\leq p$($\mathcal{C}_2$-size of f) for every $f \in \mathcal{F}$. So we have, for every $f \in \mathcal{F}$, a circuit $C_{1,f}$ in $\mathcal{C}_1$ that computes f and such that $|C_{1,f}| \leq p(|\psi(D_{2,f})|) = p(\varphi(\mathcal{C}_2)\text{-size of } f)$. But then, since $|C_{1,f}| = |\varphi(C_{1,f})|$, and since $\varphi(C_{1,f})$ is a circuit in $\varphi(\mathcal{C}_1)$ computing f, we would have that ($\varphi(\mathcal{C}_1)$-size of f) $\leq p$($\varphi(\mathcal{C}_2)$-size of f), a contradiction. □

We can then lift several succinctness results for circuits in NNF to monotone ACs. For instance we have that d-DNNF $\not\leq$ DNNF [BCMS16] so by Proposition 24 we have $\text{dD-AC}_m \not\leq \text{D-AC}_m$. Moreover since $\text{dD-AC}_m \subseteq \text{D-AC}_m$, it follows that $\text{D-AC}_m < \text{dD-AC}_m$. Weak decomposability has not been studied as widely as decomposability for NNF, so we here draw the map with the additional classes wDNNF, s-wDNNF, d-wDNNF and sd-wDNNF. Then, using Proposition 24, we will obtain the succinctness map for monotone AC shown in Figure 6.2. On that figure, an arrow $C_1 \to C_2$ means that $\mathcal{C}_1 < \mathcal{C}_2$, a double line $C_1 = C_2$ means that $\mathcal{C}_1 \simeq \mathcal{C}_2$, and the absence of connector between two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ means either that the succinctness relation is derived from transitivity or that the two classes are incomparable, i.e., $\mathcal{C}_1 \not\leq \mathcal{C}_2$ and $\mathcal{C}_2 \not\leq \mathcal{C}_1$.

Theorem 23. *The results of Figure 6.2 hold.*

Section 3.3 is dedicated to the proof of Theorem 23.

6.4 Succinctness Maps

6.4.1 Succinctness Map for Circuits in NNF

Using Proposition 24, many succinctness relations between classes of monotone ACs can be inferred from the relations between the corresponding classes of circuits in NNF. So, as a first step towards Theorem 23, we show the correctness of the map for subclasses of NNF showed in Figure 6.3.

Theorem 24. *The results of Figure 6.3 hold.*

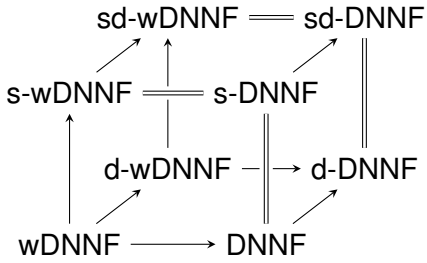


Figure 6.3: Succinctness map for different subclasses of NNF.

It was shown by Darwiche and Marquis [DM02] that s-DNNF and DNNF are equally succinct, and that sd-DNNF and d-DNNF are equally succinct, the paper also contains the statement DNNF $<$ d-DNNF conditioned on standard complexity theoretic assumptions. The result was made unconditional in [BCMS16]. So we already have the right face of the cube-like succinctness map of Figure 6.3.

Lemma 71. wDNNF $<$ DNNF.

Proof. Since DNNF $\subseteq$ wDNNF there only is DNNF $\not\leq$ wDNNF to prove. It is readily verified that monotone circuit in NNF, that is, circuit in NNF with only non-negative literal inputs, are in wDNNF. In [BCMS14] and [Cap16], the separation DNNF $\not\leq$ CNF is shown finding a denumerable class of monotone 2-CNF formulas that have polynomial size but whose equivalent circuits in DNNF all have exponential size. Monotone CNF formulas can be seen as circuits in wDNNF so this proves DNNF $\not\leq$ wDNNF. □

Peharz et al. [PTPD15] give a polynomial-time algorithm to transform any smooth *weakly* decomposable monotone AC into an equivalent smooth decomposable monotone AC[8]. A careful examination of the algorithm of Peharz et al. shows that it can be adapted to turn any s-wDNNF into an equivalent s-DNNF in polynomial time (the existence of the transformation actually derives from Lemmas 69 and 70). Examining the algorithm even further, one sees that it preserves determinism, so the adapted variant for circuits in NNF also gives a polynomial time transformation from sd-wDNNF to sd-DNNF. Thus we have:

Lemma 72. s-wDNNF $\simeq$ s-DNNF *and* sd-wDNNF $\simeq$ sd-DNNF. *But* wDNNF $<$ s-wDNNF.

Proof. As explained above, s-DNNF $\leq$ s-wDNNF and sd-DNNF $\leq$ sd-wDNNF comes from the algorithm in [PTPD15]. Since s-DNNF $\subset$ s-wDNNF and sd-DNNF $\subset$ sd-wDNNF, the first two relations follow. As for the third one, it holds for otherwise wDNNF $<$ DNNF would be violated by transitivity. □

Lemma 73. d-wDNNF $\not\leq$ DNNF.

Proof. In [Sau03], Sauerhoff consider a class $\mathcal{F}$ of functions whose DNNF size is polynomial in the number of variables but whose d-DNNF size is exponential on the number of variables, as shown by Bova et al. in [BCMS16]. More precisely, all $f \in \mathcal{F}$ on n variables have d-DNNF size at least $2^{\Omega(\sqrt{n})}$. For an assignment a, let $w(a)$ (the *weight* of a) be the number of variables set to 1 by a, that is, $w(a) = |a^{-1}(1)|$. For an integer k, let $D_k(f)$ be the smallest circuit in d-DNNF computing the function f_k whose models are exactly the models f of weight k. Since the circuit $\bigvee_{k=0}^{n} D_k(f)$ is in d-DNNF and represents f,

[8] Peharz et al. work on sum product networks (SPNs) with indicator variables inputs. Their SPNs differ from our monotone AC in that the non-negative constants are not inputs of the circuit but weights associated with edges. Such SPNs are converted into our monotone AC in polynomial time by replacing each weighted edge by a $\times$-node whose children include the weight.

there must be a function $\kappa : \mathcal{F} \to \mathbb{N}$ such that $|D_{\kappa(f)}(f)| = 2^{\Omega(\sqrt{|var(f)|})}$ holds for all f. Define the class $\mathcal{F}^* = \{f_{\kappa(f)} \mid f \in \mathcal{F}\}$.

We claim that in any circuit in wDNNF representing a satisfiable function in $\mathcal{F}^*$, all term subcircuits have the same variables. Consider a circuit in wDNNF representing f_k. Let T be one of its term subcircuit and assume $var(f) \setminus var(T) \neq \emptyset$. Let $x \in var(f) \setminus var(T)$, T has a model a that maps x to 0 and another model a' identical to a except that it maps x to 1. But $w(a) \neq w(a')$ so a and a' cannot both satisfy f_k, a contradiction. So all term subcircuits contain all variables.

So by Lemma 68, there exists a polynomial p such that (sd-wDNNF size of f) $\leq p$(d-wDNNF size of f) for every $f \in \mathcal{F}^*$. By Lemma 72, the sd-wDNNF size and the sd-DNNF size of f are polynomially related, and we know that the sd-DNNF size and the d-DNNF size of f are also polynomially related. So all functions of $\mathcal{F}^*$ have exponential d-wDNNF size. Since circuits in DNNF support polynomial-time restriction to models of fixed-weight – see for instance the proof of [ABJM17, Proposition 4.1] which can easily be adapted to circuits in DNNF – the functions in $\mathcal{F}^*$ also have polynomial DNNF size. So the class $\mathcal{F}^*$ gives us d-wDNNF $\not\leq$ DNNF. □

Lemma 74. DNNF $\not\leq$ d-wDNNF.

Proof. We consider the class $\mathcal{F}$ of monotone 2-CNF formulas used in [BCMS14] to prove that DNNF $\not\leq$ CNF. Let F be a monotone 2-CNF from $\mathcal{F}$ on n variables $x_1, \ldots, x_n$, $F = \bigwedge_{k=1}^{m}(x_{k_0} \vee x_{k_1})$. The size of F is polynomial in n but Bova et al. proved that its DNNF size is exponential in n. Now consider m fresh variables $Z = \{z_1, \ldots, z_m\}$ and define $F' = \bigwedge_{k=1}^{m}((\neg z_k \wedge x_{k_0}) \vee (z_k \wedge x_{k_1}))$. F' is a circuit in d-wDNNF, and $\exists Z.F' \equiv F$. Since DNNF circuits support polynomial-time variables forgetting [DM02], the circuits in DNNF computing F' have exponential size. Thus the class of the circuits $\{F' \mid F \in \mathcal{F}\}$ proves the separation DNNF $\not\leq$ d-wDNNF. □

Lemma 75. wDNNF $<$ d-wDNNF *and* d-wDNNF $<$ d-DNNF *and* d-wDNNF $<$ sd-wDNNF.

Proof. For the first relation, d-wDNNF $\subset$ wDNNF implies wDNNF $\leq$ d-wDNNF. We have d-wDNNF $\not\leq$ wDNNF for otherwise we would have d-wDNNF $\simeq$ wDNNF, which would imply d-wDNNF $\leq$ DNNF and thus would contradict Lemma 73.

For the second relation, d-DNNF is a subclass of d-wDNNF so d-wDNNF $\leq$ d-DNNF. And d-DNNF $\not\leq$ d-wDNNF holds for otherwise DNNF $\not\leq$ d-wDNNF would be violated by transitivity (because DNNF $\leq$ d-DNNF).

For the third relation, sd-wDNNF is a subclass of d-wDNNF so d-wDNNF $\leq$ sd-wDNNF. Since sd-wDNNF, sd-DNNF and d-DNNF are equally succinct, there must be sd-wDNNF $\not\leq$ d-wDNNF otherwise d-DNNF $\not\leq$ d-wDNNF would be violated by transitivity (because sd-wDNNF $\simeq$ d-DNNF). □

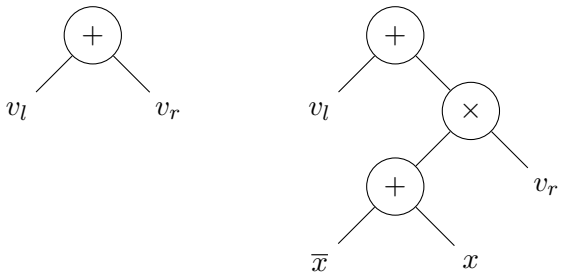
This last lemma finishes the proof of Theorem 24.

6.4.2 Succinctness Map for Monotone ACs

Now to prove Theorem 23 it suffices to show the variant for monotone AC of all lemmas used to prove the correctness of Theorem 24. First it is folklore that the transformation to smooth a circuit DNNF is easily adapted for monotone decomposable ACs:

Lemma 76. sD-AC$_m$ $\simeq$ D-AC$_m$ *and* sdD-AC$_m$ $\simeq$ dD-AC$_m$.

Proof. Let C be a monotone AC. For every $+$-node v of C that is not smooth, denote by v_l and v_r its children, for every $x \in var(v_l) \setminus var(v_r)$, insert a $\times$-node between v and v_r and whose second child is $x + \overline{x}$ as shown in the following figure:



This transformation does not modify the function computed by the circuit since the new $\times$-node is a multiplication by 1. The new $\times$-node is decomposable since $x \notin var(v_r)$ and the new $+$-node is smooth and deterministic. For every other node of the circuit, the transformation modifies neither the number of variables under the node nor the function computed by the circuit at that node. Thus the transformation preserves decomposability and determinism. The transformation is repeated until the circuit is smooth. At most $|var(C)|$ transformations are required for every $+$-node of C so the final circuit has size $O(|var(C)||C|)$. □

Then using Proposition 24 and the fact that $\mathsf{dD\text{-}AC}_m \subset \mathsf{D\text{-}AC}_m$ and $\mathsf{D\text{-}AC}_m \subset \mathsf{wD\text{-}AC}_m$, we derive the following from $\mathsf{DNNF} < \mathsf{d\text{-}DNNF}$ and $\mathsf{wDNNF} < \mathsf{DNNF}$.

Corollary 5. $\mathsf{D\text{-}AC}_m < \mathsf{dD\text{-}AC}_m$.

Corollary 6. $\mathsf{wD\text{-}AC}_m < \mathsf{D\text{-}AC}_m$.

To compare smooth weak decomposable classes of monotone AC to their smooth decomposable counterparts, we are again helped by the results of Peharz et al. [PTPD15].

Lemma 77. $\mathsf{swD\text{-}AC}_m \simeq \mathsf{sD\text{-}AC}_m$ *and* $\mathsf{sdwD\text{-}AC}_m \simeq \mathsf{sdD\text{-}AC}_m$. *But* $\mathsf{wD\text{-}AC}_m < \mathsf{swD\text{-}AC}_m$.

Proof. Peharz et al. show that $\mathsf{sD\text{-}AC}_m \leq \mathsf{swD\text{-}AC}_m$ in [PTPD15]. Since $\mathsf{sD\text{-}AC}_m \subset \mathsf{swD\text{-}AC}_m$, we also have $\mathsf{swD\text{-}AC}_m \leq \mathsf{sD\text{-}AC}_m$ and thus $\mathsf{swD\text{-}AC}_m \simeq \mathsf{sD\text{-}AC}_m$. Moreover the algorithm in [PTPD15] to go from smooth weakly decomposable AC to smooth decomposable AC preserve determinism so $\mathsf{sdD\text{-}AC}_m \leq \mathsf{sdwD\text{-}AC}_m$. Again we have that $\mathsf{sdD\text{-}AC}_m \subset \mathsf{sdwD\text{-}AC}_m$ so $\mathsf{sdwD\text{-}AC}_m \leq \mathsf{sdD\text{-}AC}_m$.

As for $\mathsf{wD\text{-}AC}_m < \mathsf{swD\text{-}AC}_m$. $\mathsf{wD\text{-}AC}_m \leq \mathsf{swD\text{-}AC}_m$ comes from $\mathsf{swD\text{-}AC}_m \subset \mathsf{wD\text{-}AC}_m$, and $\mathsf{swD\text{-}AC}_m \not\leq \mathsf{wD\text{-}AC}_m$ holds for otherwise $\mathsf{D\text{-}AC}_m \not\leq \mathsf{wD\text{-}AC}_m$ would be violated by transitivity (because $\mathsf{swD\text{-}AC}_m \simeq \mathsf{sD\text{-}AC}_m \simeq \mathsf{D\text{-}AC}_m$). □

Finally, using Proposition 24 and the inclusion relation between classes, the remaining results are easy corollaries of Lemmas 73, 74 and 75

Corollary 7. $\mathsf{dwD\text{-}AC}_m \not\leq \mathsf{D\text{-}AC}_m$ *and* $\mathsf{D\text{-}AC}_m \not\leq \mathsf{dwD\text{-}AC}_m$.

Corollary 8. $\mathsf{wD\text{-}AC}_m < \mathsf{dwD\text{-}AC}_m$ *and* $\mathsf{dwD\text{-}AC}_m < \mathsf{dD\text{-}AC}_m$ *and* $\mathsf{dwD\text{-}AC}_m < \mathsf{sdwD\text{-}AC}_m$.

All other relations follow by transitivity, so Theorem 23 is proved.

6.4.3 Starting a Map for Positive ACs

In this section, we will start drawing a succinctness map for positive ACs. Recall that positive ACs compute non-negative functions but allow for negative constants. In a sense, positive ACs have access to a third operation, namely subtraction. It is known that adding subtraction to ACs can decrease their size exponentially [Val80], so $\mathsf{AC}_p < \mathsf{AC}_m$.

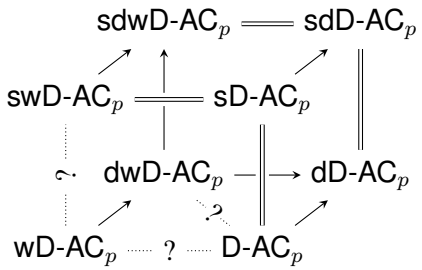


Figure 6.4: Partial succinctness map for classes of positive ACs.

Since there is no apparent mapping between positive ACs and a class of Boolean circuits similar to the mapping ϕ introduced in Section 6.3, we do not obtain a succinctness map for positive ACs in the same way we did for monotone ACs. We here solve some of the relations on the corresponding map, leaving its completion for future work.

Lemma 78. *Let C be a (smooth) (weakly) decomposable deterministic positive AC. Switching the signs of all negative constants in C yields an equivalent (smooth) (weakly) decomposable deterministic monotone AC. Therefore the relation* d-γ-AC$_p \simeq$ d-γ-AC$_m$ *holds for any* $\gamma \in \{\mathsf{D}, \mathsf{wD}, \mathsf{sD}, \mathsf{swD}\}$.

Proof. The transformation does not modify the variables below any node of the AC, so smoothness and (weak) decomposability are preserved by the transformation. By determinism, no two term subcircuits of C can compute a non-zero value on the same assignment, and the sum of the functions computed by term subcircuits is that computed by C. So each term subcircuit T computes a positive function. Since that function can be written $\alpha \prod_{\ell \in lit(T)} \ell$ where α is the product of all constants labelling inputs of T, the negative constants in T must be in even number. But then switching the signs of negative constants in C does not change the function computed by any term subcircuit. Thus the monotone AC we get is equivalent to C and, since its term subcircuits still have pairwise disjoint support, it is deterministic. □

Lemma 79. γ-AC$_p <$ d-γ-AC$_p$ *for any* $\gamma \in \{\mathsf{D}, \mathsf{w}, \mathsf{sD}, \mathsf{swD}\}$.

Proof. Monotone ACs are positive ACs so d-γ-AC$_p \leq$ d-γ-AC$_m$. Using Lemma 78 and Theorem 23, we get γ-AC$_p \leq \gamma$-AC$_m <$ d-γ-AC$_m \simeq$ d-γ-AC$_p$, hence the result. □

Lemma 80. D-AC$_p \simeq$ sD-AC$_p \simeq$ swD-AC$_p$.

Proof. The polynomial-time algorithm of [PTPD15] to transform smooth weakly decomposable ACs into equivalent smooth decomposable ACs remains sound when negative constants are allowed in the circuits. So smooth weakly decomposable positive ACs can be turned into equivalent smooth decomposable positive ACs in polynomial time, hence sD-AC$_p \leq$ swD-AC$_p$. Since sD-AC$_p$ is also a subclass of swD-AC$_p$, it follows that sD-AC$_p \simeq$ swD-AC$_p$.

For D-AC$_p \simeq$ sD-AC$_p$, one just has to observe that the transformation to make smooth a decomposable monotone AC also works for positive AC. □

The above lemmas are summarized in Figure 6.4. Three relations, indicated by question marks in the figure are open.

6.5 Lower Bounds for Positive AC

In this section we adapt the lower bounds techniques of Bova et al. from circuits in NNF to positive ACs. We then show lower bounds on the size of *structured*-decomposable positive ACs.

6.5.1 Sum of Decomposable Products

For circuits in NNF, structured decomposability is defined with help of a v-tree (variable tree) [PD08] but the definition usually assumes that constant inputs have been propagated away in the circuit. Recall that we can propagate (away) the constants in circuits in NNF without modifying the functions they compute. We do not have an equivalent notion of constant propagation for ACs, so we use the vtree-free definition from [VCL+21]. The definition assumes smoothness for simplicity.

Definition 44. *An AC C is called smooth* structured-decomposable *when it is smooth and decomposable and, for all $Y \subseteq var(C)$ there is a partition $Y = Y_0 \cup Y_1$ such that, for every $\times$-nodes v in C with $var(C_v) = Y$, calling v_l and v_r the children of v, we have $var(C_{v_l}) = Y_i$ and $var(C_{v_r}) = Y_{1-i}$ for some $i \in \{0, 1\}$.*

Structured-decomposability is a useful property that renders tractable several operations that are generally intractable, for instance taking the product of two ACs with the same vtree. In particular, PSDD circuits, for probabilistic sentential decision diagrams, form a class of circuit that uses structured-decomposability and that has applications in practice.

Definition 45. *Let Z be a set of variables. A* decomposable product *over Z is a function from Z to $\mathbb{R}$ that can be written as a product $f(X) \times h(Y)$ where (X, Y) is a partition of Z and f and h are functions to $\mathbb{R}$.*

A common approach to proving lower bounds for decomposable AC analyses representations of the function it computes in terms of sums of balanced decomposable products. Roughly put, the idea is that the more summands are needed in such a representation, the larger the ACs that compute it. This technique has been used in recent and not so recent articles, see e.g. [Val80, RY11, MM14]. Translated to Boolean circuits, decomposable products correspond to *combinatorial rectangles*, a tool that we have already used several times in previous chapters.

Variations of the next theorem have been shown several times independently in the literature, see for instance [MM14, Theorem 38]. The structured case follows from a small refinement of the proof, the rough idea is that each decomposable product is built from a different node of the circuit and, thanks to structured decomposability, all these nodes have the same set of variables, which eventually yields the same partition for the decomposable products.

Theorem 25. *Let F be a non-negative Pseudo-Boolean function over at least three variables computed by a structured decomposable smooth AC C. Then F can be written as a sum of N decomposable products over $var(F)$ with respect to the same balanced partition (X, Y), with $N \leq |C|$.*

Proof. We construct the sum of decomposable products from C such that each $f_i \times h_i$ corresponds to a different node of C. Let $\mathcal{T}$ be the set of all term subcircuits of C. First we need the following claim:

Claim 19. *There is a set S of $\times$-nodes in C such that all $var(C_v)$ for $v \in S$ are identical and such that every term subcircuit of C contains at least one node in S.*

Proof. Consider the set $\mathcal{T} = \{T_1, \ldots, T_K\}$ of term subcircuits of C. Let v_i be the root of the term subcircuit T_i. Let S be the sequence $(v_1, \ldots, v_K)$. Initially all the elements of the sequence are identical so we call $X = var(v_1) = \cdots = var(v_K)$. Clearly at this stage $|X| \geq |var(C)|/3$. Using the definition of smooth structured decomposable ACs, for every $Y \subseteq var(C)$ with (Y_0, Y_1) the *unique* partition respected by all $\times$-nodes of C whose set of variables is Y, we define $\lambda(Y) = Y_0$ if $|Y_0| \geq |Y_1|$ and $\lambda(Y) = Y_1$ otherwise. Thus $\lambda(Y) \geq \lceil |Y|/2 \rceil$. Now do the following steps in order:

1. for every i, while v_i is a $+$-node, replace v_i by its unique child in T_i – observe that by smoothness we still have $X = var(C_{v_i})$ and $|X| \geq |var(C)|/3$.

2. for every i, if v_i is a $\times$-node then, if $|X| \leq 2|var(C)|/3$ then return S after removing identical elements and stop there. If however $|X| > 2|var(C)|/3$ then replace every v_i in S by its child whose set of variables is $\lambda(X)$. At the end of this step, if we have not returned S then we have $var(v_1) = \cdots = var(v_K) = \lambda(X)$ and $|\lambda(X)| \geq |var(C)|/3$. Replace X by $\lambda(X)$ and repeat steps 1. and 2.

At the end of both 1. and 2., all nodes in S have the same set of variables X. The procedure can never skip 2. Indeed $|X|$ is always greater than 2, so at the end of 1. all nodes in S are $\times$-nodes (none of them are $+$-nodes by definition of 1., and none of them is labelled by a literal because their set of variables is X). Since the procedure never skips 2. and since the size of X decreases strictly at the end of 2., at some point $|X| \leq 2|var(C)|/3$ occurs and the procedure ends. By construction the set S returned contains a node of each term subcircuits and all nodes in S have the same set of variables. □

Let S be the set obtained by the claim above. Let X be such that $var(C_v) = X$ for every $v \in S$. Note that each term subcircuit T of C contains a single node in S. Indeed if $v, v' \in S$ were both in T then the highest common ancestor of v and v' in T would be a non-decomposable $\times$-node since $var(v) \neq var(v')$.

Let $v \in S$ and let T be a subcircuit of C containing v. Recall that T is shaped like a tree since C^i is decomposable and that T_v denotes the subcircuit of T under v. Let $T_{\overline{v}}$ be the circuit obtained replacing T_v by a node labelled by 1 in T. Replacing that fresh node in $T_{\overline{v}}$ by T_v gives T back, hence the notation $T = (T_v, T_{\overline{v}})$. Both T_v and $T_{\overline{v}}$ compute functions whose support is a single assignment on X and Y, respectively (due to C being smooth) and T computes $T_v \times T_{\overline{v}}$. If T and T' are two term subcircuits of C containing v, then so are $(T_v, T'_{\overline{v}})$ and $(T'_v, T_{\overline{v}})$, thus the disjunction of all term subcircuits of C containing v may be written

$$\sum_{T\,:\,v\in T} T = \Big(\sum_{T\,:\,v\in T} T_v\Big) \times \Big(\sum_{T\,:\,v\in T} T_{\overline{v}}\Big).$$

On the right-hand side, in the first sum, every summand is a function over X while in the second one, every summand is a function over Y, thus the product can be written $f_v(X) \times h_v(Y)$. Since every term subcircuit of C contains exactly one node in S, it follows that

$$\sum_{T\in\mathcal{T}} T = \sum_{v\in S} \sum_{T\,:\,v\in T} T = \sum_{v\in S} f_v(X) \times h_v(Y)$$

And the result follows from $F = \sum_{T\in\mathcal{T}} T$ and $|S| \leq |C|$. □

6.5.2 Lower Bounds for Structured Decomposable Positive AC

In this section we prove the following lower bound.

Theorem 26. *There is a class of positive Pseudo-Boolean functions $\mathcal{F}$ such that, for all $F \in \mathcal{F}$, the smallest AC computing F has size polynomial in $|var(F)|$ but the smallest smooth structured decomposable AC computing F has size $2^{\Omega(|var(F)|)}$.*

By Theorem 25, the smallest N for which one can write F as $F = \sum_{i=1}^{N} f_i(X) \times h_i(Y)$ where $f_i(X) \times h_i(Y)$ are decomposable products with respect to the balanced partition (X, Y) of $var(F)$, is a

lower bound on the size of all smooth structured decomposable ACs computing F. Thus, proving Theorem 26 boils down to finding non-negative functions where the smallest such N depends exponentially on the number of variables.

Let us fix a function F and a partition (X, Y). The *value matrix* of F with respect to (X, Y) is a $2^{|X|} \times 2^{|Y|}$ matrix M_F whose rows (resp. columns) are uniquely indexed by assignments to X (resp. Y) and such that, for each pair of indices (a_X, a_Y), the entry of M_F at the a_X row and a_Y column is $F(a_X \cup a_Y)$.

Lemma 81. *Let $F = \sum_{k=1}^{N} f_k(X) \times h_k(Y)$ where for all k we have $f_k \times h_k \neq 0$. Let M_F be the value matrix for F and let M_i denote the the value matrix for $f_i \times h_i$ with respect to partition (X, Y). Then*

$$\text{rk}(M_F) \leq \sum_{k=1}^{N} \text{rk}(M_k) = N.$$

Proof. By construction, $M_F = \sum_{k=1}^{N} M_k$, so $\text{rk}(M_F) \leq \sum_{k=1}^{N} \text{rk}(M_k)$ holds by sub-additivity of the rank. We now show that $\text{rk}(M_k) = 1$ holds for each k. Since $f_k \times h_k \neq 0$, there is a row in M_k which is not a 0-row. Say it is the row indexed by a_X. Then the entries in that row are $f_k(a_X) \times h_k(a_Y)$ for varying a_Y. In any other row indexed by a'_X, the entries are $f_k(a'_X) \times h_k(a_Y) = (f_k(a'_X)/f_k(a_X)) \times f_k(a_X) \times h_k(a_Y)$ for varying a_Y. Consequently, all rows are multiples of the a_X-row, in other words, all rows of M_k are linearly dependent, hence $\text{rk}(M_k) = 1$. □

Using Lemma 81, one sees that proving Theorem 26 boils down to finding functions whose value matrices with respect to *any* balanced partition (X, Y) have rank exponential in the number of variables.

The functions we construct are based on graphs. Let $G = (V, E)$ be a graph, let $n = |V|$ and, for each vertex v_i in V, create a Boolean variable x_i. We consider the function:

$$F_G(x_1, \ldots, x_n) = \prod_{(v_i, v_j) \in E} (1 + \max(x_i, x_j)). \tag{6.1}$$

Essentially, for each edge of G, if at least one of its endpoints is mapped to 1 via an assignment, then the edge contributes a factor 2 to the product, otherwise it contributes a factor 1. Regardless of the choice of G, the function F_G has a small positive AC: one just has to write $\max(x_i, x_j) = x_i + x_j - x_i x_j$ and see that the number of $\times$ and $+$ operations needed to compute F_G is polynomial in n.

An *induced matching* of G is a set $E' \subseteq E$ of edges with pairwise disjoint endpoints, whose set we denote V', such that all edges of G connecting vertices in V' are in E'.

Lemma 82. *Let F_G be as described by (6.1), let (X, Y) be a partition of $var(F_G)$ and (V_X, V_Y) be the corresponding partition of V. If there is an induced matching m in G between vertices V_l and V_r such that $V_l \subseteq V_X$ and $V_r \subseteq V_Y$, then*

$$\text{rk}(M_{F_G}) \geq 2^{|m|}$$

where M_{F_G} is the value matrix of F_G for the partition (X, Y) and $|m|$ is the number of edges in m.

Proof. Rename $M = M_{F_G}$. Identify each vertex with its variable in $var(F_G)$ and let $(x_i, y_i)_{i \in [|m|]}$ be the edges of m, with $x_i \in X$ and $y_i \in Y$. Order the variables in X as $X = (x_1, \ldots, x_{|X|})$ and the variables in Y as $Y = (y_1, \ldots, y_{|Y|})$, so that the $|m|$ first variables in each set correspond to the nodes in the matching. Permutations of rows or columns do not change the rank of a matrix so we assume that the assignments indexing the rows and the columns are ordered so that, when seeing the assignments as tuples of 0 and 1, the integers encoded in binary by the tuples are ordered. More formally a_X is before a'_X if and only if $\sum_k a(x_k) 2^{k-1} < \sum_k a'(x_k) 2^{k-1}$. Now consider all $2^{2|m|}$ truth assignments to

$var(F_G)$ where variables corresponding to vertices not in $V_l \cup V_r$ are set to 0. Let M^* be the $2^{|m|} \times 2^{|m|}$ sub-matrix of M obtained by keeping only rows and columns indexed by these assignments. The rank of a sub-matrix is always at most that of the matrix, so $\text{rk}(M^*) \leq \text{rk}(M)$. To prove the lemma, it is enough to show that $\text{rk}(M^*) = 2^{|m|}$, which holds if and only if $\det(M^*) \neq 0$. For $0 \leq i \leq |m|$, let M_i^* be the matrix containing the first 2^i rows and first 2^i columns of M^*. We prove by induction on i that all M_i^* have non-zero determinant, which will prove that M^* (which is $M_{|m|}^*$) has non-zero determinant, and therefore full rank. For the base case, $M_0^* = (1)$ has determinant 1. For the general case, assume that $\det(M_i^*) \neq 0$ and observe that $M_{i+1}^* = \left(\begin{array}{c|c} M_i^* & 2M_i^* \\ \hline 2M_i^* & 2M_i^* \end{array}\right)$. The determinant of M_{i+1}^* is

$$\det\left(\begin{array}{c|c} M_i^* & 2M_i^* \\ \hline 2M_i^* & 2M_i^* \end{array}\right) = \det\left(\begin{array}{c|c} -M_i^* & 2M_i^* \\ \hline 0 & 2M_i^* \end{array}\right) = \det(-M_i^*)\det(2M_i^*) = (-2)^{2^i}\det(M_i^*)^2 \neq 0.$$

□

So if, for *every* balanced partition of V, we have a large enough induced matching M between the two sides, then the rank of the value matrix for F_G for any balanced partition is large, thus many balanced decomposable products are needed in a sum representing F_G. The only thing left is to find graphs G with this "large enough matching" property, which turn out to be *expander graphs*, which we have already met in previous chapters. Recall that a d-regular graph is a graph whose vertices all have degree d. A (c, d)-expander graph on vertices V is a d-regular graph such that for any $S \subseteq V$ of size $|S| \leq |V|/2$, it holds that $|N(S)| \geq c|S|$, where $N(S) = \{v \in V \setminus S \mid (u, v) \in E, u \in S\}$.

Theorem 27. *[AS00, Section 9.2] There is, for some $c > 0$, an denumerable sequence of $(c, 3)$-expander graphs $(G_i)_{i\in\mathbb{N}}$.*

We use these expander graphs for our lower bound.

Lemma 83. *Let $G = (V, E)$ be a $(c, 3)$-expander graph with $n = |V|$, and let $V = V_1 \uplus V_2$ be a balanced partition of V. Then there exists an induced matching m of size $\Omega(n)$ between V_1 and V_2.*

Proof. V_1 or V_2 has size at most $n/2$, say $|V_1| \leq n/2$. Then $N(V_1) \subseteq V_2$ and $|N(V_1)| \geq c|V_1| \geq cn/3$ where the last inequality comes from the partition being balanced. So at least $cn/3$ edges connect V_1 to V_2. Since G is 3-regular, at least a third of these edges form an matching in G, and a third of these matching edge share no endpoint in V_1, and finally a third of these edges share no endpoint in V_2 either. So we obtain a induced matching between V_1 and V_2 of size at least $cn/81$. □

Combining Theorems 25 and 27 with Lemmas 81, 82 and 83 yields Theorem 26.

6.6 Conclusion and Perspectives

We have started drawing succinctness maps for arithmetic circuits modeled after proposed for circuits in NNF in [DM02]. Due to the great amount of recent work on practical applications of ACs with specific structural restrictions, we have studied classes of ACs for combinations of four key restrictions: decomposability, weak decomposability, determinism and smoothness. Using a mapping between monotone ACs and circuits in NNF, we have drawn the full succinctness map for monotone ACs by lifting the existing map for circuits in NNF and extending it to incorporate new classes defined with weak decomposability. In certain cases we could show that positive and monotone ACs have the same expressive power, which gave us some succinctness results between classes of positive ACs for free. We leave the challenging task of determining the remaining relationships between classes of positive AC as an open question.

Open question 7. Determine the succinctness relationships labelled by a question mark in Figure 6.4.

Several succinctness relationships between classes of positive ACs and their monotone counterparts are also missing. In particular it is a long standing open question whether $\mathsf{D\text{-}AC}_p < \mathsf{D\text{-}AC}_m$ holds. Perhaps a separation can be found in the case when the circuits are structured-decomposable. In the last section of this chapter, we have introduced techniques to prove lower bounds on positive ACs and applied them to the case of smooth structured-decomposable ACs. This could be the first step towards answering the following:

Open question 8. Prove an exponential separation between structured-decomposable positive ACs and structured-decomposable monotone ACs or, alternatively, prove that the two classes of circuits are equally succinct.

Conclusion

To conclude this thesis, let us give a final overview of our contributions. Our first contributions belong to the long tradition of studying the L size of selected functions for L a class of representations, here a compilation language. We have shown new lower bounds on the DNNF size of particular pseudo-Boolean constraints and on the DNNF size of satisfiable Tseitin formulas. To obtain these new lower bounds, we have used the well-established methods of cover by balanced rectangles and we have improved upon it to create a refined method that takes the form of a two-player adversarial game. In a less standard direction, we have also taken into account the compilation of approximations when the DNNF size of the function to be represented is too important. We have started from an existing notion of approximation that comes with guarantees on the approximation error, that we have called weak approximation. We have generalized existing results on functions whose weak approximations all have exponential OBDD sizes to show that they also have exponential d-DNNF sizes. Then, we have shown limitations of weak approximation that disqualify it as a good approximation notion in many settings, in particular if it is to be used with approximate model counting in mind. To circumvent these limitations, we have introduced the new notion of strong approximation and we have shown that, for some functions for which weak approximations can be of poor quality (typically approximations by the function that is uniformly 0), compiling strong approximations is not an option as their d-DNNF sizes are exponential in the number of variables. Notably, using our improved method for finding lower bounds on the DNNF size, we have shown that particular satisfiable Tseitin formulas are examples of such functions.

In the second part of the thesis, we have exploited further the lower bounds on the DNNF size of satisfiable Tseitin formulas in two applications. The first application deals with the space efficiency of bottom-up knowledge compilation. Indeed, we can use our lower bound to show that str-DNNF $(\wedge,r)$-compilations of unsatisfiable Tseitin formulas whose graphs have bounded degree all generate intermediate circuits of size exponential in the treewidths of the graphs. A simple trick allows us to prove that there exists a class of satisfiable CNF formulas that have constant str-DNNF sizes but whose str-DNNF $(\wedge,r)$-compilations all generate intermediate circuits of size exponential in the primal treewidth. The second application of our lower bound makes a connection to the origin of Tseitin formulas. Grigori Tseitin introduced unsatisfiable Tseitin formulas with the idea that they would be hard formulas to refute in the resolution proof system. This was later confirm by subsequent research not only for the resolution proof system but for other proof systems as well. We have contributed to this research strand by showing that the number of clauses in every regular resolution refutation of unsatisfiable Tseitin formulas whose graphs have bounded degrees is at least exponential in the treewidths of the graphs. The exponential dependence in the treewidth of this lower bound matches that of known upper bounds, so the unsatisfiable Tseitin formulas whose graphs have bounded degree have regular resolution refutations of length polynomial in the number of variables n if and only if the treewidth of the graph is at most logarithmic in n.

After this small detour in proof complexity, we went back to knowledge compilation in the last part of the thesis. There we have explored new horizons for knowledge compilation. The first direction has been to study new enumeration queries for existing compilation languages. On the one hand, we

have shown that enumerating the prime implicants and the prime implicates of circuits in dec-DNNF is feasible in incremental polynomial-time. On the other hand, we have also shown that enumerating specific prime implicants, namely sufficient reasons for a given variable assignment and subset-minimal abductive explanations, is unlikely to be feasible in output-polynomial time even for DTs and OBDDs, which are among the most constrained function representations studied in knowledge compilation. In a second direction, we have explored knowledge compilation for pseudo-Boolean functions. In the spirit of the knowledge compilation map, we have defined several classes of arithmetic circuits by forcing on them combinations of properties (decomposability, monotonicity, etc.), and we have studied the succinctness relationships between these classes. Using a tight connection between classes of monotone ACs and the corresponding classes of circuits in NNF, we have drawn the complete succinctness map for the eight classes of monotone ACs considered. We have started extending the map to non-monotone positive ACs, but we could not fully complete it.

Finally, on the one hand, this thesis pursues the study of knowledge compilation in the traditional way by studying the hardness of representing selected functions in several existing compilation languages or by looking at new queries for these languages. On the other hand, it also explores some areas of the domain that are not as well studied, like approximate knowledge compilation, and even shows that knowledge compilation can be connected in surprising way to other areas of computer science, especially proof complexity theory.

Bibliography

[AAC+19] S. AKSHAY, Jatin ARORA, Supratik CHAKRABORTY, Shankara Narayanan KRISHNA, Divya RAGHUNATHAN, and Shetal SHAH. « Knowledge Compilation for Boolean Functional Synthesis ». In Clark W. BARRETT and Jin YANG, editors, *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, pages 161–169. IEEE, 2019. 139

[ABB+21a] G. AUDEMARD, S. BELLART, Louenas BOUNIA, F. KORICHE, J.-M. LAGNIEZ, and P. MARQUIS. « On the Explanatory Power of Decision Trees ». *CoRR*, abs/2108.05266, 2021. 128

[ABB+21b] Gilles AUDEMARD, Steve BELLART, Louenas BOUNIA, Frédéric KORICHE, Jean-Marie LAGNIEZ, and Pierre MARQUIS. « On the Explanatory Power of Decision Trees ». *CoRR*, abs/2108.05266, 2021.

[ABdR+18] Albert ATSERIAS, Ilario BONACINA, Susanna F. de REZENDE, Massimo LAURIA, Jakob NORDSTRÖM, and Alexander A. RAZBOROV. « Clique is hard on average for regular resolution ». In Ilias DIAKONIKOLAS, David KEMPE, and Monika HENZINGER, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 866–877. ACM, 2018. 102

[ABJM17] Antoine AMARILLI, Pierre BOURHIS, Louis JACHIET, and Stefan MENGEL. « A Circuit-Based Approach to Efficient Enumeration ». In Ioannis CHATZIGIANNAKIS, Piotr INDYK, Fabian KUHN, and Anca MUSCHOLL, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 111:1–111:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. 1, 8, 11, 16, 112, 144

[ACH+21] Dimitris ACHLIOPTAS, Amin COJA-OGHLAN, Max HAHN-KLIMROTH, Joon LEE, Noëla MÜLLER, Manuel PENSCHUCK, and Guangyan ZHOU. « The number of satisfying assignments of random 2-SAT formulas ». *Random Struct. Algorithms*, 58(4):609–647, 2021. 63

[ACMS20] A. AMARILLI, F. CAPELLI, M. MONET, and P. SENELLART. « Connecting Knowledge Compilation Classes and Width Parameters ». *Theory Comput. Syst.*, 64(5):861–914, 2020. 4, 13, 40, 46, 47, 138

[AGMS16] Ignasi ABÍO, Graeme GANGE, Valentin MAYER-EICHBERGER, and Peter J. STUCKEY. « On CNF Encodings of Decision Diagrams ». In Claude-Guy QUIMPER, editor, *Integration of AI and OR Techniques in Constraint Programming - 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29 - June 1, 2016, Proceedings*, volume 9676 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2016. 4, 13, 41

[AHU74] Alfred V. AHO, John E. HOPCROFT, and Jeffrey D. ULLMAN. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. 113

[AJPU07] Michael ALEKHNOVICH, Jan JOHANNSEN, Toniann PITASSI, and Alasdair URQUHART. « An Exponential Separation between Regular and General Resolution ». *Theory Comput.*, 3(1):81–102, 2007. 102

[Ajt05] Miklós AJTAI. « A Non-linear Time Lower Bound for Boolean Branching Programs ». *Theory of Computing*, 1(1):149–176, 2005. 61, 62

[AKM20] G. AUDEMARD, F. KORICHE, and P. MARQUIS. « On Tractable XAI Queries based on Compiled Representations ». In *Proc. of KR'20*, pages 838–849, 2020. 112

[AKV04] Albert ATSERIAS, Phokion G. KOLAITIS, and Moshe Y. VARDI. « Constraint Propagation as a Proof System ». In Mark WALLACE, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004.

[ANO+12] Ignasi ABÍO, Robert NIEUWENHUIS, Albert OLIVERAS, Enric RODRÍGUEZ-CARBONELL, and Valentin MAYER-EICHBERGER. « A New Look at BDDs for Pseudo-Boolean Constraints ». *J. Artif. Intell. Res.*, 45:443–480, 2012. 42

[AR11] Michael ALEKHNOVICH and Alexander A. RAZBOROV. « Satisfiability, Branch-Width and Tseitin tautologies ». *Comput. Complex.*, 20(4):649–678, 2011. 8, 15, 103

[ARMS02] Fadi A. ALOUL, Arathi RAMANI, Igor L. MARKOV, and Karem A. SAKALLAH. « Generic ILP versus specialized 0-1 ILP: an update ». In *IEEE/ACM International Conference on Computer-aided Design, ICCAD*, pages 450–457, 2002. 41

[AS00] Noga ALON and Joel H. SPENCER. *The Probabilistic Method, Second Edition*. John Wiley, 2000. 150

[Bac07] Fahiem BACCHUS. « GAC Via Unit Propagation ». In *Principles and Practice of Constraint Programming - CP 2007*, pages 133–147, 2007. 41

[BBI12] Paul BEAME, Christopher BECK, and Russell IMPAGLIAZZO. « Time-space tradeoffs in resolution: superpolynomial lower bounds for superlinear space ». In Howard J. KARLOFF and Toniann PITASSI, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 213–232. ACM, 2012. 102

[BBR09] Olivier BAILLEUX, Yacine BOUFKHAD, and Olivier ROUSSEL. « New Encodings of Pseudo-Boolean Constraints into CNF ». In *Theory and Applications of Satisfiability Testing - SAT 2009*, pages 181–194, 2009. 41

[BCMS14] Simone BOVA, Florent CAPELLI, Stefan MENGEL, and Friedrich SLIVOVSKY. « Expander CNFs have Exponential DNNF Size ». *CoRR*, abs/1411.1995, 2014. 4, 13, 38, 46, 143, 144

[BCMS16] Simone BOVA, Florent CAPELLI, Stefan MENGEL, and Friedrich SLIVOVSKY. « Knowledge Compilation Meets Communication Complexity ». In Subbarao KAMBHAMPATI,

editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1008–1014. IJCAI/AAAI Press, 2016. 5, 14, 32, 33, 34, 48, 55, 138, 142, 143

[Ben02] Eli BEN-SASSON. « Hard examples for the bounded depth Frege proof system ». *Comput. Complex.*, 11(3-4):109–136, 2002. 103

[BHK01] László BABAI, Thomas P. HAYES, and Peter G. KIMMEL. « The Cost of the Missing Bit: Communication Complexity with Help ». *Combinatorica*, 21(4):455–488, 2001. 62

[BHMR99] Endre BOROS, Peter L. HAMMER, Michel MINOUX, and David J. RADER. « Optimal Cell Flipping to Minimize Channel Density in VLSI Design and Pseudo-Boolean Optimization ». *Discret. Appl. Math.*, 90(1-3):69–88, 1999. 41

[BHMS84] R. K. BRAYTON, G. D. HACHTEL, C. T. MCMULLEN, and A. L. SANGIOVANNI-VINCENTELLI. *Logic Minimization Algorithms for VLSI Synthesis*, volume 2 of *The Kluwer International Series in Engineering and Computer Science*. Springer, 1984.

[BI13] Christopher BECK and Russell IMPAGLIAZZO. « Strong ETH holds for regular resolution ». In Dan BONEH, Tim ROUGHGARDEN, and Joan FEIGENBAUM, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 487–494. ACM, 2013. 102

[BK06] Hans L. BODLAENDER and Arie M. C. A. KOSTER. « Safe separators for treewidth ». *Discret. Math.*, 306(3):337–350, 2006. 52, 53, 82

[BLS02] Randal E. BRYANT, Shuvendu K. LAHIRI, and Sanjit A. SESHIA. « Deciding CLU Logic formulas via Boolean and Pseudo-Boolean encodings ». In *Intl. Workshop on Constraints in Formal Verification, CFV*, 2002. 41

[BLW95] Beate BOLLIG, Martin LÖBBING, and Ingo WEGENER. « Simulated Annealing to Improve Variable Orderings for OBDDs ». In *IN INT'L WORKSHOP ON LOGIC SYNTH*, pages 5–5, 1995. 78

[BM12] Lucas BORDEAUX and João MARQUES-SILVA. « Knowledge Compilation with Empowerment ». In *Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM*, pages 612–624, 2012. 41

[BN21] Sam BUSS and Jakob NORDSTRÖM. « Proof Complexity and SAT Solving ». *2nd edition of Handbook of Satisfiability*, 2021.

[Bod96] Hans L. BODLAENDER. « A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth ». *SIAM J. Comput.*, 25(6):1305–1317, 1996. 39

[Bod98] Hans L. BODLAENDER. « A Partial k-Arboretum of Graphs with Bounded Treewidth ». *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. 39

[Bov16] Simone BOVA. « SDDs Are Exponentially More Succinct than OBDDs ». In Dale SCHUURMANS and Michael P. WELLMAN, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 929–935. AAAI Press, 2016. 28

[Bre13] Alain BRETTO. *Hypergraph Theory - An Introduction*. Springer International Publishing, 2013. 129

[Bry86] Randal E. BRYANT. « Graph-Based Algorithms for Boolean Function Manipulation ». *IEEE Trans. Computers*, 35(8):677–691, 1986. 3, 12, 58, 74

[BS17a] Simone BOVA and Stefan SZEIDER. « Circuit Treewidth, Sentential Decision, and Query Compilation ». In Emanuel SALLINGER, Jan Van den BUSSCHE, and Floris GEERTS, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 233–246. ACM, 2017. 4, 13, 39, 40

[BS17b] Simone BOVA and Stefan SZEIDER. « Circuit Treewidth, Sentential Decision, and Query Compilation ». In Emanuel SALLINGER, Jan Van den BUSSCHE, and Floris GEERTS, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 233–246. ACM, 2017. 4, 13, 39, 40, 46

[BSW02] Beate BOLLIG, Martin SAUERHOFF, and Ingo WEGENER. « On the Nonapproximability of Boolean Functions by OBDDs and Read-k-Times Branching Programs ». *Inf. Comput.*, 178(1):263–278, 2002. 5, 14, 59, 60, 61, 62

[BW96] Beate BOLLIG and Ingo WEGENER. « Improving the Variable Ordering of OBDDs Is NP-Complete ». *IEEE Trans. Computers*, 45(9):993–1002, 1996. 75

[BW01] Eli BEN-SASSON and Avi WIGDERSON. « Short proofs are narrow - resolution made simple ». *J. ACM*, 48(2):149–169, 2001. 103

[Cap16] Florent CAPELLI. « *Structural restriction of CNF-formulas: application to model counting and knowledge compilation* ». PhD thesis, 2016. 38, 39, 40, 143

[CBLM05] Sylvie COSTE-MARQUIS, Daniel Le BERRE, Florian LETOMBE, and Pierre MARQUIS. « Propositional Fragments for Knowledge Compilation and Quantified Boolean Formulae ». In Manuela M. VELOSO and Subbarao KAMBHAMPATI, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 288–293. AAAI Press / The MIT Press, 2005. 40, 112

[CC13] Chandra CHEKURI and Julia CHUZHOY. « Large-treewidth graph decompositions and applications ». In Dan BONEH, Tim ROUGHGARDEN, and Joan FEIGENBAUM, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 291–300. ACM, 2013. 90, 91

[CD03] Hei CHAN and Adnan DARWICHE. « Reasoning about Bayesian Network Classifiers ». In *Conference in Uncertainty in Artificial Intelligence, UAI*, pages 107–115, 2003. 58

[CD06] Hei CHAN and Adnan DARWICHE. « On the Robustness of Most Probable Explanations ». In *UAI*. AUAI Press, 2006. 32, 139

[CD08] Mark CHAVIRA and Adnan DARWICHE. « On probabilistic inference by weighted model counting ». *Artif. Intell.*, 172(6-7):772–799, 2008. 137

[CD13] Arthur CHOI and Adnan DARWICHE. « Dynamic Minimization of Sentential Decision Diagrams ». In Marie DESJARDINS and Michael L. LITTMAN, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013. 6, 15, 28, 74, 75, 77, 78

[CDLS02] Marco CADOLI, Francesco M. DONINI, Paolo LIBERATORE, and Marco SCHAERF. « Preprocessing of Intractable Problems ». *Inf. Comput.*, 176(2):89–120, 2002.

[CKD13] Arthur CHOI, Doga KISA, and Adnan DARWICHE. « Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams ». In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU*, volume 7958, pages 121–132, 2013. 58, 137

[CLLM06] Sylvie COSTE-MARQUIS, Daniel LE, Florian LETOMBE, and Pierre MARQUIS. « Complexity Results for Quantified Boolean Formulae Based on Complete Propositional Languages ». *J. Satisf. Boolean Model. Comput.*, 1(1):61–88, 2006.

[CM78] A.K. CHANDRA and G. MARKOWSKY. « On the number of prime implicants ». *Discrete Mathematics*, 24:7–11, 1978. 116

[CM19] Florent CAPELLI and Stefan MENGEL. « Tractable QBF by Knowledge Compilation ». In Rolf NIEDERMEIER and Christophe PAUL, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 1, 11

[CS21] Florent CAPELLI and Yann STROZECKI. « Enumerating models of DNF faster: Breaking the dependency on the formula size ». *Discret. Appl. Math.*, 303:203–215, 2021. 8, 16, 112

[CT20] Karine CHUBARIAN and György TURÁN. « Interpretability of Bayesian Network Classifiers: OBDD Approximation and Polynomial Threshold Functions ». In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2020, Fort Lauderdale, Florida, USA, January 6-8, 2020*, 2020. 58, 59, 64, 71

[Dar01a] Adnan DARWICHE. « Decomposable negation normal form ». *J. ACM*, 48(4):608–647, 2001. 1, 2, 3, 4, 5, 11, 12, 13, 14, 24, 39, 53, 58, 59

[Dar01b] Adnan DARWICHE. « On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision ». *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001. 39, 58

[Dar01c] Adnan DARWICHE. « Recursive conditioning ». *Artif. Intell.*, 126(1-2):5–41, 2001.

[Dar02a] Adnan DARWICHE. « A Compiler for Deterministic, Decomposable Negation Normal Form ». In Rina DECHTER, Michael J. KEARNS, and Richard S. SUTTON, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 627–634. AAAI Press / The MIT Press, 2002. 29, 38, 69

[Dar02b] Adnan DARWICHE. « A Logical Approach to Factoring Belief Networks ». In Dieter FENSEL, Fausto GIUNCHIGLIA, Deborah L. MCGUINNESS, and Mary-Anne WILLIAMS,

editors, *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, pages 409–420. Morgan Kaufmann, 2002. 9, 16

[Dar03] Adnan DARWICHE. « A differential approach to inference in Bayesian networks ». *J. ACM*, 50(3):280–305, 2003. 9, 16, 137

[Dar04] Adnan DARWICHE. « New Advances in Compiling CNF into Decomposable Negation Normal Form ». In Ramón López de MÁNTARAS and Lorenza SAITTA, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 328–332. IOS Press, 2004. 29

[Dar11] Adnan DARWICHE. « SDD: A New Canonical Representation of Propositional Knowledge Bases ». In Toby WALSH, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 819–826. IJCAI/AAAI, 2011. 1, 3, 6, 11, 12, 14, 27, 39, 40, 58, 74

[Dar22] Adnan DARWICHE. « Tractable Boolean and Arithmetic Circuits ». *CoRR*, abs/2202.02942, 2022. 1, 11

[dC20] Alexis de COLNET. « A Lower Bound on DNNF Encodings of Pseudo-Boolean Constraints ». In Luca PULINA and Martina SEIDL, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 2020. 4, 13, 41, 64

[dCM20] Alexis de COLNET and Stefan MENGEL. « Lower Bounds for Approximate Knowledge Compilation ». In Christian BESSIERE, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1834–1840. ijcai.org, 2020. 4, 13, 60

[dCM21a] Alexis de COLNET and Stefan MENGEL. « Characterizing Tseitin-Formulas with Short Regular Resolution Refutations ». In Chu-Min LI and Felip MANYÀ, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2021. 4, 8, 13, 15, 41, 47, 56

[dCM21b] Alexis de COLNET and Stefan MENGEL. « A Compilation of Succinctness Results for Arithmetic Circuits ». In Meghyn BIENVENU, Gerhard LAKEMEYER, and Esra ERDEM, editors, *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, pages 205–215, 2021. 9, 16, 139

[dCM22a] Alexis de COLNET and Pierre MARQUIS. « On the Complexity of Enumerating Prime Implicants from dec-DNNF Circuits ». In *To appear in the proceedings of IJCAI 2022*, 2022. 8, 16, 113

[dCM22b] Alexis de COLNET and Stefan MENGEL. « Lower Bounds on Intermediate Results in Bottom-Up Compilation ». In *To appear in the proceedings of AAAI 2022*, Lecture Notes in Computer Science, 2022. 6, 14, 79, 80

[Den16] Aaron W. DENNIS. « *Algorithms for Learning the Structure of Monotone and Nonmonotone Sum-Product Networks* ». PhD thesis, Brigham Young University, 2016. 137

[DF59] B. DUNHAM and B. FRIDSHAL. « The problem of simplifying logical expressions ». *Journal of Symbolic Logic*, 1959. 116

[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):21, 2007.

[DH20] A. DARWICHE and A. HIRTH. « On the Reasons Behind Decisions ». In *Proc. of ECAI'20*, pages 712–720, 2020. 9, 16, 116, 128

[DHJ+04] Pavol DURIS, Juraj HROMKOVIC, Stasys JUKNA, Martin SAUERHOFF, and Georg SCHNITGER. « On multi-partition communication complexity ». *Inf. Comput.*, 194(1):49–75, 2004. 64, 65

[Die12] Reinhard DIESTEL. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. 53

[dK86] J. de KLEER. « An assumption-based TMS ». *Artificial Intelligence*, 28:127–167, 1986.

[dKMR92] J. de KLEER, A. K. MACKWORTH, and R. REITER. « Characterizing diagnoses and systems ». *Artificial Intelligence*, 56:197–222, 1992. 116

[DLL62] Martin DAVIS, George LOGEMANN, and Donald W. LOVELAND. « A machine program for theorem-proving ». *Commun. ACM*, 5(7):394–397, 1962. 102

[DM02] Adnan DARWICHE and Pierre MARQUIS. « A Knowledge Compilation Map ». *J. Artif. Intell. Res.*, 17:229–264, 2002. 1, 3, 5, 8, 11, 13, 16, 24, 26, 28, 29, 54, 62, 112, 114, 115, 117, 128, 138, 139, 143, 144, 150

[DM07] Rina DECHTER and Robert MATEESCU. « AND/OR search spaces for graphical models ». *Artif. Intell.*, 171(2-3):73–106, 2007. 136

[DM21] A. DARWICHE and P. MARQUIS. « On Quantifying Literals in Boolean Logic and Its Applications to Explainable AI ». *J. Artif. Intell. Res.*, 72:285–328, 2021. 128

[DP60] Martin DAVIS and Hilary PUTNAM. « A Computing Procedure for Quantification Theory ». *J. ACM*, 7(3):201–215, 1960. 102

[EG95] Th. EITER and G. GOTTLOB. « The Complexity of Logic-Based Abduction ». *Journal of the ACM*, 42(1):3–42, 1995. 9, 16, 115, 126

[EMG08] Th. EITER, K. MAKINO, and G. GOTTLOB. « Computational aspects of monotone dualization: A brief survey ». *Discret. Appl. Math.*, 156(11):2035–2049, 2008.

[FK96] Michael L. FREDMAN and Leonid KHACHIYAN. « On the Complexity of Dualization of Monotone Disjunctive Normal Forms ». *J. Algorithms*, 21(3):618–628, 1996. 135

[FM06] Hélène FARGIER and Pierre MARQUIS. « On the Use of Partially Ordered Decision Graphs in Knowledge Compilation and Quantified Boolean Formulae ». In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 42–47. AAAI Press, 2006. 112

[FM14] Hélène FARGIER and Pierre MARQUIS. « Disjunctive closures for knowledge compilation ». *Artif. Intell.*, 216:129–162, 2014. 114, 115

[FX13] Luke FRIEDMAN and Yixin XU. « Exponential Lower Bounds for Refuting Random Formulas Using Ordered Binary Decision Diagrams ». In Andrei A. BULATOV and Arseny M. SHUR, editors, *Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings*, volume 7913 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2013. 6, 14, 78

[GI17] Ludmila GLINSKIH and Dmitry ITSYKSON. « Satisfiable Tseitin Formulas Are Hard for Nondeterministic Read-Once Branching Programs ». In Kim G. LARSEN, Hans L. BODLAENDER, and Jean-François RASKIN, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. 35

[GI21] Ludmila GLINSKIH and Dmitry ITSYKSON. « On Tseitin Formulas, Read-Once Branching Programs and Treewidth ». *Theory Comput. Syst.*, 65(3):613–633, 2021.

[GIRS19] Nicola GALESI, Dmitry ITSYKSON, Artur RIAZANOV, and Anastasia SOFRONOVA. « Bounded-Depth Frege Complexity of Tseitin Formulas for All Graphs ». In Peter ROSSMANITH, Pinar HEGGERNES, and Joost-Pieter KATOEN, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 49:1–49:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 103

[GK98] Dima GRIGORIEV and Marek KARPINSKI. « An Exponential Lower Bound for Depth 3 Arithmetic Circuits ». In Jeffrey Scott VITTER, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 577–582. ACM, 1998. 138

[GK99] V. GURVICH and L. KHACHIYAN. « On Generating the Irredundant Conjunctive and Disjunctive Normal Forms of Monotone Boolean Functions ». *Discret. Appl. Math.*, 96-97:363–373, 1999. 128

[GKM+11] Parikshit GOPALAN, Adam R. KLIVANS, Raghu MEKA, Daniel STEFANKOVIC, Santosh S. VEMPALA, and Eric VIGODA. « An FPTAS for #Knapsack and Related Counting Problems ». In *IEEE Symposium on Foundations of Computer Science, FOCS*, pages 817–826, 2011. 41, 58, 59, 64

[GM94a] J. GERGOV and C. MEINEL. « Efficient analysis and manipulation of OBDDs can be extended to FBDDs ». *IEEE Transactions on Computers*, 43(10):1197–1209, 1994.

[GM94b] Jordan GERGOV and Christoph MEINEL. « On the Complexity of Analysis and Manipulation of Boolean Functions in Terms of Decision Graphs ». *Inf. Process. Lett.*, 50(6):317–322, 1994. 78

[GM09] Martin GROHE and Dániel MARX. « On tree width, bramble size, and expansion ». *J. Comb. Theory, Ser. B*, 99(1):218–228, 2009. 70, 79, 80

[GTT20] Nicola GALESI, Navid TALEBANFARD, and Jacobo TORÁN. « Cops-Robber Games and the Resolution of Tseitin Formulas ». *ACM Trans. Comput. Theory*, 12(2):9:1–9:22, 2020. 8, 15, 103

[HCD06] Jinbo HUANG, Mark CHAVIRA, and Adnan DARWICHE. « Solving MAP Exactly by Searching on Compiled Arithmetic Circuits ». In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 1143–1148. AAAI Press, 2006. 137

[HD04] Jinbo HUANG and Adnan DARWICHE. « Using DPLL for Efficient OBDD Construction ». In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004. 38, 77

[HD05] Jinbo HUANG and Adnan DARWICHE. « DPLL with a Trace: From SAT to Knowledge Compilation ». In Leslie Pack KAELBLING and Alessandro SAFFIOTTI, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 156–162. Professional Book Center, 2005. 5, 14, 74

[HD07] Jinbo HUANG and Adnan DARWICHE. « The Language of Search ». *J. Artif. Intell. Res.*, 29:191–219, 2007. 25, 40

[HLW06] Shlomo HOORY, Nathan LINIAL, and Avi WIGDERSON. « Expander graphs and their applications ». *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006. 79, 103

[HTKY97] Kazuhisa HOSAKA, Yasuhiko TAKENAGA, T. KANEDA, and Shuzo YAJIMA. « Size of Ordered Binary Decision Diagrams Representing Threshold Functions ». *Theor. Comput. Sci.*, 180(1-2):47–60, 1997. 40, 41, 42, 64

[HW17] Daniel J. HARVEY and David R. WOOD. « Parameters Tied to Treewidth ». *J. Graph Theory*, 84(4):364–385, 2017. 20, 91

[IKRS20] Dmitry ITSYKSON, Alexander KNOP, Andrei E. ROMASHCHENKO, and Dmitry SOKOLOV. « On OBDD-based Algorithms and Proof Systems that Dynamically Change the order of Variables ». *J. Symb. Log.*, 85(2):632–670, 2020. 6, 7, 14, 15, 78

[INAM20] A. IGNATIEV, N. NARODYTSKA, N. ASHER, and J. MARQUES-SILVA. « On Relating 'Why?' and 'Why Not?' Explanations ». *CoRR*, abs/2012.11067, 2020. 128

[INM19] A. IGNATIEV, N. NARODYTSKA, and J. MARQUES-SILVA. « Abduction-Based Explanations for Machine Learning Models ». In *Proc. of AAAI'19*, pages 1511–1519, 2019. 9, 16, 128

[IO13] Dmitry ITSYKSON and Vsevolod OPARIN. « Graph Expansion, Tseitin Formulas and Resolution Proofs for CSP ». In Andrei A. BULATOV and Arseny M. SHUR, editors, *Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings*, volume 7913 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2013. 8, 15, 103

[IRS22] Dmitry ITSYKSON, Artur RIAZANOV, and Petr SMIRNOV. « Tight Bounds for Tseitin Formulas ». In *To appear in the proceedings of SAT 2022*, 2022. 46, 47, 56, 80, 99, 109

[IRSS21] Dmitry ITSYKSON, Artur RIAZANOV, Danil SAGUNOV, and Petr SMIRNOV. « Near-Optimal Lower Bounds on Regular Resolution Refutations of Tseitin Formulas for All Constant-Degree Graphs ». *Comput. Complex.*, 30(2):13, 2021. 8, 15, 46, 103, 104, 105, 106, 107

[Ivă65] Peter L IVĂNESCU. « Some network flow problems solved with pseudo-boolean programming ». *Operations Research*, 13(3):388–399, 1965. 41

[JS82] Mark JERRUM and Marc SNIR. « Some Exact Complexity Results for Straight-Line Computations over Semirings ». *J. ACM*, 29(3):874–897, 1982. 137

[JS12] Abhay Kumar JHA and Dan SUCIU. « On the tractability of query compilation and bounded treewidth ». In Alin DEUTSCH, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 249–261. ACM, 2012. 4, 13, 39

[KBLM16] Frédéric KORICHE, Daniel Le BERRE, Emmanuel LONCA, and Pierre MARQUIS. « Fixed-Parameter Tractable Optimization Under DNNF Constraints ». In Gal A. KAMINKA, Maria FOX, Paolo BOUQUET, Eyke HÜLLERMEIER, Virginia DIGNUM, Frank DIGNUM, and Frank van HARMELEN, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1194–1202. IOS Press, 2016. 112

[KCL+19] Pasha KHOSRAVI, YooJung CHOI, Yitao LIANG, Antonio VERGARI, and Guy Van den BROECK. « On Tractable Computation of Expected Predictions ». In Hanna M. WALLACH, Hugo LAROCHELLE, Alina BEYGELZIMER, Florence D'ALCHÉ-BUC, Emily B. FOX, and Roman GARNETT, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11167–11178, 2019. 137

[KdBCD14] Doga KISA, Guy Van den BROECK, Arthur CHOI, and Adnan DARWICHE. « Probabilistic Sentential Decision Diagrams ». In Chitta BARAL, Giuseppe De GIACOMO, and Thomas EITER, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014. 136, 137, 138, 139

[KLMT13a] F. KORICHE, J.-M. LAGNIEZ, P. MARQUIS, and S. THOMAS. « Knowledge Compilation for Model Counting: Affine Decision Trees ». In *Proc. of IJCAI'13*, pages 947–953, 2013. 128

[KLMT13b] Frédéric KORICHE, Jean-Marie LAGNIEZ, Pierre MARQUIS, and Samuel THOMAS. « Knowledge Compilation for Model Counting: Affine Decision Trees ». In Francesca ROSSI, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 947–953. IJCAI/AAAI, 2013. 29

[KN97] Eyal KUSHILEVITZ and Noam NISAN. *Communication complexity*. Cambridge University Press, 1997. 61

[KPS93] Dimitris J. KAVVADIAS, Christos H. PAPADIMITRIOU, and Martha SIDERI. « On Horn Envelopes and Hypergraph Transversals ». In Kam-Wing NG, Prabhakar RAGHAVAN, N. V. BALASUBRAMANIAN, and Francis Y. L. CHIN, editors, *Algorithms and Computation, 4th International Symposium, ISAAC '93, Hong Kong, December 15-17, 1993, Proceedings*, volume 762 of *Lecture Notes in Computer Science*, pages 399–405. Springer, 1993. 58, 129

[Kra95] Jan KRAJÍCEK. *Bounded arithmetic, propositional logic, and complexity theory*, volume 60 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1995. 103, 104

[Kra08] Jan KRAJÍCEK. « An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams ». *J. Symb. Log.*, 73(1):227–237, 2008. 6, 14, 78

[KS96] David R. KARGER and Clifford STEIN. « A New Approach to the Minimum Cut Problem ». *J. ACM*, 43(4):601–640, 1996. 59

[KS19] Petr KUČERA and Petr SAVICKÝ. « Propagation complete encodings of smooth DNNF theories ». *CoRR*, abs/1909.06673, 2019. 4, 13, 41, 56

[KSW99] Matthias KRAUSE, Petr SAVICKÝ, and Ingo WEGENER. « Approximations by OBDDs and the Variable Ordering Problem ». In *International Colloquium Automata, Languages and Programming, ICALP*, pages 493–502, 1999. 59, 60

[LD08] Daniel LOWD and Pedro M. DOMINGOS. « Learning Arithmetic Circuits ». In David A. MCALLESTER and Petri MYLLYMÄKI, editors, *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*, pages 383–392. AUAI Press, 2008. 137

[LLM16] Jean-Marie LAGNIEZ, Emmanuel LONCA, and Pierre MARQUIS. « Improving Model Counting by Leveraging Definability ». In Subbarao KAMBHAMPATI, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 751–757. IJCAI/AAAI Press, 2016.

[LM17] Jean-Marie LAGNIEZ and Pierre MARQUIS. « An Improved Decision-DNNF Compiler ». In Carles SIERRA, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 667–673. ijcai.org, 2017. 1, 11, 29, 38, 81

[LMMW18] Daniel LE BERRE, Pierre MARQUIS, Stefan MENGEL, and Romain WALLON. « Pseudo-Boolean Constraints from a Knowledge Representation Perspective ». In *International Joint Conference on Artificial Intelligence, IJCAI*, pages 1891–1897, 2018. 40, 42

[Mar93] P. MARQUIS. « Skeptical Abduction ». *Int. J. Artif. Intell. Tools*, 2(4):511–540, 1993. 117

[Mar00] P. MARQUIS. « *Consequence finding algorithms* », volume 5 of *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, Chapter 2, pages 41–145. Kluwer Academic Publisher, 2000.

[MC91] J.-C. MADRE and O. COUDERT. « A Logically Complete Reasoning Maintenance System Based on a Logical Constraint Solver ». In *Proc. of IJCAI'91*, pages 294–299, 1991. 117

[McC56] E.J. McCluskey. « Minimization of Boolean functions ». *Bell System Technical Journal*, 35(6):1445–1446, 1956.

[Men16] Stefan Mengel. « Parameterized Compilation Lower Bounds for Restricted CNF-Formulas ». In *International Conference Theory and Applications of Satisfiability Testing, SAT*, 2016. 4, 13, 65

[MGC+21] J. Marques-Silva, Th. Gerspacher, M. C. Cooper, A. Ignatiev, and N. Narodytska. « Explanations for Monotonic Classifiers ». In *Proc. of ICML'21*, pages 7469–7479, 2021. 128

[Mil19] T. Miller. « Explanation in artificial intelligence: Insights from the social sciences ». *Artificial Intelligence*, 267:1–38, 2019. 116

[MM14] James Martens and Venkatesh Medabalimi. « On the Expressive Efficiency of Sum Product Networks ». *CoRR*, abs/1411.7717, 2014. 147

[Mol19] Ch. Molnar. *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable*. Leanpub, 2019. 116

[MW19] Stefan Mengel and Romain Wallon. « Revisiting Graph Width Measures for CNF-Encodings ». In *Theory and Applications of Satisfiability Testing, SAT*, 2019. 45

[NKR+18] N. Narodytska, S. Prasad Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. « Verifying Properties of Binarized Deep Neural Networks ». In *Proc. of AAAI'18*, pages 6615–6624, 2018. 116

[Nor17] Joakim Alme Nordstrand. « *Exploring Graph Parameters Similar to Tree-Width and Path-Width* ». PhD thesis, University of Bergen, Department of Informatics, 2017. 20

[NW97] Noam Nisan and Avi Wigderson. « Lower Bounds on Arithmetic Circuits Via Partial Derivatives ». *Comput. Complex.*, 6(3):217–234, 1997. 9, 16

[NW07] Nina Narodytska and Toby Walsh. « Constraint and Variable Ordering Heuristics for Compiling Configuration Problems ». In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 149–154, 2007. 6, 14, 77

[OD14] U. Oztok and A. Darwiche. « On Compiling CNF into Decision-DNNF ». In *Proc. of CP'14*, pages 42–57, 2014.

[OD15] Umut Oztok and Adnan Darwiche. « A Top-Down Compiler for Sentential Decision Diagrams ». In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3141–3148. AAAI Press, 2015. 28, 29, 38

[Pap77] Spiros G. Papaioannou. « Optimal Test Generation in Combinational Networks by Pseudo-Boolean Programming ». *IEEE Trans. Computers*, 26(6):553–560, 1977. 41

[PD07] Knot Pipatsrisawat and Adnan Darwiche. « Clone: Solving Weighted Max-SAT in a Reduced Search Space ». In Mehmet A. Orgun and John Thornton, editors, *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial*

Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings, volume 4830 of *Lecture Notes in Computer Science*, pages 223–233. Springer, 2007. 1, 5, 11, 14, 58

[PD08] Knot PIPATSRISAWAT and Adnan DARWICHE. « New Compilation Languages Based on Structured Decomposability ». In Dieter FOX and Carla P. GOMES, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 517–522. AAAI Press, 2008. 2, 5, 12, 14, 24, 74, 138, 147

[PD10] Knot PIPATSRISAWAT and Adnan DARWICHE. « Top-Down Algorithms for Constructing Structured DNNF: Theoretical and Practical Implications ». In Helder COELHO, Rudi STUDER, and Michael J. WOOLDRIDGE, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 3–8. IOS Press, 2010. 38

[PD11] Hoifung POON and Pedro M. DOMINGOS. « Sum-product networks: A new deep architecture ». In *IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011*, pages 689–690. IEEE Computer Society, 2011. 136, 137

[Pre21] Steven PRESTWICH. « CNF Encodings ». *2nd edition of Handbook of Satisfiability*, 2021. 63

[Pro90] G.M. PROVAN. « The computational complexity of multiple-context truth maintenance systems ». In *Proc. of the 9th European Conference on Artificial Intelligence (ECAI'90)*, pages 522–527, Stockholm, 1990.

[PS95] Shipra PANDA and Fabio SOMENZI. « Who are the variables in your neighborhood ». In Richard L. RUDELL, editor, *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1995, San Jose, California, USA, November 5-9, 1995*, pages 74–77. IEEE Computer Society / ACM, 1995. 78

[PSP94] Shipra PANDA, Fabio SOMENZI, and Bernard PLESSIER. « Symmetry detection and dynamic variable ordering of decision diagrams ». In Jochen A. G. JESS and Richard L. RUDELL, editors, *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1994, San Jose, California, USA, November 6-10, 1994*, pages 628–631. IEEE Computer Society / ACM, 1994. 78

[PTPD15] Robert PEHARZ, Sebastian TSCHIATSCHEK, Franz PERNKOPF, and Pedro M. DOMINGOS. « On Theoretical Properties of Sum-Product Networks ». In Guy LEBANON and S. V. N. VISHWANATHAN, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, volume 38 of *JMLR Workshop and Conference Proceedings*. JMLR.org, 2015. 139, 140, 143, 145, 146

[Qui52] W.V.O. QUINE. « The problem of simplifying truth functions ». *American Mathematical Monthly*, 59:521–531, 1952.

[Qui55] W.V.O. QUINE. « A way to simplify truth functions ». *American Mathematical Monthly*, 62:627–631, 1955.

[Qui59] W.V.O. QUINE. « On cores and prime implicants of truth functions ». *American Mathematical Monthly*, 66:755–760, 1959.

[Raz09] Ran RAZ. « Multi-linear formulas for permanent and determinant are of super-polynomial size ». *J. ACM*, 56(2):8:1–8:17, 2009. 137, 138

[Raz10] Ran RAZ. « Elusive Functions and Lower Bounds for Arithmetic Circuits ». *Theory Comput.*, 6(1):135–177, 2010. 138

[Raz14] Igor RAZGON. « On OBDDs for CNFs of Bounded Treewidth ». In Chitta BARAL, Giuseppe De GIACOMO, and Thomas EITER, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014. 40

[Raz16] Igor RAZGON. « On the Read-Once Property of Branching Programs and CNFs of Bounded Treewidth ». *Algorithmica*, 75(2):277–294, 2016. 47, 56

[Raz21] Igor RAZGON. « Classification of OBDD Size for Monotone 2-CNFs ». In Petr A. GOLOVACH and Meirav ZEHAVI, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 25:1–25:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 40, 46

[RdK87] R. REITER and J. de KLEER. « Foundations of assumption-based truth maintenance systems: preliminary report ». In *Proc. of AAAI'87*, pages 183–188, 1987. 115

[RKG14] Tahrima RAHMAN, Prasanna KOTHALKAR, and Vibhav GOGATE. « Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees ». In Toon CALDERS, Floriana ESPOSITO, Eyke HÜLLERMEIER, and Rosa MEO, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II*, volume 8725 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2014. 136

[RL16] Amirmohammad ROOSHENAS and Daniel LOWD. « Discriminative Structure Learning of Arithmetic Circuits ». In Dale SCHUURMANS and Michael P. WELLMAN, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 4258–4259. AAAI Press, 2016. 137

[RP13] Igor RAZGON and Justyna PETKE. « Cliquewidth and Knowledge Compilation ». In Matti JÄRVISALO and Allen Van GELDER, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 335–350. Springer, 2013. 4, 13, 39

[Rud93] Richard RUDELL. « Dynamic variable ordering for ordered binary decision diagrams ». In Michael R. LIGHTNER and Jochen A. G. JESS, editors, *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993, Santa Clara, California, USA, November 7-11, 1993*, pages 42–47. IEEE Computer Society / ACM, 1993. 6, 15, 78

[RY11] Ran RAZ and Amir YEHUDAYOFF. « Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors ». *J. Comput. Syst. Sci.*, 77(1):167–190, 2011. 147

[Sau03] Martin SAUERHOFF. « Approximation of boolean functions by combinatorial rectangles ». *Theor. Comput. Sci.*, 301(1-3):45–78, 2003. 143

[SCD18a] A. SHIH, A. CHOI, and A. DARWICHE. « Formal Verification of Bayesian Network Classifiers ». In *Proc. of PGM'18*, pages 427–438, 2018. 116

[SCD18b] A. SHIH, A. CHOI, and A. DARWICHE. « A Symbolic Approach to Explaining Bayesian Network Classifiers ». In *Proc. of IJCAI'18*, pages 5103–5111, 2018. 9, 16, 128

[SCD19a] A. SHIH, A. CHOI, and A. DARWICHE. « Compiling Bayesian Networks into Decision Graphs ». In *Proc. of AAAI'19*, pages 7966–7974, 2019. 116

[SCD19b] Andy SHIH, Arthur CHOI, and Adnan DARWICHE. « Compiling Bayesian Network Classifiers into Decision Graphs ». In *AAAI Conference on Artificial Intelligence, AAAI*, pages 7966–7974, 2019. 58

[Sch89] Petra SCHEFFLER. « *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme* ». PhD thesis, 12 1989. 54

[Sch15] Nicolas SCHMIDT. « *Compilation de préférences application à la configuration de produit* ». PhD thesis, 2015. 78

[Seg08] Nathan SEGERLIND. « On the Relative Efficiency of Resolution-Like Proofs and Ordered Binary Decision Diagram Proofs ». In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity, CCC 2008, 23-26 June 2008, College Park, Maryland, USA*, pages 100–111. IEEE Computer Society, 2008. 6, 14, 78

[Seg14] Luc SEGOUFIN. « A glimpse on constant delay enumeration (Invited Talk) ». In Ernst W. MAYR and Natacha PORTIER, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 13–27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. 114

[Sie02] Detlef SIELING. « The Nonapproximability of OBDD Minimization ». *Inf. Comput.*, 172(2):103–138, 2002. 75

[Sin02] Carsten SINZ. « Knowledge compilation for product configuration ». In *Proceedings of the Configuration Workshop, 15th European Conference on Artificial Intelligence*, pages 23–26, 2002. 1, 11

[SK91] Bart SELMAN and Henry A. KAUTZ. « Knowledge Compilation using Horn Approximations ». In Thomas L. DEAN and Kathleen R. MCKEOWN, editors, *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 2*, pages 904–909. AAAI Press / The MIT Press, 1991. 4, 13, 38, 57, 58, 59

[SK96] Bart SELMAN and Henry A. KAUTZ. « Knowledge Compilation and Theory Approximation ». *J. ACM*, 43(2):193–224, 1996. 1, 4, 5, 11, 13, 14, 38, 57, 58, 59, 72

[SL90] B. SELMAN and H. LEVESQUE. « Abductive and default reasoning: a computational core ». In *Proc. of AAAI'90*, pages 343–348, 1990. 9, 16, 115, 126

[Som09] Fabio SOMENZI. « CUDD: CU decision diagram package-release 2.4. 0 ». *University of Colorado at Boulder*, 2009. 74, 77

[Str19] Y. STROZECKI. « Enumeration Complexity ». *Bull. EATCS*, 129, 2019. 8, 9, 16, 112, 113, 114

[STV14] Sigve Hortemo SÆTHER, Jan Arne TELLE, and Martin VATSHELLE. « Solving MaxSAT and #SAT on Structured CNF Formulas ». In Carsten SINZ and Uwe EGLY, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2014. 4, 13, 39

[SW95] Detlef SIELING and Ingo WEGENER. « Graph Driven BDDs - A New Data Structure for Boolean Functions ». *Theor. Comput. Sci.*, 141(1&2):283–310, 1995. 3, 12

[SY10] Amir SHPILKA and Amir YEHUDAYOFF. « Arithmetic Circuits: A survey of recent results and open questions ». *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388, 2010. 137, 138

[Tie70] James C. TIERNAN. « An efficient search algorithm to find the elementary circuits of a graph ». *Commun. ACM*, 13(12):722–726, 1970. 112

[Tse68] GS TSEITIN. « The complexity of a deduction in the propositional predicate calculus ». *Zap. Nauchn. Sem. Leningrad Otd. Mat. Inst. Akad. Nauk SSSR*, 8:234–259, 1968. 7, 15, 102

[Tse83] G. S. TSEITIN. « *On the Complexity of Derivation in Propositional Calculus* », pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. 7, 15, 102

[TSZ10] Olga TVERETINA, Carsten SINZ, and Hans ZANTEMA. « Ordered Binary Decision Diagrams, Pigeonhole Formulas and Beyond ». *J. Satisf. Boolean Model. Comput.*, 7(1):35–58, 2010. 78

[Urq87] Alasdair URQUHART. « Hard examples for resolution ». *J. ACM*, 34(1):209–219, 1987. 8, 15, 35, 102, 103

[Val80] Leslie G. VALIANT. « Negation can be Exponentially Powerful ». *Theor. Comput. Sci.*, 12:303–314, 1980. 1, 9, 10, 11, 16, 17, 136, 137, 145, 147

[VCL+21] Antonio VERGARI, YooJung CHOI, Anji LIU, Stefano TESO, and Guy Van den BROECK. « A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference ». In Marc'Aurelio RANZATO, Alina BEYGELZIMER, Yann N. DAUPHIN, Percy LIANG, and Jennifer Wortman VAUGHAN, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 13189–13201, 2021. 1, 9, 11, 16, 137, 138, 147

[VD15] Guy VAN DEN BROECK and Adnan DARWICHE. « On the Role of Canonicity in Knowledge Compilation ». In Blai BONET and Sven KOENIG, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 1641–1648. AAAI Press, 2015.

[Was16] Kunihiro WASA. « Enumeration of Enumeration Algorithms », 2016. 112

[Weg00] Ingo WEGENER. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000. 3, 12, 13, 45, 59

Résumé

Le thème de la thèse est la compilation de connaissances, une approche pour la résolution de problèmes difficiles à résoudre du point de vue du calcul et qui vise à réduire cette complexité en pré-traitant (en compilant) une partie du problème connue à l'avance et modélisée par une fonction. Il s'agit de trouver une représentation de la fonction dans une classe de représentations appelée langage de compilation. Pour un langage L donné, la taille d'une fonction dans L désigne la taille de sa plus petite représentation dans L. Dans cette thèse, nous étudions différents aspects de la compilation dans le langage DNNF, le langage où les fonctions Booléennes sont représentées par des circuits sous forme normale négative décomposable (DNNF).

Dans la première partie de la thèse, nous montrons des bornes inférieures sur la taille de fonctions particulières dans le langage DNNF. Pour certaines fonctions, notamment des contraintes pseudo-Booléennes, nous obtenons des bornes inférieures par l'application de techniques usuelles pour l'analyse de circuits en DNNF. Pour d'autres fonctions, ces mêmes techniques s'avèrent insuffisantes. En particulier, pour les formules de Tseitin, qui sont des formules CNF représentant des systèmes d'équations linéaires structurés par des graphes, nous avons amélioré les techniques existantes pour montrer une borne exponentielle sur leur taille dans le langage DNNF. Quand la taille d'une fonction dans un langage est trop grande pour envisager sa compilation, on peut tenter de compiler une approximation de la fonction, on parle alors de compilation de connaissances approchée. Nous avons étudié deux notions d'approximation qui offrent des garanties sur l'erreur d'approximation. Pour chacune de ces notions, nous avons trouvé des exemples de fonctions dont toutes les approximations ont une taille trop grande dans de nombreux langages de compilation.

Dans la seconde partie de la thèse, nous donnons des applications des nos bornes inférieures. Ces applications font le lien entre la compilation de connaissances et le domaine de la complexité des preuves. Notamment, nous utilisons notre borne inférieure sur la taille des formules de Tseitin dans le langage DNNF pour obtenir une caractérisation des formules de Tseitin non-satisfiables (dont les graphes sont de degré borné par une constante) pour lesquelles il existe des réfutations par résolution dites régulières de taille polynomiale en le nombre de variables. Une seconde application concerne à la fois certains systèmes de preuve et les compilateurs utilisant la compilation dite ascendante. La compilation ascendante a pour particularité qu'elle construit le circuit représentant la fonction initiale en combinant des circuits intermédiaires. Dans le cas de formules non-satisfiables, la séquence de circuits intermédiaires générés lors d'une compilation ascendante peut être vue comme une réfutation de la formule. Nous analysons des cas de compilations ascendantes de formules (pour certaines non-satisfiables) ayant de petites représentations dans le langage cible, mais pour lesquelles des circuits intermédiaires de taille exponentielle sont inévitables.

Dans les derniers chapitres de la thèse, nous tentons d'élargir le champ de recherche en compilation de connaissances tout en restant dans l'esprit de la « carte de compilation de connaissances », un modèle proposé par Darwiche et Marquis pour comparer les langages de compilation en termes de compacité (quels sont les langages permettant d'obtenir les plus petites représentations ?) et en termes d'efficacité calculatoire (pour quels langages existe-t-il des algorithmes en temps polynomial pour répondre aux requêtes ?). Nous introduisons de nouvelles requêtes pour les langages Booléens, plus précisément des requêtes d'énumération, et nous initions des « cartes de compacité » pour des langages pour des fonctions non-Booléennes.

Abstract

The subject of the thesis is knowledge compilation, an approach for computationally hard problems that aims to work around general lower bounds results by preprocessing (or compiling) an input function. The aim is to generate an equivalent representation of the function in a class of representations called the compilation language. Given a compilation language L, we call "L size of a function" the size of the smallest representation in L for that function. In this thesis, we study different aspects of the compilation language DNNF of Boolean circuits in decomposable negation normal form (DNNF).

In the first part of the thesis, we prove exponential lower bounds on the DNNF size of particular functions and even on the DNNF size of their approximations. For some functions, especially some pseudo-Boolean constraints, we prove lower bounds using classical techniques to analyse circuits in DNNF using tools from communication complexity. But for other functions, the same techniques do not yield good lower bounds. In particular for Tseitin formulas, which are CNF formulas representing systems of parity constraints structured by graphs, we had to improve existing techniques to prove a lower bound on their DNNF sizes that is exponential in the treewidth of the underlying graphs. Since the DNNF size of functions is sometimes an impediment for compilation in practice, we study the complexity of representing approximations of functions in DNNF. We use and compare two notions of approximations that guarantee a bounded error of approximation. For both notions, we give examples of functions that are not only hard to represent exactly in many compilation languages, but whose approximations are also all hard to represent in the same languages.

In the second part of the thesis, we give applications of our lower bounds that establish connections between knowledge compilation and proof complexity theory. First, using our lower bound on the DNNF size of Tseitin formulas, we give a characterization of unsatisfiable Tseitin formulas whose underlying graphs have maximal degree bounded by a constant that have small refutations in the regular resolution proof system. A second application deals with the analysis of compilers that respect the "bottom-up" paradigm. A particularity of bottom-up compilers is to generate intermediate representations or circuits and, for some compilation languages, the sequence of intermediate circuits generated in a bottom-up compilation of an unsatisfiable formula can be seen as a refutation of the formula. We show that bottom-up compilers in a language L can be very inefficient even when the functions to compile have a small representation in L since large intermediate circuits are sometimes unavoidable.

In the last part of the thesis, we explore new horizons for knowledge compilation. We follow the spirit of the "knowledge compilation map", a framework created by Darwiche and Marquis to compare compilation languages in term of succinctness (which languages yields the smaller representations?) and in terms of efficiency for reasoning on function (for which languages are there polynomial-time algorithms for given queries?). We introduce new queries for Boolean languages, more precisely enumeration queries, and we initiate a "succinctness map" for languages for non-Boolean functions.